AUTOMATIC CONFIGURATION OF QUEUEING NETWORK MODELS FROM BUSINESS PROCESS DESCRIPTIONS

By
UMA MAHESHWAR CHALAVADI
Bachelor of Engineering (Hons)
Birla Institute of Technology & Science
Pilani, India
2002

Submitted to Graduate College of the
Oklahoma State University
in partial fulfillment
of the requirements for
the Degree of

MASTER OF SCIENCE
December, 2004

AUTOMATIC CONFIGURATION OF QUEUEING NETWORK MODELS FROM BUSINESS PROCESS DESCRIPTIONS

Thesis Approved:

Dr. Manjunath Kamath

Thesis Adviser

Dr. William J. Kolarik

Dr. Nikunj P. Dalal

Dr. A. Gordon Emslie

Dean of the Graduate College

ACKNOWLEDGEMENTS

My sincere thanks are due to my advisor Dr. Manjunath Kamath, without his motivation, encouragement and support this thesis would not have been possible. I am greatly indebted for his guidance both on personal and professional fronts. He was a huge inspiration for me. I really consider myself lucky to have gotten a chance to be associated so closely with a wonderful person like Dr. Kamath. Sir, thanks for being there for me always.

I would also like to thank my committee members Drs. Nikunj P. Dalal and William J. Kolarik, for serving on my committee. Their guidance and suggestions were invaluable to my work.

It was a pleasure to work with all my colleagues at the Center for Computer Integrated Manufacturing Enterprises (CCiMe). I extend my appreciation to all my colleagues for patiently listening to all the related presentations and for offering their critique. My special thanks are due to Karthik Ayodhiramanujan for his help and support. I especially thank the DIME team for reviewing my work and for their suggestions and constructive criticism. Special thanks are due to Chetan Yadati and Parthiban Dhananjeyan. I also would want to thank my friends Partha Ayalasomayajula, Dheeraj Babu Gulluru and Vamsi Maramreddy for helping me with the document.

I dedicate this work to my parents for their love, support and encouragement through all my endeavors.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure **Page**

ABBREVIATIONS

BPML	Business Process Modeling Language
BPMI	Business Process Modeling Initiative
CIM	Computer Integrated Manufacturing
DIME	Distributed Integrated Process Modeling of Next Generation Enterprises
DISS	DIME Intelligent Support System
DTD	Document Type Definition
DXL	Dynamics eXchange Language
eBXML	Electronic Business using eXtensible Markup Language
IPM	Integrated Process Management
IPM-PDL	IPM-Process Definition Language
MXL	Multimodeling eXchange Language
OASIS	Organization for the Advancement of Structured Information Standards
PNML	Petri Net Markup Language
QNML	Queueing Network Markup Language
RAQS	Rapid Analysis of Queueing Systems
UEML	Unified Enterprise Modeling Language
XRL	eXchangebale Routing Language
XML	eXtensible Markup Language
XSLT	eXtensible Style Sheet Language Transformation
XSD	eXtensible Schema Definition

Chapter 1

Introduction

A business process is an ordered sequence of tasks/activities involving people, materials, energy, equipment, or information, designed to achieve a specific business outcome. A business process defines what a business does and also determines how well the business does what it does. Processes are critical components of almost all types of systems supporting enterprise-level and business-critical activities. As identified by [8], business processes are increasingly recognized as important corporate assets that need to be managed throughout their life cycle. Especially, interests in next generation enterprise structures such as e-businesses, virtual enterprises, B2B, B2C & B2G electronic commerce companies, and globally dispersed supply-chains are driving current research in this area.

Business process modeling relates to the representation and specification of an enterprise's operations. Information on cost drivers and process performance measures, including time, quality and efficiency are critical for a holistic view of business processes in an enterprise [6]. Analysis of these processes is important to identify improvement opportunities. Existing process modeling techniques are descriptive and lack the much needed prescriptive capabilities [6]. Also, they do not provide business modelers or system architects with a formal theoretical base from which business processes can be analyzed in a rigorous, quantitative manner. As identified in [17], the Web promises a different way of encoding model components, finding information, and a novel way of performance modeling using client-server mechanisms. It is also suggested that XML and related Web technologies can be used for model specification and performance analysis using, for example, queueing or computer simulation. It is now possible to run a queueing model inside a Web browser. Using process modeling techniques in conjunction with performance modeling techniques such as queueing and simulation, enterprise issues related to cost and time can be addressed in an integrated manner for a distributed environment.

The emerging next-generation enterprise systems can be effective and scalable only if their construction is guided by a strong theory-driven framework [10, 11, 12] that takes an approach to link description with formal qualitative or quantitative analyses in an integrated manner. This thesis significantly extends the analysis capability of such an enterprise modeling framework [11] to include queueing models. To this end, a generic model transformation scheme to support queueing analysis is developed. The transformation scheme uses the process control flow and task resource requirements to create a view where process instances flow through a network of resources. Two alternative approaches were explored to automatically configure a queueing network model from a business process description. The first approach generates a queueing network model from a business process markup language description. The second approach generates a queueing network model from a formal Petri-net based business process representation, which is described using the Petri Net Markup Language (PNML). Currently, there does not exist a portable and open representation to capture the specifications of a queueing network model. Hence, this research has also led to the development of an XML-based markup language called the Queueing Network Markup Language (QNML) to store the specifications of a queueing network model.

Distributed Integrated Process Modeling and Analysis of Next Generation Enterprises (DIME) framework [10, 11, 12], developed at the Center for Computer Integrated Manufacturing Enterprises (CCiMe) at Oklahoma State University, is a framework that supports the design, analysis, automation and management of business processes. This thesis was motivated by the need to extend the analysis capability of DIME to include queueing models by making use of the existing graphical modeling language, namely, the DIME Descriptive Language (DDL) [5, 10, 11, 12], and the Petri net theory base. Petri nets were chosen to provide the theoretical foundation for the DIME framework. Current analysis capability within the DIME framework is limited to qualitative analyses using the Petri net representation. One of the key goals of the DIME effort is to support performance analysis using multiple tools (e.g. queueing and simulation). Since the goals of this thesis were motivated by the DIME effort, this thesis has addressed its objectives by developing two alternative approaches to achieve queueing analysis capability within the DIME framework. It is important to note the reason queueing analysis was chosen instead of simulation. Simulation models employ process-centric views. There is a reasonable one-to-one correspondence between the elements of a process model and the constructs of a simulation model developed using commercial simulation software such as Arena [15]. Whereas, a queueing model employs a resource-centric view of the process model, and provides a different perspective. This new perspective provides the user/modeler with a better insight to the process in consideration. The approaches developed here result in adding queueing analysis capability within the DIME framework. However, the approaches are general in nature and can be applied to any general purpose process modeling framework. In summary, the creative effort needed to develop a mapping scheme between the process-centric and resource-centric views and the potential to generalize the approach were the key reasons behind the focus on queueing analysis

