Institute of Architecture of Application Systems

University of Stuttgart
Universitätsstraße 38
D-70569 Stuttgart

Master Thesis

Concepts and Tools for Measuring the Complexity of Service Choreography Models

Sayan Banerjee

Course of Study:	Computer Science
Examiner:	Prof. Dr. Dr. h.c. Frank Leymann
Supervisor:	Dipl.-Inf. Michael Hahn
Commenced:	December 8, 2017
Completed:	June 8, 2018
CR-Classification:	D.2.8, D.2.11, H.4.1

Abstract

With the increasing acceptance of Service-oriented Architectural (SOA) style to deliver services over the network by businesses in today's competitive world have shaped the two major paradigms to realize service compositions: choreographies and orchestrations. Services are almost ubiquitous in carrying a unit of software functionality to the end user. Although both the paradigms of service composition, namely orchestration and choreography, are devised to execute a conversation protocol between the concerned participants, nevertheless the service choreography designed using a modelling language is quite convoluted. With the increase in importance of data in modern computing, TraDE concepts were adapted for data-aware service choreographies to decouple the data flow from the intricacies of the control flow associated in the message exchange between participants of the choreography. The TraDE middleware assists in controlling the data transfer among participants in a straightforward and automated manner. The service choreography models in this thesis are framed using BPMN 2.0 and BPEL4Chor modelling languages for studying their degree of complexity.

In this research work we analyse existing complexity metrics (custom built for process models) and to which extent they are efficient in predicting the complexity of service choreographies that are modelled by incorporating the TraDE approach. We devise our complexity metrics based on adaption from software engineering, graph theory, and business process model research domain to measure the level of complexity for TraDE concepts applied service choreography model. The complexity metrics we adapted for applying to the service choreography model are Size Metrics, McCabe's Cyclomatic Complexity, Control-flow Complexity, Interface Complexity, Coefficient of Network Complexity, Durfee Square and Perfect Square Metric, Connectivity Level between Activities, Halstead-based Choreography Complexity, and Structural Metrics. We also develop the prototypical implementation of the framework for computing all the devised complexity metrics applied onto a user specified service choreography model and persistently store the generated metric results. The developed framework is evaluated for its efficacy in computing the complexity metrics on a collection of service choreography models that are designed in both BPMN 2.0 and BPEL4Chor modelling notations.

Acknowledgements

I would like to take this opportunity to express my gratitude and heartfelt thanks to my supervisor Dipl.-Inf. Michael Hahn from the Institute of Architecture of Application Systems (IAAS) at the University of Stuttgart for his valuable feedback and positive guidance throughout the course of my master thesis.

I would also like to thank Professor Dr. Dr. h. c. Frank Leymann for offering me the opportunity to accomplish my master thesis. I thank them for their constant encouragement and priceless advice to succeed in reaching my master thesis goals.

My deep appreciation to my family and friends for their never ending support and love throughout my life which helped me in achieving my master thesis.

Contents

CHAPTER 1

Introduction

Composition of multiple services from a global perspective is termed as a service choreography. Service choreographies take into account the complete conversation (between the services) constraints (sequence of service call and which data element to manipulate) and responsibilities from an overall aspect. Choreography is a significant part for service oriented architecture (SOA), where SOA is an architectural paradigm for developing complex software structure or enterprise applications composed of discrete unit functionality known as services. Services are independent blocks of functions offered by business organizations over the network to the consumers which can be accessed from the published list of services. There has been an extensive acceptance and utilization of service oriented architecture in various business fields and research domain with continuous growth. For instance in Cloud Computing, Business Process Management (BPM), Internet of Things (IoT), Home Healthcare and scientific workflow domain [HBK+17].

Service choreographies are modelled, for example, using Business Process Model and Notation (BPMN 2.0) [Obj18] and Business Process Execution Language for Choreography (BPEL4Chor) [DKL+07] modelling languages. Service choreography models can be of varying complexity and can impact the design effectiveness of the model. Complexity has a negative impact on the accuracy, comprehensiveness, maintainability, effectiveness, optimization potential, and understandability of service choreography models. In spite of its significance in this context, research in this area of measuring complexity of choreography models is a rather new domain with a very few contributions.

There is a growing need for understanding how the complexity of a choreography model affects the error production rate, maintenance issues, and ease of alteration. There are some data indicating that complexity can determine the error probability of a business process [MMN+06]. In this thesis we focus on adapting some of the process model complexity metrics, based from the software engineering domain and graph theory, for applying them with some modifications tailor-made for service choreography models.

In this chapter we discuss the objectives of the topic and describe the problems which are required to be addressed. We provide an overall perspective and main goals of this thesis and finally outline the structure of our research work.

1.1 Problem Statement

We give a brief discussion about the topic's fundamental concepts and motivation in the last section. Here we discuss about solving the specific tasks at hand for accomplishing our thesis's objectives. With rapid growth in IT, for instance in the field of eScience and scientific workflow domain, scientists and researchers are engulfed with an ever increasing need to address the problem of optimization techniques and controlling the complexity of service choreography models. They essentially desire for a framework which will be able to compute the complexity of a service choreography model, enriched with a wide array of complexity metric options. The framework application should take a choreography model as an input given by the user to compute the user's choice of complexity metric results. The user of the framework should be able to access the results instantly, as the output of the metric computation is displayed at once as well as stored in the file system for persistence storage.

We have to identify existing complexity metrics which can be adapted, altered, and tailor-made for applying to the service choreography models. The complexity metrics should also take into account the data-flow between participants, with respect to the sharing and access of data elements in a centralized manner. The complexity metrics should reflect the choreography model's true intricacies and the degree of difficulty in understanding and comprehending the model.

1.2 Main Goal

The main goal of this thesis work is to frame the complexity metrics fitting for the service choreography models (especially with respect to data perspective) and to develop a framework for computing the devised complexity metrics applied on a user given (as input) choreography model and producing the desired metric results. Finally we will evaluate the results of the devised complexity metrics in the prototypical implementation (developed framework) applied on a collection of choreography models.

1.3 Thesis Objectives

The specific thesis objectives are listed as follows:

- Analysis of the current state of the art in research and technologies in order to identify to which extent existing complexity metrics, e.g., from software engineering or process models, can be applied for measuring the complexity of service choreography models.

- Identification and description of viable complexity metrics for (data-aware) service choreographies.
- Prototypical implementation of the identified complexity metrics for a selected choreography modelling notation.
- Evaluation of the identified metrics and the resulting prototype through a set of (collected) choreography models.

1.4 Outline

The remaining thesis document is structured as follows:

Chapter 2 – Background and Fundamentals: Discusses the key concepts underlying our research topic.

Chapter 3 – Related Work: Gives a brief explanation about the existing research contributions with respect (related) to the thesis topic.

Chapter 4 – Complexity Measurement of Service Choreography Models: Describes the devised complexity metrics adapted specifically for service choreography models in both modelling languages (BPMN and BPEL4Chor).

Chapter 5 – Implementation: Specifies the details of the prototypical implementation of the framework for computing the complexity metrics.

Chapter 6 – Evaluation: Illustrates and analyses the complexity metric results applied to a set of choreography models.

Chapter 7 - Conclusion: This chapter concludes our research work with some brief discussion about future work.

CHAPTER 2

Background and Fundamentals

In this chapter we discuss the basic concepts and fundamental theory elemental for our research work. We give a brief introduction of the essential topics underlying our thesis work from varied fields of research. This chapter consists of four main sections each describing about the concepts and technologies related to our thesis work. In Section 2.1 we describe the basic concepts of orchestration and choreography. In Section 2.2 we discuss about the key concepts related to business process management and the modelling language Business Process Model and Notation (BPMN). We explain the concepts behind BPEL4Chor modelling language in Section 2.3. Section 2.4 discusses the transparent data exchange (TraDE) approach in data-aware service choreography models.

2.1 Choreography and Orchestration

Service-oriented architectures (SOA) is an architectural style used for creating complex software systems composed of a collection of autonomous web services. Services are some sort of basic functionality which are offered by the businesses to consumers over the network. There has been recent advances of SOA in several industries like the scientific domain, Cloud Computing, Internet of Things (IoT) etc.

Service choreography and service orchestration are two different archetypes which defines the style for composing a number of services into a concrete entity using different modelling languages [HBK+17]. Service orchestration are generally deployed in private business processes where one central coordinator process or service is in charge of other participating services and regulates their execution. They are designed from the perspective of a single master service or node which coordinates the whole operation. The orchestration is operated in a centralized manner controlled by the master node aware of all the operations, goals, and the sequence of service execution. Business Process Model and Notation (BPMN) and Business Process Execution Language (BPEL) are the two most notable orchestration modelling language [HBK+17].

Contrary to the service orchestrations, service choreography are not based on the concept of a central coordinator. They are modelled based on a global perspective, where all participating services are aware of their execution roles and conversation pattern with other

participants. The participants interact among themselves through the exchange of messages between them [HBK+17].

There are two distinct modelling approaches for choreographies: *interaction models* and *interconnected interface behaviour models*. Basic conversations between two participants are the fundamental units for interaction models. The dependencies are defined from a global perspective. Let's Dance [ZBD+06] and Web Services Choreography Description Language (WS-CDL) [KBR+05] are the two modelling languages for choreographies based on the interaction model. Whereas in the case of interconnected interface behaviour paradigm, each participant is made aware about its association with other participants through the help of message flows and control flows (specified per participant). BPMN and BPEL4Chor are the two modelling languages for choreographies based on the interconnected interface behaviour model [DKB08].

2.2 Business Process Modelling

2.2.1 Process

The word *process* originates from the Latin term *processus* or *processioat*, which denotes an executed action to reach its objective, and the manner in which it is accomplished. For this reason a process is said to be a set of interdependent tasks and activities which are commenced in response to some triggered initiating event in order to attain a particular goal (outcome of the project) for the beneficiary (user) of the process [RSS14]. Process occurs in our day to day activities and is a common phenomenon taking place around us. Processes are the cornerstone of all actions that encompasses notions such as time, space, and motion, and they shape and adapt to the real world scenario in which we live [RSS14]. Every process is a designated part of larger functions of a whole system.

For instance a process can be any household chores we do in our daily lives. Suppose you are working late night and you need a cup of coffee to keep you going. The notion of fatigue in your body sparked and initialized the process of making a cup of coffee. The process includes (1) you go to the kitchen, (2) place the pot on the stove, (3) put water, coffee powder, milk and sugar into the pot, (4) switch on the stove, (5) boil the mixture for five minutes, (6) switch off the stove, (7) strain the prepared coffee to your cup, and (8) go to your room and drink the coffee. These activities stitched together forms a successful process naturally. This is a very simple and self-explanatory example to illustrate the concept of a *process* in our day to day lives.

2.2.2 Business Process

A business process is defined as the set of inter-related, well-defined activities or tasks, executed in a pre-defined order, which is essential to accomplish a common business goal for the beneficiary of the whole process [Kir17]. These activities are placed in a defined sequence to result in a service or product (specific project outcome) for the distinguished client or clients of the organization. The activities or tasks may be enacted by people (manual intervention) or systems (automated process) and can be executed in parallel or in sequence. The business process is associated with distinctly specified inputs and a solitary output. Business processes are made up of core tasks and activities which are linked with one another, and are sorted and grouped [RSS14].

A business process can also consist of minor activities within the business process itself, and in such a case, these minor activities are called subprocesses (process decomposition), which is initiated by the prior subprocess (or the initial commencing event) to produce a beneficial outcome for the later subprocess or the end user (customer) and his or her processes [Kir17]. A business process can not only activate numerous tasks and subprocesses, it can also trigger other processes [RSS14].

A certain person can be given charge of a particular business process, who is known as the process owner, is accountable and responsible for ensuring the proper functioning of the process from its inception till the termination of the process. [Kir17]. Hence a complete business process is a collection of steps, executed and looked after by a number of stakeholders to reach the process goal.

A business process can be manually executed with the help of human intervention or in an automatic fashion. For manual process execution, the process reaches its target without any outside help of automation technology, whereas in the case of automated process execution, users (the process owner) implement the process in a more precise and efficient way with the support of assisting technology [Pnm18].

Business processes exist across industries, both vertical and horizontal, and it can cover all types of business operations, for example in manufacturing and production sector, finance domain, health sector, essential banking activities, tourism industry , defence, human resource (HR) operations, public sector, compliance industry etc. [Pnm17]. In general the business processes can be categorized into three broad types according to Kirchmer [Kir17]. They are as follows:

1. Operational process, whose core objective is to foresee and ensure the correct functioning of the operational tasks of an organization; in this category the staff of the organization “get the things done”.
2. Management process, which oversees and ensures the proper execution of the operational processes; here the managers of the operations “ensure efficient and accurate work processes”.

3. Governance process, which ensures that the organization is functioning adhered to the business rules and regulations, guidelines, and stakeholders expectations; this is where executives ensure the "rules and guidelines for business success" are followed by appropriate management of the process execution terms and conditions.

A business process can usually be diagrammatized (modelled) as a flowchart of a series of activities embedded with a number of control structures (splits or joins) or as a process grid of a sequence of activities with applicable guidelines on the basis of data values in the process [RSS14]. It can also be envisioned as a workflow of logical steps. Business Process Management is the managerial branch which aims to formalize this approach [Pnm17].

2.2.3 Business Process Management

BPM is the acronym for the term Business Process Management. Business Process Management is defined as a discipline of design, modelling, execution, monitoring, and optimization of business process for streamlining of business operations and growth of profitability [Itk18]. According to Kirchmer [Kir17],

"BPM is a management discipline that provides governance for a process-oriented organization with the goal of agility and operational performance. Therefore it uses methods, policies, metrics, management practices, and software tools to manage and continuously improve an organization's business processes."

BPM can be more aptly described as a branch or a discipline area instead of a software tool or technology. It is not just a software which we can buy off the shelves in response to an organization's requirements, rather the technical developers and managers of the entity takes on a key role in business process management. BPM is a managerial way to channel the organization's efforts towards focusing on customer and project requirements. It is a comprehensive controlling methodology that encourages "business effectiveness and efficiency while striving for innovation, flexibility, and integration with technology." BPM focusses on continuously improving the processes, hence could be described as a "process optimization process" [Itk18].

Business Process Management's lifecycle consists of five phases: design, modelling, execution, monitoring, and optimization as shown in Figure 2.1. They are briefly described as follows [Itk18]:

- Design – Process modelling task involves distinctly identifying the existing processes and figuring out the design of planned upcoming processes. The main goal in this phase is structuring the sequence of activities in the process flow, identifying the actors in the process, alerts and notifications, moving an issue or bug up in the hierarchy, standard operating procedures, service level agreements, and task hand-over methodology.

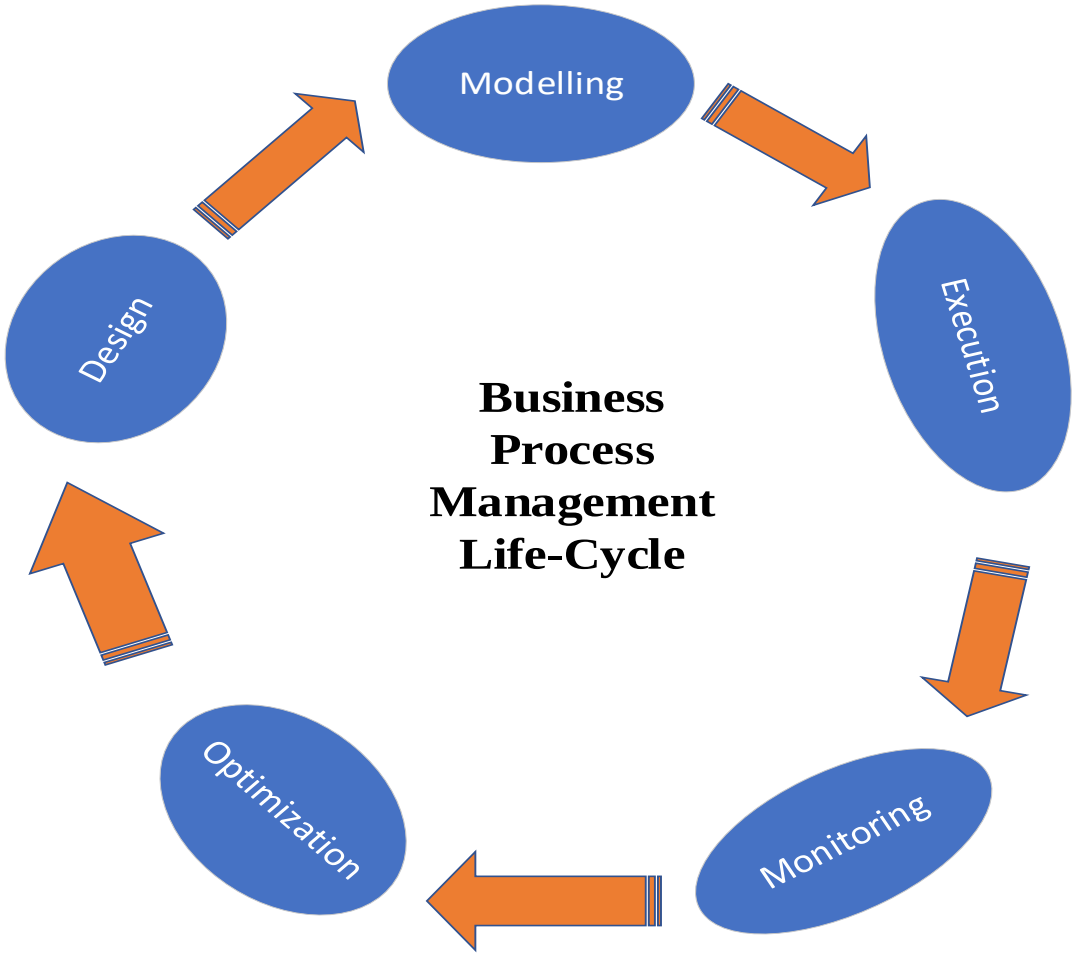

Figure 2.1: Phases of Business Process Management Lifecycle.

- Modelling – The abstract design prepared in the previous phase is taken up for modelling and it is supplemented with the required variables (for example, variation in prices of raw materials or labour costs will impact the project's budget and the organization's profit margin as a whole; this regulates the process' functioning under different situations).
- Execution – For automating the process we can either implement or buy a software application that will accomplish the necessary process steps, or devise an execution procedure by involving people who are assisted by the software.
- Monitoring – In this phase the performance of the business process is continuously tracked, assessed, and analysed. Monitoring task involves the tracking of individual processes to collect their operating state and performance data, along with the real time monitoring of the parameter values used by the process. The gathered tracking

data is utilized to generate the performance statistics of a single or multiple processes to be analysed and adapted for process designing in future.

- Optimization – Process optimization phase involves accessing the process performance data and associated statistics information from the monitoring phase, finding the possible or existing bottleneck regions in the process flow and the plausible opportunities for overall cost reduction or other betterment areas; and then incorporating those incremental changes by adapting them in the process design.

2.2.4 Business Process Model and Notation

Business processes, as discussed in earlier sections, are modelled using various methodology and modelling techniques. Business process modelling is the task of expressing the processes and process flow of an organization, for analysing and improving the current process structure. This is generally implemented by business analysts and managers for boosting the process efficiency.

The Business Process Model and Notation (BPMN) is a standard for business process modelling, which offers the flowchart based graphical notation that illustrates the business process steps. BPMN expresses the end to end of a business process flow. “The notation has been specifically designed to coordinate the sequence of processes and messages that flow between different process participants in a related set of activities” [Obj18].

BPMN is developed by the Business Process Management Initiative (BPMI) to encourage and develop the usage of Business Process Management by laying down standards for process design, modelling, execution, maintenance, and optimization of processes. The main aim of BPMN is to offer a notation which can be understood comprehensively by all business users (business analysts, managers, and technical developers) [OR03].

A standard BPMN empowers businesses with the competence to comprehend their own business procedures in a graphical depiction which will help them to analyse and communicate the performance, collaborations and business transaction between the organizations [Obj18]. Another critical goal of BPMN is to enable the XML languages developed for the business process execution, such as Business Process Execution Language for Web Services (BPEL4WS) [OAS18] and Business Process Modelling Language (BPML) can be graphically depicted with a standard notation [OR03]. BPMN offers the ability to transform the business level notation to an executable Business Process (BPEL) [Whi06].

BPMN 1.0 specification was released to the public in May, 2004, it was then called Business Process Modelling Notation, which later in February, 2006 was adopted by OMG as a standard. BPMN 2.0 specification was adopted in January, 2011, which included some new features like choreographies and correlations.

There are four groups of elements in a Business Process Diagram (BPD) modelled using BPMN, which are Flow Objects, Connecting Objects, Swimlanes, and Artifacts. Flow Objects include activities, events, and gateways. An activity is some task performed in a business process, and it can be either atomic or non-atomic (compound) activity. They are generally named in the structure of a Verb-Noun combination. Activity are basically of two kinds, task and sub-process. They are represented as rounded rectangles and can also be defined to contain loops (for multiple execution). An activity can also possess multiple instances of it, mostly for parallel execution. A task is an atomic activity embedded in a process. It is the lowest-level process which cannot be broken down further. Tasks for special purposes are provided in the BPMN specification like the Send Task for sending messages, Receive Task for receiving messages, User Task where human intervention is required to perform the task, Manual Task are not managed and tracked by business process engine, and Service Task for invoking some service like web services or an application. These different types of tasks are distinguished based on the appropriate icons which are placed on the top left corner of the rectangle as depicted in the Figure 2.2.

Sub- process facilitates to draw hierarchical process diagrams. A sub-process is non-atomic (compound), meaning it cannot be decomposed into more detailed structure level. To hide the details of a sub-process for reducing verbosity (less clutter), a collapsed sub-process version can be used which contains a *plus* symbol in the bottom-centre of the rectangle, whereas to make the sub-process details visible in the diagram we use an expanded sub-process version for greater clarity. Figure 2.2 illustrates the different types of activities (tasks and sub-processes).

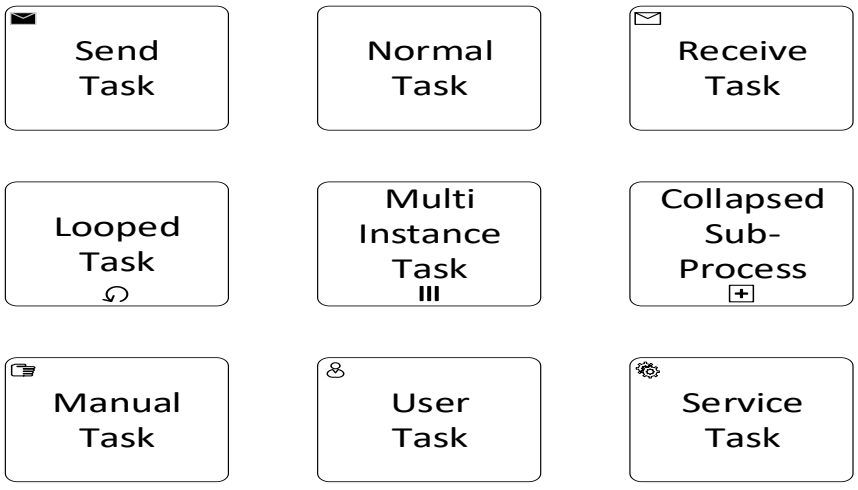

Figure 2.2: Types of Activity (Task).

Events, another type of flow objects, are depicted using a circle. Events are like triggers that occurs during the execution of a business process and it impacts the process flow as a result. They are of three types: start event, end event, and intermediate event; which are identified based on the type of circle boundary. Start event indicates the inception of a

process flow and are depicted by a single narrow boundary border. End event indicates the termination of a process flow with the generated result and are depicted by a single thick boundary border. Intermediate event represents about something occurred during the course of the process flow and are depicted by a tramline boundary border. Figure 2.3 shows the different event types.

Figure 2.3: Types of Event.

Gateways are the third type of flow object and they are modelled using diamond shaped object. Gateways are used to decide the execution sequence of the process flow as they converge and diverge in the process. The gateway types are distinguished based on the icons or markers placed inside the diamond shape. The main functionality of gateways are either to split or merge the process flow. The three main types of gateway are exclusive, inclusive and parallel. Exclusive gateways are used to build alternative paths in a process flow where only one of the many possible paths can be executed. They are depicted using a "X" marker inside the diamond. Inclusive gateways are used for decisions where there are multiple outcomes which can be simultaneously possible and are depicted using an "O" marker inside the diamond. They are used to create both alternative as well as parallel paths in the process flow. Inclusive gateways are generally paired (diverging gateway have a corresponding merging gateway). Parallel gateways are for defining multiple parallel process flow paths for simultaneous execution and are depicted using a "+" icon inside the diamond. One main functionality for parallel gateway is to synchronize parallel paths. Figure 2.4 illustrates the different gateway types.

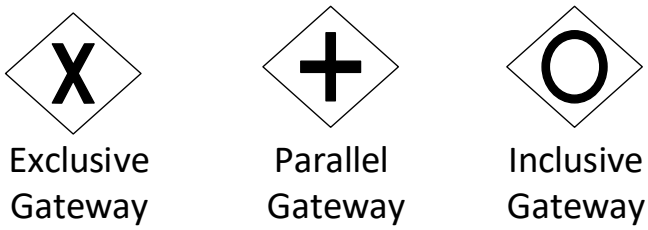

Figure 2.4: Types of Gateway.

Connecting Objects or Connectors are used to link two flow objects together, like between two activities or between an activity and a gateway. They are mainly of three types: sequence flow, message flow, and association. A sequence flow defines the order in which the activities will be executed in a process and it cannot pass over a pool or sub-process'

boundary limit. It helps to designate the process flow, indicating the control flow transfer sequence. The source and the target of a sequence flow belongs to one of the flow objects. Sequence flow is represented by a solid line containing a solid arrowhead mark at the end as shown in the Figure 2.5.

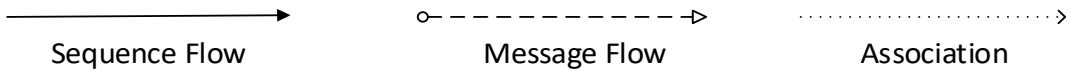

Figure 2.5: Types of Connector.

Message flow is used to depict the transfer of messages between two process participants that are capable of sending and receiving messages. Message flow goes beyond a participant's or a pool's boundary and they are not permitted to link two objects inside the same participant (in BPMN, pools are used to denote separate participants). Message flow are denoted using a dashed line having an open (empty) arrowhead at the end. An association is used to connect and associate between data objects, text and other artifacts with flow objects. Associations are used to indicate the inputs and outputs of an activity and they are represented using a dotted line with a line arrowhead. Figure 2.5 illustrates the different connector types.

Swimlanes are used to isolate or separate activities (disjoint sets) into distinct sections to denote different roles and organizational units. Swimlanes are of two main construct types: pool and lane. Pools are used to denote individual participants in a process. They are represented using a hollow rectangular container for separating a collection of activities generally in the context of business to business (B2B) process model. Communication across pool boundary is implemented using message flows. Lanes are used to indicate the sub-partitions of the construct elements inside a pool to differentiate and categorize among distinct activity sets. Sequence flows can cut across a lane boundary. Figure 2.6 illustrates the different swimlane types.

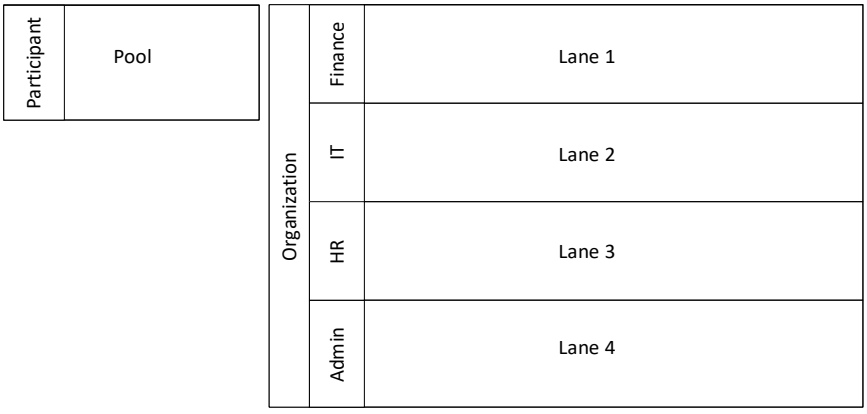

Figure 2.6: Types of Swimlane (Pool and Lane).

Artifacts enable to depict additional information appropriate for the context apart from the regular process flow structure. There are three standard types of artifacts: data object, group and annotation. Data objects are used to represent how data is manipulated in a process, that means the manner in which the data is consumed (input data) or produced (output data) by a certain activity. They are linked to activities by associations. Groups are denoted using a curved corner rectangle having a dashed line as its border. They are used to categorize activities and sequence flow can cross a group boundary. Annotations are used to provide extra textual information about a construct element in the process. This helps the reader of the business process diagram to clearly understand the process structure. Annotations are denoted using an open rectangle possessing annotation text. Figure 2.7 illustrates the different artifact types.

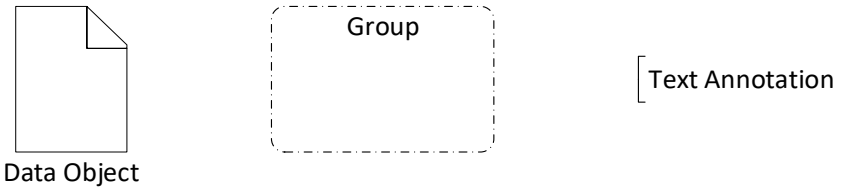

Figure 2.7: Types of Artifact.

2.3 BPEL4Chor

The Business Process Execution Language for Web Services (BPEL4WS or commonly called as BPEL) is a standard execution language used for defining and running the business process, which are employing web services, and transforming them (orchestration) to a sole business process. BPEL is based on the foundation of XML and Web services. BPEL provides the platform to directly and uncomplicatedly orchestrate multiple Web services into a new synthesized services known as business process [Jur06].

Because of BPEL's popularity and widespread acceptance as a standard, Decker et al. [DKL+07] utilized BPEL as a stepping stone for defining choreographies. A supplementary tier of software called BPEL4Chor was put on to change BPEL to a choreography based language from an orchestration language. Decker et al. [DKL+07] illustrated how BPEL is extended and adapted for specifying choreographies. Their principal extensions differentiates among three main facets [DKL+07]:

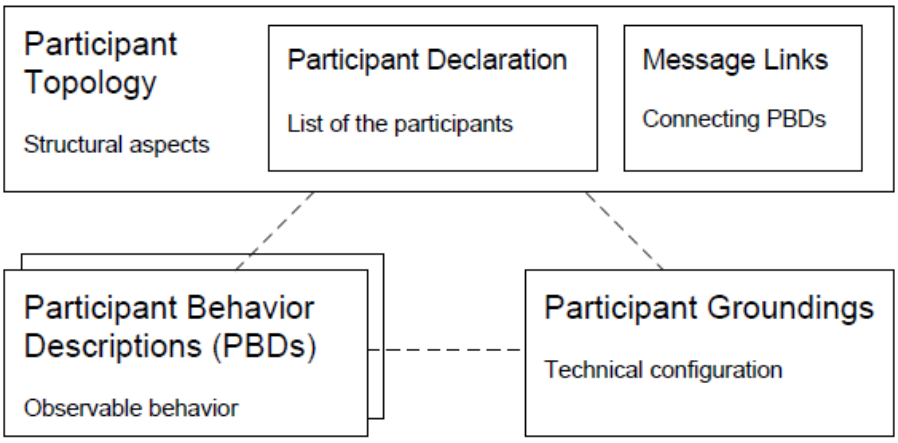

Figure 2.8: BPEL4Chor Artifacts [DKL+07].

- Participant Behaviour Description – specifies and structure the control flow dependencies in between the activities, especially among the activities assigned with the communication role within a particular participant.
- Participant Topology – details the structural notions of a choreography by defining participant types, participant references, and message links. Similar participants have to offer identical collection of communication activities. Message link enables to connect the communication activities of distinct participants.
- Participant Groundings – specifies the substantive technical aspects of the choreography, mainly the port types and the data formats.

BPEL4Chor is based on the interconnected interface behaviour model approach for defining choreographies. The three BPEL4Chor artifacts, as discussed above, enables to decouple the core aspects of choreographies, like the communication activities and their link dependencies, from their technical definition details. This separation of aspects helps to accomplish greater reusability of choreography models. Figure 2.8 depicts the three different BPEL4Chor artifact types [DKL+07].

2.3.1 Participant Behaviour Descriptions

The first artifact type, Participant Behaviour Description, defines a participant's linking activities which are responsible for communication with other participants; for instance the message send and receive activities are specified along with their control-flow dependencies, that is pivotal in choreography designing. For managing control and data flow BPEL already offers a wide array of constructs that are reused in BPEL4Chor [DKL+07].

Decker et al. [DKL+07] introduced the Abstract Process Profile for Participant Behaviour Descriptions by specifying the prerequisites needed to define the behaviour of a participant. To distinctly identify and access the activities from abstract process models, identifiers are added for each activity in each process; especially communication activities and onMessage branch contains an identifier for linking matching send and receive activities of separate participants. Conditional expressions for alternate process flow can be composed in plain text. Correlation techniques of BPEL are also used in BPEL4Chor without any alteration. Correlations are defined in the participant behaviour description.

2.3.2 Participant Topology

The second type of artifact is the participant topology that gives an overall structure of the existent participants and the joining among the participants through message links [DKL+08]. The top-level structural aspect of the choreography is apprehended in participant topology specifications. The Participant topology establishes the concepts of participant type, participant reference, and message link [DKL+07]. Participants are recorded in the participant declaration section where participants and participant sets are differentiated [DKL+08].

Every participant belongs to a certain participant type specified by the corresponding participant behaviour description which defines the roles and responsibility of the participant. As a result this participant behaviour description bears upon all participants of the same type [DKL+07]. A Participant reference is for identifying participants from the participant declaration list. A forEach construct can be employed to loop over (iteration) all the participants in a participant set [DKL+08].

The relationship (connection ratio) between numbers of participants belonging to a certain participant type can be categorized into three distinct instance types. The first instance can be one to one ratio; meaning for a particular participant type there exists just one participant in a specific instance of a choreography conversation. For example, there is only one interviewee and just one interviewer participating in an interview conversation. The second instance of the relationship can be one to n ratio; which means that for a certain participant type there are multiple participants (must be more than one) occurring together in a single conversation instance, where the participant count (total number of participants) data is available (either already known or can be computed from the requirement modelling phase) during the designing phase. For example, there are five car variants for a particular car model type in a manufacturing conversation; where the participant type is the car variant for a specific car model having five participants (for example: Lxi, Dxi, Exi, Exiz, and DLZi) [DKL+07].

The final instance of the relationship type can also be one to n ratio, but here in this scenario the number of participants is unknown at design-time (unlike the second relationship type) and the precise participant count can only be computed at runtime. For example the exact

number of employees in a particular department of the organization keeps on changing (new employee gets hired continuously and attrition also occurs in the meantime); so the exact number can only be determined at a particular instance of the conversation. For providing support for all the three different relationship between participants and participant types, the *participant sets* construct is offered [DKL+07].

The messages communicated between participants are implemented through the help of message link constructs. A message link joins two participants and annotates the message (name and type) which is transmitted through it [DKL+08].

2.3.3 Participant Groundings

The participant groundings basically offer the mapping to web service specific technical configuration [DKL+07]. The technical setup specifications are represented in the participant grounding, in which the actual port types and operations for the message links are defined. The individual message link are assigned to a particular port type and operation, which enables to accomplish one participant by the means of multiple port types [DKL+08]. For a legitimate participant grounding all message links must be grounded. Once a choreography is fully grounded, each one of the participant behaviour description can feasibly be converted to an executable BPEL process based on the Abstract Process Profile [DKL+07].

2.4 Transparent Data Exchange in Service Choreographies

The significance of data is paramount now in modern scientific researches and increasing steadily with the current progress in data science and technology and the advent of Big Data and Internet of Things (IoT). For reducing the complexity and rigidity of service choreography models and making them more flexible with respect to data context in the run time, Hahn et al. [HBK+17] introduced the concepts for designing and execution of data-aware choreographies with the help of the *Transparent Data Exchange (TraDE)* approach. The inherent purpose of the TraDE approach is to make the data flow and data management logic completely separate from the actual control flow logic between participants in service choreographies by modelling the data-flow on the level of the choreography [HBK+17].

The TraDE method concepts are applicable for both the choreography modelling languages, such as BPMN 2.0 and BPEL4Chor as discussed in earlier sections (Section 2.2 and 2.3). It can also be applied to other choreography modelling languages which are based on the interconnected interface behaviour modelling procedure [HBK+17].

2.4.1 Motivation

Without TraDE approach the data flow between two participants (using message-based data exchange) is specified in a different manner in comparison to the data flow within a participant (using simple data associations). The message-based data exchange method for data transactions between participants not only requires extra control flow and modelling constructs for implementing such scenario, but also necessitates the maintaining of multiple copies of the same data objects in each participant which are associated with the data object [HBK+17]. This makes the visual representation of the choreography model more verbose and complicated to understand.

Another disadvantage of the message-based data exchange method is that the actual data transfers between participants at run-time have to be defined during the modelling phase, which makes it difficult to optimize the inter-participant data flow due to their rigid binding in the message flow definitions. Hence Hahn et al. [HBK+17] introduced the TraDE approach to openly provide the common data flow management and exchange functionality embedded in the run-time environment for discarding such drawbacks in the message-based data exchange approach [HBK+17]. In the following we illustrate in detail the underlying theory and concepts of the TraDE approach.

2.4.2 TraDE Modelling

Hahn et al. [HBK+17] introduced the choreography data model (CDM) to express the common choreography data structures in an enclosing container, which are agreed upon by all the participants in a choreography model. The cross-partner data objects are modelled as distinct constructs which holds the choreography data that are mutually shared and accessed by all participants. This enables to avoid modelling multiple copies of the same data object which are accessed by several participants in the choreography, thereby making it less redundant in the choreography model diagram and reduces visual clutter [HBK+17].

The cross-partner data flow facilitates natural representation of the data exchange between participants, independent from the message flow. This decoupling of data flow from the message flow constructs (thereby from control-flow as well) and its associated message handling activities gives more focus and flexibility on the data model shared by multiple participants of a choreography. A cross-partner data object possesses an identifier attribute for uniquely accessing it from any participant of a choreography, and it is empowered to accommodate multiple data elements in it [HBK+17].

A data element represents a unit of shared data and its type is based on its structural definition. Both the cross-partner data object and cross-partner data flow are defined to be modelled as a language independent construct. The Figure 2.9 shows a choreography example modelled based on the BPMN 2.0 specification, illustrating the applied TraDE concepts [HBK+17].

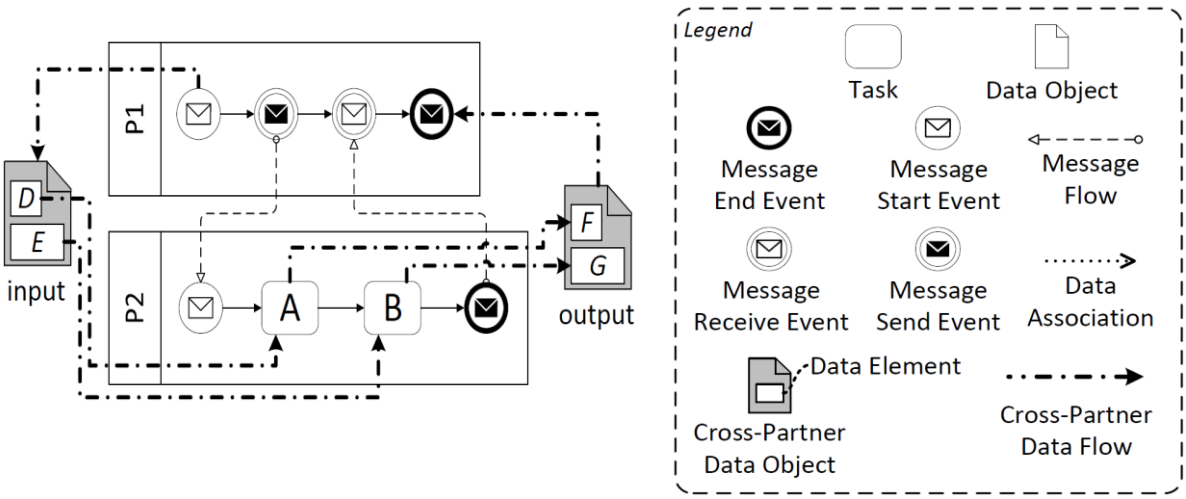

Figure 2.9: A BPMN 2.0 based choreography model applied with TraDE concepts [HBK+17].

Since we are focussing on the modelling perspective, the execution (TraDE Refinement and Middleware are described in the paper [HBK+17]) details are omitted in this document.

CHAPTER 3

Related work

This chapter explains some elementary concepts regarding existing complexity metrics in software engineering and business process model domain which is the quintessential cornerstone of this research document and used in the following chapters. In the process of building the groundwork for understanding the major concepts of the topic, numerous references are cited for finer clarity of the main topics. In the following sections, some of the current state of the art complexity metrics are discussed which can be adapted and applied for measuring the complexity of service choreography models.

3.1 Complexity Metrics

A complexity metric provides a numerical indication of a model's complexity and its structure. At the same time it is not equivalent to Computational complexity, which analyses the effectiveness of the logic of an algorithm with respect to solving the existing problem. We are interested in analysing the current state of the art metrics that can enlighten us about the model's difficulty level in perspective of its understandability and maintainability, and how to adapt them specifically for applying to service choreography model. In case the metric value is above a certain threshold, the model should be redesigned from scratch or the complex and heavier modules could be broken down into simpler fragments. A complexity metric provides a quantitative measurement which eases the process of forecasting bugs in the model and lowering maintenance expenses.

3.1.1 Model Size: Lines of Code Metrics

Lines of Code (LOC) metric, an elementary measure, is known since the early days of complexity measurement and is considered to be simple and easy to compute. It gives the program size based on the simple count of the number of lines of code of a program. This metric continues to have a large acceptance base, mainly due to its simplicity. As stated by Cardoso et al. [CMN+06], the basis of the LOC measure is that program length can be used as a predictor of program characteristics such as errors occurrences, reliability, and ease of maintenance.

Like in an assembler programming language or any low level language a Lines of Code metric gives the total number of instruction statements, similarly for high level language program the Lines of Code indicates to the number of executable statements (not counting the comments, syntactical structure statements, line breaks etc.) [Kan02]. Considering process activity to be equivalent to an executable statement of a program code, Cardoso et al. [CMN+06] derives a very simple metric *M1,* that just counts the number of activities (*NOA*) in a process model.

M1: NOA = Number of activities in a process. [CMN+06]

Worth mentioning here is that the *NOA* metric provides exclusively one particular aspect of size, the length. This metric fails to incorporate other aspects like complexity or the structure of the model. As we know exact source code vary among different programming languages, due to their individual programming constructs requirement, for accomplishing the same task. As some programming languages offers more control flow structures compared to others, providing more flexibility and available options for achieving the same result, hence LOC varies depending on languages. Whereas NOA metric is independent of languages since activity denote the actual business function of the (group of) executable statements which is defined in any particular language. Any activity is defined for fulfilling a particular task or resulting in some outcome, not considering the manner how the goal is reached or how and who actually performs the task. Activity count varies with the manner in which an activity is defined, possessing varying complexity. Hence in comparison to the primitive LOC metric, the NOA metric is not language dependent, thereby enabling the user with easier comprehension of the metric.

Cardoso et al. [CMN+06] also expresses another adaptation of the LOC metric by considering not only activities as program statements, but also taking into account process control-flow elements, such as various control structures (namely AND, OR, XOR gateways, triggering events etc.) in a process model, which impacts the execution sequence among activities. This control flow statements alters the path of execution depending on some business logic and unlike other computational statements they do not possess values.

Cardoso et al. [CMN+06] defines two distinct metrics based on structuredness of the process. For well-structured process we simply count the control structures with respect to splits assuming that an associated join exists. Well-structured process are similar to software programs like the statement blocks in a program are confined within either braces or begin and end instruction statements. Incorporating all these features in designing the metric *M2* which counts the activities and control-flow elements of a process.

M2: NOAC = Number of activities and control-flow elements in a process.[CMN+06]

Alternatively Cardoso et al. designed a third metric (M3) for not well-structured process. Modelling languages which allow such process constructions permit splits to exist without an associated join, making it more challenging to comprehend the process structure resulting in increased design errors.

M3: NOAJS = Number of activities, joins and splits in a process. [CMN+06]

We show an example here of how these metrics are computed. Complying with the BPMN 2.0 specification the process model in Figure 3.1 is illustrated. The above mentioned three complexity metrics (M1, M2, and M3) are applied to the process model in Figure 3.1 for better understanding. As can be seen in Figure 3.1, the model contains total seven activities. This results M1 (NOA) to be seven. There are total five control flow structures (three XOR-gateways and two OR-gateways) present in the process model. Hence this results M2 (NOAC) to be twelve. Since the model depicted in Figure 3.1 is of a well-structured process, hence M3 (NOAJS) is equivalent to M2 here, which is twelve.

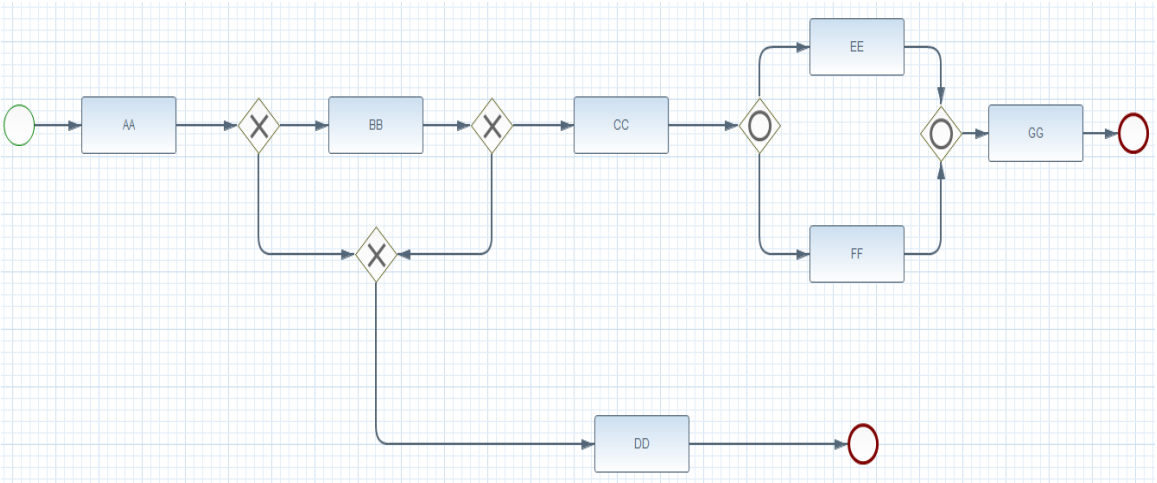

Figure 3.1: An example of a BPMN process model

3.1.2 McCabe's Cyclomatic Complexity

A cyclomatic number, devised by McCabe [Mcc76], maps program complexity to the number of control paths through a program module. The metric is computed from graph theory utilizing the cyclomatic number which gives the number of linearly independent paths in a program [CMN+06]. This metric is not dependent on any programming language and format, which makes it to be one of the most popular software metrics currently used.

Since this metric has been applied to millions of lines of code, a huge empirical knowledge base exists. Using this model developers fine-tune their own software complexity measurements relative to the documented data. McCabe's cyclomatic complexity (MCC) value gives us a program's control-flow complexity. A low number means that the program is easier to comprehend and change if any alteration required. This cyclomatic number provides an in-depth perception of testability, since it is equivalent to total number of test cases required for attaining full path coverage [GL06]. It is found that there is a major

correlation between the cyclomatic number of a piece of software and its defect level [GL06].

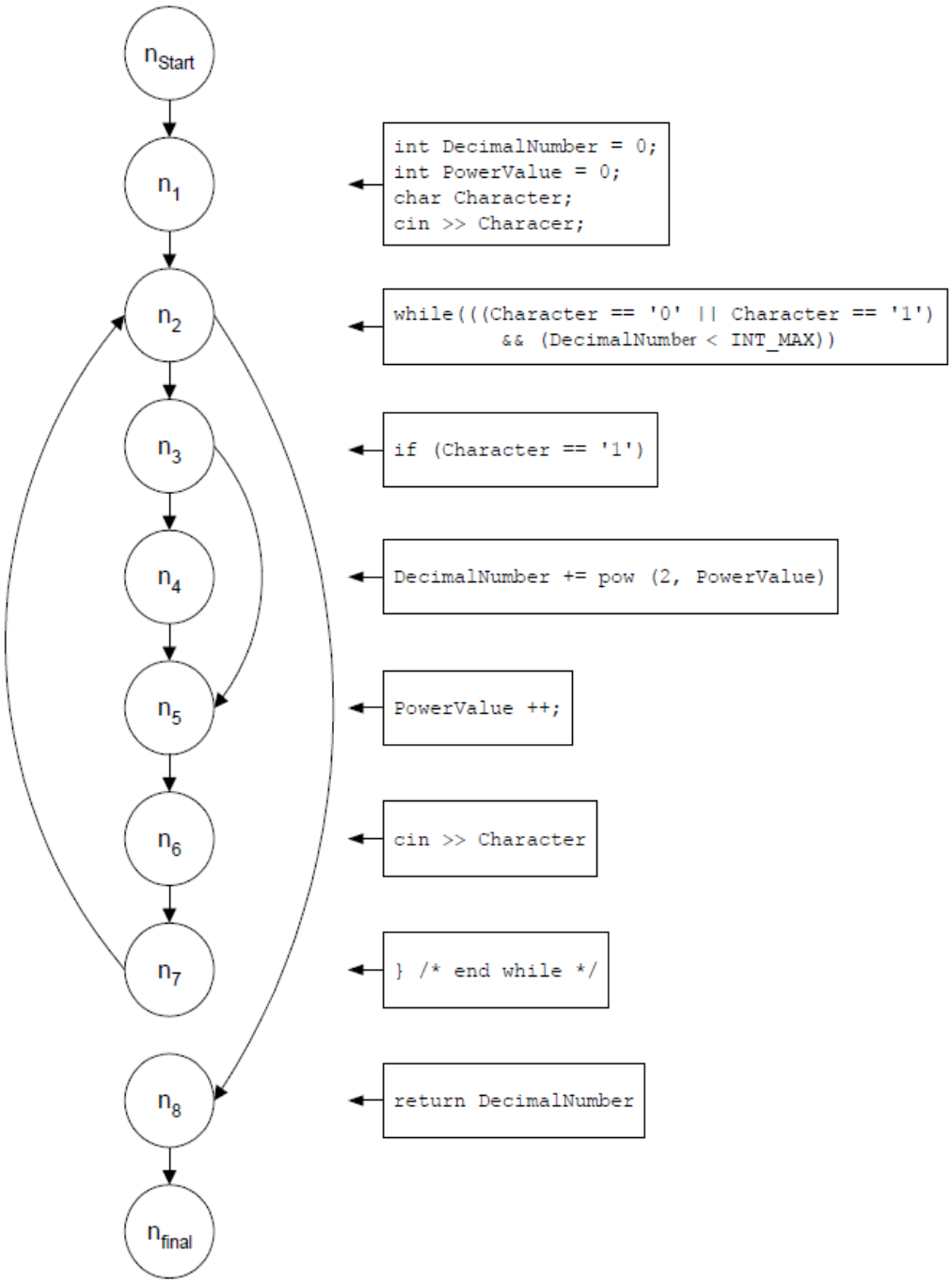

Figure 3.2: Code snippet of a C++ program and its corresponding control flow graph [Wey16].

As proposed by McCabe [Mcc76], the cyclomatic complexity for each module is defined to be:

$MCC = e - n + 2$;

where *e* is the number of edges and *n* is the number of nodes in the control flow graph. Control flow graphs expresses the sequence of execution (path) in a program structure. The nodes depict the computational statements or control flow logic structures. The edges are the connections between two nodes. Every feasible sequence path in a process module has an associated path from the start to the terminating node in the control flow graph. For example in Figure 3.2 [Wey16], the MCC of the control flow graph for the C++ program (converting a binary number into a decimal number) given is 11 – 10 + 2 = 3.

3.1.3 The Control-flow Complexity Metric

Used much like the MCC metric but also for assessing process' complexity, Cardoso et al. developed this metric [CMN⁺06]. Similar to the cyclomatic number, the CFC metric for process accounts for the decision nodes in the control flow. Nodes in the MCC control flow graph (Figure 3.1) have same denotations, whereas process nodes (split and join control-flow structures) possesses dissimilar denotations (like XOR-joins, OR-splits, AND-splits, etc.). Cardoso et al.'s CFC metric considers this vital deviation.

The CFC metric is devised on the analysis of XOR-splits, OR-splits, and AND-splits control flow structures since each split brings on the concept of mental states in process. The control-flow complexity (CFC) of process as defined by Cardoso is "the number of mental states that needs to be considered when a designer develops a process" [GL06]. Cardoso states that the notion of mental states is crucial based on some theories implying that complexity over a certain threshold setback the whole point of human brains capability of picturising the process' numerous control-flow paths which ultimately leads to error.

Computation of control-flow complexity's additive nature makes it relatively easy by just adding the CFC of all split elements. Mathematical computation of control-flow complexity is given as follows where *P* is a process and *A* is an activity [CMN⁺06]:

$$CFC\ (P) = \sum_{A \in P,\ A\ is\ a\ XOR-split} CFC_{XOR}(A) + \sum_{A \in P,\ A\ is\ an\ OR-split} CFC_{OR}(A) + \sum_{A \in P,\ A\ is\ an\ AND-split} CFC_{AND}(A)$$

The $CFC_{XOR\text{-}split}$, $CFC_{OR\text{-}split}$, $CFC_{AND-split}$ functions is computed as follows [CMN+06]:

- $CFC_{XOR\text{-}split}$ (A) = fan–out(A). The control-flow complexity of XOR split is calculated as the total number of outgoing transitions which can emerge after execution of the XOR – split.
- $CFC_{OR-split}$ (A) = $2^{fan-out(A)}$ - 1. The control-flow complexity of OR splits is determined by number of possible states which may result from execution of an OR–split construct. There exists 2^n – 1 chances to process minimum one and maximum n of the emergent transitions of an OR split.
- $CFC_{AND-split}$ (A) = 1. Entire transitions set outgoing from an AND–split have to be executed, so only one possible state emerges from execution of an AND split construct.

As the value of $CFC_{XOR\text{-}split}$, $CFC_{OR\text{-}split}$, and $CFC_{AND-split}$ increases, the process design gets to be more complex. CFC analysis seeks to evaluate complexity without direct execution of process [CMN+06].