Within the DIME framework, a process model is described using the DIME Descriptive Modeling Language constructs. The process model is stored in a computer processable format using the DIME markup language for later retrieval, and sharing. This transformation is achieved using a XSL Transformation (already developed). From the available DML format, the process model can now be transformed into a Petri net model and a queueing network model for further analysis as a result of this thesis effort. The first version of a mapping scheme for transformation from DML to Petri nets (or equivalent PNML) has been developed as part of the DIME research. The mapping schemes from DML to QNML, the newly developed Queueing Network Markup Language and from PNML to QNML are the ones that were developed as part of this thesis. The development of the mapping schemes involved the study of the linkages between the process modeling language constructs, DML, PNML and QNML. It was essential to work thorough a variety of examples to understand and formalize the linkages. To give a brief overview of the various representations studied as part of the current effort, a process description from [15] is used in the following example. The activities in this example have dedicated resources. Hence, the transformation into a queueing network model is straightforward.

Example 1.1

The following process description is used as an example to show the possible formal representations envisioned by the DIME framework. *An office that dispenses automotive license plates has divided its customers into three categories based on the location. There is one clerk assigned to each of the three areas who processes application forms and collects payments. When a customer arrives at the office, a computer generates a token based on the location information entered by the customer. The token has the information as to which clerk processes the application and collects payment. Based on this, the customer goes to the corresponding clerk. After completion of this step, all customers are sent to a head clerk who checks the forms and issues the plates. The interarrival time of customers is exponential with a rate of 0.25 customers per minute. Processing times for clerks are Uniform(8,10) minutes. Service time of the head clerk is Uniform(2.66,3.33) minutes.*

Figure 1.1 represents the process description using the DIME Descriptive Modeling Language constructs. The corresponding Petri net representation is shown in Figure 1.2. Figure 1.3 presents the equivalent queueing network representation. This effort focused on the generation of queueing network model represented in Figure 1.3 from models represented in Figures 1.1 and 1.2. As mentioned earlier, this example was chosen for illustrative purposes, and hence the generation of the queueing network model is greatly simplified.

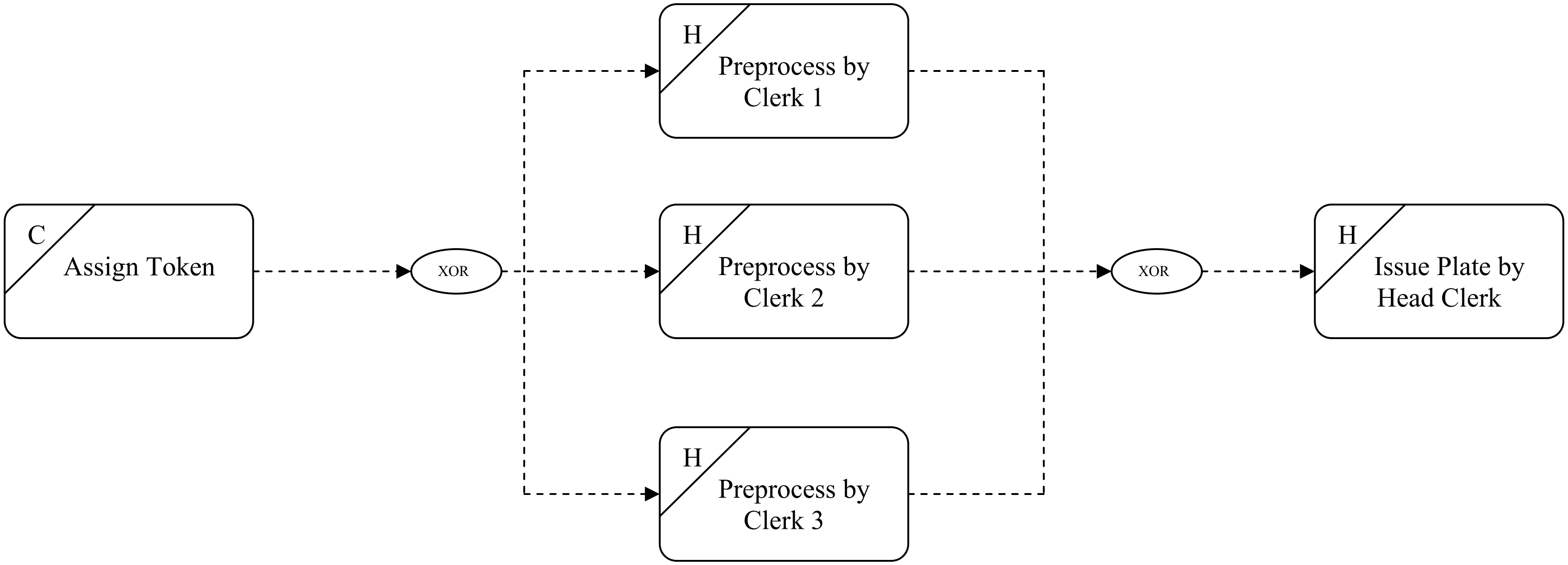
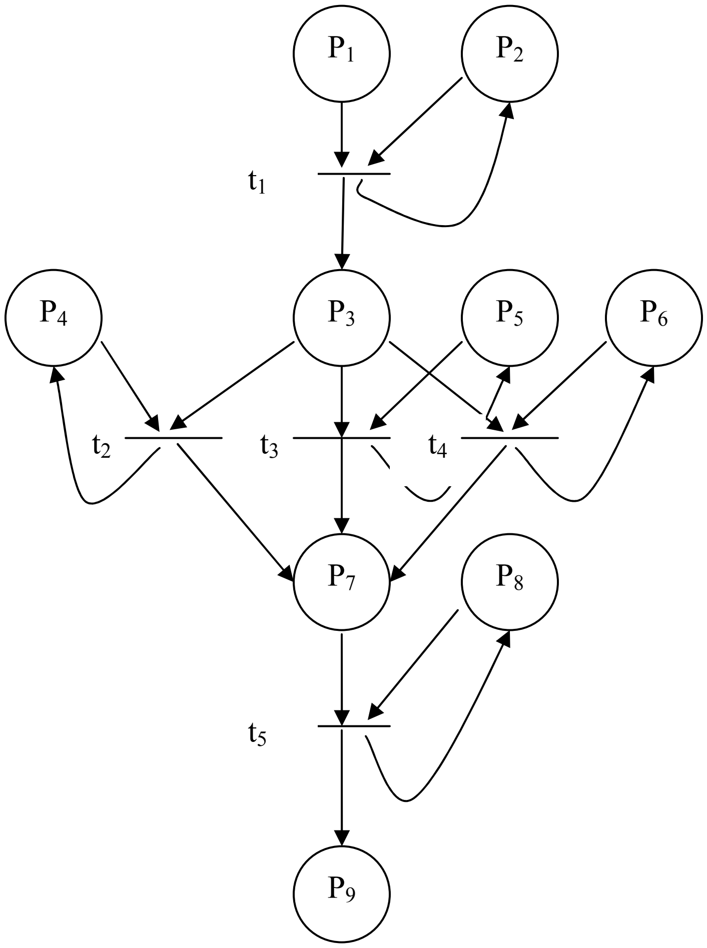


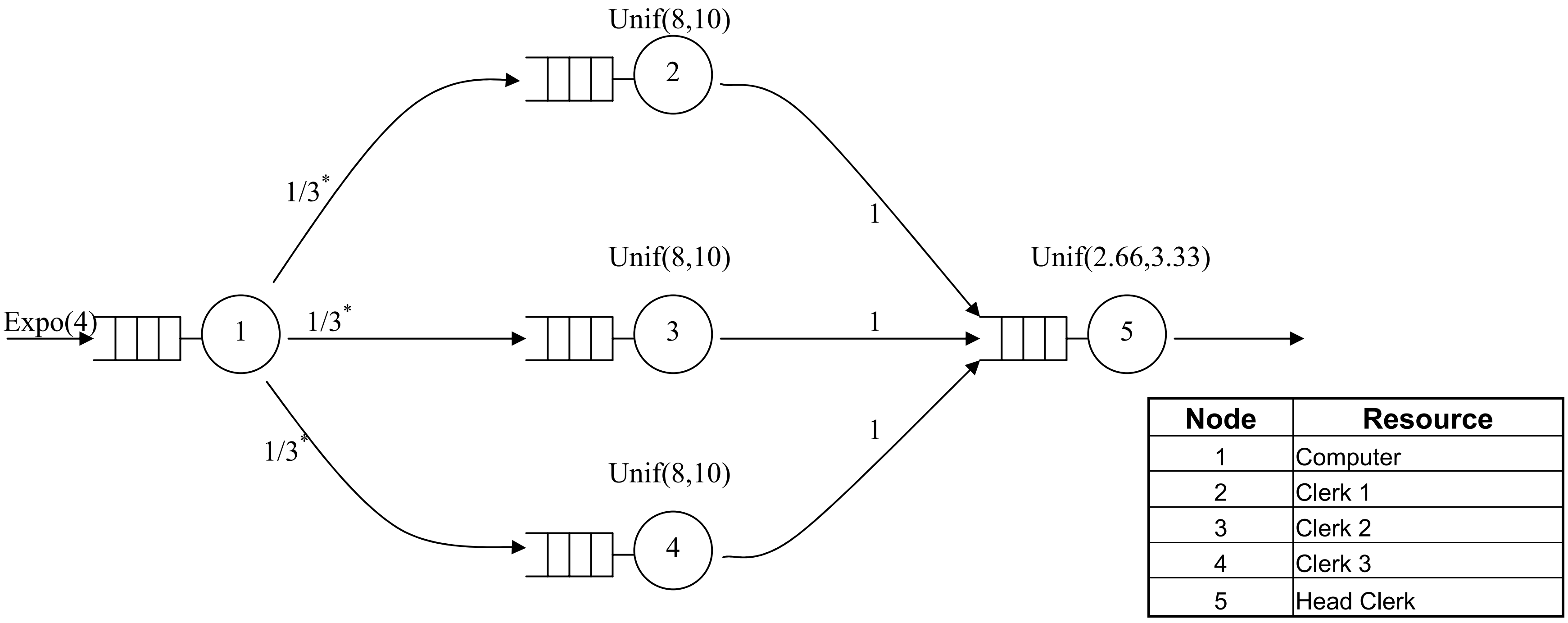
Figure 1.1. DDL Representation of Example 1.1



Place	Condition
P_1	Customer Arrived
P_2	Computer Available
P_3	Token Assigned
P_4	Clerk 1 Free
P_5	Clerk 2 Free
P_6	Clerk 3 Free
P_7	Payment Collected
P_8	Head Clerk Available
P_9	License Plate Issued

Transition	Activity
t_1	Assign Token
t_2	Preprocess by clerk 1
t_3	Preprocess by clerk 2
t_4	Preprocess by clerk 3
t_5	Issue License Plate

Figure 1.2. Petri net Representation of Example 1.1



Node	Resource
1	Computer
2	Clerk 1
3	Clerk 2
4	Clerk 3
5	Head Clerk

* Equal routing probabilities are assumed

Figure 1.3. Open Network Queueing Model of Example 1.1

The rest of the document is organized as follows. Chapter 2 is a review of XML and related technologies with a focus on their contribution to the area of enterprise modeling. Chapter 3 presents some background on the DIME framework, and briefly describes the DIME Descriptive Language and the various layers that make up the DIME framework. Chapter 4 presents the research statement, and the research approach and methodology that were followed to accomplish the goals of the thesis. Chapter 5 presents the research work on the development of the Queueing Network Markup Language (QNML). Chapter 6 presents the mapping schemes required for the single-step approach (described in Section 3.3) and the corresponding transformation algorithm for configuring a queueing network model from a business process description. Chapter 7 presents the mapping schemes required for the multi-step approach (described in Section 3.3) and the corresponding transformation scheme for configuring a queueing network model from a business process described using a Petri net model. Chapter 8 presents a qualitative comparison of the two approaches developed as part of this effort. Chapter 9 presents the contributions made by this thesis effort and explores some related areas for future work.