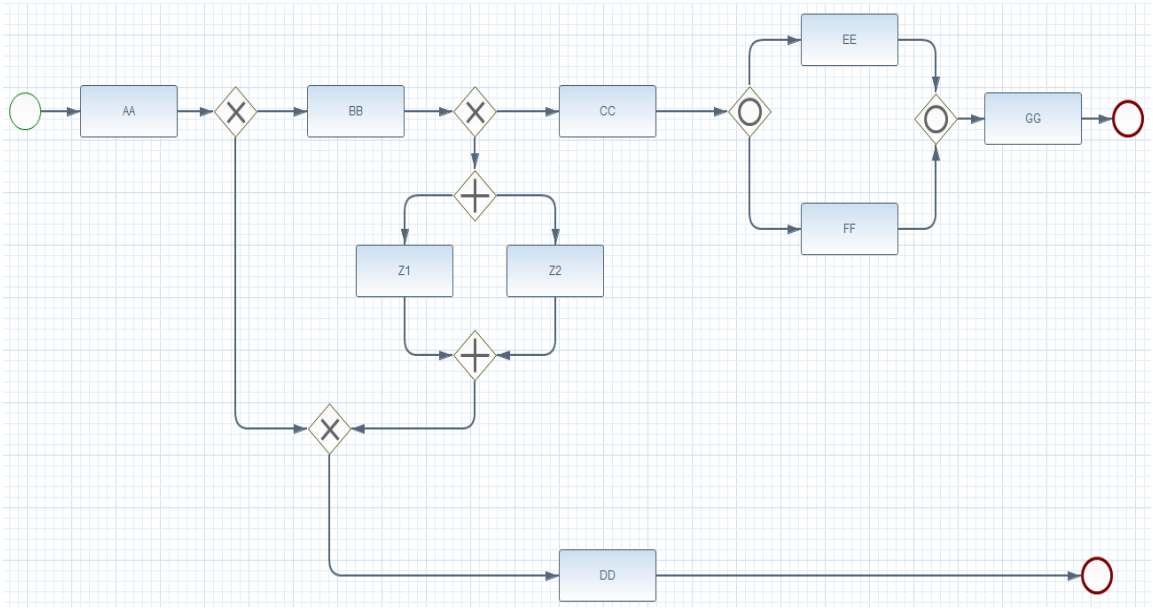

Figure 3.3: An example of a BPMN process model

We provide here a short example illustrating the metric computation. The process model in Figure 3.3 is depicted according to the BPMN 2.0 specification. As evident from the figure there are two AND gateways (BPMN parallel gateway), three XOR gateways (BPMN exclusive gateway), and two OR gateways (BPMN inclusive gateway). Assuming G to be as gateways and P is the process model, we show the computation as follows:

$$\sum_{G \in P,\ G \text{ is a } XOR-split} CFC_{XOR}(G) = 2 + 2 + 1 = 5$$

$$\sum_{G \in P,\ G \text{ is an } OR-split} CFC_{OR}(G) = 3 + 1 = 4$$

$$\sum_{G \in P,\ G \text{ is an } AND-split} CFC_{AND}(G) = 1 + 1 = 2$$

CFC(P) = 5 + 4 + 2 = 11

Hence the control flow complexity of the process model shown in Figure 3.3 is 11.

3.1.4 Information Flow Metric by Henry and Kafura

Henry and Kafura [HK81] developed a metric incorporating the impact of data flow in and out of every software module in a program structure. This approach takes into account the frequency of calls to a particular module from the remaining modules. *Fan-in* is the local data coming in to a module. Fan-out is the total number of calls made to other modules from the given module, that is the local data moving out of the module. The formula for defining the complexity value of a procedure (PC) is:

$PC := Length * (Fan - in * Fan - out)^2$ [HK81].

The expression (fan-in * fan-out) denotes the count of number of combinations of an input source to an output destination. The length value can be computed from simple LOC metric or more effective and elaborate ones like MCC or the longest control flow path in the execution sequence. Generally modules having a bigger fan-in value are the more generic modules which do some basic computations and hence are called upon by many other modules. Alternatively modules with higher fan-out value maybe bigger ones performing more complicated computations requiring calls to other modules for fundamental or basic jobs. If a module simultaneously possess higher fan-in and fan-out values, then for improving the complexity structure a redesign might be necessary.

Cardoso et al. [CMN+06] adapts this metric for complexity evaluation in process. If activities are perceived as black boxes then the length cannot be computed, hence it is assumed to be 1. Whereas activities being white boxes, the length is computed relative to its actual source code length by applying some size metric like LOC and MCC. Here the source code of an activity may mean the underlying activity implementation source code in a corresponding programming language, but if it is so then the source code length is heavily dependent upon the underlying process engine and programming language used for their implementation. Hence it may not be effective enough to be used as a reference value for comparisons. Similarly at process level with the CFC metric one can only differentiate among three types of activities (XOR, OR, and AND gateways). Also in the case of NOAJS,

it always assigns an activity length of 1. Therefore accounting for the above mentioned situations regarding the activity length, we hereby assume it to be 1.

The fan-in and fan-out are mapped directly to the incoming and outgoing control flow connectors of the activities [CMN+06]. When the inputs to an activity (fan-in) are ready and it is scheduled for execution, an activity is called. After finishing execution of an activity, its output data is transferred to the activities connected to it through transitions [CMN+06]. Cardoso et al. proposes a metric named as interface complexity (IC) of an activity which is computed as [CMN+06]:

$$IC := Length * (number\ of\ inputs * number\ of\ outputs)^2$$

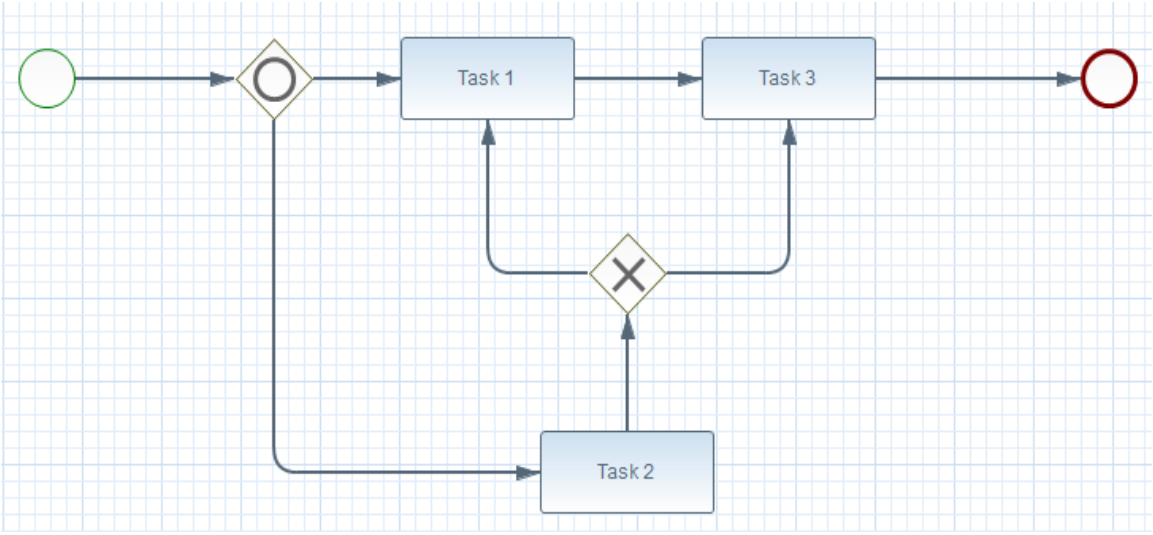

Figure 3.4: An example of a BPMN process model with three Task Activities.

An example is provided for better understanding of the metric computation. The process model in Figure 3.4 is depicted according to the BPMN 2.0 specification. As shown in Figure 3.4 we compute the *IC* for the three Task Activities. We assume Length to be one for all three Tasks. The first Task has two incoming connectors and one outgoing connector, hence *IC* results to be four. The second Task has one incoming connector and one outgoing connector, hence *IC* results to be one. The third Task has two incoming connectors and one outgoing connector, hence *IC* results to be four.

Benefits of the IC metric are favourable especially for the data driven process as it can be computed after the design phase because of the already known (before implementation begins) control flow connectors associated with each activity.

3.1.5 The Coefficient of Network Complexity Metric

A. M. Latva-Koivisto [Lat01] formulates the Coefficient of Network complexity (CNC) metric for graph. This metric is applied widespread in network analysis domain and is meant to assess the complexity level of a critical path network.

Cardoso et al. [CMN+06] adapts this metric for complexity computation of process models. It is calculated as the number of arcs divided by total count of nodes. In this context, nodes represents activities, joins, and splits. Formally it is defined as follows [CMN+06]:

$$CNC := \frac{Number\ of\ arcs}{Number\ of\ activities, joins, and\ splits}$$

We apply this metric to the process model shown in Figure 3.4 as an example for better illustration. As evident from the process model, it has eight arcs in total and seven nodes (Tasks, XOR and OR gateways, Start and End events). According to the formula *CNC* results to be "$\frac{8}{7}$".

3.1.6 Durfee Square Metric and Perfect Square Metric

Despite the fact that simple metrics such as NOAJS and Depth (Maximum nesting of structured blocks in a process model) could be effortlessly computed, however these metrics do not incorporate the heterogeneity of component structures present in the process model. On the other hand some metrics like CNC or Connectivity Level between Activities (CLA) metric (see Section 3.1.7 for reference) which considers the structural variation of the model, might be cumbersome and complicated to compute or difficult to comprehend for a designer.

On the basis of simple measures, Kluza et al. [KN14] proposes a measure which takes into account both types of process elements and their number. The idea comes from concepts like h-index or Durfee Square. On the basis of the spread of process elements: the Durfee Square Metric (DSM) is equal to *d* if there are *d* types of elements which occur at least *d* times in the model (each), and the remaining element types do not occur more than *d* times (each). For a profound depiction of the distribution pattern, Kluza et al. [KN14] propose to use the Perfect Square Metric (PSM) based on the g-index. PSM is defined on a given set of element types ranked in decreasing order on the number of their instances, the PSM is the (unique) largest number such that the top *p* types occur (together) at least p^2 times.

DSM and PSM are devised to compute the quantity and variation of the process elements at the same time. The metrics result in a natural number which makes it easier for designers to understand the complexity of the model. For calculation illustration we take the process model depicted in Figure 3.5, presented in the work of Kluza and Nalepa [KN14], as

example and compute the metrics for it. The process model is illustrated according to the BPMN 2.0 specification.

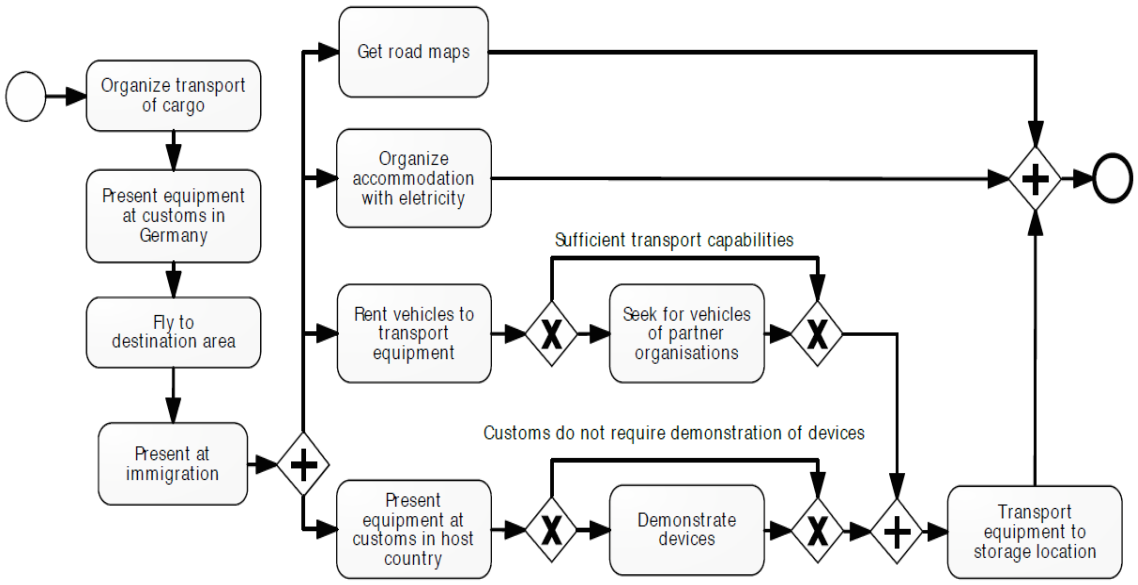

Figure 3.5: An example of a BPMN process model showing transfer of vehicles in an emergency situation. [KN14].

For DSM and PSM calculation purpose, now we form a list of all the element types present in the given model and their associated frequency of occurrence ranked in descending order as shown in Table 3.1.

Table 3.1: Element types and their frequency in the BPMN process model in Figure 3.5.

Element types	**Frequency**
Activity	11
XOR-Gateway	4
AND-Gateway	3
Start event	1
End event	1

In DSM computation, we can see from Table 3.1 that there are three types of elements (Activity, XOR-Gateway, and AND-Gateway) each occurring at least three times each in the model and the remaining two types (Start and End events) do not occur more than three times. Hence the DSM metric result for this model is **three.**

In the same way for PSM computation, if we take *p:*=3 then the top three types (Activity, XOR-Gateway, and AND-Gateway) occur 18 times together, satisfying the boundary condition of p^2 times, that is at least nine occurrences. Since PSM is defined to be the unique largest number, next we assume *p:*=4. The top four element types (Activity, XOR-Gateway, AND-Gateway, and Start event) occur 19 times together, again satisfying the boundary condition of p^2 times that is at least 16 occurrences. When assuming *p:*=5, the top five element types occurs only 20 times together, thus failing the minimum occurrence requirement of at least 25 times. Hence the PSM metric results in **four,** the unique largest number satisfying the required conditions of the PSM metric. Both metrics help designers to understand the structural complexity of the model and its composition, thereby supports for the changes required to reduce maintenance costs.

3.1.7 Connectivity Level between Activities

This metric basically measures the number of control calls made amongst activities relative to the activity count. It can also be described as the level of coupling among activities in the given model. Rolon et al. [ARG+06] presents a great number of simple metrics for evaluating various conceptual models. The metrics defined there can be divided into two primary categories: Base and Derived Measures. The Base Measures are coined as the count of each kind of BPMN elements that a business process model can contain, e.g. Number of Task Looping (NTL) – the total number of tasks with loop characteristics in the model. Using the Base Measures they came up with many Derived Measures that utilize those simple measurement metrics to indicate the proportion of occurrence between the different model elements.

One of such Derived Measure is Connectivity level between Activities (CLA) [ARG+06]. It is defined as:

$$CLA := \frac{TNA}{NSFA};$$

where TNA stands for Total Number of Activities and NSFA stands for Number of Sequence Flows between Activities. TNA is defined as:

$$TNA := TNT + TNCS;$$

where TNT stands for Total Number of Task in the model and TNCS stands for Total Number of Collapsed Sub-Process in the model.

TNT is composed by addition of some Base Measures. It is computed as:

$$TNT := NT + NTL + NTMI + NTC;$$

where NT stands for Number of Tasks, NTL stands for Number of Task with Loop, NTMI stands for Number of Task Multiple Instances, and NTC stands for Number of Task Compensation.

Similarly TNCS is computed as:

$TNCS := NCS + NCSL + NCSMI + NCSC + NCSA$;

Table 3.2: Overview of all Base and Derived Measure [ARG+06]

Name of Basic Measures	Description
NTL	Number of Task Looping
TNT	Total Number of task of the Model
TNCS	Total number of Collapsed Sub-Process of the model
NT	Number of Tasks
NTMI	Number of Task Multiple Instances
NTC	Number of Task Compensation
NCS	Number of Collapsed Sub-Process
NCSL	Number of Collapsed Sub-Process Looping
NCSMI	Number of Collapsed Sub-Process Multiple Instance
NCSC	Number of Collapsed Sub-Process Compensation
NCSA	Number of Collapsed Sub-Process Ad-Hoc
NSFA	Number of Sequence Flows between Activities

Where NCS stands for Number of Collapsed Sub-Process, NCSL stands for Number of Collapsed Sub-Process Looping, NCSMI stands for Number of Collapsed Sub-Process Multiple Instance, NCSC stands for Number of Collapsed Sub-Process Compensation, and NCSA stands for Number of Collapsed Sub-Process Ad-Hoc.

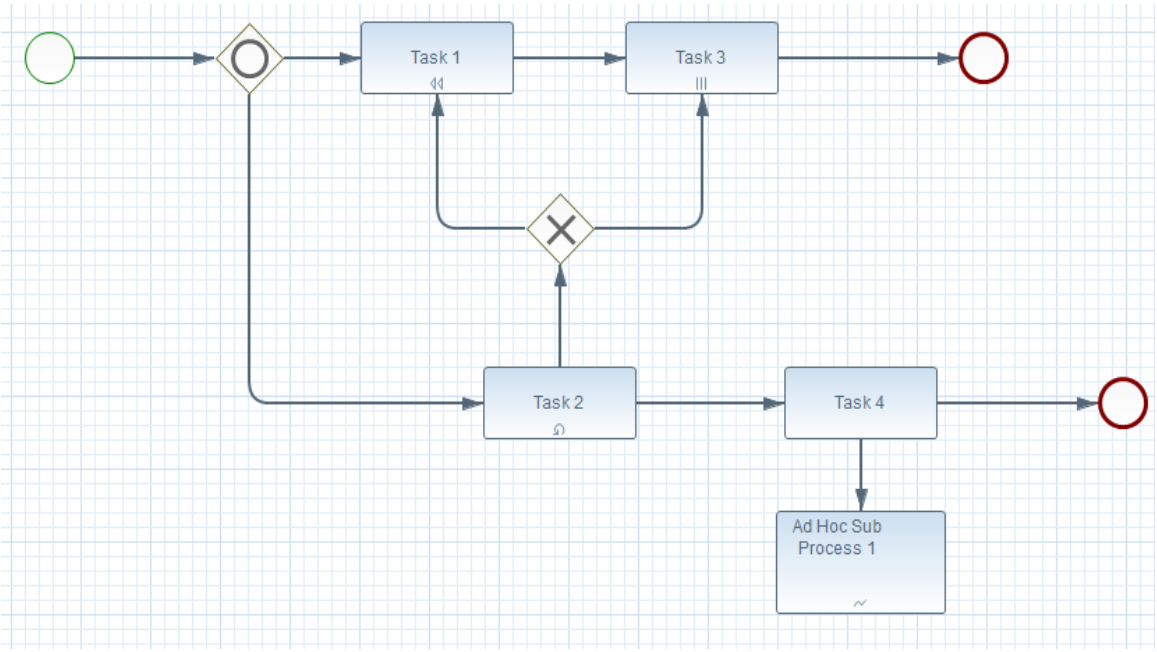

Figure 3.6: An example of a BPMN process model showing different kinds of Task and Sub-Process Activities.

NSFA, a Base Measure, denotes the connectivity between activities. Table 3.2 provides an overview of all the basic measures which are used to calculate TNT and TNCS.

An example is given for easier understanding of the metric computation. The process model in Figure 3.6 is depicted according to the BPMN 2.0 specification showing different kinds of tasks and sub-process activities. We compute this CLA metric on the process model in Figure 3.6. As seen from the model there is one standard Task, one Task with loop characteristics, one Task with compensation, one Task having multiple instances, and one Ad-Hoc Sub-Process present. Hence TNA amounts to five, and NSFA amounts to six. Therefore CLA results to be "$\frac{5}{6}$".

3.1.8 Halstead-based Process Complexity

The Halstead [Hal87] measures are quite popular and used widespread as it is one of the well-known composite measure in software complexity domain. For a quantitative measurement of the program complexity, Halstead devised these metrics based on the count and types of program operands (variables and constants) and program operators (numerical operators and constructs that change the path of program execution). This metric consists of a group of primitive measures (n1, n2, N1, and N2) which can be computed from the source code [Hal87] as follows:

- n1 = number of unique operators (if, while, =, break, GOTO, etc.);
- n2 = number of unique operands (variables and constants);
- N1 = Total count of operator occurrences;
- N2 = Total count of operand occurrences;

The small "n" accounts for the distinct operators and operands present in the program whereas Capital "N" depicts their total number of occurrences in the program. Cardoso et al. [CMN+06] suggest to link the business process elements to Halstead's group of primitive measures. The set of primitive measures coined by Halstead are adapted for business process complexity computation as follows:

- n1 = the number of unique activities, splits, and joins, and other control-flow elements of a business process.
- n2 = the number of unique data containers that are manipulated by the process and its activities.
- N1 = the total number of type "n1" occurrences.
- N2 = the total number of data containers (count of "n2").

Using these adapted primitive measures for business process, Cardoso et al. [CMN+06] introduce the notion of Halstead-based Process Complexity (HPC) measures for forecasting

the Process Length, Volume, and Difficulty. Thus the HPC measures are computed as follows:

- Process Length: $N := (n1 * \log_2 n1) + (n2 * \log_2 n2)$
- Process Volume: $V := (N1 + N2) * \log_2(n1 + n2)$
- Process Difficulty: $D := \left(\frac{n1}{2}\right) * \left(\frac{N2}{n2}\right)$

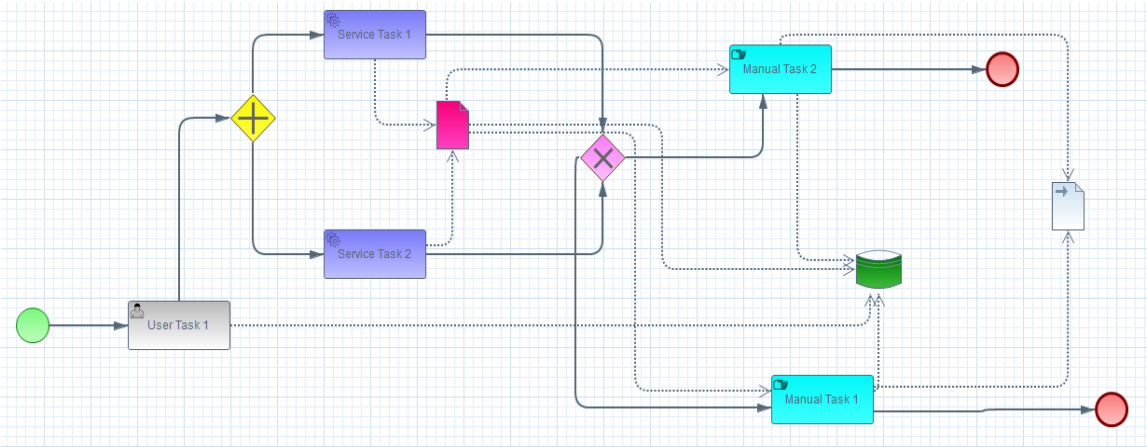

Figure 3.7: An example of a BPMN process model depicting distinct modelling constructs.

An example is given for easier understanding of the metric computation. The process model in Figure 3.7 is depicted according to the BPMN 2.0 specification. Different modelling constructs are represented in distinct colours for easier comprehension of the unique modelling construct nodes as depicted in Figure 3.7. We first calculate the four basic parameters (n1, n2, N1, and N2) which are required for the metric computation (Process Length, Volume, and Difficulty). As evident from the Figure 3.7, there are three unique activity types like User Task in grey colour, Service Task in deep blue colour and Manual Task in sky blue colour. There are two distinct types of gateways present, Exclusive gateway (XOR) in pink colour and Parallel gateway (AND) in yellow colour. Also two event types are evident, Start event in light green colour and End event in red colour. Hence *n1* amounts to seven. With respect to Data containers there are three distinct types present, Data Store Reference in deep green colour, Data Object in deep pink colour, and Data Output in light blue colour. Hence *n2* amounts to three. By calculating their total number of occurrences in the process model, *N1* amounts to ten, and *N2* amounts to three.

Applying the above calculated four parameters in their respective formulae, we get the Process Length for the model shown in Figure 3.7 to be approximately “24.44”, the Process Volume results to be approximately “43.18”, and the Process Difficulty results to be “3.5”.

There are many benefits of HPC measures for process, like it is not required to do an in-depth analysis of process structures, prediction of error rate and effort necessary for maintenance, easy computation, and can be applied to many process modelling languages.

3.1.9 Structural Metrics

These metrics are devised for understanding the structural complexity of a process model and ease of alteration if required. Sanchez et al. [SGM+10] considers a group of metrics for a sequence of experiments conducted on process models. Some of those metrics which can be adapted for process choreographies are:

- Diameter: The length of the longest path from a start node to an end node.
- The Coefficient of Connectivity (CoC): Ratio of the total *number of arcs* in a process model to its total *number of nodes*.

Both the above mentioned metrics have negative correlation with understandibility and modifiability of process models. For illustration purpose we apply this two metrics on the model given in Figure 3.5. There the longest path is “13” and the model contains 24 arcs and 20 nodes. This results in a Diameter of 13 and a CoC value of “1.2”.

CHAPTER 4

Complexity Measurement of Service Choreography Models

In the last chapter we have described about some of the current state of the art complexity metrics from software engineering and process model domain, which can be adapted and applied to service choreography models as well. These complexity metrics provides us with a comprehensive idea about the model's difficulty level for understanding and maintenance purposes. In this chapter we are going to adapt some of those metrics which deems to be fit for service choreography models and apply them to evaluate their utility.

Based on the Kinetic Monte Carlo (KMC) simulation from Hahn et al. [HBK+17], Figure 4.1 shows a choreography model which we use as an example for applying the different complexity metrics and computing the values for comparison.

4.1 An example Choreography model from eScience

4.1.1 Introduction

This simulation model is made using the custom-made simulation software Ostwald ripening of Precipitates on an Atomic Lattice (OPAL) [BS03]. OPAL simulates the formation of copper precipitates, which is the development of atom clusters, within a lattice due to thermal aging. There are five fundamental elementary units in this simulation that are represented as five participants in the choreography model as shown in Figure 4.1. Complying with the BPMN 2.0 specification for a choreography archetype, the communication among five participants and the transfer of control flow is realized using BPMN message flows. The data items that are created and used are depicted using BPMN data objects (DO). The solid line with a closed solid arrowhead represents the sequence flow and the dotted line with the closed hollow arrowhead depicts the message flow between participants for data exchange. The dotted line with open arrowhead represents the data associations between the data objects and the service tasks or the message events. The function modules of the OPAL simulation software are depicted using respective service tasks. Some data objects are designed to hold multiple items, which are represented as data collection objects.

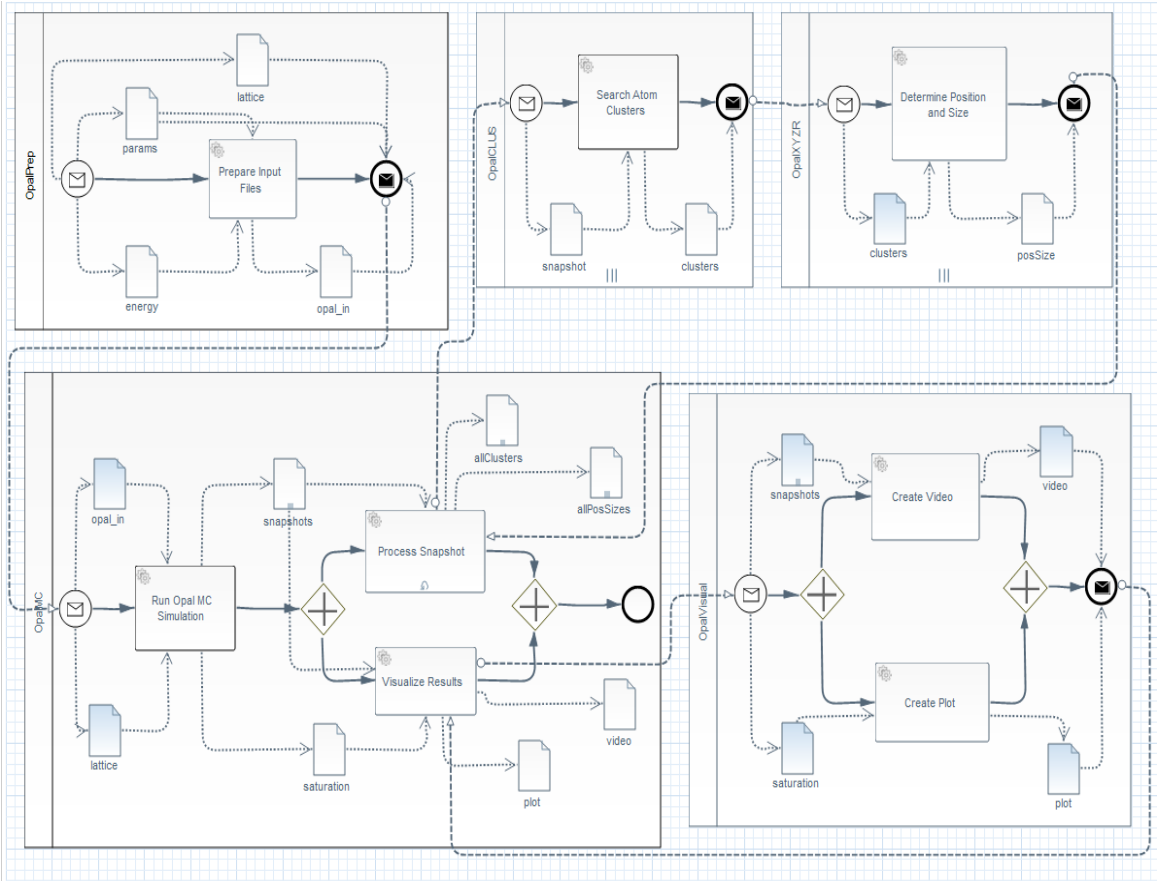

Figure 4.1: An example of a Choreography performing a thermal aging simulation [HBK+17].