Chapter 2

Literature Review

Driven by the Computer Integrated Manufacturing (CIM) initiative, enterprise modeling was born in the United States in the early 80's and gained prominence through large efforts like the ICAM project on IDEF [12]. From then on, enterprise modeling has been an active research area through which many modeling languages, tools and approaches have been developed. Examples of these are CIMOSA, PERA, IEM, GERAM, and GRAI-GIM [12]. These techniques for enterprise modeling focused on different dimensions of the business process life cycle. With the widespread use of the Internet, the need for modeling techniques to support a distributed infrastructure, wherein users are able to create, modify, and analyze process models from any location across the globe, has increased. With the advent of XML and related technologies, and protocols like SOAP [2], current research efforts are directed towards enabling enterprise modeling in a distributed environment.

A brief overview of XML and related technologies is presented in Section 2.1, as these technologies enable the implementation of next generation enterprise modeling. The modeling techniques that use these technologies in the context of a distributed environment are presented in Section 2.2. Section 2.3 is a brief discussion of the DIME approach, and its similarities and differences with respect to the techniques discussed in Section 2.2.

2.1 XML and Related Technologies

XML is a markup language that allows users to define a set of tags that describe the structure of a document. XML provides a basic syntax but does not define the actual tags. The significant feature of XML is its extensibility. XML allows for custom tag-sets specific to corporations, scientific disciplines and other such domains [3]. For example, a user may specify logical tags such as Employee, SSN, Name, and Address to describe data related to an employee. These logical data structures specific to a corporation can be called as vocabularies. XML vocabularies provide more easily searchable documents and databases, and a way to exchange information between many different organizations and computer applications. Also additional validity constraints of these data can be stored in an associated file called the Document Type Definition (DTD) file [4] or the XML Schema (XSD) file [24].

XML provides a structural representation of data that can be implemented broadly and is easy to deploy. XML is a subset of SGML (Standard Generalized Markup Language), modified and optimized for delivery over the Web. This standard has been defined by the World Wide Web Consortium [4]. XML, which provides a data standard that can encode the content, semantics, and schemata for a wide variety of cases, ranging from simple to complex, can be used to mark up a purchase order, an invoice, a payment advice, information about people and organizations, etc. Thus, XML ensures that structured data will be uniform and understandable across a variety of applications, software vendors and customers. XML is valuable to the Internet because it provides interoperability using a flexible, open, standards-based format, with new ways of accessing legacy databases and delivering data to Web clients. Applications can be built more quickly, are easier to maintain, and can easily provide multiple views on the structured data. This resulting interoperability, maintainability and flexibility are the key to the next generation philosophy of modeling over the Internet.

2.2 Modeling Techniques based on XML

Business Process Markup Language (BPML) [1] suggested by the Business Process Management Initiative (BPMI) is a meta language for modeling business processes. This initiative has delivered a schema for BPML. This language provides a standard method to model mission-critical business processes. It also provides an abstracted execution model for collaborative and transactional business processes based on the concept of a transactional finite-state machine (FSM).

Electronic Business using eXtensible Markup Language (ebXML) was started in 1999 as an initiative of OASIS and the United Nations/ECE agency CEFACT [15]. The original project envisioned and delivered five layers of substantive data specification, including XML standards for business processes. ebXML is a modular suite of specifications that enables enterprises of any size and in any geographical location to conduct business over the Internet. Using ebXML, companies now have a standard method to exchange business messages, conduct trading relationships, communicate data in common terms and define and register business processes [7]. eBXML mainly addresses the aspects related to software design.

Unified Enterprise Modeling Language (UEML) [19] project was setup to contribute to the solving of problems arising from the existence of multiple enterprise modeling languages. The long-term objective of this project includes the definition of a Unified Enterprise Modeling Language that will provide new means to improve interoperability between business models, modeling languages and tools. The main aim of the UEML project is to achieve interoperability between existing supporting tools as well as newly developed tools.

Fishwick [16] presents the Rube Architecture that focuses on multimodeling and customization. The primary purpose of the architecture is to facilitate dynamic multimodel construction and reuse within 3D, immersive environment, which is a major ingredient of the next generation philosophy of modeling. MXL and DXL are the two modeling specification languages that are created to achieve the above stated purpose. However, Fishwick's [16] research to date is predicated on specifying and presenting models and not on the analysis part of it.

eXchangeable Routing Language (XRL) [21] is a XML-based process definition language which provides support for process routing between trading partners in order to provide Internet-based electronic commerce services. The core feature of XRL is that it provides a mechanism to describe processes at an instance level and not at the class level, which enables partial ordering of tasks for one specific instance. The semantics of XRL are expressed in terms of Petri nets for which powerful analysis techniques are available [21]. A prototype workflow management system called the XRL/flower is developed using Petri net based semantics.

Integrated Process Management (IPM) [8] is a business process management paradigm that aims at integrating processes using extensible Markup Language and supporting design, analysis, automation, and management of business process knowledge. IPM-PDL is an XML-based process definition language for integrated process management. Process definitions and related data are integrated using XML, which will be translated to a colored Petri net. Various analyses and simulation can be performed to check the validity of a new process and estimate its performance.

2.3 DIME's Approach to Enterprise Modeling

The XML-based techniques mentioned above are mainly designed for exchangeability, that is, to exchange process definitions between two different systems. Some of them just focus on process simulation, narrowly defined as execution of models, rather than specification or presentation of them. Hence, they are not suitable for supporting the whole process lifecycle since they concentrate on specific aspects of lifecycle such as process definition, model specification or presentation, execution, and process simulation, which by themselves are a part of the whole.

The overall approach of DIME, elaborated further in Chapter 3, is most similar to that of XRL/flower and IPM in that all three adopt XML for the process definition language. DIME Descriptive Modeling Language or DDL, shares similarities between XRL and IPM-PDL. However, the fundamental difference is that the Petri nets are used as the theoretical base in the DIME framework, whereas they are used for process simulation in XRL/flower and IPM. Other formal techniques such as queueing and simulation are suggested as potential tools for process analysis in the DIME framework.

The advantage in this approach is that the specification of the business rules specific to process analysis are postponed (not required) until the analysis technique or tool is actually decided. The previous approaches are limited to tools that support analysis using Petri nets, whereas the DIME's approach allows for compatibility with various tools that are designed to work for Petri nets, queueing or simulation. Thus the DIME approach has a broader scope in terms of performance analysis and improvement of business processes.

Chapter 3

Overview of the Distributed Integrated Process Modeling of Next Generation Enterprises (DIME) Framework

The DIME framework is a result of research funded by the National Science Foundation through grant # DMI-0075588, under the Scalable Enterprise Systems Initiative. The DIME framework aims at integrating processes using eXtensible markup language (XML) and supporting the design, analysis, automation, and management of business processes. This chapter presents the background material on the DIME framework. Section 3.1 describes the conceptual model that forms the basic framework of DIME. Section 3.2 explains the layers that comprise the DIME framework in detail. Section 3.3 presents the current status of the DIME project.

3.1 Conceptual Model

The following conceptual model for the framework is presented here for completeness. For a detailed treatment of this subject the reader is encouraged to refer to [6,10,11,12]. The emphasis here is on business users and specialized modelers, who create, modify, analyze, and use enterprise process models. The model comprises of at least two layers: front-end graphical and back-end formal. Additional layers could be added for analysis. A theoretical base is established by well-defined mappings between the user-oriented graphical model at the front end and corresponding formal representation at the back end. The mapping is two-way; a formal representation can be generated from a user's graphical model and vice versa.

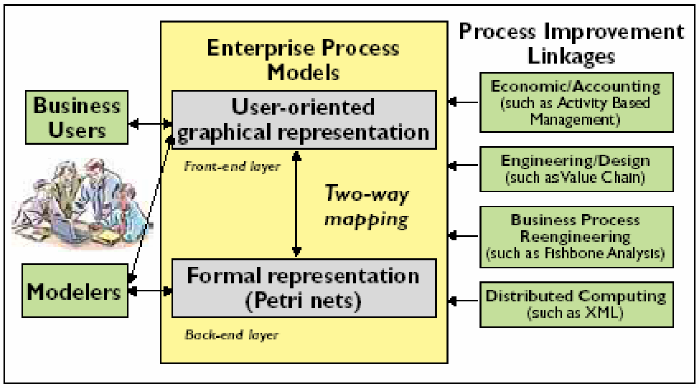

Enterprise Process Models
Process Improvement Linkages
Business Users
User-oriented graphical representation
Front-end layer
Two-way mapping
Modelers
Formal representation (Petri nets)
Back-end layer
Economic/Accounting (such as Activity Based Management)
Engineering/Design (such as Value Chain)
Business Process Reengineering (such as Fishbone Analysis)
Distributed Computing (such as XML)

Figure 3.1. Conceptual Model of the DIME Framework [6]

3.2 The DIME Framework

Based on the conceptual model described in Section 3.1, the DIME framework has been developed. It aims at integrating processes using the eXtensible mark-up language (XML) and supporting the design, analysis, automation, and management of business process knowledge. The DIME framework (Figure 3.2) is composed of three layers:

1. Descriptive Modeling Layer
2. Scalable Representation Layer and
3. Enterprise Analysis Layer

3.2.1 Descriptive Modeling Layer

A graphical front-end language called the DIME Descriptive Modeling Language (DDL) is defined in this layer. Following are the advantages of DDL over the existing modeling languages: ease of modeling, control flow representation, accuracy of modeling, differentiation between physical and electronic data, clarity of semantics, clarity of syntax, self-contained technique, separation of data and control flows, support for hierarchical modeling, and ability to support formal analysis [5].

DDL uses a 3-tiered approach for enterprise process modeling. A basic description of the process flow is created in the basic descriptive tier, which is mandatory. In the transformational tier the user specifies several technical and business parameters for each of the activities. In the tracking tier the user specifies target values or operating ranges for enterprise performance measures and links to the enterprise system to obtain real-time data or historical enterprise performance metrics. However, the transformational tier and the tracking tier are optional. The tiered approach marks the extensibility or scalability of the DDL.

3.2.2 Scalable Representation Layer

The scalable representation layer contains computer-processable representations that enable internal (within the enterprise) and external (such as suppliers, and customers) user groups to share information about processes. This layer includes XML representations of the descriptive and formal models and the DIME mappings that are achieved between these representations. The XML representation of the descriptive model is captured using the DIME Markup Language (DML). DML is consistent with emerging standards such as Resource Description Framework (RDF) and business-process modeling language or BPML. DML is automatically created from the DDL using a browser-resident program.