4.1.2 Simulation Flow and Model Description

On receiving a new request message, the *OpalPrep* participant creates a new simulation instance. The first request message consists the initiating parameters (*params* DO) like the number of snapshots to take, an initial energy configuration (*energy* DO) and a lattice (*lattice* DO) [HBK+17]. The *Prepare Input Files* service task takes *param* DO and *energy* DO as input and produces *opal_in* DO as output. Then the *params* DO, *lattice* DO, and *opal_in* DO are wrapped in a message to call the participant *OpalMC*.

The *RunOpalMC* participant consumes *lattice* and *opal_in* DO and results in *saturation* and *snapshot* (collection) DO in accordance to the input data. Parallelly the snapshots are analysed further in service task *Process Snapshot* and visualized in service task *Visualize Results*. Multiple instances of the *OpalCLUS* and *OpalXYZR* participants are created by the *Process Snapshots* service task depending on the number of snapshots [HBK+17]. The *OpalCLUS* participant produces *clusters* DO and calls the *OpalXYZR* participant which generates *posSize* DO. Then the control flow goes back to the *Process Snapshot* task of the *OpalMC* participant with the analysed result (resulting cluster and position data).

Simultaneously the visualization participant *OpalVisual* also returns the resulting media data back to the *Visualize Results* task and enables the *OpalMC* participant to finally finish the execution of the simulation instance.

4.2 Application of Complexity Metrics

In the following sections we are going to apply the complexity metrics, discussed in the last chapter for the process models, adapted for the choreography models in general. We use the OPAL choreography (BPMN based model shown in Figure 4.1) as an example for the application of the complexity metrics.

4.2.1 The Size Metrics

We hereby apply the three size metrics which were discussed in Section 3.1.1, namely NOA (M1), NOAC (M2), NOAJS (M3). The first metric (M1), Number of Activities (NOA), is defined to be the total number of activities present in the service choreography model. From a global perspective, each participant can be taken as an activity as a whole from a broad scale, assuming each participants finer implementation details are unknown (black box). For the choreography model shown in Figure 4.1, the NOA amounts to just five. When considering the inner details of each participants design, NOA for choreography model is defined to incorporate other constructs as an activity like the message events, data objects, and the service tasks. We do not take into account the control flow structures (gateways) as activities while considering the count of activities (NOA). Considering the choreography model exposes its inner details for each participant, the NOA metric for the model in Figure 4.1 amounts to 38 according to the later definition. So there is a huge difference between the two calculations when considering the participant's design details.

The second metric (M2), Number of Activities and Control flow elements (NOAC) in a service choreography model is defined to be the count of control flow structures in addition to the activity count in NOA. The control-flow structures include the gateways (Exclusive-XOR, Inclusive-OR, and Parallel-AND). There are four parallel gateways (AND) in the choreography model (as shown in Figure 4.1), so four control structures in addition to NOA count, hence NOAC results to be 42.

The third metric (M3), Number of Activities, Joins, and Splits (NOAJS) for a choreography model is defined to be the count of both join and split gateways in addition to activities for not well-structured choreography models. Since the choreography model (Figure 4.1) in this scenario being well-structured (38 activities, 2 joins, and 2 splits), NOAJS results equivalent to NOAC, that is 42. These three metrics gives us the complexity based on the size and volume of the service choreography model. The difference between the values of NOAJS and NOA also gives us an understanding about the complexity based on the number

of control-flow structures with proportion to the number of activities. This tells us a great deal about the model's complexity in terms of volume as well as its ease of comprehension of the model.

We adapt and use the three metrics discussed above to devise another metric (M4) for service choreography model as follows:

$$M4 := \frac{NOAJS - NOA}{NOA}$$

This ratio is easily computed from the previous metrics, hence M4 of our choreography model in Figure 4.1 is approximately "0.11". In the future work we can conduct an empirical study to find a threshold value for M4 above which a choreography model can be perceived as real complex and further model re-designing would be necessary.

4.2.2 McCabe's Cyclomatic Complexity

As discussed in Section 3.1.2, McCabe's Cyclomatic Complexity (MCC) metric enlightens us in comprehending a process model's control flow complexity and its count of independent paths from start node to the terminating node. Here we adapt and define the MCC metric for the service choreography model in general. The MCC equation as defined earlier is as follows:

$$MCC = e - n + 2$$

For applying the MCC metric to the choreography model, we define the edges (*e*) parameter in the MCC equation to represent the sequence flow, data associations, and message flow in a choreography model. Similarly the nodes (*n*) in the equation depicts the service tasks, data objects or containers, message events, and parallel gateways in the choreography model. Nodes also represents other constructs in choreography modelling according to BPMN 2.0 specification like all types of Tasks (manual, user, script, send, receive etc.), Gateways (inclusive, exclusive, event based etc.), Data items (data store reference, data input, data output etc.), and Sub-Process (Ad-Hoc, Transaction etc.).

We can compute the MCC metric for each participant individually and then sum up to get the complexity value for the whole model from a global perspective. Another alternative is to calculate for the choreography model as a whole ignoring the participant's boundaries.

First we compute MCC for each participant in the service choreography model shown in Figure 4.1. There are five participants in the model as shown in Figure 4.1. We denote them as follows: OpalPrep as P1, OpalMC as P2, OpalCLUS as P3, OpalXYZR as P4, and OpalVisual as P5. For P1 there are twelve edges (*e*=12) and seven nodes (*n*=7), hence MCC results to be seven. For P2 there are twenty five edges (*e*=25) and fifteen nodes (*n*=15), hence MCC results to be twelve. For P3 there are eight edges (*e*=8) and five nodes (*n*=5), hence MCC results to be five. For P4 there are eight edges (*e*=8) and five nodes (*n*=5),

hence MCC results to be five. For P5 there are sixteen edges (e=16) and ten nodes (n=10), hence MCC results to be eight. Now we sum up the individual participant's MCC value, which results to be 37 for the choreography model as a whole.

Now we compute MCC for the choreography model from a global perspective. Here we get number of edges to be sixty three (e=63) and nodes to be forty two (n=42), hence MCC results to be 23. We can see that when calculating for individual participant separately and then summing up the MCC value is quite higher when compared to global computation (choreography model as a whole). This is because of the plus two in the MCC equation ($e-n+2$), it adds up every time the MCC is computed for each participant whereas for the global perspective the plus two is added only once. Another difference is that for each message flow, it is accounted for twice (once in the participant where it's outgoing and once where it's incoming) when computing for individual participant and summing up. This explains the difference of fourteen for the MCC value in the two alternative computation procedures as shown. As can be seen from the above discussion, the MCC computation result based on global perspective better reflects the cyclomatic complexity of a choreography model compared to the other MCC computation procedure (based on each participant separately-local perspective).

The MCC metric adaptation for the service choreography model helps us to understand the model's alternative flow path complexity. For example, from the computation shown above, we can conclude that the model's complexity is not too critical (since it's below 50).

4.2.3 The Control-flow Complexity Metric

The Control-flow Complexity (CFC) metric measures the split and join control flow structures present in the choreography model. It can be directly adapted for service choreography models as the decision split structures are equivalent to the Exclusive (XOR), Inclusive (OR), and Parallel (AND) gateways as discussed in Section 3.1.3. This metric is additive and the final value indicates the choreography model's complexity of the existing gateways. The CFC equation is:

$$CFC\ (P) = \sum_{A \in P,\ A\ is\ a\ XOR-split} CFC_{XOR}(A) + \sum_{A \in P,\ A\ is\ an\ OR-split} CFC_{OR}(A) + \sum_{A \in P,\ A\ is\ an\ AND-split} CFC_{AND}(A)$$

Where $CFC_{XOR\text{-}split}(A) = fan\text{–}out(A)$, $CFC_{OR\text{–}split}(A) = 2^{fan\text{–}out(A)} - 1$, and $CFC_{AND\text{–}split}(A) = 1$. For computing the CFC metric of the choreography model as a whole we need to calculate the metric value for each individual gateway separately according to the formula given for their respective gateways. In the choreography model shown in Figure 4.1, we have only one kind of gateway present (AND). We have four AND gateways and since $CFC_{AND\text{-}split}$ value is always one, hence CFC metric value for the model is **four**. This metric gives us an

insight on the kind and number of gateways present and their complexity for each based on their associated outgoing sequence flow count.

4.2.4 Interface Complexity Metric

The Interface Complexity (IC) metric as discussed in Section 3.1.4 is defined for each activity in process model, but in this section we adapt and alter this metric specifically for the service choreography model. The IC metric is computed for each participant in the choreography model and then summed up for all the participants in the model to result into a final complexity metric value of the whole model. The IC equation for choreography model is defined as follows:

$$IC := Length * (number\ of\ inputs * number\ of\ outputs)^2$$

The *length* parameter in the IC equation for choreography model represents the participant's length. This participant length is equivalent to the number of construct nodes existing within the participant. The construct node represents any kind of Tasks, Gateways, Events, Sub Process, and Global Tasks. Data items and Connectors are not accounted for in the participant length computation. The *number of inputs* parameter in the IC equation (for choreography model in general) is defined to be the incoming data flow count, meaning the number of data items consumed by the participant. Similarly the *number of outputs* parameter for choreography model is defined to be the outgoing data flow count, meaning the number of data items produced or modified (of the incoming data item) and send out from the participant to other participants in the same choreography model. If there are no incoming or outgoing data flow occur for any participant, we assume the corresponding parameter value to be taken as one instead of zero. This assumption is made because of the participants which might not have any incoming or outgoing data flow, which results the second part of the IC equation to be zero (due to multiplication by zero) and hence the IC metric value will not correctly reflect the participant's true complexity. This situation arises for terminating participant which does not have any outgoing data flow. With this assumption we prevent this kind of situations from giving a deceptive IC metric value.

We hereby compute the IC metric value for the choreography model depicted in Figure 4.1. First we compute the IC value for each of the five participants as present in the model. For participant P1 (OpalPrep), the *length* value is three, *number of inputs* is three and the *number of outputs* is also three. Hence the IC value for P1 results to be 243. For participant P2 (OpalMC), the *length* value is seven, *number of inputs* is two and the *number of outputs* is zero due to being a terminating participant, so we assume it to be one. Hence the IC value for P2 results to be 28. For participant P3 (OpalCLUS), the *length* value is three, *number of inputs* is one and the *number of outputs* is also one. Hence the IC value for P3 results to be three. For participant P4 (OpalXYZR), the *length* value is three, *number of inputs* is one and the *number of outputs* is also one. Hence the IC value for P4 results to be three. For participant P5 (OpalVisual), the *length* value is six, *number of inputs* is two and the *number*

of outputs is also two. Hence the IC value for P5 results to be 96. After calculation of IC value for each participant, we sum them up for the choreography model from a global perspective, which results to be 373. This metric, as defined for service choreography models, helps us to gain an insight about the model's data consumption and generation pattern and the associated flow complexity.

4.2.5 The Coefficient of Network Complexity Metric

As discussed in Section 3.1.5, the Coefficient of Network Complexity (CNC) metric measures the flow network complexity (arcs) relative to the count of other construct nodes. Here we adapt this metric for measuring the complexity of service choreography models. The CNC equation is as follows:

$$CNC := \frac{Number\ of\ arcs}{Number\ of\ activities, joins, and\ splits}$$

The parameter *number of arcs* in the CNC equation for service choreography model is defined to be the count of all connector types (in the BPMN 2.0 specification) like the Message Flow, Data Association, and Sequence Flow. Similarly the *number of activities, joins, and splits* parameter in the CNC equation for service choreography model is defined to be the count of all types of Tasks, Gateways, Events, and Sub Process.

We now compute the CNC value for the choreography model shown in Figure 4.1. As CNC is defined above for choreography model, the *number of arcs* is 63 and the *number of activities, joins, and splits* is 22. Therefore, CNC value results to be approximately "2.86". This gives us the insight about its flow connector's complexity and how dense the connections are.

4.2.6 Durfee Square Metric and Perfect Square Metric

The Durfee Square Metric (DSM) and the Perfect Square Metric (PSM) is directly adapted for measuring the complexity of service choreography models. As discussed in Section 3.1.6 the DSM and PSM computation for process models, we define DSM for choreography model to be equal to *d*, if there are *d* types of construct elements occurring at least *d* times (each) in the choreography model. The construct elements are defined to be any Task types, Data Item types, Gateway types, Sub Process types, and Event types.

According to the above definition of DSM for choreography model, we hereby compute DSM for the model depicted in Figure 4.1. For this purpose we form a table containing a list of all the construct element types in the model as well as their associated frequency of occurrence ranked in descending order as shown in Table 4.1.

Table 4.1: Construct Element types and their frequency in the choreography model in Figure 4.1.

Element types	Frequency
Data Objects	20
Service Tasks	8
Message Start Event	5
Message End Event	5
Parallel (AND) Gateway	4

For DSM computation, we can see from Table 4.1 that there are four types of construct elements (Data Objects, Service Tasks, Message Start Event, and Message End Event) each occurring at least four times each in the model and the remaining one type (Parallel Gateway) do not occur more than four times. Hence, the DSM result for this choreography model is **four.**

Similarly, PSM for choreography model is defined on a given set of construct element types ranked in descending order on the number of their instances. The PSM is the (unique) largest number such that the top p types occur together at least p^2 times. The construct element types are similar to as defined for the DSM.

As evident from Table 4.1, when assuming p to be 5, then all the top five construct element types occur together for 42 times, satisfying the boundary condition of p^2 times, that is at least 25 occurrences. Since our assumed p value is the unique largest number possible for our concerned model (total five construct element types), hence PSM value results to be **five.** Both the metrics helps us to get an insight about the choreography model's construct element composition and their occurrence distribution, which eases our comprehension of the model's complexity.

4.2.7 Connectivity Level between Activities

The Connectivity Level between Activities (CLA) metric discussed in Section 3.1.7 is directly adapted for service choreography models. The CLA equation for choreography model is defined as follows:

$$CLA := \frac{TNA}{NSFA}$$

The Total Number of Activities (TNA) parameter in the CLA equation for choreography model is defined to be the total number of task and collapsed sub process in the choreography model. The Number of Sequence Flows between Activities (NSFA) parameter (in the CLA equation for choreography model) denotes two kinds of connectors

in BPMN 2.0 specification; namely the Sequence Flow and the Message Flow between Tasks or Sub Process in the choreography model.

We compute the CLA metric for the choreography model portrayed in Figure 4.1; where the TNA value is eight and NSFA value is 25. Thus, the CLA metric value results to be approximately "0.32". This metric is pretty straight forward to compute and gives us the complexity of connection between the various tasks and sub process.

4.2.8 Halstead-based Choreography Complexity

The Halstead-based Process Complexity (HPC) metric is directly adapted for choreography models, named as Halstead-based Choreography Complexity (HCC). The four sets of primitive measures (n1, n2, N1, and N2) in the Halstead complexity metric are adapted for choreography model as follows:

- n1:= the number of unique Task types, Gateway types, Event types, and Sub Process types in a choreography model.
- n2:= the number of unique Data Items which are either consumed or produced by the participants in the choreography model.
- N1:= the total number of n1 occurrences.
- N2:= the total number of n2 occurrences.

The HCC measures for computing the choreography length, volume, and the difficulty can be calculated by utilizing the above defined four primitive parameters. The formula for calculating these measures are as follows:

- Choreography Length: $N := (n1 * \log_2 n1) + (n2 * \log_2 n2)$
- Choreography Volume: $V := (N1 + N2) * \log_2(n1 + n2)$
- Choreography Difficulty: $D := \left(\frac{n1}{2}\right) * (\frac{N2}{n2})$

Now we will compute the HCC measures for the choreography model depicted in Figure 4.1. The first parameter *n1* amounts to four (Service Task, Message Start Event, Message End Event, and Parallel Gateway). The second parameter *n2* amounts to two (Data Object, and Data Object with Collection). By calculating their total number of occurrences in the choreography model, *N1* amounts to 22, and *N2* amounts to 20.

Applying the above calculated four parameters in their respective formulae, we get the Choreography Length for the model shown in Figure 4.1 to be ten, the Choreography Volume results to be approximately "108.57", and the Choreography Difficulty results to be "20". This complexity metric helps us to get an insight about the choreography model's understanding difficulty as well as its size.

4.2.9 Structural Metrics

As discussed in Section 3.1.9, Structural metrics such as Diameter and the Coefficient of Connectivity (CoC) are directly adapted for the choreography model. These metrics are simple to compute and helps in a great way to comprehend the robustness of the structure of the model.

The Diameter for choreography model is defined as the length of the longest path that can be taken from the start node (maybe Message Start Event) to the terminating node (maybe Message End Event). The CoC for choreography model is defined as the ratio of number of arcs in a choreography model (arcs are any kind of Connectors like Message Flow, Sequence Flow, Data Association etc.) to the total number of construct nodes (all types of Tasks, Gateways, Data Items, Events, and Sub Process).

We now compute both these complexity metrics for the choreography model depicted in Figure 4.1. The longest path is "21" in this scenario and the choreography model contains 63 arcs and 42 nodes. Hence, the Diameter results to be 21 and the CoC value is "1.5". The Diameter value reflects that it is a pretty long choreography model. The CoC value gives us an acumen about the connectivity complexity, how its every construct node is associated with 1.5 arcs on an average.

4.3 Complexity Metrics for Data-aware Choreography Models

In this section we apply the complexity metrics discussed in the previous section to data-aware service choreography models. As discussed by Hahn et al. [HBK+17], data aware service choreography is realized through the Transparent Data Exchange (TraDE) approach where data containers are used as a language independent name for a modelling construct that allows the specification of data, e.g., BPMN 2.0 data objects or BPEL variables. A cross-partner data object (DO) comprises of multiple data elements which are shared among various participants in the choreography model.

In the following sections we apply the complexity metrics to the data-aware choreography model shown in Figure 4.2 [HBK+17]. The figure depicts the same choreography model as in Figure 4.1 using the TraDE approach. The data items in the choreography model are defined independently of any participant as shared cross-partner data objects, which enables the user of the choreography model to easily comprehend and distinguish the consumed and produced data.

As evident from the Figure 4.2 the count of data objects has been greatly reduced as compared to Figure 4.1, which diminishes the choreography model's graphical complexity. After applying the complexity metrics to this model in Figure 4.2, we compare and summarise the metric values obtained with the previous model in Figure 4.1.

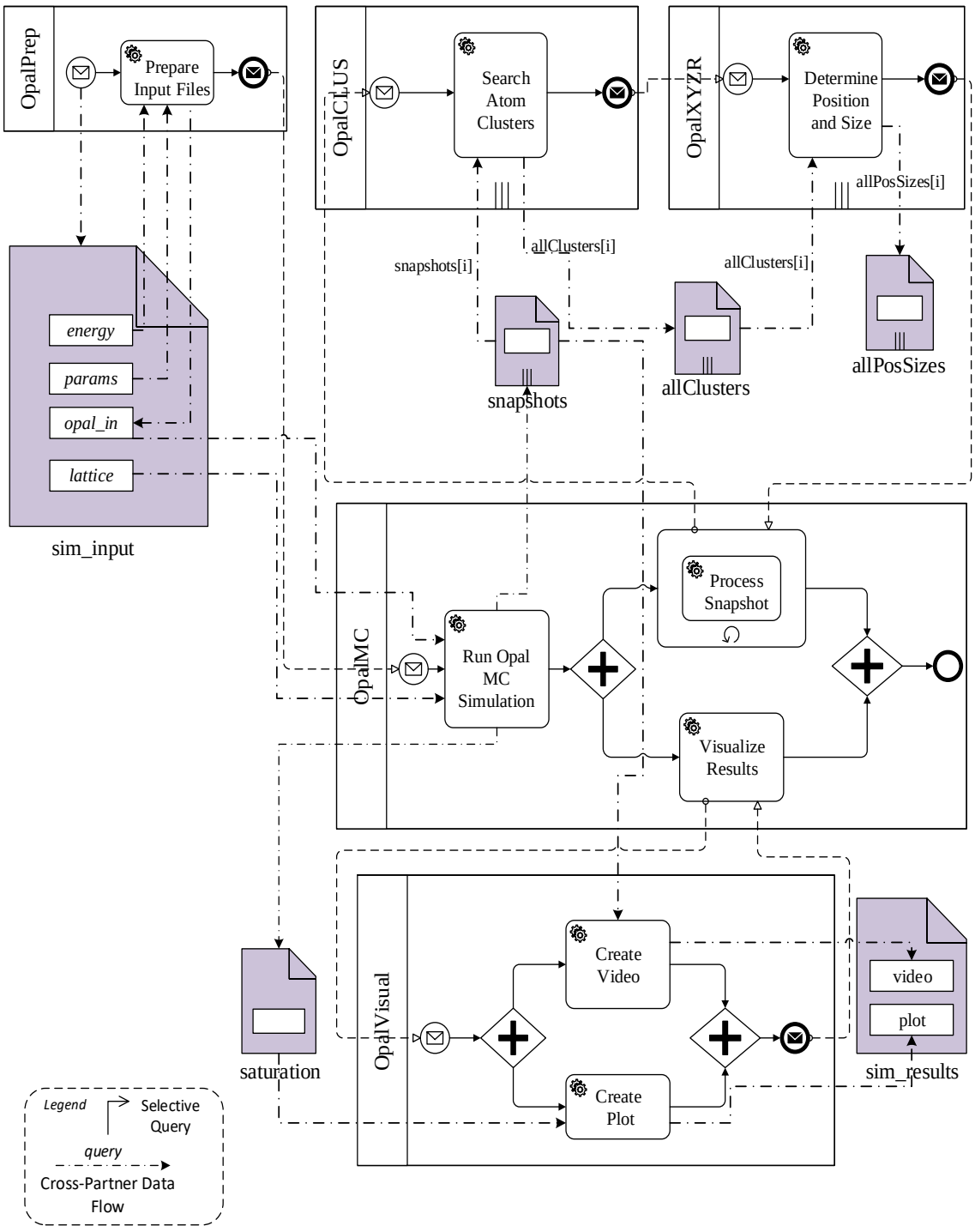

Figure 4.2: Data-aware OPAL simulation choreography after applying TraDE concepts [HBK+17].

4.3.1 The Size Metrics

As the four size metrics (M1, M2, M3, and M4) discussed in Section 4.2.1 are applied to the choreography model in Figure 4.1, similarly we apply them in this model depicted in Figure 4.2. Applying the first metric (M1), Number of Activities is 24. We see a strong reduction in NOA (as compared to 38) value since there are 14 fewer Data Objects (DO) in the data-aware choreography model which greatly lessen the overall NOA count.

Similarly in the second metric (M2), the Number of Activities and Control flow elements amounts to 28, accounting for the four parallel Gateways, which is also a reduction of 14 from the previous NOAC count of 42 due to the shared cross partner Data Objects. The third metric (M3), Number of Activities, Joins, and Splits amounts same as of NOAC, which is 28, equivalent to NOAC due to well-structured model. The fourth metric (M4) is computed from M1 and M3, which amounts to approximately "0.16", slightly greater than the previous model due to the lower NOA count. As we can see with the size metrics, this data-aware model performs better than the previous model.

4.3.2 McCabe's Cyclomatic Complexity

Similarly as discussed in Section 4.2.2, here we apply the McCabe's Cyclomatic Complexity (MCC) metric on the data-aware choreography model depicted in Figure 4.2. For this MCC computation we take the second approach (as discussed in Section 4.2.2) by calculating for the choreography model as a whole ignoring the participant's boundaries.

The total number of edges (*e*) amounts to 41 and total number of nodes amounts to 28. Therefore, MCC metric value results to be 15. As evident from the above calculation, data-aware service choreography model performed better than the previous model in Figure 4.1 for the MCC metric, whose value is eight lesser than the previous model. Hence, this metric as well asserts the fact that the data-aware choreography model's complexity is less than the previous non data-aware choreography model.

4.3.3 The Control-flow Complexity Metric

As discussed in Section 4.2.3, the Control-flow Complexity (CFC) metric is applied to the model in Figure 4.2. As evident from the data-aware choreography model depicted in Figure 4.2, there exists only one kind of Gateway (Parallel Gateway), occurring four times in total. As the value of $CFC_{AND\text{-}split}$ is one, therefore the CFC metric value for the whole choreography model is **four**. So there is no difference in the CFC value compared to the previous model in Figure 4.1.

4.3.4 Interface Complexity Metric

The Interface Complexity (IC) metric is computed for each participant in the choreography model represented in Figure 4.2. The *length, number of inputs,* and *number of outputs* parameters are calculated in a similar way as discussed in Section 4.2.4. The assumption of taking value "one" instead of "zero" for parameters *number of inputs* and *number of outputs,* when there are no such data flow, is also maintained in this scenario.

We compute the IC value for each of the five participants separately. For participant P1 (OpalPrep) the *length* value is three, *number of inputs* is two and the *number of outputs* is one. Hence the IC value for P1 results to be 12. For participant P2 (OpalMC) the *length* value is seven, *number of inputs* is two and the *number of outputs* is two. Hence the IC value for P2 results to be 112. For participant P3 (OpalCLUS) the *length* value is three, *number of inputs* is one and the *number of outputs* is also one. Hence the IC value for P3 results to be three. For participant P4 (OpalXYZR) the *length* value is three, *number of inputs* is one and the *number of outputs* is also one. Hence the IC value for P4 results to be three. For participant P5 (OpalVisual) the *length* value is six, *number of inputs* is two and the *number of outputs* is also two. Hence the IC value for P5 results to be 96. Now we add up the IC value for each participant to get the data-aware choreography model's IC value as a whole, which results in 226.

We can see a contrasting difference in the IC metric value when compared to the previous choreography model's (in Figure 4.1) IC value of 373. There is a sharp decrease in the interface complexity due to the consolidation of data objects into a shared cross-partner data objects, which reduces the data flow count and its associated complexities. Especially for participant P1, the IC value decreased from 243 in the previous model to only 12 in this data-aware choreography model, which paved the way for an overall IC value reduction of 147 in between the two model's. Clearly this data-aware choreography model performed way better than the previous model with respect to the IC metric.

4.3.5 The Coefficient of Network Complexity metric

We apply the Coefficient of Network Complexity (CNC) metric to the data-aware choreography model in Figure 4.2. The cross-partner data flow is also accounted for the computation of the parameter *number of arcs* in the CNC equation along with the other connector types as mentioned in Section 4.2.5.

The first parameter in the CNC equation *number of arcs* amounts to 41 (19 Sequence flows, 6 Message flow, and 16 Cross-partner data flow). The second parameter *number of activities, joins, and splits* amounts to 22. Therefore, the CNC value for the choreography model is approximately "1.86", which is lower than the previous choreography model's (in Figure 4.1) approximate CNC value of "2.86". As evident from the calculations, this data-aware choreography model possesses lower complexity value than the previous model. Due

to less number of data association connectors leading to lower value of the first parameter *number of arcs* which results the CNC value to be lower in this scenario. This is because of the consolidated cross-partner data objects resulting in fewer data flow connection to the activities (Service Tasks).