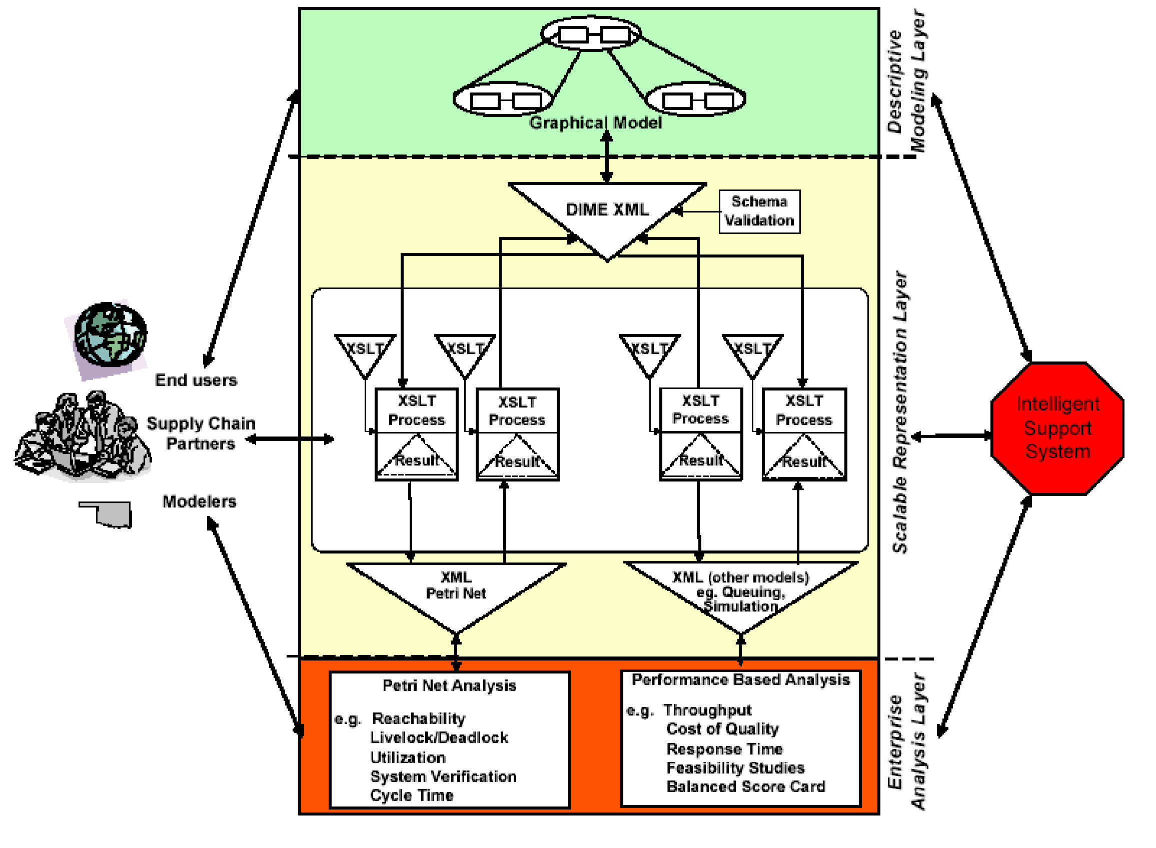


Figure 3.2. DIME Framework [13]

Typical DML statements are shown below [13]:

```
<Activity id="T1">
     <ActivityName>Issue Token</ActivityName>
     <Classification>M</Classification>
     <ActivityDuration>1</ActivityDuration>
     <SCV>1</SCV>
     <INPUT>
          <RESOURCES>
               <Resource num="1" ResID="#R1">
                    <UnitsRequired>1</UnitsRequired>
               </Resource>
          </RESOURCES>
     </INPUT>
 </Activity>
```

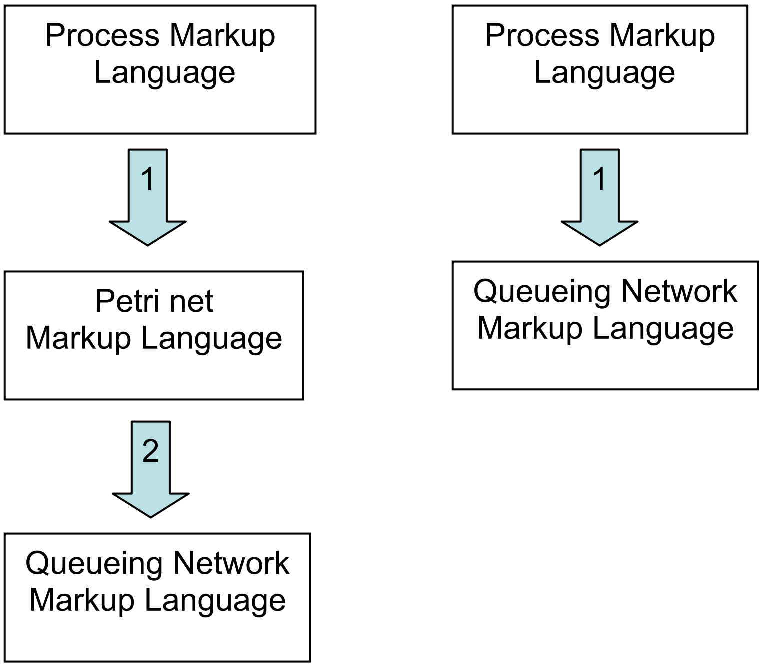
The other XML representations within this layer are for (1) Petri net models, (2) other formal models such as queueing and simulation, and (3) formal views, such as cost, resource, and productivity views. With Petri nets providing the theory base, the two-way mapping between the DML and the Petri net representation is fundamental to this approach.

3.2.3 Enterprise Analysis Layer

This layer includes the various types of formal analyses that can be done with the models created in the Scalable Representation Layer. System modelers, business process analysts and engineering personnel are the primary users that would interact with this layer. A knowledge-based expert system within DIME Intelligent Support System (DISS) assists the users in the selection of the appropriate analysis technique. The analyses in this layer include qualitative and quantitative analyses using Petri net models. However, due to the layered structure of DIME, other modeling approaches can be supported as well.

3.3 Current Status of the DIME Project

The first version of the graphical modeling language, the DIME Descriptive Modeling Language (DDL) has been developed [5]. It builds on the strengths of existing process modeling techniques such as data flow diagrams, IDEF techniques, and SAP's Event-driven Process Chain technique. The syntax and semantics of this language also incorporate the knowledge derived from Petri net representations of workflow constructs. A preliminary version of XML-based schema, for the DIME Markup Language (DML), that stores the graphical process models has been completed [13]. The theory behind the two-way mapping scheme between the graphical process modeling language (DDL) and the Petri net representations has been designed and tested. A proof-of-concept prototype has been developed wherein a process model can be built using the DDL constructs. The high level process model can further be expanded to describe the lower level processes. This prototype allows the conversion of the graphical process models to formal Petri net representations. This is achieved in two steps. In the first step, the information specific to the model is captured in a computer processable format using the DIME Markup Language (DML). In the second step, the DML representation that has been so obtained is transformed to an equivalent Petri-net representation using the Petri Net Markup Language (PNML). This transformation is accomplished using a XSL transformation.


Process Markup Language
1
Petri net Markup Language
2
Queueing Network Markup Language
Process Markup Language
1
Queueing Network Markup Language

(a) Multi-Step Approach **(b) Single-Step Approach**

Figure 3.3. Approaches to Analysis within the DIME Framework

In the initial version of the DIME framework, Petri nets serve as the theoretical base. Petri nets also provide the backend representation to enable further analyses. That is, the other formal representations for queueing, simulation, etc., should be generated from the Petri net representation. This multi-step approach is illustrated in Figure 3.3(a). In addition to the DDL capturing all the required information for the chosen analysis technique, if further validation is also incorporated within the analysis tool, then it is not always mandatory to make a transformation into the Petri net representation.

This leads to a new approach for generating formal representations directly from the graphical modeling language constructs. This single-step approach is illustrated in Figure 3.3(b). There was some ongoing research in this regard at the beginning of this thesis, and the validity of the single-step was being debated. Hence, the current research focused on generating formal representations for queueing models using both the multi-step and single-step approaches with an aim to evaluate the pros and cons of the two approaches. Eventually, this effort forms a basis for using either the multi-step approach or the single-step approach to support the multiple analysis capability within the DIME framework.