4.3.6 Durfee Square Metric and Perfect Square Metric

We first compute the Durfee Square Metric (DSM) on the data-aware choreography model depicted in Figure 4.2. As discussed in Section 4.2.6, we form a table containing a list of all the construct element types in the model along with their associated frequency of occurrence ranked in descending order as represented in Table 4.2. The only difference compared to Table 4.1 is the Data Objects here is replaced by Cross-Partner Data Objects.

Table 4.2: Construct Element types and their frequency in the choreography model in Figure 4.2.

Element types	**Frequency**
Service Tasks	8
Cross-Partner Data Object	6
Message Start Event	5
Message End Event	5
Parallel (AND) Gateway	4

From the Table 4.2 we can see that there exists four construct element types (Service Tasks, Cross-Partner Data Objects, Message Start Event, and Message End Event) each occurring at least four times in the model and the remaining one type (Parallel Gateway) does not occur more than four times. Therefore, the DSM result for this model is **four**.

For the Perfect Square Metric (PSM) computation, as can be observed from the Table 4.2, there are five construct element types in the choreography model which occur together for 28 times. Hence assuming $p:=5$, it satisfies the boundary condition of p^2 times, that is at least 25 occurrences in this scenario. Since our assumed p value is the unique largest number possible applicable to this scenario, therefore PSM value results to be **five**. As evident, there is no difference in complexity value for both the metrics between the two choreography models (Figure 4.1 and Figure 4.2).

4.3.7 Connectivity Level between Activities

The Connectivity Level between Activities (CLA) metric as discussed in Section 4.2.7, is applied to the data-aware choreography model in Figure 4.2. The first parameter, Total Number of Activities (TNA) amounts to eight, and the second parameter Number of Sequence Flows between Activities (NSFA) amounts to 25. Therefore, computing the CLA metric for the choreography model results to be approximately "0.32". The CLA value for this model is equivalent to the CLA value for the previous choreography model in Figure 4.1. There is no change in values due to the fact that the tasks and the connectors accounted for in the CLA equation are the same since the introduction of the cross-partner data objects did not impact the connectivity between activities in this scenario.

4.3.8 Halstead-based Choreography Complexity

As discussed in Section 4.2.8, the Halstead-based Choreography Complexity (HCC) measures are applied to the data-aware choreography model shown in Figure 4.2. We first compute the four sets of primitive measures (n1, n2, N1, and N2) as defined in the Section 4.2.8 for the model. The first primitive parameter *n1* amounts to four (Service Task, Message Start Event, Message End Event, and Parallel Gateway). The second parameter *n2* amounts to two (Cross-Partner Data Object, and Cross-Partner Data Object with Collection). By calculating their total number of occurrences in the choreography model, *N1* amounts to 22, and *N2* amounts to 6 as observed from the model.

Having calculated the primitive sets of measures, we now compute the three HCC metrics by applying them in their respective formulae. The Choreography Length for the model results to be ten, the Choreography Volume results to be approximately "72.38", and the Choreography Difficulty results to be six. As evident from the above computations the data-aware choreography model have less Choreography Volume and Difficulty values compared to their respective HCC values of the previous model in Figure 4.1. There is a strong reduction of 14 in Choreography Difficulty value between the two models, meaning that there is a 70% decrease in difficulty complexity level for the data-aware choreography model. Similarly there is an approximate 33.3 % fall in Choreography Volume levels as compared to the previous model. Although the Choreography Length remains the same between the two models, but overall the HCC measures for this model is lower than the previous model.

4.3.9 Structural Metrics

We hereby compute both the Structural metrics as discussed in Section 4.2.9, namely Diameter and the Coefficient of Connectivity (CoC). The longest path is "21" in this model and therefore the Diameter results to be "21". For CoC computation, we also take into

account the cross-partner data flow when calculating the parameter *number of arcs in a choreography model*. With the above assumption, the data-aware choreography model contains 41 (19 Sequence flows, 6 Message flows, and 16 Cross-Partner data flows) arcs and 28 nodes. Therefore, the CoC metric value results to be approximately "1.46". Although the Diameter value remains unchanged, however there is a slight decrease in the Coefficient of Connectivity value when compared to the previous model in Figure 4.1. We can hence say that the data-aware choreography model performed marginally better with respect to the Structural metrics as compared to the non-data-aware choreography model.

4.3.10 Summary of the BPMN model's Complexity Metric Results

We summarise all the complexity metric results (rounded to two decimal places), which were discussed in Section 4.2 and 4.3, in the Table 4.3 for both the BPMN based OPAL choreography model shown in Figure 4.1 (standard) and Figure 4.2 (with TraDE approach).

Table 4.3: Summary of the BPMN based OPAL choreography model's Complexity Metric Results

Complexity Metrics		**Standard Model (Figure 4.1)**	**TraDE Model (Figure 4.2)**
Size Metrics	**NOA**	38	24
	NOAC	42	28
	NOAJS	42	28
	M4	0.11	0.16
MCC		23	15
CFC		4	4
IC		373	226
CNC		2.86	1.86
DSM		4	4
PSM		5	5
CLA		0.32	0.32
HCC	**Length**	10	10
	Volume	108.57	72.38
	Difficulty	20	6
Structural Metrics	**CoC**	1.5	1.46
	Diameter	21	21

4.4 Application of complexity metrics to BPEL4Chor model

In this section we are going to illustrate how the BPMN choreography model shown in Figure 4.2 is manually transformed into BPEL4Chor model by taking into account the TraDE approach in designing the model and computing the complexity metrics on it. In the following sections, we are going to analyse the difference of complexity metric results between the BPEL4Chor models with the TraDE approach and the standard one (without TraDE approach). All the complexity metrics discussed so far in this chapter are applied to the BPEL4Chor model, which is transformed from the BPMN based choreography model shown in Figure 4.2.

Recollect the service choreography model (BPMN) example demonstrated in Figure 4.1 from Section 4.1, which illustrates the simulation model by employing the OPAL software. Since in the prototypical implementation we have used the BPEL4Chor modelling language for incorporating the TraDE methodology, the need arises for transforming the BPMN choreography model to BPEL4Chor model to be fed into the framework as an input. As we have implemented the framework to function for both the BPMN choreography models (without the data-aware TraDE method) as well as for the BPEL4Chor models (including the TraDE method), our complexity metrics designed for the choreography models are language and platform independent. The complexity metrics are devised in a generic fashion, so that a choreography model designed in any modelling language could be mapped according to the required input parameters for computing a specific complexity metric. Hence they can be applied across any modelling and notation language used for designing the service choreography models. In this section, we are also going to illustrate the mapping for BPEL4Chor models to the complexity metric's parameters.

4.4.1 Transformation to BPEL4Chor model

The BPEL4Chor model shown in Figure 4.3, denotes the transformed choreography model as shown from Figure 4.1 designed in BPMN. This choreography model is the standard one, meaning it is without taking into account the TraDE approach. In the later section we are going to show the same BPEL4Chor model adapted for the data-aware TraDE approach, which is the transformed version of the BPMN choreography model (with TraDE approach) shown in Figure 4.2. Since BPEL4Chor by default has no visual notation, we are showing graphical representation of all the BPEL4Chor models using the ChorDesigners [WAS+13] visual notation throughout the document. The general mapping rules and constraints applied in transforming BPMN constructs to BPEL4Chor elements for a choreography model (only those rules which are required for transforming the OPAL choreography model from Figure 4.1 to Figure 4.3) are briefly described in the following section.

The message start event in the BPMN model is transformed to a receive activity in the BPEL4Chor model. The receive activity is a trigger initiating event, which receives the call

to initiate the participant, and is generally the first activity element in a participant. The receive activity is inside the sequence activity block and belongs to a participant behaviour description (PBD) activity set.

The service task in the BPMN model is transformed to an invoke activity for the BPEL4Chor model. The invoke activity calls for the required service to be executed and it is also part of the PBD activity set. The invoke activity can call another participant by specifically transferring the control flow to the called participant. The asynchronous invoke activity (which is used in OPAL choreography model as shown in Figure 4.3) in BPEL4Chor can have only a message link originating from it and not a terminating message link.

The message end event in the BPMN model is transformed to an invoke activity in the BPEL4Chor model when there is an outgoing message link attached to it, otherwise it is transformed to a reply activity (when there is no outgoing message link and the control-flow for the concerned participant terminates there). The reply activity also belongs to the PBD activity set and is generally the last activity of a participant.

The data objects in the BPMN model are transformed to BPEL participant variables in the BPEL4Chor model (without TraDE approach). The message flow construct in the BPMN model is transformed to a message link element in the BPEL4Chor model. The parallel gateway in BPMN is transformed to a flow activity in the BPEL4Chor model.

The looping construct scenario in BPMN is modelled using a ForEach activity (it also belongs to the PBD activity set) in BPEL4Chor inside a scope activity. Each scope activity contains a sequence activity in it to signify the execution sequence. Any activity inside the ForEach is executed multiple times until the terminating condition is met. Also the ForEach is restricted to contain only one activity in it. The ForEach by its default structure contains a scope activity, inside which we can place any activity belonging to the PBD set.

The five participants in the BPMN model shown in Figure 4.1 are transformed into their respective BPEL4Chor model participants (based on the above mentioned generic BPMN to BPEL4Chor mapping rules). Each participant block contains a process element in it which encapsulates the other activities. Inside each process element there is an activity of type Sequence which contains the other Participant Behaviour Description activities inside it. All activities inside the Sequence activity block are executed in their displayed order as shown in Figure 4.3, going from top to bottom.

Next we discuss the transformation to BPEL4Chor model adapted for the data-aware TraDE approach. The BPEL4Chor model shown in Figure 4.4 illustrates the Opal Simulation scenario incorporated with the TraDE methodology. The cross-partner data object and the cross-partner data flow are the two quintessential aspect of the TraDE applied BPEL4Chor model which are represented by the data objects and the data connectors in the choreography model shown in Figure 4.4.

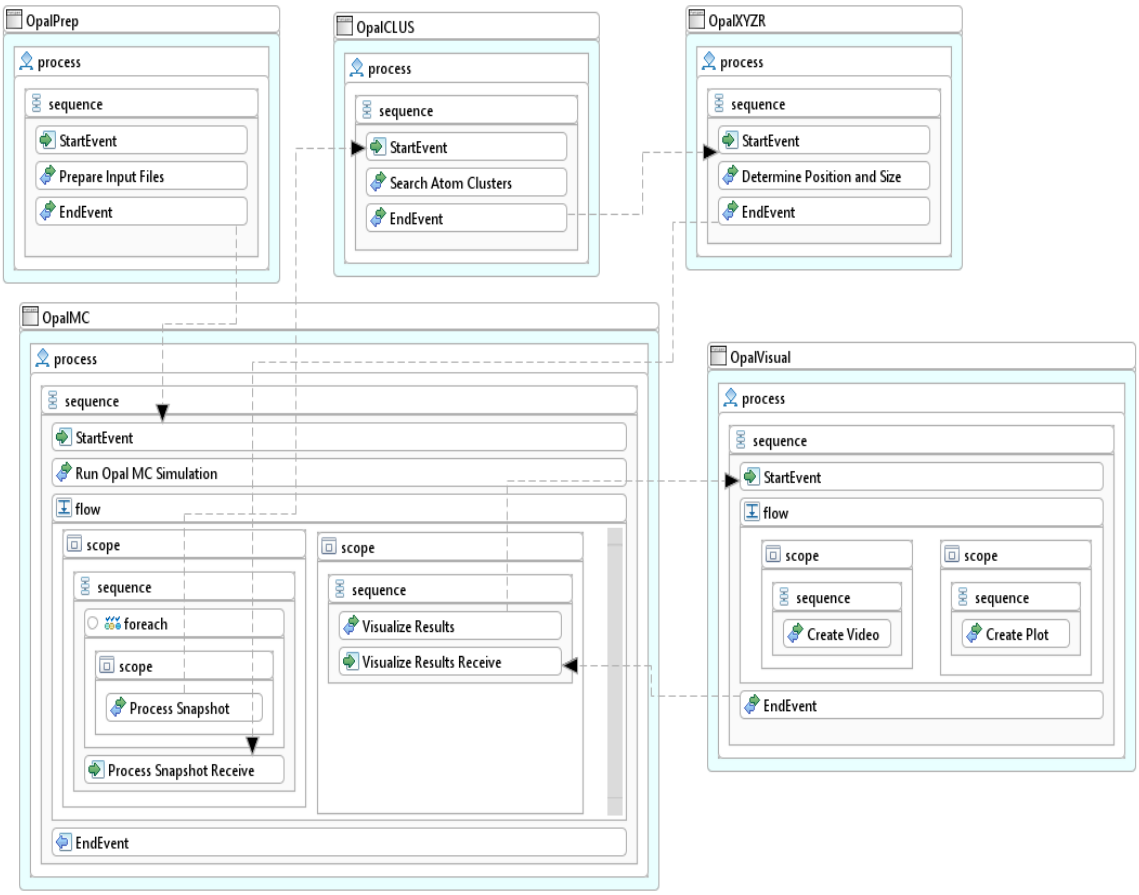

Figure 4.3: BPEL4Chor model of KMC Simulation using Opal Software [WAS+13].

The data objects are represented in blue colour heading as can be seen in the Figure 4.4. Similarly the data connectors are sketched by blue dotted lines having an arrowhead at the receiver's end. The data objects can contain multiple data elements in it which are shared amongst the participants. The data elements are represented by white rectangular boxes inside a data object.

The cross partner data objects and data flow shown in the BPMN model of Opal simulation in Figure 4.2 are transformed to data objects and data connectors respectively in the BPEL4Chor model shown in Figure 4.4. As can be seen there are total six data objects in the BPEL4Chor model as was in the BPMN model.

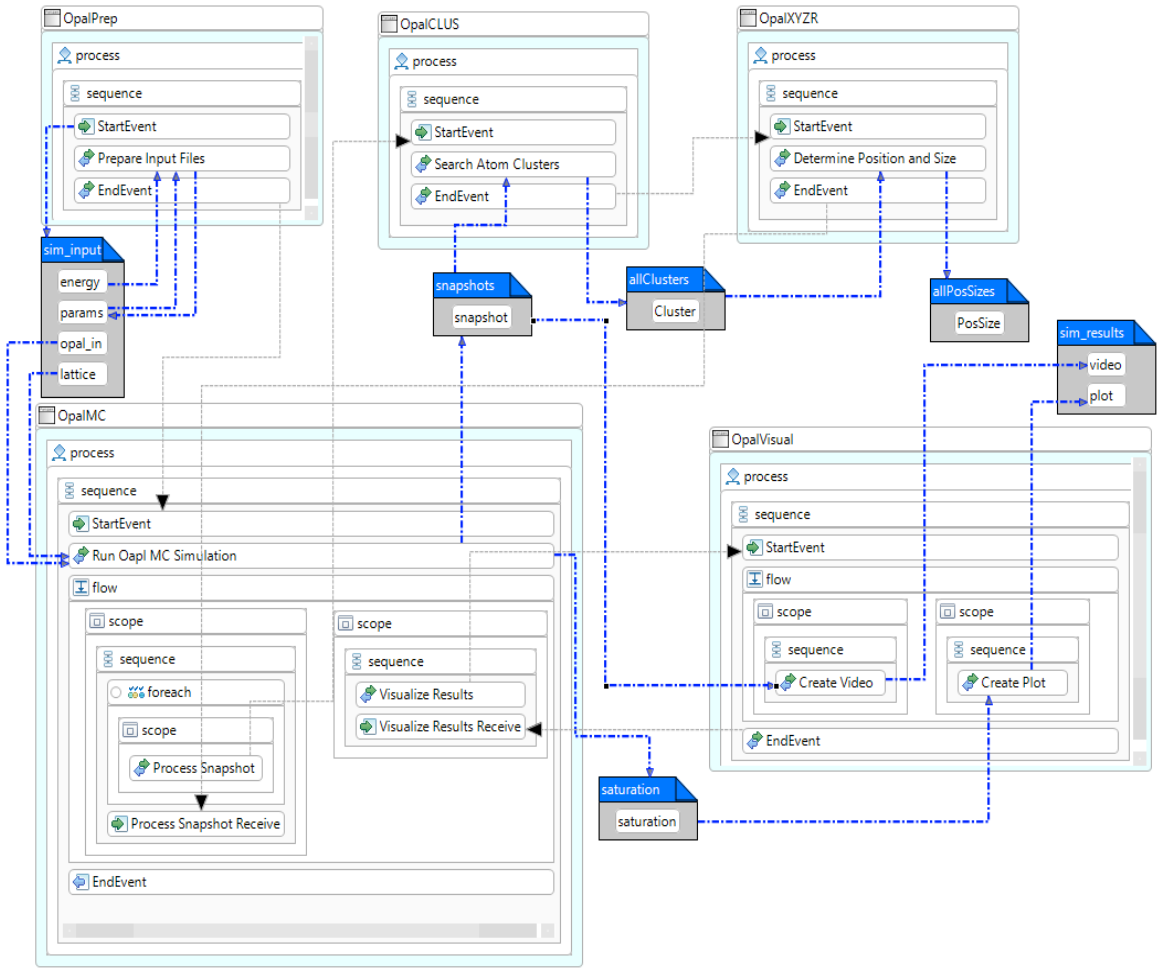

Figure 4.4: TraDE based BPEL4Chor model of KMC Simulation using Opal Software [WAS+13].

4.4.2 Complexity of the BPEL4Chor model

The complexity metrics for the service choreography model in BPMN are applied here to the BPEL4Chor model for the same Opal Simulation scenario. We compute all the metrics for both the BPEL4Chor models shown in Figure 4.3 (standard one without TraDE) and Figure 4.4 (with TraDE approach) and then analyse and compare the metric results for the two models.

For the reason of simplicity and verbosity reduction we refer the standard BPEL4Chor model (without TraDE) as the first model (Model 1) and the one with the TraDE approach as the second model (Model 2).

4.4.2.1 The Size Metrics

We apply all the four size metrics which were discussed in the Section 4.2.1 to both the BPEL4Chor models as mentioned before. The first metric (M1), Number of Activities (NOA), counts the total number of activities contained in the BPEL4Chor model which includes the Invoke activity, Reply activity, Receive activity, ForEach activity, OpaqueActivity, While activity and Data Objects. There are seven receive activities, twelve Invoke activities, one Reply activity, and one ForEach activity common in both the BPEL4Chor models. The result of NOA for the first model is 21 and for the second model is 27. There are six data objects in the second model which are represented as variables in the first model. Due to the implicit data flow in BPEL and not taking BPEL variables into account as data containers, the difference between the two model's results is six.

The second metric (M2), Number of Activities and Control flow elements (NOAC) also takes into account the control flow structures in addition to the activity count in NOA, which includes the gateway representative structures like the Flow activity (parallel gateway-AND) and the If/ElseIf/Else activity (exclusive gateway-XOR). There are two flow activities each in both the first and the second model, resulting in two control structures in addition to the NOA count for the NOAC computation. Hence the result of NOAC for the first model is 23 and for the second model is 29.

The third metric (M3), Number of Activities, Joins and Splits (NOAJS) is same as of the NOAC count in the case of BPEL4Chor models since there are no concepts of joins (converging) and splits (diverging) control structures (gateways - Flow and If/ElseIf/Else) for BPEL4Chor models. Hence in this scenario the NOAJS result for the first model is 23 and for the second model is 29.

The fourth metric (M4) computes from the previous results (M1 and M3). The result of M4 for the first model is "0.10" and for the second model is approximately "0.07".

4.4.2.2 McCabe's Cyclomatic Complexity

We apply the McCabe's Cyclomatic Complexity (MCC) metric to both the BPEL4Chor models as described in the Section 4.2.2. The edges (e) from the MCC equation represents the sequence flows, data connectors, and message links in the case of BPEL4Chor model. The sequence flow is computed according to the number of activities inside a sequence activity block and the number of sequence flow required for connecting each of them depending on the nature of the activity defined. For example in the case of a flow activity, the sequence flow is computed based on the number of scope activities present in it. The data connectors and message links are directly computed from their total occurrence count in the respective models.

The nodes (*n*) in the MCC equation depicts some activities belonging to the PBD set like the Invoke, Receive, Reply, ForEach, OpaqueActivity, If/ElseIf/Else and the Flow activity as well. The count of data objects are also accounted for in the nodes computation.

We compute the MCC metric for the BPEL4Chor model as a whole (from a global perspective). The number of edges for the first model amounts to 27 (21 sequence flows and six message links) and the number of nodes amounts to 23 (seven Receive activities, twelve Invoke activities, one Reply activity, one ForEach activity, and two Flow activities). Hence the MCC result for the first model is six.

Similarly we compute the MCC metric for the second model. The number of edges for the second model amounts to 43 (21 sequence flows, six message links, and 16 data connectors) and the number of nodes amounts to 29 (seven Receive activities, twelve Invoke activities, one Reply activity, one ForEach activity, two Flow activities and six Data objects). Hence the MCC result for the second model is 16.

As evident from the above computation of the MCC metric for both models, there is a difference of ten between the two results due to the count of data objects and data connectors occurring only in the second model (because in the first model data connectors are expressed as implicit data flow in BPEL and data objects are represented as BPEL variables – both are not accounted for in the metric computation since they are not explicitly specified in the model), pushing the edges count to 43 whereas the node count is increased to only 29. This explains the difference of ten between the MCC results of the two models.

4.4.2.3 The Control-flow Complexity Metric

In this section we will compute the Control-Flow Complexity (CFC) metric for both the BPEL4Chor models as described in the Section 4.2.3. The CFC for exclusive (XOR) gateway can be applied to the IF/ElseIf/Else activity structure in the case of BPEL4Chor models since the exclusive gateway in BPMN models is equivalent to the IF/ElseIf/Else activity in BPEL4Chor models. Similarly the CFC for parallel (AND) gateway can be applied to the Flow activity for BPEL4Chor models since the parallel gateway in BPMN models is equivalent to the Flow activity in BPEL4Chor models. The CFC for inclusive (OR) gateway in BPMN models has no such equivalent structure here in the case for the BPEL4Chor models, hence we omit the CFC for OR gateway calculation in this scenario.

So the CFC metric for the BPEL4Chor models are computed only for the CFC_{XOR} and CFC_{AND} components. For the CFC_{XOR} computation in BPEL4Chor models, the fan-out is directly correlated to the number of If, ElseIf and Else conditions present in the IF/ElseIf/Else activity structure, since it represents the possible number of outcomes branching emerging after the execution of IF/ElseIf/Else activity. The CFC_{AND} computation in BPEL4Chor models is a straightforward calculation for every Flow activity structure, which results to one.

There are two Flow activities each in both the BPEL4Chor models. Hence the CFC result for both the first and the second model is two since each Flow activity accounts for only one in the CFC computation.

4.4.2.4 Interface Complexity Metric

In this section we will compute the Interface Complexity (IC) metric for both the BPEL4Chor models as described in the Section 4.2.4. The three parameters in the IC equation (*length, number of inputs,* and *number of outputs*) is computed for each participant in the BPEL4Chor model.

The participant's *length* in the case of a BPEL4Chor model is equivalent to the number of activities (not all activities in BPEL4Chor modelling are accounted for) specified inside the participant. The activities which are accounted for computing the participant's *length* in the BPEL4Chor model are Receive, Invoke, Reply, OpaqueActivity, Flow, and If. The *number of inputs* parameter in the IC equation represents the incoming data connector's count, which means the number of data connectors terminating on or in the participant. Similarly the *number of outputs* parameter is computed by the count of outgoing data connectors, meaning the number of data connectors originating from the participant. If there are no incoming or outgoing data connectors associated with a participant, we assume the corresponding parameter value to be taken as one instead of zero (refer the Section 4.2.4 for an elaborate discussion on the reason for this assumption).

We now compute the IC metric for the first BPEL4Chor model by calculating the IC value for each participant in the model and finally sum them up to get the IC result for the whole BPEL4Chor model. Since BPEL does not support the specification of explicit data flow and therefore no data connectors can exist in the first model, we thereby assume the *number of inputs* and the *number of outputs* parameter value to be one (instead of zero) for computing the IC metric for each of the participants. For participant P1 (OpalPrep) the *length* value is three, *number of inputs* is one and the *number of outputs* is also one. Hence the IC value for P1 results to be three. For participant P2 (OpalMC) the *length* value is eight, *number of inputs* is one and the *number of outputs* is also one. Hence the IC value for P2 results to be eight. For participant P3 (OpalCLUS) the *length* value is three, *number of inputs* is one and the *number of outputs* is also one. Hence the IC value for P3 results to be three. For participant P4 (OpalXYZR) the *length* value is three, *number of inputs* is one and the *number of outputs* is also one. Hence the IC value for P4 also results to be three. For participant P5 (OpalVisual) the *length* value is five, *number of inputs* is one and the *number of outputs* is also one. Hence the IC value for P5 results to be five. The IC metric value for the first model results to be 22.

Now we compute the IC metric for the second model in a similar procedure as followed for the first model. For the second model there are data connectors associated with each participant, hence the parameters in the IC equation *number of inputs* and the *number of*

outputs values are not zero in this scenario, although the parameter *length* values of each participant is same as of the first model. For participant P1 (OpalPrep) the *length* value is three, *number of inputs* is two and the *number of outputs* is also two. Hence the IC value for P1 results to be 48. For participant P2 (OpalMC) the *length* value is eight, *number of inputs* is two and the *number of outputs* is also two. Hence the IC value for P2 results to be 128. For participant P3 (OpalCLUS) the *length* value is three, *number of inputs* is one and the *number of outputs* is also one. Hence the IC value for P3 results to be three. For participant P4 (OpalXYZR) the *length* value is three, *number of inputs* is one and the *number of outputs* is also one. Hence the IC value for P4 also results to be three. For participant P5 (OpalVisual) the *length* value is five, *number of inputs* is two and the *number of outputs* is also two. Hence the IC value for P5 results to be 80. The IC metric value for the second model results to be 262.

We can see a strong difference of the IC metric results between the two BPEL4Chor models, resulting the IC value of the second model to be 240 more than the first model's IC value. This is solely due to the fact that there are no data connectors associated with any of the participants in the first model (since data connectors are expressed as implicit data flow in BPEL and hence they are not accounted for in the metric computation), which leads to the assumption of taking the value as one for two crucial parameters in the IC equation resulting in lower IC values for each of the participants in the first model.

4.4.2.5 The Coefficient of Network Complexity Metric

In this section we will compute the Coefficient of Network Complexity (CNC) metric for both the BPEL4Chor models as described in the Section 4.2.5. The parameter *number of arcs* in the CNC equation counts all the connector types available under BPEL4Chor modelling specification, which includes the message links, data connectors and the sequence flows which we have adapted for BPEL4Chor models from the BPMN 2.0 specification. The number of sequence flows in a BPEL4Chor model is computed by an algorithm to count the number of activities present inside a sequence activity block depending on their nature and structure style. The second parameter in the CNC equation *number of activities, joins, and splits* is calculated by the count of Invoke activity, Reply activity, Receive activity, ForEach activity, OpaqueActivity, While activity, Flow activity, and If/ElseIf/Else activity, where the Flow and If/ElseIf/Else activities represents the joins and splits in the parameter and rest of the mentioned activity types belongs to the activities in the parameter.

For the first BPEL4Chor model, the *number of arcs* is 27 (21 sequence flows and six message links), and the *number of activities, joins, and splits* is 23 (seven Receive activities, twelve Invoke activities, one Reply activity, one ForEach activity, and two Flow activities). Hence the CNC result for the first model is approximately "1.17".

Similarly for the second BPEL4Chor model, the *number of arcs* is 43 (21 sequence flows, six message links, and 16 data connectors), and the *number of activities, joins, and splits* is 23 (seven Receive activities, twelve Invoke activities, one Reply activity, one ForEach activity, and two Flow activities). Hence the CNC result for the second model is approximately "1.87".

The CNC result for the second model is significantly higher compared to the first model since the second parameter's (*number of activities, joins, and splits)* value of the CNC equation is same for both the models while the first parameter's (*number of arcs*) value of the CNC equation have increased from 27 in the first model to 43 in the second model (since data connectors in the first model are expressed as implicit data flow in BPEL and hence they are not accounted for in the metric computation) due to the addition of data connectors in the second model.

4.4.2.6 Durfee Square Metric and Perfect Square Metric

In this section we will compute the Durfee Square Metric (DSM) and Perfect Square Metric (PSM) for both the BPEL4Chor models as described in the Section 4.2.6. The construct elements (refer to the definition of DSM and PSM from Section 4.2.6) mentioned in the DSM and PSM definitions, in case of the BPEL4Chor models represents the Invoke activity, Reply activity, Receive activity, ForEach activity, OpaqueActivity, Scope activity, While activity, Flow activity, If/ElseIf/Else activity, and Data Objects.

For the purpose of DSM and PSM computation we form a table for each of the BPEL4Chor models containing a list of all the construct element types present in the choreography model along with their associated frequency of occurrence ranked in descending order. The Table 4.4 represents the construct elements for the first BPEL4Chor model whereas Table 4.5 represents the construct elements for the second BPEL4Chor model.

Table 4.4: Construct Element types and their frequency in the BPEL4Chor model in Figure 4.3.

Element types	**Frequency**
Invoke	12
Receive	7
Scope	4
Parallel Gateway (Flow)	2
Reply	1
Other Task	1

For DSM computation of the first BPEL4Chor model, we can see from Table 4.4 that there are three types of construct elements (Invoke, Receive, and Scope) each occurring at least three times each in the model and the remaining three types (Parallel Gateway-Flow, Reply, and Other Task) do not occur more than three times. Hence the DSM result for the first BPEL4Chor model is three**.**

For PSM computation of the first BPEL4Chor model, as can be seen from Table 4.4 that when assuming *p* to be 5, the combined occurrence of the top five construct element types (Invoke, Receive, Scope, Parallel Gateway-Flow, and Reply) is 26 times which satisfies the boundary condition of p^2 times, meaning at least 25 occurrences. Since our assumed *p* value is the unique largest number possible for our concerned model (assuming *p* to be 6, combined occurrence is only 27, failing to satisfy the boundary condition of at least 36 occurrences), hence the PSM results to be five**.**

Table 4.5: Construct Element types and their frequency in the BPEL4Chor model in Figure 4.4.

Element types	**Frequency**
Invoke	12
Receive	7
Data Object	6
Scope	4
Parallel Gateway (Flow)	2
Reply	1
Other Task	1

For DSM computation of the second BPEL4Chor model, we can see from Table 4.5 that there are four types of construct elements (Invoke, Receive, Data Object, and Scope) each occurring at least four times each in the model and the remaining three types (Parallel Gateway-Flow, Reply, and Other Task) do not occur more than four times. Hence the DSM result for the second BPEL4Chor model is four**.**

For PSM computation of the second BPEL4Chor model, as can be seen from Table 4.5 that when assuming *p* to be 5, the combined occurrence of the top five construct element types (Invoke, Receive, Data Object, Scope, and Parallel Gateway-Flow) is 31 times which satisfies the boundary condition of p^2 times, meaning at least 25 occurrences. Since our assumed *p* value is the unique largest number possible for our concerned model (assuming *p* to be 6, combined occurrence is only 32, failing to satisfy the boundary condition of at least 36 occurrences), hence the PSM results to be five**.**

The DSM result (four) for the second model is one more than the corresponding result (three) for the first model due to the fact that there are six more data objects (data objects in the first model are expressed as BPEL variables and since they are not explicitly specified

in the choreography model, they are not accounted for in the metric computation) in the second model than in the first model (rest of the construct elements are same); whereas the PSM result is same for both the BPEL4Chor models.

4.4.2.7 Connectivity Level between Activities

In this section we will compute the Connectivity Level between Activities (CLA) metric for both the BPEL4Chor models as described in the Section 4.2.7. The first parameter *Total Number of Activities* (TNA) in the CLA equation denotes the Invoke activity, Reply activity, Receive activity, ForEach activity, OpaqueActivity, and While activity in the case of the BPEL4Chor models. The second parameter *Number of Sequence Flows between Activities* (NSFA) in the CLA equation represents the sequence flows between activities and the message links between the participants in the BPEL4Chor models.

For the first BPEL4Chor model, the TNA parameter value is 21 (seven Receive activities, twelve Invoke activities, one Reply activity, and one ForEach activity) and the NSFA parameter value is 27 (21 sequence flows and six message links). Hence the CLA metric for the first model results to be approximately “0.78”.

For the second BPEL4Chor model, the TNA parameter value is 21 (seven Receive activities, twelve Invoke activities, one Reply activity, and one ForEach activity) and the NSFA parameter value is 27 (21 sequence flows and six message links). As can be seen both the parameter (TNA and NSFA) values are same as of the first model, since the CLA metric does not take into account the count of the data objects or the data connectors, which is the major difference between the two models. Hence the CLA metric for the second model also results to be approximately “0.78”.

4.4.2.8 Halstead-based Choreography Complexity

In this section we will compute the Halstead-based Choreography Complexity (HCC) metric for both the BPEL4Chor models as described in the Section 4.2.8. First we need to calculate the four sets of primitive measures (*n1, n2, N1, N2*) by adapting them for the BPEL4Chor models.

The first variable *n1* represents the number of unique activity types like the Receive activity, Invoke activity, Reply activity, Scope activity, ForEach activity, While activity, IF/ElseIf/Else activity and Flow activity. The second variable *n2* represents the number of unique data container types like the Data Objects and Data Elements. The third variable *N1* denotes the total number of *n1* occurrences. The fourth variable *N2* denotes the total number of *n2* occurrences.

We now calculate the four variables (*n1, n2, N1, N2*) for the first BPEL4Chor model. The first variable *n1* amounts to six (Receive activity, Invoke activity, Reply activity, Scope

activity, ForEach activity, and Flow activity). The second variable *n2* amounts to null since there are no data container types present. But we make an assumption here that if *n2* is zero we take it as one. Since *n2* becoming zero would mean that *N2* also becomes zero automatically, hence we also assume *N2* to be one in this scenario. This assumption is established to negate the deceptive outcome of the Choreography Length, Volume, and Difficulty values, since n2 and N2 being zero therefore it will impact the formulae in a considerable manner. The third variable *N1* amounts to 27 (seven Receive activities, twelve Invoke activities, one Reply activity, four Scope activities, one ForEach activity, and two Flow activities). The fourth variable *N2* should be zero but due to our aforementioned assumption it is taken to be one.

By using the values of the above computed four variables the HCC metrics are computed for the first BPEL4Chor model as follows: the Choreography Length results to be approximately "15.51", the Choreography Volume results to be approximately "78.61", and the Choreography Difficulty results to be "3.00".

Similarly we calculate the four variables (*n1, n2, N1, N2*) for the second BPEL4Chor model. The first variable *n1* amounts to six (Receive activity, Invoke activity, Reply activity, Scope activity, ForEach activity, and Flow activity). The second variable *n2* amounts to two (Data Object and Data Element).The third variable *N1* amounts to 27 (seven Receive activities, twelve Invoke activities, one Reply activity, four Scope activities, one ForEach activity, and two Flow activities). The fourth variable *N2* amounts to 16 (six Data Objects and ten Data Elements).

By using the values of the above computed four variables the HCC metrics are computed for the second BPEL4Chor model as follows: the Choreography Length results to be approximately "17.51", the Choreography Volume results to be "129.00", and the Choreography Difficulty results to be "24.00".

As can be seen from the HCC results of the two BPEL4Chor models, there is a slight increase of Choreography Length of the second model compared to the first model. Similarly the Choreography Volume has also increased significantly for the second model when compared to the first model. But the most significant increase has been for the Choreography Difficulty value for the second model in comparison to the first model, it is an eight fold rise. The increase in values of the HCC metrics for the second model is due to the fact that the data objects and the data elements are added only in the second model for the TraDE approach whereas in the case of the first model these elements are expressed as BPEL participant variables which are not accounted for in the complexity measures as they are not explicitly specified in the choreography model.

4.4.2.9 Structural Metrics

In this section we will compute the two (Diameter and Coefficient of Connectivity) Structural metrics for both the BPEL4Chor models as described in the Section 4.2.9. The

Diameter is directly adapted for the BPEL4Chor models by the usage of sequence flows. The first parameter in the Coefficient of Connectivity (CoC) equation *number of arcs* represents the count of connectors like the sequence flows, the message links and the data connectors in the case of BPEL4Chor models. The second parameter *total number of nodes* represents the count of all construct nodes like the Invoke activity, Reply activity, Receive activity, ForEach activity, OpaqueActivity, While activity, Flow activity, If/ElseIf/Else activity, and Data Objects.

For the first BPEL4Chor model the Structural metrics are computed as follows. The Diameter results to be 22. The first parameter of CoC equation *number of arcs* is 27 (21 sequence flows and six message links) and the second parameter *total number of nodes* is 23 (seven Receive activities, twelve Invoke activities, one Reply activity, one ForEach activity, and two Flow activities). Hence the CoC metric results to be approximately "1.17".

For the second BPEL4Chor model the Structural metrics are computed as follows. The Diameter results to be 22. The first parameter of CoC equation *number of arcs* is 43 (21 sequence flows, six message links, and 16 data connectors) and the second parameter *total number of nodes* is 29 (seven Receive activities, twelve Invoke activities, one Reply activity, one ForEach activity, two Flow activities, and six Data Objects). Hence the CoC metric results to be approximately "1.48".

The Diameter value is same for both the BPEL4Chor models but CoC value is greater in case of the second model compared to the first model since *number of arcs* parameter value is quite higher (43 in second model and 27 in first model) due to the large number of data connectors in the second model (since data connectors are expressed as implicit data flow in BPEL and hence they are not accounted for in the metric computation for first model).

4.4.3 Comparison of the Complexity Metric Results

We summarise all the complexity metric results (rounded to two decimal places), which were discussed in Section 4.4.2, in the Table 4.6 for both the aforementioned BPEL4Chor models.

Table 4.6: Summary of the BPEL4Chor based OPAL choreography model's Complexity Metric Results

Complexity Metrics		**Model 1 (Figure** 4.3**)**	**Model 2 (Figure** 4.4**)**
Size Metrics	**NOA**	21	27
	NOAC	23	29
	NOAJS	23	29
	M4	0.10	0.07
MCC		6	16
CFC		2	2

IC		22	262
CNC		1.17	1.87
DSM		3	4
PSM		5	5
CLA		0.78	0.78
HCC	**Length**	15.51	17.51
	Volume	78.61	129
	Difficulty	3	24
Structural Metrics	**CoC**	1.17	1.48
	Diameter	22	22

CHAPTER 5

Implementation

This chapter discusses the details of the prototypical implementation of all the complexity metrics, as discussed in chapter 4, applied to a collection of service choreography models we have designed. The following sections present the details about the prototype's architecture as well as the finer details of the complexity metrics along with its assumptions cater to meet the needs of the TraDE approach integrated in the choreography models.

5.1 Architecture

The prototypical implementation of the complexity metrics are realized through a classical 3-tier architecture of the framework. Figure 5.1 demonstrates the overall architecture of the prototype in three separate layers.

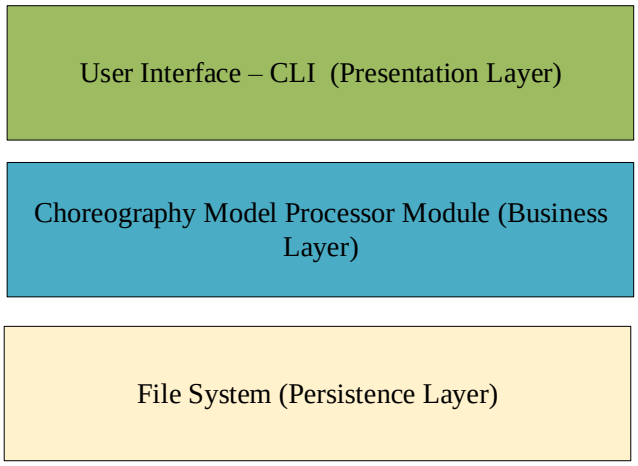

Figure 5.1: Generic 3-tier architecture of the framework's prototypical implementation.

In the presentation layer, the user interface is designed as the command line interface pattern where the user gives the choreography model's file path and selects the complexity metrics to be applied on it as input. Once the user has entered the input details, the choreography

model is loaded, parsed and the complexity metrics are computed in the business layer. The results of the complexity metrics are displayed to the user instantly on the console as well as written to a result file which is stored in the file system for later retrieval. The latter is realized in the third layer, which is the storage or persistence layer.

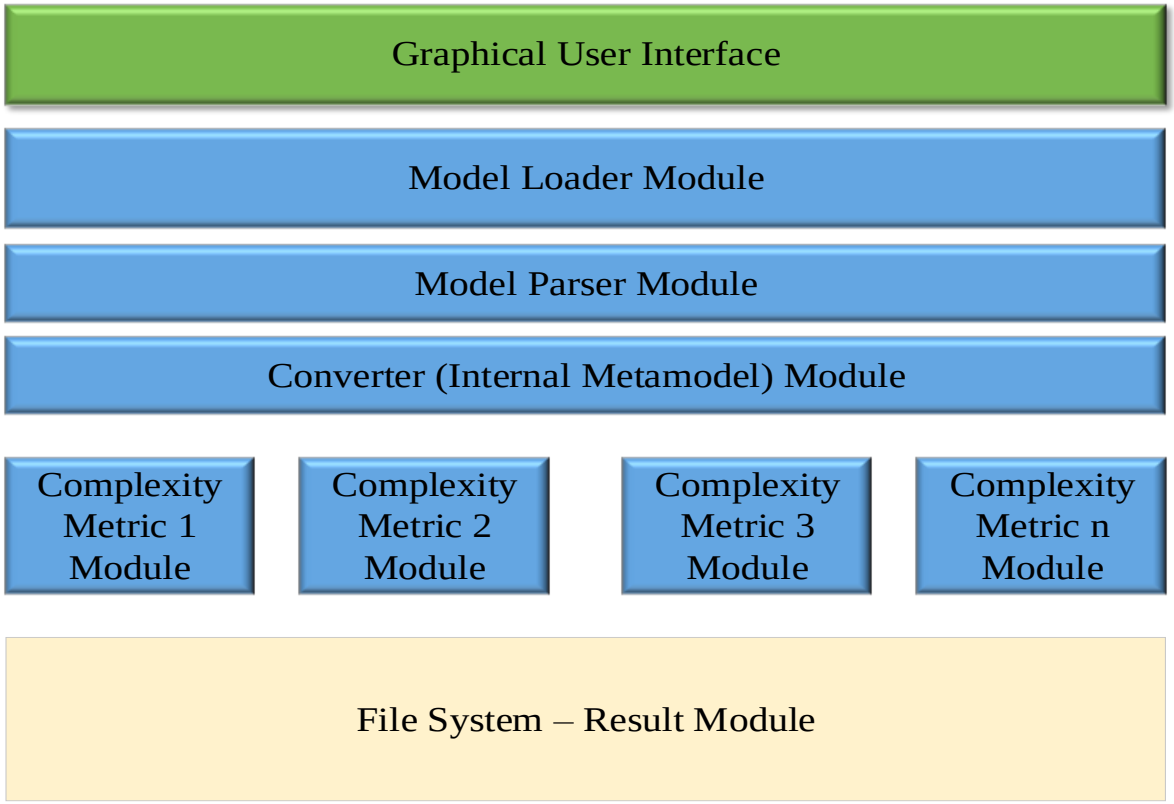

Figure 5.2: The detailed 3-tier architecture of the framework's prototypical implementation.

In Figure 5.2 we can see the architecture of the prototype in finer details. The first layer represents the *Graphical User Interface* with which the user interacts with the tool.

The second layer, business logic layer, has apparently been subdivided into four layers. The *Model Loader Module* retrieves the choreography model identified by the file path given by the user. It loads the choreography model into the environment for further processing. The next sublayer *Model Parser Module* parses the choreography model's XML file into its root elements and associated subparts. It parses the XML file according to the element tag associated with each construct nodes.

The next sublayer in this business layer is the *Converter Module* that transforms the parsed choreography model's XML file into an internal meta-model, which is suitable for feeding

into the *Complexity Metric Modules* in the next sublayer for computation of the metrics. The internal meta-model is designed in such a way that the input parameters for the complexity metrics computation can easily be accessed for passing it into the subsequent metric computation modules as required by the corresponding function module.

The next sublayer contains several modules, each representing a particular complexity metric computation module. These modules are in the same sublayer for parallel processing, resulting in faster response. Each module in this sublayer apply their respective complexity metrics on the selected choreography model and displays the computation results instantaneously in the console through the first layer, *Graphical User Interface*.

The output from the previous *Complexity Metric Modules* are passed onto the next layer, which is the third layer in the architecture, the *Result Module*. This layer represents the file system, and the *Result Module* stores the output of the different complexity metrics persistently in a result file. The result file is stored in the file system for easy accessibility in the future. The result file also contains the input choreography model's details such as the filename and file path to correctly identify the particular choreography model corresponding to the metric results.

5.2 Implementation Details

For the prototypical implementation of the complexity metrics we have used Java as the main programming language. The storage layer is implemented as the file system. This is used for storing the application tool related files, the service choreography model files which are used as input to the tool, and the result file containing the output. For accessing and operating with the file system we use a set of standard Java libraries. For running the prototype we depend on the file system of the computer it operates on.

We have used the Business Process Model and Notation (BPMN) 2.0 for designing and building the standard service choreography models according to the BPMN 2.0 specification. For parsing the BPMN xml file, we have used the Eclipse Modelling Framework [EF18a] and the available BPMN EMF model [EF18b] files for accessing the model's subparts. For implementing the complexity metrics on the choreography models designed by the TraDE approach, BPMN 2.0 could not be used for building the models since BPMN 2.0 does not support the TraDE approach by default as of now.

The two quintessential aspect of the TraDE approach, the cross partner data objects and the cross partner data flow is not supported by the BPMN 2.0 specification. For implementing these two critical aspects of the TraDE method we had to extend the metamodel of the modelling framework which is out of scope for our work. So we replace the implementation of the TraDE approach with the BPEL4Chor technology instead of BPMN 2.0 for smoother transition.

BPEL4Chor is the choreography extension for the Business Process Execution Language (BPEL). BPEL4Chor can also be perceived as an interchange format which helps in designing choreographies in all stages. Since TraDE method is not supported by BPMN 2.0, the input choreography models are manually translated from BPMN choreographies into our TraDE supported BPEL4Chor model.

After the mapping of BPMN to BPEL4Chor models, all the complexity metrics are implemented again in Java for the BPEL4Chor model. Every BPEL4Chor model have an associated XML file, which is similar to the BPMN model's XML file. But the constructs for the BPEL4Chor model are somewhat different compared to the BPMN model. The XML file of the BPEL4Chor model is parsed similar to the BPMN with the help of BPEL EMF model. The parsed file is used to generate the parameters required for the complexity metrics implementation. The input parameters for the complexity metric's computation modules varies according to its requirement.

The User Interface (UI) for the BPEL4Chor model's complexity metric implementation framework have been designed in the same way as of the BPMN framework to maintain consistency. The result of the complexity metrics applied on the BPEL4Chor model are similarly written to an output result file.

The transformation of BPMN choreography models to BPEL4Chor choreography model is not always straightforward like a one to one mapping. In this section, we are going to describe the mapping rules we have applied for translating from BPMN to BPEL4Chor models.

The Service Task (for synchronous invocation) and the Send Task (for asynchronous invocation) in BPMN are mapped to Invoke activities in BPEL4Chor. Invoke activity is a type of Participant Behaviour Description (PBD). If it is a black box task, it can be mapped to an OpaqueActivity.

In looping mechanisms, ForEach and While under BPEL4Chor are available for mapping the loop element in BPMN. Single Participant are mapped similar to BPMN, whereas in BPEL4Chor we have an extra option as Participant Set for denoting multiple Participants. Start events are transformed to Receive activities when it is a non-triggering event. A trigger event of the type multiple is mapped to a Pick activity.

End events are mapped to either an Invoke activity or a Reply activity depending on the outgoing transitions and the pattern in the choreography model. If the end event contains a synchronous call to another participant, we can use the Reply activity and for an asynchronous call we transform it into an Invoke activity. Also the subprocess in BPMN are mapped to the Invoke activity for BPEL4Chor.

The basic gateway transformation are realized as the following. For parallel gateways (AND) in BPMN, it is mapped to a Flow activity in BPEL4Chor which may possess multiple parallel executable paths. Flow Activity Link in BPEL4Chor are used for building continuing parallel paths inside a Flow activity for a particular sequence flow path. For exclusive gateways (XOR) in BPMN are mapped to an If/ElseIf/Else activity in

BPEL4Chor. For inclusive gateways (OR) in BPMN, there are no specific mappings in BPEL4Chor. It is transformed based on the pattern and type of flow object that follows the inclusive gateway.

Swimlanes in BPMN are generally not mapped to BPEL4Chor. The Data Objects in the BPMN (standard choreography model) are mapped to the participant variables. There are three types of variables specified in BPEL4Chor notation, namely the Simple type, Message type, and Element type. The Simple type of variable can possess an XML schema of simple data types, the Message type can hold a WSDL message and the Element type holds an XML schema element. In our case for the TraDE approach, the Data Objects in BPMN are mapped directly to the Data Objects in the extended BPEL4Chor notation adapted for the TraDE method. An actual data element is mapped onto the Data Element contained inside a Data Object for the BPEL4Chor TraDE method. For the TraDE approach, extended BPEL4Chor allows for holding multiple Data Elements inside a single Data Object.

The Data Associations in the BPMN model are mapped to Data Connectors only in the BPEL4Chor model with TraDE approach extension. The sequence flow in BPMN are not mapped directly for BPEL4Chor, instead Sequence is used as a housing or container activity for BPEL4Chor which contains several other activities inside it which are executed in sequence. The message flow in BPMN are directly mapped to Message Links for BPEL4Chor.

Scope activity in BPEL4Chor can be used to map some activities in BPMN which occur together inside the same scope, meaning Scope is again a type of container activity. There are some other constructs available in BPEL4Chor like the Pick, OnMessage, OnAlarm etc. which were not required in our case for mapping BPMN choreography models to BPEL4Chor, and hence are omitted here from the discussion for simplicity.

With the help of above defined transformation rules for converting the BPMN choreography models to the BPEL4Chor models, we have mapped our input BPMN models according to the BPEL4Chor model specification. These transformed BPEL4Chor models are then used as an input into the framework for computing the complexity metrics applied on our TraDE adapted service choreography models.

5.3 Dynamic Aspect of Interaction with the Framework

In the Figure 5.3 we show an example illustrating the standard interactions between the User and the parts of the Framework which are actually responsible for carrying out the tasks at hand. The Figure 5.3 shows an UML sequence diagram denoting the interactions involved between the User and the Framework for a particular scenario of computing the complexity metric Diameter (Structural Metric) in this case.

The UML sequence diagram illustrates the dynamic control-flow transfer from one module to another. Here the User communicates with the Graphical User Interface (GUI) of the

framework, which is the interaction point for users of the tool. The User does not need to know the actual functioning of the internal modules involved in this scenario, like the File System and the Model Manager. The Model Manager module, as presented in the sequence diagram, is a representational encapsulation of the functionalities of the three modules (Model Loader, Model Parser, and Converter Module) contained in the business logic layer of the prototype's architecture. The GUI acts as an abstraction layer to the User of the framework.

The sequence diagram does not account for all the possible interactions and use case for this particular complexity metric computation scenario and the probable problems which might occur in between the interaction. For example, potential fault might occur due to the wrong User input or possible failure of the File system due to the incorrect file path of the input choreography model given by the User. Another cause of error can be due to the missing file in the file path given by the User. It can also happen such that the input model file specified by the User is not a valid choreography model's XML file (BPMN or BPEL4Chor) as expected by the framework, or the access to the file is restricted for the reason of protection. For all the above mentioned causes of potential failure in loading the appropriate choreography model's XML file, a corresponding suitable error message is displayed to the User. The error reporting is done by the GUI.

Once the corresponding file is accessed and loaded by the Model Manager through the use of the File System, the loading confirmation is displayed to the User by the Interaction Point (GUI) and asks for the User's choice for selecting the complexity metrics which are to be applied on the choreography model. Once the User enters the list of metric choices for the complexity computation, the input is passed to the Model Manager by the GUI. The Model Manager then calls the corresponding metric computation module for the respective user's choice. In this case of the sequence diagram, the Diameter Metric Module is called for computing the Diameter of the choreography model, which is executed using a recursive algorithm.

Once the Diameter Metric module finishes computing the Diameter of the model, the result of the metric is written to the output file by the File System object. On successful writing of the result to the file, the control is passed onto the GUI for displaying the final result to the User on the console.

5.4 Result File

The framework computes the respective metrics and displays the result of each metrics to the User instantly on the console. Although the result is shown to the User on the console, the complete result is also finally written to an output text file for permanent storage and later access, since the result shown on the console is temporary.

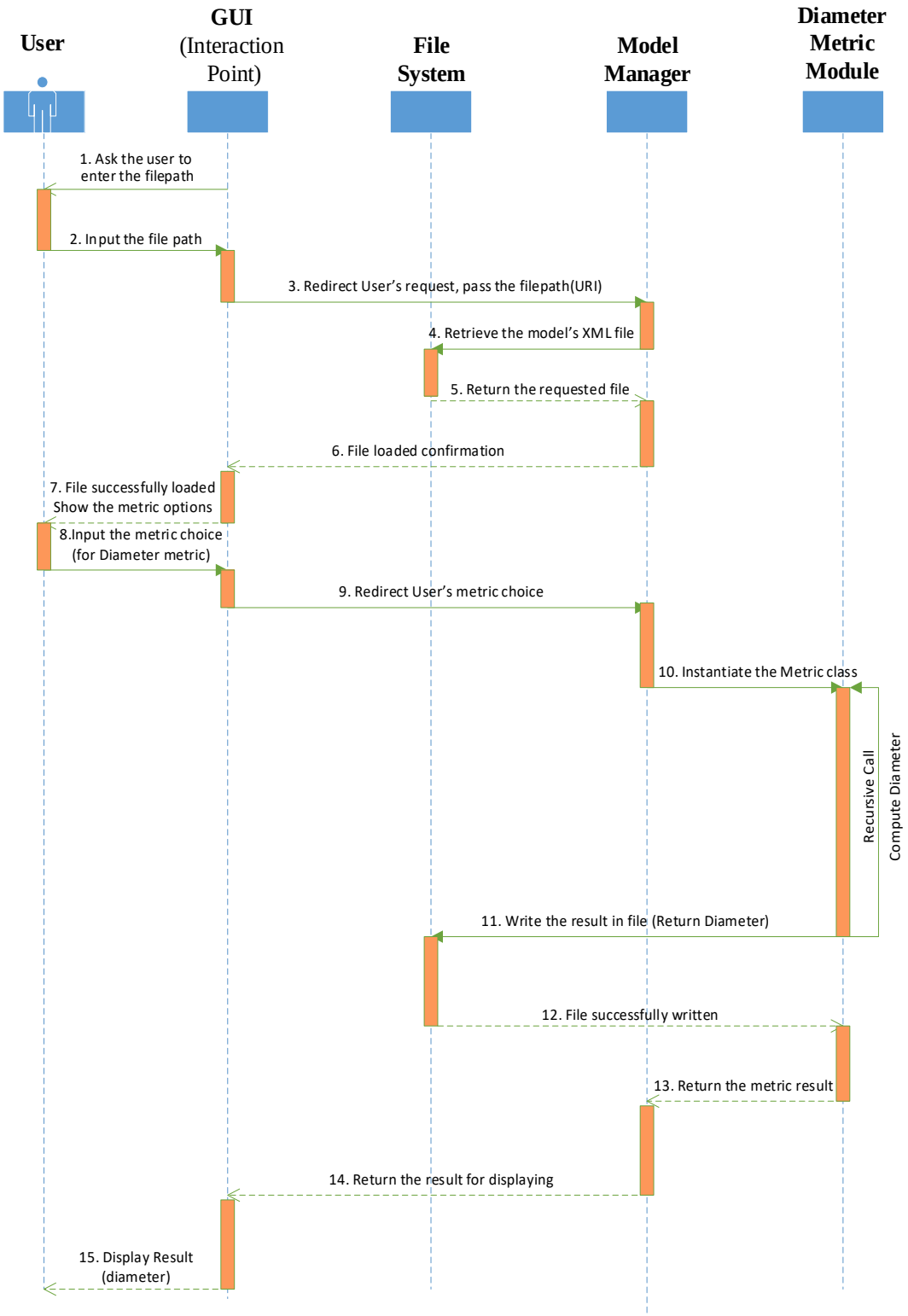

Figure 5.3: An UML sequence diagram for the complexity metric Diameter computation.