Chapter 4

Research Statement and Methodology

Existing enterprise process modeling frameworks do not provide adequate support for modeling and performance analysis of business processes in a distributed and integrated environment. This is primarily due to their concentration on specific aspects of the process lifecycle such as process definition, model specification or presentation, and execution. Specifically, there is a definite need for an approach to enable performance-based analysis of business processes using techniques like queueing and simulation in a distributed and integrated environment. Particularly, within the DIME framework, there is a need to extend analysis to include queueing and simulation techniques.

This research was an integral part of a larger process modeling research program and partially addressed the above issue. As explained in Section 3.3, the initial version of the DIME framework envisioned the generation of formal representations for queueing models using the Petri net representation as the backend. However, in this thesis effort, the formal representation for queueing analysis will be generated from both the Petri net representation and the DIME Descriptive Modeling Language. This thesis also evaluated and compared these two approaches called the multi-step and single-step approaches, respectively.

4.1 Research Objectives

The purpose of this thesis was to design, develop, and test ways to generate queueing network models from both the existing graphical modeling language description and the backend Petri net representation. To achieve this, the following objectives were identified.

1. To develop a Queueing Network Markup Language (QNML), that allows for computer processable apecification of queueing network models.
2. To generate a corresponding schema for QNML.
3. To conduct a thorough evaluation of the DIME Descriptive Modeling Language, in order to analyze its features and contribution to the modeling task, and to enhance its constructs to support quantitative and qualitative analyses.
4. To study linkages for mappings between Petri Net Markup Language [2, 9] (PNML) and Queueing Network Markup Language (QNML) and generate appropriate transformation schemes for the multi-step approach as illustrated in Figure 3.3(a).
5. To study linkages for mappings between the DIME Descriptive Modeling Language (DDL) and Queueing Network Markup Language (QNML) and generate appropriate transformation schemes for the single-step approach as illustrated in Figure 3.3(b).
6. To recommend an approach to generate formal representations for queueing analysis within the DIME framework, by evaluating the pros and cons of the multi-step and the single-step approaches as described in Section 3.3.

4.2 Scope of the Research

The purpose of this research was to develop formal representations within the DIME framework to support performance analysis using queueing theory. The central idea was to develop a general purpose representation which could be used either as a part of an integrated environment or used in isolation. The current DIME framework is designed to use Petri net representation as the back-end to generate other formal representations. However, as part of the current effort, two-way mappings schemes between DIME Descriptive Modeling Language and queueing models were explored in addition to the mappings between the Petri nets and queueing models. Other theoretical bases were not explored as part of this research. Also, formal representation for analysis using simulation techniques, which is a part of the DIME framework, was not explored as part of this thesis effort.

4.3 Research Methodology

In order to accomplish the objectives stated in Section 4.1, the effort was divided into the following stages.

<u>Stage 1:</u> The existing modeling approaches were studied and explored in detail to gain an understanding of their purpose, strengths and limitations.

<u>Stage 2:</u> In this stage, the linkages between process modeling constructs, their representations and corresponding queueing models were identified. Specifically, linkages between the DIME Modeling Language, corresponding Petri net representation and queueing network models were studied.

Stage 3: Based on the information collected above, a set of elements that are required for the queueing network models were identified, and a meta model was developed. Based on the meta model a markup language (QNML) for queueing networks was specified.

Stage 4: In this stage, the mapping schemes between DDL and QNML, and between PNML and QNML were designed. Transformations were accomplished through general transformation algorithms developed based on the mapping schemes.

Stage 5: In this stage, the pros and cons of the approaches developed in the previous stage were evaluated, and an approach is recommended to generate formal representations for queueing analysis within the DIME framework.

Stage 6: This stage involved potential enhancements and extensions to the DIME Descriptive Modeling Language (DDL) and the DIME framework to incorporate the changes required.

Chapter 5

Queueing Network Markup Language (QNML)

This chapter presents the work on the development of a markup language for queueing network models, called the Queueing Network Markup Language (QNML). Section 5.1 gives a brief introduction to the QNML. The building blocks or the elements of the QNML are presented in Section 5.2. To standardize the QNML and to validate QNML documents, a schema is proposed in Section 5.3.

5.1 Introduction

Queueing Network Markup Language (QNML) proposed here is an XML-based interchange format for queueing network models. The QNML described here is a starting point for a standard interchange format for queueing network model specifications. Although the QNML developed here focuses on supporting queueing analysis within the DIME framework, it has enough generality to be used in other settings as well. This section presents a preliminary meta model for QNML and describes the individual QNML elements. Figure 5.1 shows the meta model of basic QNML in UML notation (see note in Appendix A for details of the UML notation). The meta model for QNML consists of classes that makeup the markup language. These classes are translated to XML elements. These elements are the keywords of QNML and can be called as QNML elements. Each element is comprised of attributes and other elements. A unique identifier