The output file is named as ResultFileBPMN.txt for the service choreography models implemented in BPMN, whereas the output file for the choreography models implemented in BPEL4Chor is called as ResultFileBPEL4Chor.txt.

For maintaining consistency both the respective result file (BPMN and BPEL4Chor) is based on the same skeleton for writing the complexity metric results in a particular format for increasing readability and comprehensibility of the output.

At the beginning only the input choreography model's file description is written which contains the file name, Uniform Resource Identifier (URI) of the file, and the target namespace in the same order. The input model's file details is important for later tracking the complexity metric results to its associated choreography model on which the metrics were applied onto.

The complexity metric results are printed in the same order as it was displayed to the User when all the metrics were shown for User's choice input, although the User might have entered the metric choice options in a random manner (the User's choices are sorted in ascending order before metric computation).

In Figure 5.4 an example screenshot of the result file for BPEL4Chor implementation is shown. The formal metric name and the metric description is written initially before writing the result for each complexity metric. The complexity metric is described in short in a couple of sentences for User's benefit to recall the metric details. The description also includes any associated formulae with the metric computation and the metric definition as well. The result of the metrics is then written (under the subheading Results). Each complexity metric's result and description is separated by a dotted line for enhancing readability.

ResultFileBPEL4Chor.txt - Notepad
File Edit Format View Help
Input File Details----> Name : OpalStandard.chor , URI : resource2/TradeDig/OpalStandard.chor , Target Namespace : http://defaultTargetNameSpace

**********COMPLEXITY METRIC RESULTS**********

Metric Name : Control Flow Complexity (CFC)
Metric Description :-
CFC (P) = Σ CFC_XOR (A), where (A∈P, A is a XOR-split) + Σ CFC_OR (A), where (A∈P, A is an OR-split) + Σ CFC_AND (A), where (A∈P, A is an AND-split) - where P is the choreography model and A is an activity
CFC_XOR-split (A) = fan−out (A), CFC_OR-split (A) = (2^fan − out (A)) - 1, CFC_AND-split (A) = 1

Results :-
CFC metric for the choreography model -----> : 2.0

Metric Name : Coefficient of Network Complexity (CNC)
Metric Description :-
CNC = (Number of arcs)/(Number of activities,joins,and splits)
The parameter number of arcs in the equation represents all the connector types (in the BPMN 2.0 specification) and the number of activities, joins, and splits parameter in the equation denotes all types of Tasks, Gateways, Events, and Sub Process.

Results :-
CNC metric for the choreography model ----> : 1.87

Metric Name : Durfee Square Metric (DSM) and Perfect Square Metric (PSM)
Metric Description :-
DSM is defined to be equal to d if there are d types of construct elements occurring at least d times (each) in the choreography model. The construct elements are defined to be any Task types, Data Item types, Gateway types, Sub Process types, and Event types.
PSM is defined on a given set of construct element types ranked in descending order on the number of their instances, the PSM is the (unique) largest number such that the top p types occur together at least p^2 times.

Results :-
DSM for the choreography model -----> : 4
The top 4 construct elements with their respective count are :-
Invoke 12
Receive 7
Data Object 6
Scope 4

PSM for the choreography model -----> : 5
The top 5 construct elements with their respective count are :-
Invoke 12
Receive 7
Data Object 6
Scope 4
Parallel Gateway (Flow) 2

Figure 5.4: A screenshot of the result file named ResultFileBPEL4Chor.txt

CHAPTER 6

Evaluation

In this chapter we are going to evaluate our developed framework for computing the complexity metrics on a collection of service choreography models, designed and modelled using both BPMN and BPEL4Chor modelling notation. We are going to show the computed metric results in tables for easier comparison between the standard choreography models and the TraDE applied choreography models, as well as analyse the difference in results between the two used modelling notations (BPMN and BPEL4Chor).

6.1 Evaluation of Choreography Models

Our collection of choreography models contains four sets of BPMN and BPEL4Chor models designed and developed for feeding into our framework system as input. Out of these four sets we have already discussed in length about the first model set (Opal Simulation) in Chapter 3 and 4. In this section we are going to introduce the other three model sets and give a brief description about them before discussing the complexity metric results produced by our developed framework.

6.1.1 Pizza Delivery Scenario

Our second choreography model set is based on the pizza ordering and delivery service scenario [BPM10]. Here a customer orders a pizza from a given menu and the pizza restaurant prepares and delivers the pizza to the provided address.

The designed choreography model based on this scenario contains four participants. The first participant Pizza Customer makes the order by selecting the pizza from the Food Menu List and accomplish the payment according to the total order and thereafter eats the food. The first participant calls the second participant named as Pizza Processor and passes the payment data, order details and the delivery address.

The Pizza Processor participant receives the order details and notifies the Kitchen (third participant) about the selected pizza to be prepared and receives the required cooking time. It then informs the Delivery Unit (fourth participant) about the cooking time and delivery

address and gets back the current status of the delivery vehicles. Once it has the pizza ready to be delivered it ships the pizza using the available or nearby car depending upon the current vehicle status.

The kitchen participant gets the name of the pizza to be prepared, returns the approximate cooking time to the Pizza Processor and prepares the pizza. The Delivery Unit participant gets the delivery address and cooking time and returns back the current vehicle status to the Pizza Processor.

The Figure 6.1 shows the choreography model of this pizza delivery scenario [BPM10] based on the BPMN 2.0 specifications. This is the standard model without the TraDE concepts applied.

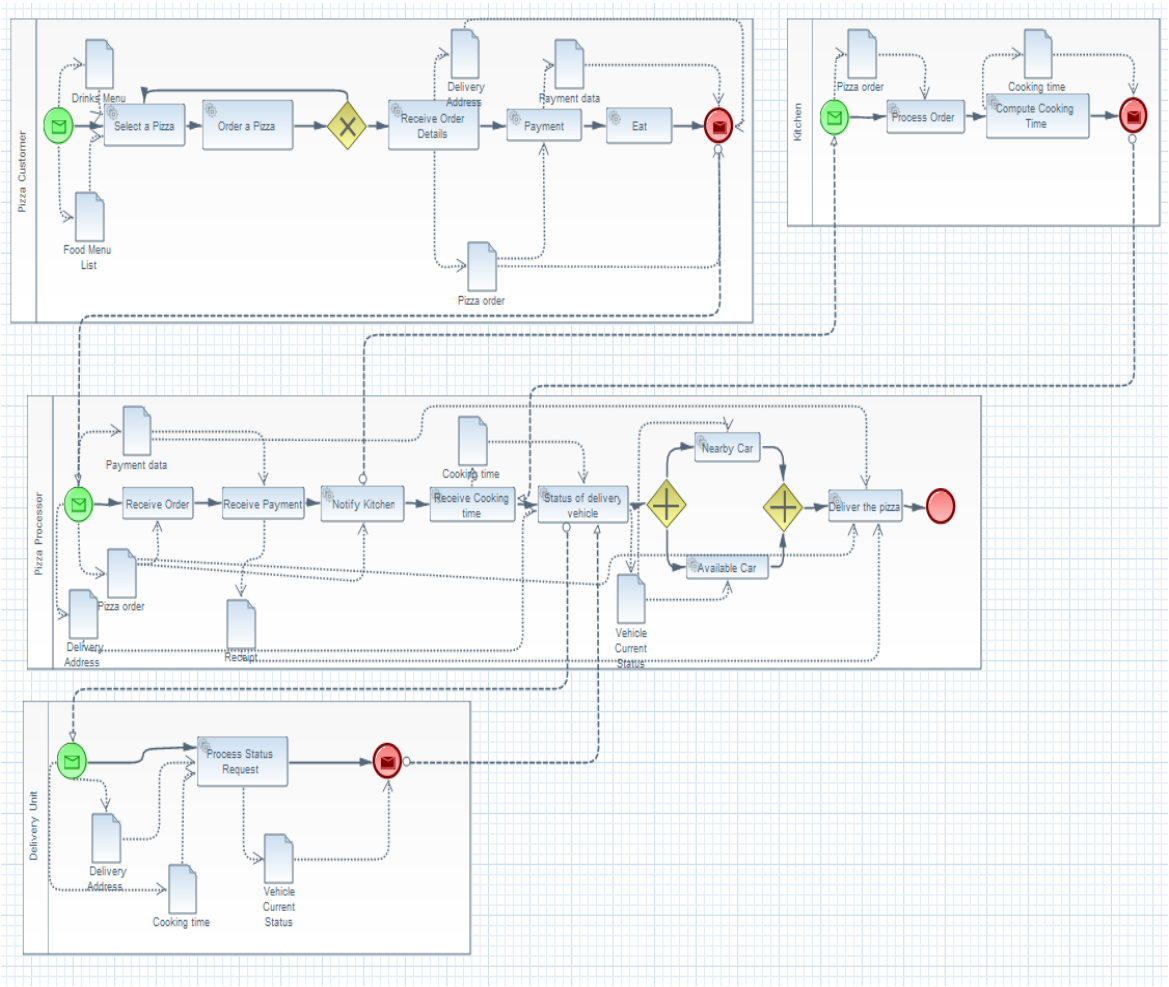

Figure 6.1: BPMN 2.0 based Choreography Model of the Pizza Delivery Scenario.

The Figure 6.2 illustrates the BPEL4Chor model version of the same pizza delivery scenario as shown in Figure 6.1. Figure 6.3 also shows the same BPEL4Chor model of the pizza delivery scenario with the TraDE concepts applied into it.

The complexity metrics are applied on both the BPMN and BPEL4Chor models of the pizza delivery scenario as discussed above. The choreography models shown in Figure 6.1, 6.2, and 6.3 are used as input to be fed into the framework for complexity computation. The Table 6.1 shows all the complexity metric results summarised into a tabular structure for easier comparison and readability.

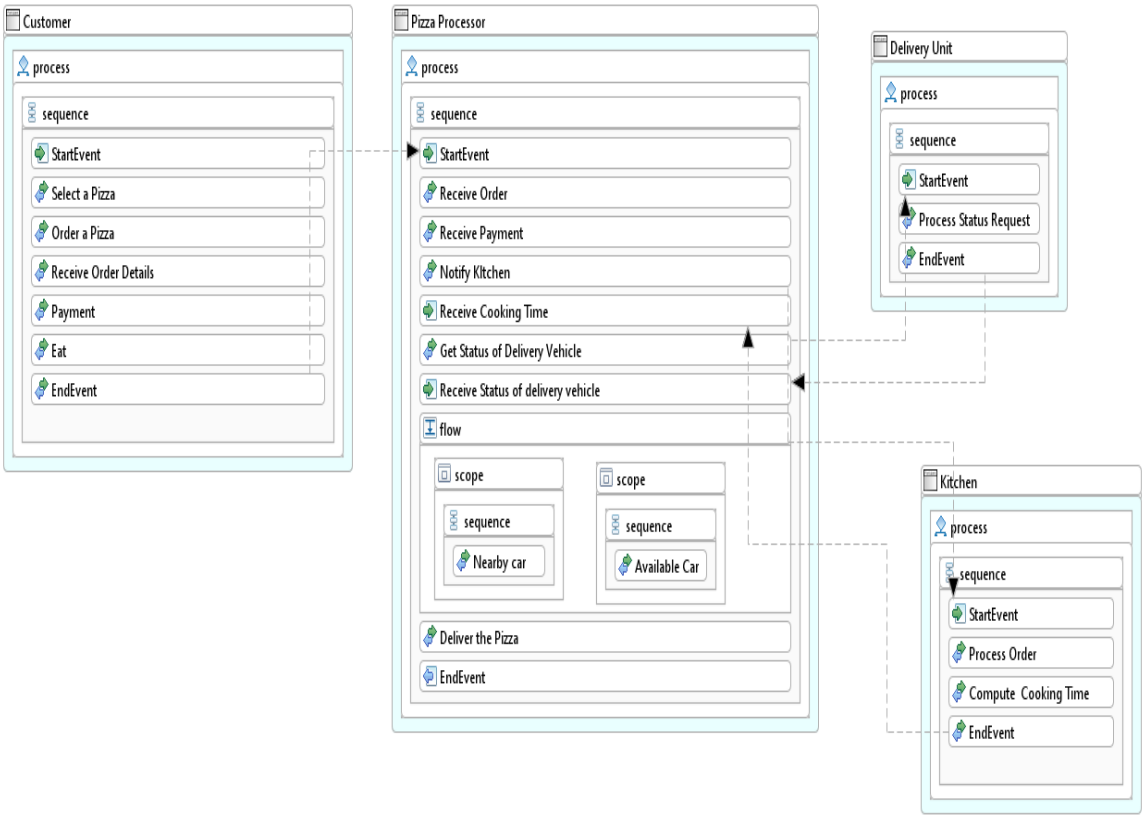

Figure 6.2: BPEL4Chor Model (Standard) of the Pizza Delivery Scenario.

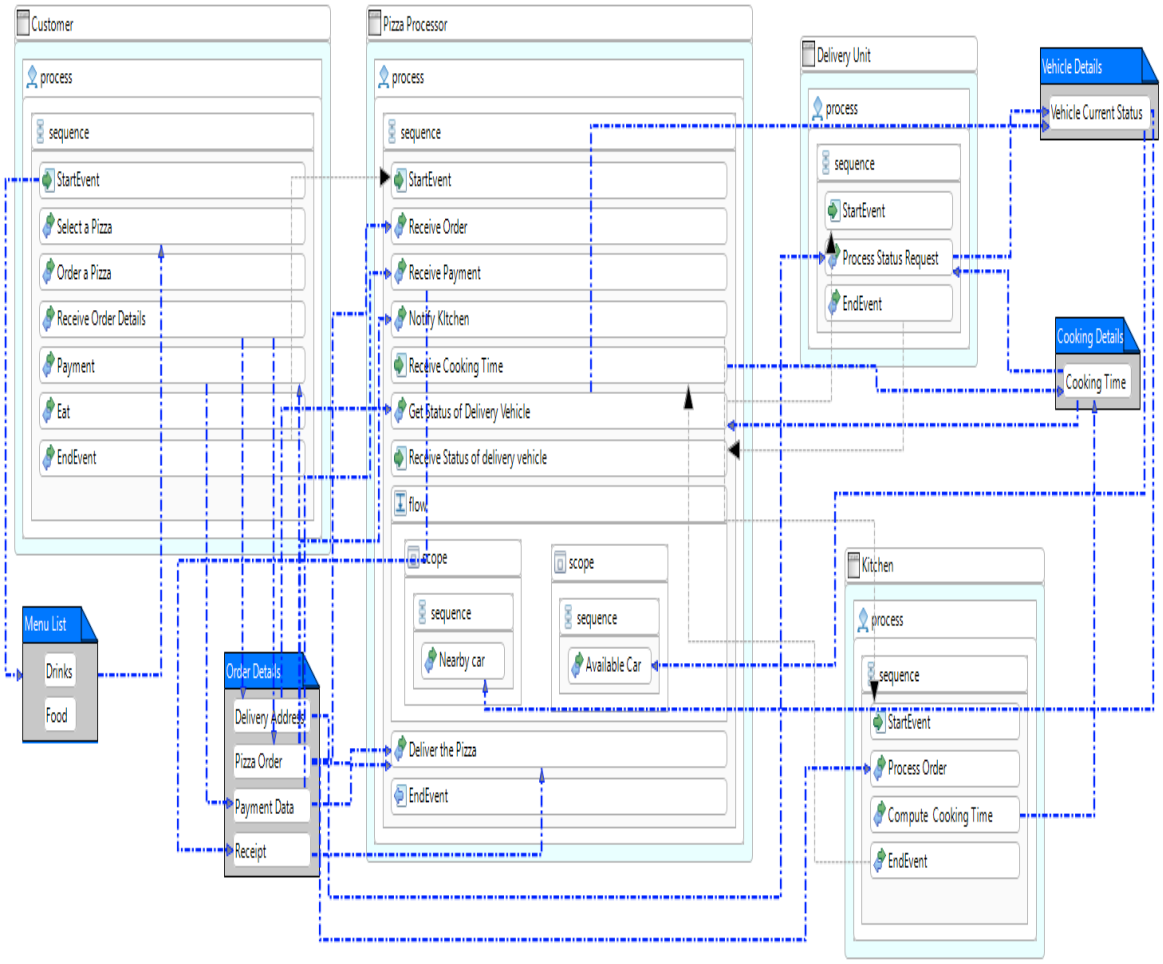

Figure 6.3: BPEL4Chor Model (TraDE applied) of the Pizza Delivery Scenario.

Table 6.1: Complexity Metric Results for Pizza Delivery.

Complexity Metrics		**BPMN model (Figure 6.1)**	**BPEL4Chor model (Figure 6.2)**	**BPEL4Chor model (TraDE) (Figure 6.3)**
Size Metrics	**NOA**	40	25	29
	NOAC	43	26	30
	NOAJS	43	26	30
	M4	0.08	0.04	0.03
MCC		26	5	25
CFC		4	1	1
IC		412	26	11264
CNC		2.48	1.12	2.04
DSM		4	2	3

PSM		6	5	5
CLA		0.53	0.86	0.86
HCC	**Length**	11.61	11.61	13.61
	Volume	105.98	74.96	112.29
	Difficulty	40	2.5	15
Structural Metrics	**CoC**	1.56	1.12	1.77
	Diameter	27	27	27

6.1.2 Taxi Booking Service

The third choreography model set is based on the online taxi booking service and trip execution scenario [WAS+13]. Here a passenger books a cab online from a list of different taxi variants and the cab arrives at the pick-up address given by the passenger in a short time.

The designed choreography model based on this scenario contains four participants. The first participant Customer is initiated by the passenger who books the cab by selecting a particular car from a list of taxi variants (economy, deluxe, or luxury) available based on the distance type (short distance travel or for inter-city transfers) provided by the passenger at first. After selecting the cab the Customer accomplish the payment according to the approximate trip cost (the credit card company blocks the approximate trip amount for security reasons) and on successful payment execution the cab arrives at the pick-up location within a short time.

The service task Receive Trip Details in the first participant Customer calls the second participant named as Taxi Order Processor and passes the trip objectives and pick-up location data. The trip objectives contains the cab type and the other booking details (passenger information like name, age etc. and distance type). The service task Payment in the first participant Customer also calls the third participant named as Payment Unit and passes the trip details data to it.

The Taxi Order Processor participant receives the trip objectives details on trigger and gets the current taxi status based on the pick-up location by calling the fourth participant named as Taxis. On receiving the taxi status it then sends the nearest (to the pick-up location) or the available cab to the passenger's given location.

The Taxis participant gets the trip objectives data and processes the request to send back the current taxi status data to the calling participant. The Payment Unit participant gets the trip data from the calling participant and executes the payment procedure based on the mode of payment opted by the passenger and finally generates the payment receipt (bill or invoice) to return it back to the calling participant.

The Figure 6.4 shows the choreography model of this taxi booking service [WAS+13] based on the BPMN 2.0 specifications. This is the standard model without the TraDE concepts applied.

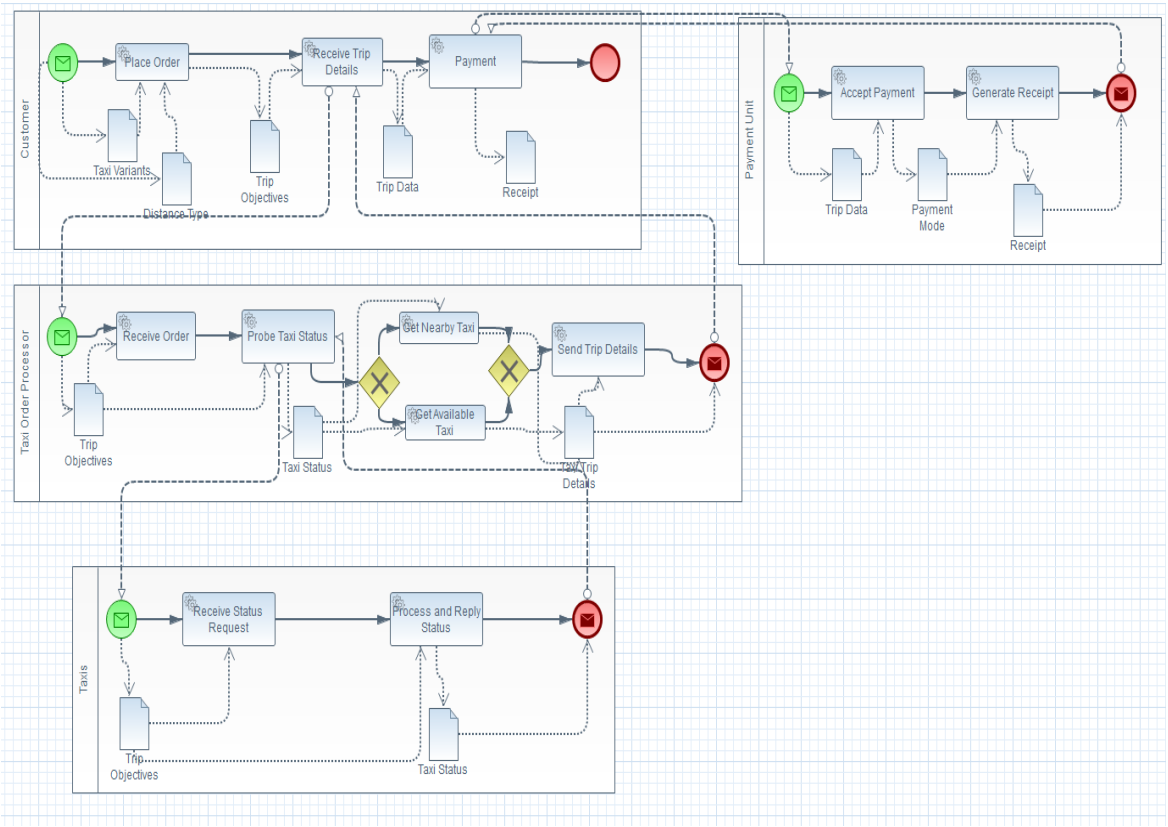

Figure 6.4: BPMN 2.0 based Choreography Model of the Taxi Booking Service.

The Figure 6.5 shows the BPEL4Chor model version of the same taxi booking service scenario as shown in Figure 6.4. Figure 6.6 also shows the same BPEL4Chor model of the taxi booking service scenario with the TraDE concepts applied into it.

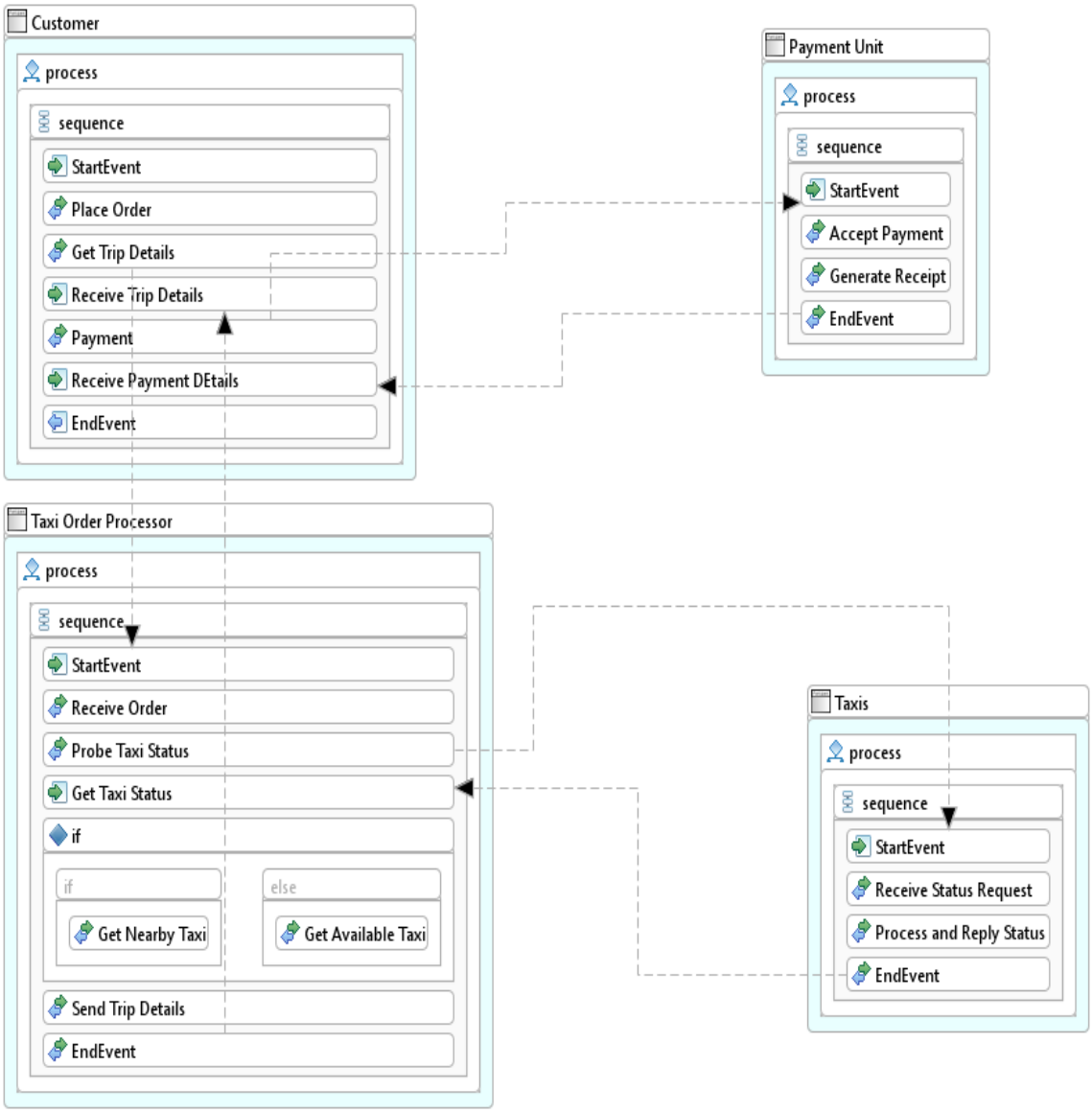

Figure 6.5: BPEL4Chor Model (Standard) of the Taxi Booking Service.

The complexity metrics are applied on both the BPMN and BPEL4Chor models of the taxi booking service scenario as discussed above. The choreography models shown in Figure 6.4, 6.5, and 6.6 are used as input to be fed into the framework for complexity computation. The Table 6.2 shows all the complexity metric results summarised into a tabular structure for easier comprehensibility.

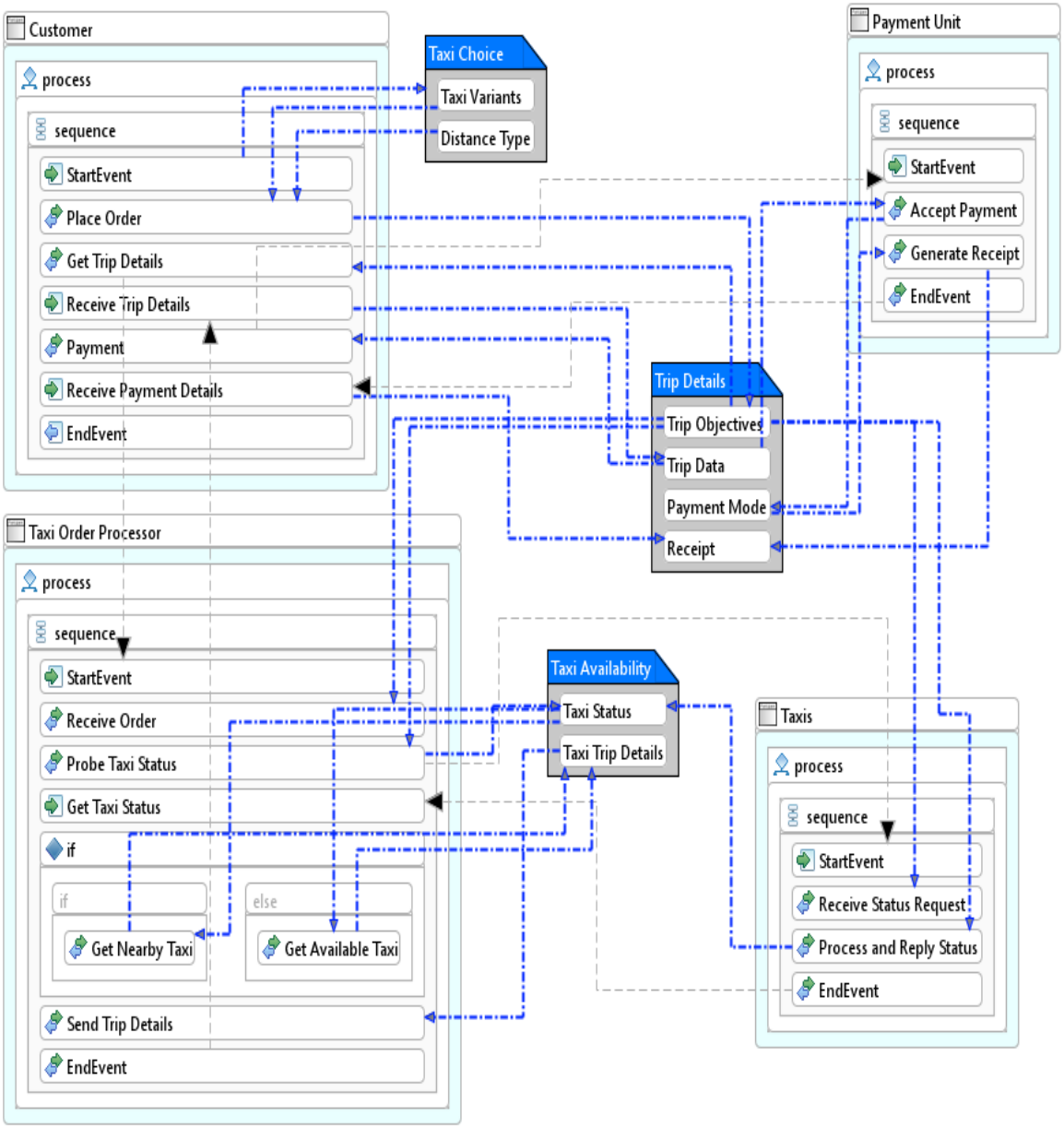

Figure 6.6: BPEL4Chor Model (TraDE applied) of the Taxi Booking Service.

Table 6.2: Complexity Metric Results for Taxi Booking Service.

Complexity Metrics		BPMN model (Figure 6.1)	BPEL4Chor model (Figure 6.2)	BPEL4Chor model (TraDE) (Figure 6.3)
Size Metrics	NOA	33	23	26
	NOAC	35	24	27
	NOAJS	35	24	27
	M4	0.06	0.04	0.04
MCC		22	2	22
CFC		3	2	2
IC		37	24	3897
CNC		2.50	1	1.96
DSM		4	2	3
PSM		5	4	5
CLA		0.48	0.96	0.96
HCC	Length	8	8	10
	Volume	81.27	58.05	90.47
	Difficulty	26	2	11
Structural Metrics	CoC	1.57	1	1.74
	Diameter	23	25	25

6.1.3 Holiday Travel Scenario

The fourth choreography model set is based on planning a trip, generating the flight booking and trip itinerary details scenario [BPM10]. The traveller submits the desired trip details (destination and dates) to the tour service agent for booking the flights and organizing the complete itinerary for the trip.

The designed choreography model based on this scenario contains four participants. The first participant Passenger is initiated by the traveller (user) who comes up with the trip destination and the desired travel dates from a brainstorming session. The service task Request Itinerary in Passenger calls the second participant, named as Tour Services, with the desired trip details data to arrange the whole trip (flight booking and itinerary generation). The Passenger then receives the confirmed flight booking and the complete itinerary for the trip from the Tour Services.

The Tour Services participant receives the trip details data (trip destination and trip dates) on trigger and then requests for the flight fare quotation list by calling the third participant named as Flight Desk (passes the trip details). On receiving the flight fare quotation list it selects the airline for flight booking based on the cheapest fare (best deal) from the fare list. The service task Buy Ticket in Tour Services calls the fourth participant Airline Company

by passing the passenger and trip details for booking the flight ticket. On receiving the flight ticket the Tour Services participant generates the itinerary and returns back both the flight ticket and itinerary to the calling participant.

The Flight Desk participant receives the trip dates and destination data on trigger and processes the request to retrieve the flight fares for all the airlines operating in the given sector (between destination and departure city). Based on the retrieved fare list the Flight Desk generates and returns back the flight fare quotation list to the calling participant.

The Airline Company participant gets the trip destination and dates along with the passenger details data on trigger from the calling participant and books the requested flight ticket. The Airline Company then generates and returns back the confirmed reservation of the flight booking to the calling participant.

The Figure 6.7 shows the choreography model of this holiday travel scenario [BPM10] based on the BPMN 2.0 specifications. This is the standard model without the TraDE concepts applied.

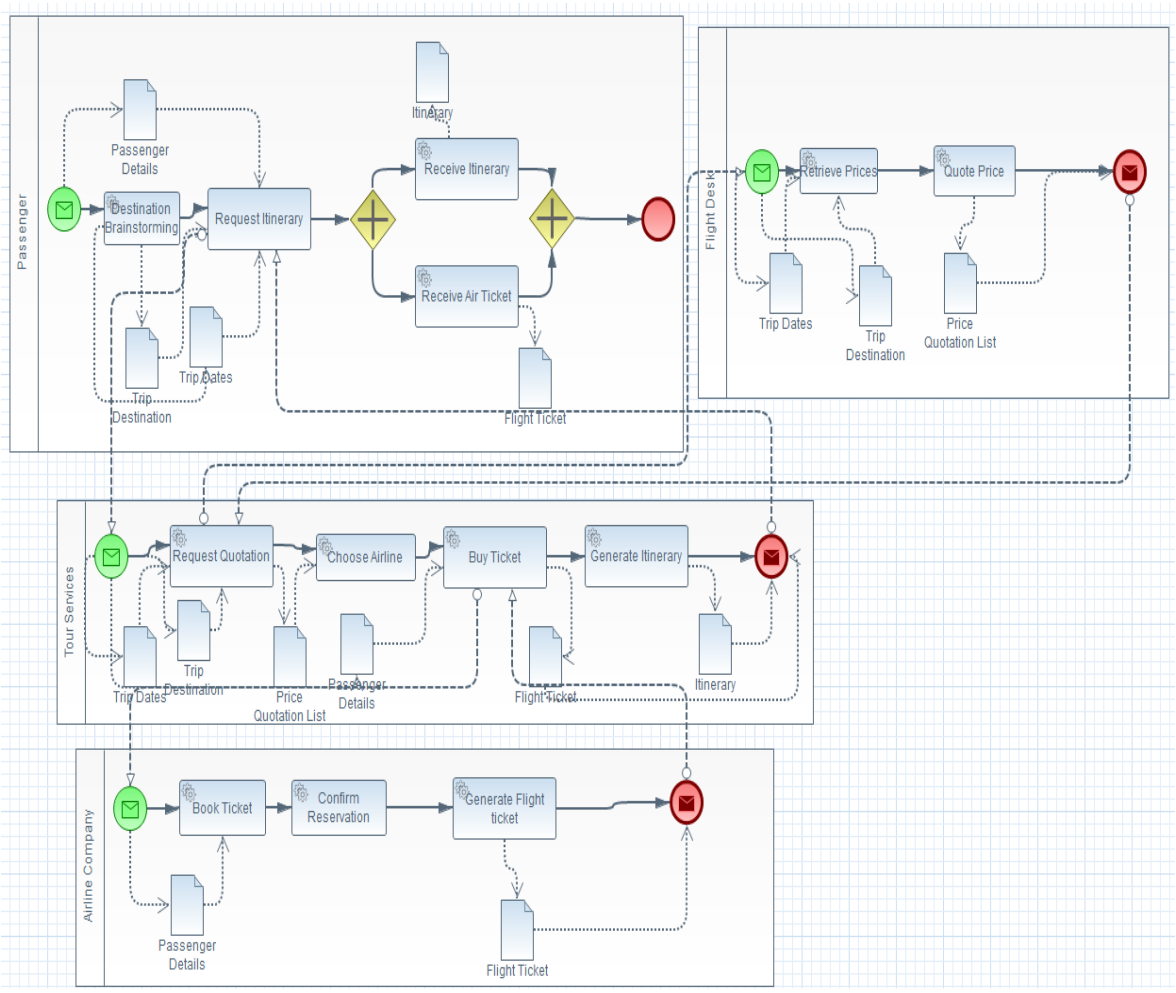

Figure 6.7: BPMN 2.0 based Choreography Model of the Holiday Travel Scenario.

The Figure 6.8 shows the BPEL4Chor model version of the same holiday travel scenario as shown in Figure 6.7. Figure 6.9 also shows the same BPEL4Chor model of the holiday travel scenario with the TraDE concepts applied into it.

The complexity metrics are applied on both the BPMN and BPEL4Chor models of the holiday travel scenario as discussed above. The choreography models shown in Figure 6.7, 6.8, and 6.9 are used as input to be fed into the framework for complexity computation. The Table 6.3 shows all the complexity metric results summarised into a tabular structure for easier comprehensibility.

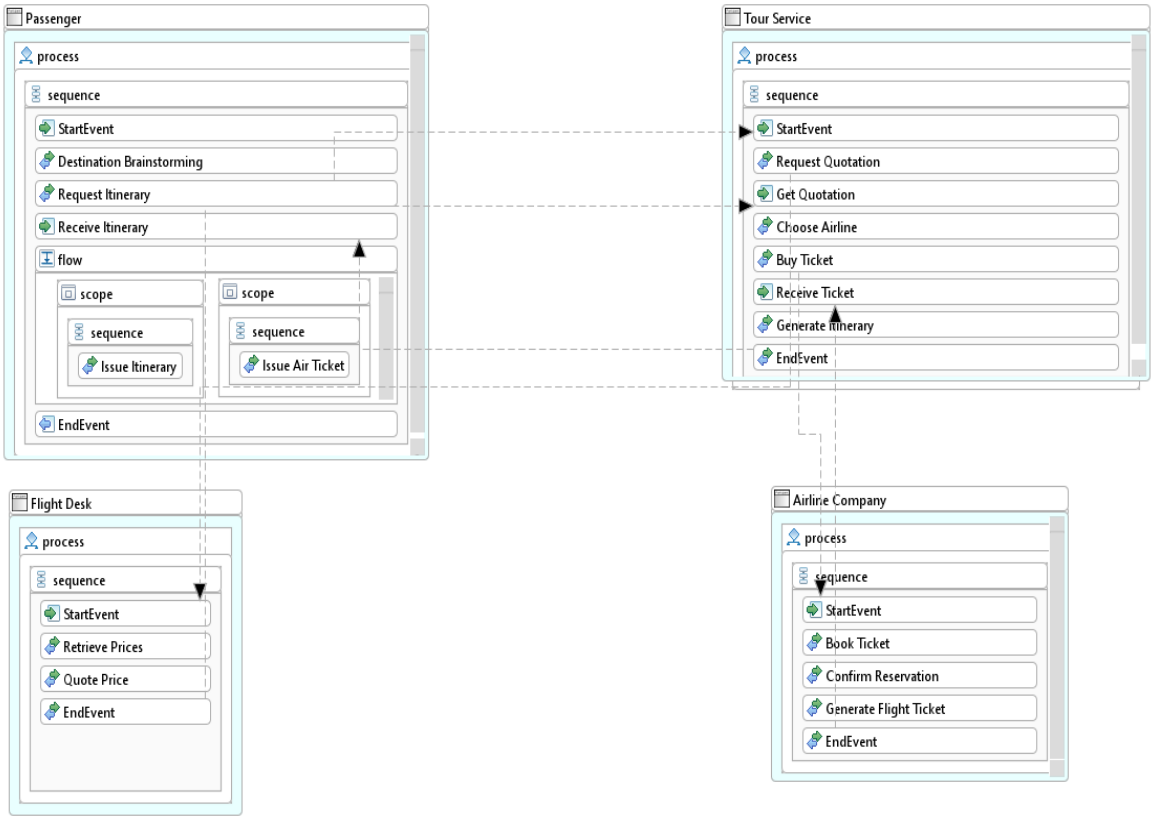

Figure 6.8: BPEL4Chor Model (Standard) of the Holiday Travel Scenario.

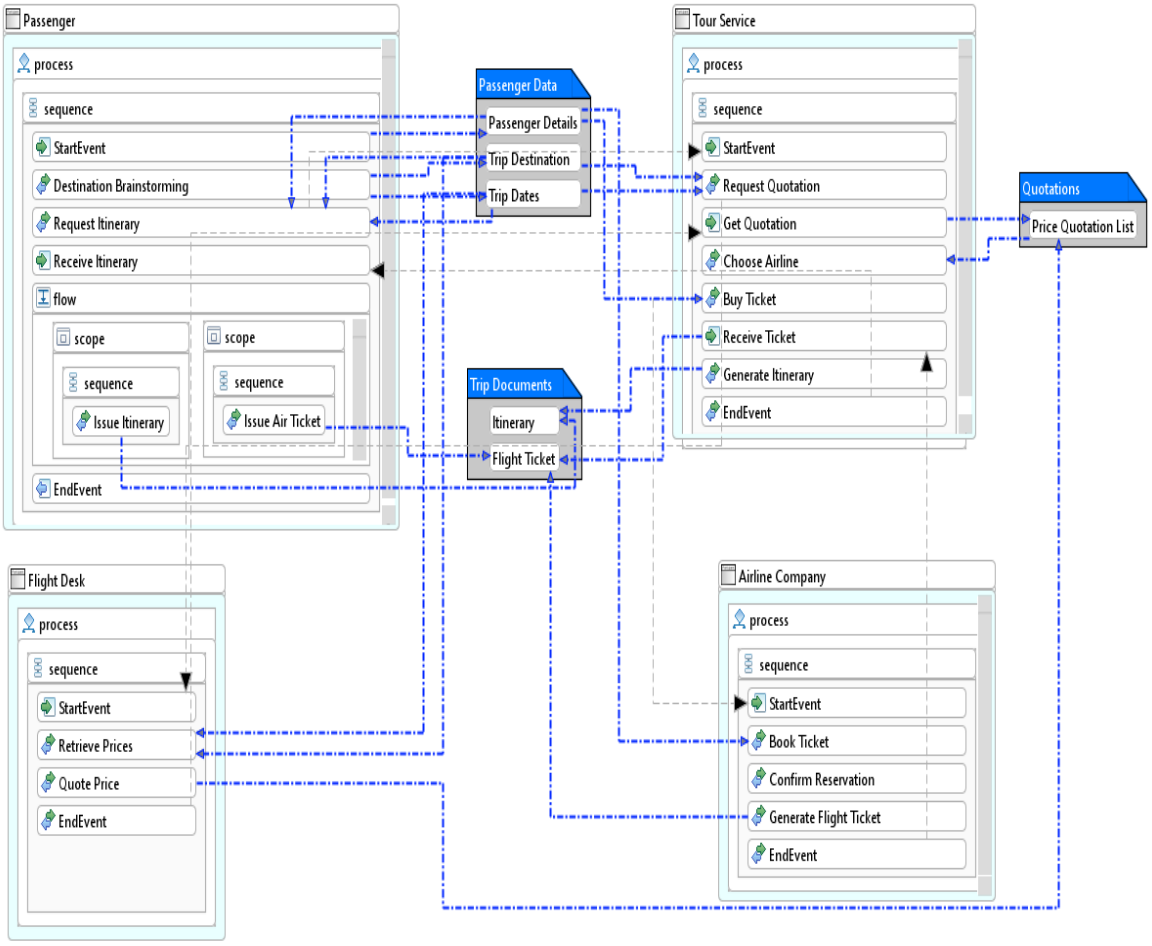

Figure 6.9: BPEL4Chor Model (TraDE applied) of the Holiday Travel Scenario.

Table 6.3: Complexity Metric Results for Holiday Travel Scenario.

Complexity Metrics		**BPMN model (Figure 6.1)**	**BPEL4Chor model (Figure 6.2)**	**BPEL4Chor model (TraDE) (Figure 6.3)**
Size Metrics	**NOA**	37	24	27
	NOAC	39	25	28
	NOAJS	39	25	28
	M4	0.05	0.04	0.04
MCC		19	6	23
CFC		2	1	1
IC		245	25	2973

CNC		2.43	1.16	1.96
DSM		4	2	3
PSM		5	5	5
CLA		0.5	0.83	0.83
HCC	**Length**	8	11.61	13.61
	Volume	88.23	72.38	101.06
	Difficulty	32	2.5	11.25
Structural Metrics	**CoC**	1.44	1.16	1.75
	Diameter	24	27	27

Chapter 7
Conclusion and Future Work

Service choreographies can be designed using many modelling notations, but in this thesis we have implemented the choreography models in BPMN 2.0 and BPEL4Chor which are most commonly used. Although the service choreographies are meant to diminish the complexity of the process structure through the use of choreographed messaging communication and separating the data flow by incorporating the TraDE methodology, the choreography model still remains complex to understand and comprehend.

We adapted some complexity metrics from software engineering and business process modelling domain and modified them to be able to apply onto the choreography models. The complexity metrics devised for the service choreography models in this thesis helps a great deal in understanding the structure and complexity of the models instantly without having to manually compute and perceive their complex structure. Some of the complexity metrics like Diameter, Halstead-based Choreography Complexity (HCC) Length, Volume and Difficulty, Interface Complexity (IC), and McCabe's Cyclomatic Complexity (MCC) are the most beneficial ones in terms of comprehending a choreography model's true complexity based on its various construct nodes and connectors. From the aforementioned metrics we have an idea about the model's volume, difficulty in understanding, and the longest execution path possible in the model.

The incorporation of TraDE concept in the service choreography model reduces the data associations or connectors by a large margin and helps to centralize the data access and sharing by making transparent data flow and separating the data transitions from the control flow. The TraDE concepts thereby reduces the overall complexity of the choreography models as discussed in Chapter 4.

In this thesis we also developed the prototypical implementation of applying the complexity metrics on the choreography models which are given as input to the framework. Our framework can handle the input choreography models in both the BPMN and BPEL4Chor modelling notations. It also takes into account and processes the TraDE constructs in the choreography models like the cross-partner data flow and the cross-partner data objects for computing the complexity metrics. In the following paragraph we discuss about the possible future work and scope of improvement for our framework.

The complexity metrics can be made to account for other aspects of choreography models like user perception and ease of understandability of a model. They can be more fine-tuned for understanding some particular characteristics of a choreography model like a data objects association complexity with each participant or its access rate etc. The framework can be extended in the future for handling the BPMN based choreography models incorporated with the TraDE concepts. This can be accomplished by extending the meta-model of the Eclipse Modelling Framework (EMF), which will be able to handle the cross-partner data objects and the cross-partner data flow in the graphical modelling tool while designing a BPMN based choreography model. Another future work possibility is to increase the complexity metric options in the framework by devising new metrics suited for choreography models.

Bibliography

[ARG+06] E. R. Aguilar, F. Ruiz, F. García, and M. Piattini. “Applying Software Metrics to Evaluate Business Process Models.” *CLEI Electronic Journal*, vol. 9, no. 1, June 2006.

[BPM10] “BPMN 2.0 by Example.” Version Alpha 8 (non-normative). OMG Document Number: dtc/2010-06-xx. 2010.

[BS03] P. Binkele, and S. Schmauder. “An atomistic Monte Carlo simulation of precipitation in a binary system” 2003.

[CMN+06] J. Cardoso, J. Mendling, G. Neumann, and H. A. Reijers. “A Discourse on Complexity of Process Models.” 2006.

[DKB08] G. Decker, O. Kopp, and A. Barros. “An Introduction to Service Choreographies.” 2008.

[DKL+07] G. Decker, O. kopp, F. Leymann, and M. Weske. “BPEL4Chor: Extending BPEL for Modelling Choreographies.” Hasso-Plattner-Institute, University of Potsdam, Germany and Institute of Architecture of Application Systems, University of Stuttgart, Germany. 2007.

[DKL+08] G. Decker, O. Kopp, F. Leymann, K. Pfitzner, and M. Weske. “Modelling Service Choreographies using BPMN and BPEL4Chor.” Hasso-Plattner-Institute, University of Potsdam, Germany and Institute of Architecture of Application Systems, University of Stuttgart, Germany. 2008.

[EF18a] Eclipse Foundation, Inc. Eclipse Modelling Framework (EMF). https://www.eclipse.org/modeling/emf/. 2018.

[EF18b] Eclipse Foundation, Inc. Eclipse Business Process Model and Notation (BPMN2). https://projects.eclipse.org/projects/modeling.mdt.bpmn2. 2018.

[GL06] V. Gruhn and R. Laue. “Complexity Metrics for Business Process Models.” 2006.

[Hal87] M. H. Halstead. “Elements of Software Science.” Elsevier, Amsterdam. 1987.

[HBK+17] M. Hahn, U. Breitenbucher, O. Kopp, and F. Leymann. “Modelling and Execution of Data-aware Choreographies: An Overview.” *In: Computer Science – Research and Development.* 2017.

[HBL+17] M. Hahn, U. Breitenbucher, F. Leymann, and A. Weiss. “TraDE – A Transparent Data Exchange Middleware for Service Choreographies.” *In: On the Move to Meaningful Internet Systems.* 2017.

[HK81] S. Henry and D. Kafura. “Software structure metrics based on information-flow.” *IEEE Transactions On Software Engineering*, 7(5):510–518. 1981.

[Hoh02] G. Hohpe. “Stairway to Heaven.” Software Centre, The University of Gothenburg; Gothenburg, Germany. 2002.

[Itk18] IT Knowledge Portal. Business Process Management. http://www.itinfo.am/eng/business-process-management/. 2018.

[Jur06] M. B. Juric. “A Hands-on Introduction to BPEL.” 2006.

[Kan02] S. H. Kan. “Metrics and Models in Software Quality Engineering.” Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. 2002.

[KBR+05] N. Kavantzas, D. Burdett, G. Ritzinger, and Y. Lafon. “Web Services Choreography Description Language Version 1.0.” W3C Candidate Recommendation. 2005.

[Kir17] M. Kirchmer. “High Performance Through Business Process Management: Strategy Execution in a Digital World.” 2017.

[KN14] K. Kluza, and G. J. Nalepa. “Proposal of Square Metrics for Measuring Business Process Model Complexity.” 2014.

[Lat01] A. M. Latva-Koivisto. “Finding a complexity for business process models.” Helsinki University of Technology, Tech. Rep., Feb 2001.

[Mcc76] T. J. McCabe. “A complexity measure.” *In: IEEE Transactions on Software Engineering*, 2(4):308–320. 1976.

[MMN+06] J. Mendling, M. Moser, G. Neumann, H.M.W. Verbeek, B.F. van Dongen, and W.M.P. van der Aalst. “A Quantitative Analysis of Faulty EPCs in the SAP Reference Model.” 2006.

[OAS18] OASIS, Advancing open standards for the information society. https://www.oasis-open.org/. 2018.

[Obj18] Object Management Group. Business Process Model and Notation. http://www.bpmn.org/. 2018.

[OR03] M. Owen and J. Raj. “BPMN and Business Process Management: Introduction to the New Business Process Modelling Standard.” Popkin Software. 2003.

[Pnm18] PNMSoft – A Genpact Company. Business Process. http://www.pnmsoft.com/resources/bpm-tutorial/business-process/. 2018.

[RSS14] M. von Rosing, A. Scheer, and H. von Scheel. “Phase 1: Process Concept Evolution.” The Complete Business Process Handbook: Body of Knowledge from Process Modelling to BPM, Volume1. 2014.

[SGM+10] L. Sánchez-González, F. García, J. Mendling, F. Ruiz, and M. Piattini. “Prediction of business process model quality based on structural metrics.” *In: Proceedings of the 29th international conference on Conceptual modeling, Vancouver, Canada*. 2010.

[SR15] K. D. Swenson, and M. von. Rosing. “What Is Business Process Management?” 2015.

[WAS+13] A. Weiss, V. Andrikopoulos, S. G. Saez, D. Karastoyanova, and K. Vukojevic-Haupt. “Modelling Choreographies using the BPEL4Chor Designer: an Evaluation Based on Case Studies.” 2013.

[Wey16] M. Weyrich. Software Engineering for Real-Time Systems. 2016.

[Whi06] S. A. White. “Introduction to BPMN.” IBM Software Group. 2006.

[ZBD+06] J. M. Zaha, A. Barros, M. Dumas, and A. Hofstede. “Let’s Dance: A Language for Service Behaviour Modelling.” 2006.

All links were last followed on June 4, 2018.

List of Figures

List of Tables

List of Abbreviations

API	Application Programming Interface
BAM	Business Activity Monitoring
BPD	Business Process Diagram
BPEL	Business Process Execution Language
BPEL4Chor	Business Process Execution Language for Choreography
BPEL4WS	Business Process Execution Language for Web Services
BPM	Business Process Management
BPMI	Business Process Management Initiative
BPML	Business Process Modelling Language
BPMN	Business Process Model and Notation
BPMS	Business Process Management System
CDM	Choreography Data Model
CFC	Control flow Complexity
CLA	Connectivity Level between Activities
CNC	Coefficient of Network Complexity
CoC	Coefficient of Connectivity
DO	Data Objects
DSM	Durfee Square Metric
EMF	Eclipse Modelling Framework
GUI	Graphical User Interface
HCC	Halstead-based Choreography Complexity
HPC	Halstead-based Process Complexity
HR	Human Resource
IC	Interface Complexity
IoT	Internet of Things

KMC	Kinetic Monte Carlo
LOC	Lines of Code
LOC	Lines of Code
MCC	McCabe's Cyclomatic Complexity
NCS	Number of Collapsed Sub-Process
NCSA	Number of Collapsed Sub-Process Ad-Hoc
NCSC	Number of Collapsed Sub-Process Compensation
NCSL	Number of Collapsed Sub-Process Looping
NCSMI	Number of Collapsed Sub-Process Multiple Instance
NOA	Number of Activities
NOAC	Number of Activities and Control-flow
NOAJS	Number of Activities, Joins and Splits
NSFA	Number of Sequence Flows between Activities
NT	Number of Tasks
NTC	Number of Task Compensation
NTL	Number of Task Looping
NTMI	Number of Task Multiple Instances
OMG	Object Management Group
OPAL	Ostwald ripening of Precipitates on an Atomic Lattice
PBD	Participant Behaviour Description
PC	Procedure Complexity
PSM	Perfect Square Metric
REST	Representational State Transfer
SOA	Service-oriented Architecture
TNA	Total Number of Activities
TNCS	Total number of Collapsed Sub-Process
TNT	Total Number of Task
TraDE	Transparent Data Exchange
UI	User Interface
UML	Unified Modelling Language
URI	Uniform Resource Identifier

URL	Uniform Resource Locator
WS-CDL	Web Services Choreography Description Language
WSDL	Web Services Description Language
XML	Extensible Markup Language

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

Place, Date, Signature