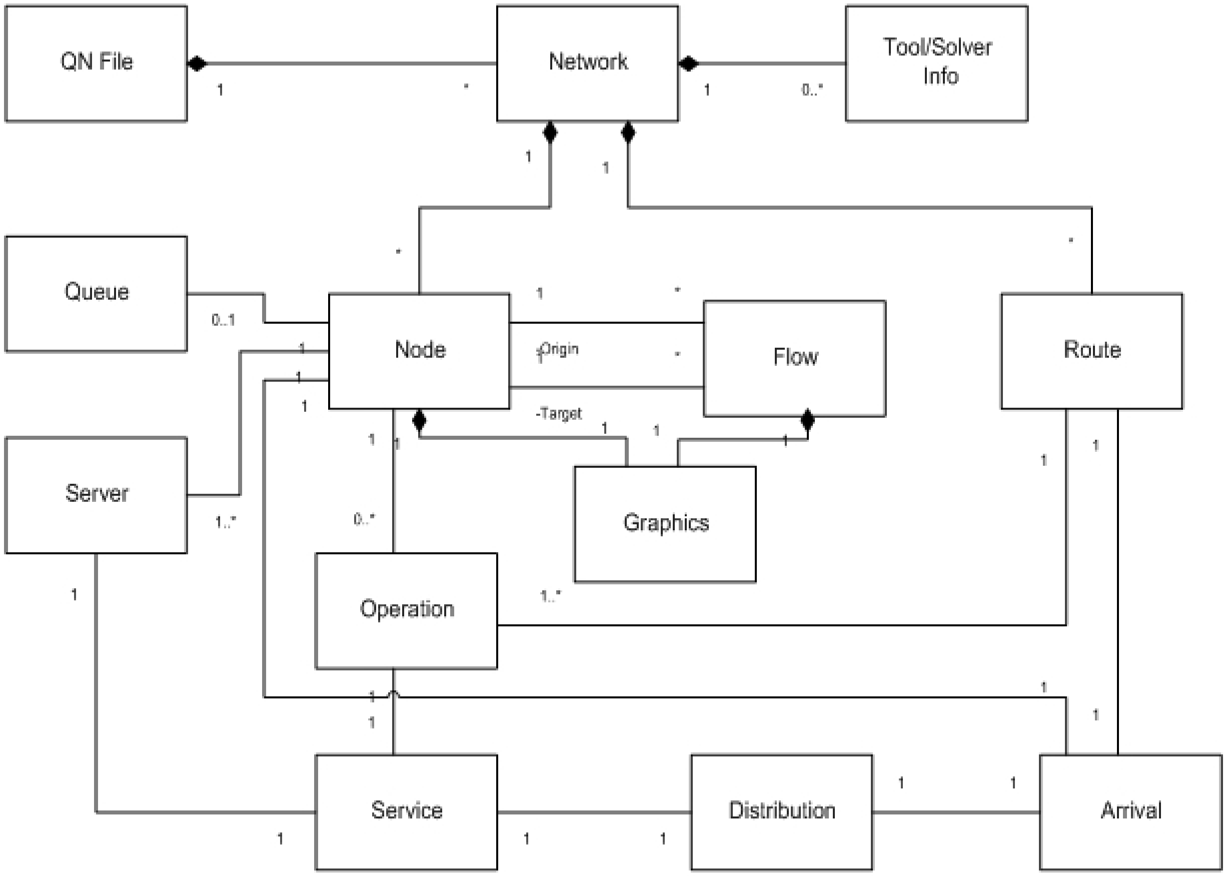


Figure 5.1. Meta Model for the Basic QNML
(Refer Appendix A for UML Notation)

is used to refer to each of the elements. The associations between the elements in the meta model define the relationships between them.

5.1.1 Description of the QNML Meta Model

A queueing network file may consist of one or more queueing network specifications. A network can be an open network, a closed network or a mixed network. Each network is made up of nodes and flow elements that connect the nodes. Each node is associated with one or more servers. A single-server node has a single server associated with it and a multi-server node has at least two servers associated with it. Also there is a queue in front of every node except a delay (or infinite server) node. A node is further comprised of arrival and service elements that correspond to the arrival and service information of the entities that visit the node. The arrival and service elements share a distribution class to describe the parameters of the arrival and service processes. A distribution element here could take a two-moment approach or include the corresponding probability distribution parameters for describing the inter arrival or service information. A flow element can be viewed as an arc connecting two nodes. Hence, it has an origin node and a destination node as its sub-elements. The flow element also captures the routing information of entities or a class of customers flowing from a node to any other node in the network. The flow of entities through the network could also be described by a collection of routes. Each route consists of a sequence of operations that correspond to a deterministic path an entity or a class of customers can take. Each operation is associated with a node in a route and can be viewed as a node visit of an entity following that route. A flow element captures the probabilistic routing information and a route captures the deterministic routing information of entities or classes of customers in the queueing network model.

5.2 QNML Elements

This section describes the individual elements of the QNML. A brief description of each of the elements is followed by its sub-elements. The data types and allowable values are also listed along with the sub-elements.

QNFile

A QNFile element is used to describe a queueing model. As a queueing model can be made up of one or more networks, a QNFile has one or more of the Network elements.

Network

A Network element represents a queueing network in the queueing model. It could be of one of the following three types: Open, Closed or Mixed. A network is composed of one or more nodes and zero or more route elements. In addition, the sub-elements listed in Table 5.1 describe the network.

Table 5.1 Sub-Elements of Network

Element	Description	Data Type	Allowable Values
Type	Describes the type of network	String	Open/Closed/Mixed
CustomerCount	Maximum number of customers allowed in the network at any one time	Integer	1, 2, …, ∞
NodeCount	Number of nodes in the network	Integer	1, 2, …, N
RoutingInformation	Describes if the routing information provided is either Probabilistic or Deterministic	String	Probabilistic/Deterministic

Node

A node represents a resource pool in the queueing network. It is typically associated with a queue and has one or more servers. In addition to the queue and server elements, other sub-elements of a Node element are listed in Table 5.2.

Table 5.2 Sub-Elements of Node

Element	Description	Data Type	Allowable Values
Name	Resource name	String	e.g. Clerk
Type	Type of the resource	String	Human,Computer,Machine
Descriptiom	Description of the resource	String	e.g. Pre-processing
ServerCount	Count of the number of units of a resource available	Integer	1, 2, …, N
Utilization	To store the utilization of the node	Double	ℝ

Queue

A Queue element is associated to a node and represents the queue that is present in front of any node. The elements listed in Table 5.3 help describe a queue.

Table 5.3 Sub-Elements of Queue

Element	Description	Data Type	Allowable Values
Size	Capacity of the queue	Integer	1, 2, …, ∞
Discipline	Service Discipline	String	e.g. FIFO/LIFOSIRO
Mean Length	Average Queue Length	Double	ℝ

Server

A Server element is associated with a Node. A server is a specific resource instance from the resource pool. The elements listed in Table 5.4 help describe a server.

Table 5.4 Sub-Elements of Server

Element	Description	Data Type	Allowable Values
Service Time	Processing time information	Service	
Utilization	Utilization of particular resource	Double	ℝ

Operation

An Operation element is associated with a Route element. It is a node visit in the route. Hence it refers to a Node element. A mean processing rate of service and a squared coefficient of variation of the service time or a distribution are needed to describe an Operation. These service attributes are readily captured by a Service element. Hence, an Operation element has Service as its sub-element.

Arrival

An Arrival element provides the arrival information to a node. It is composed of a Distribution element and also has rate and squared coefficient of variation (SCV) as its sub-elements as listed in Table 5.5.

Table 5.5 Sub-Elements of Arrival

Element	Description	Data Type	Allowable Values
ArrivalRate	External arrival rate to the node	Double	$\mathbb{R}$
SCV	Squared coefficient of variation of the interarrival time	Double	$\mathbb{R}$
Distribution	Probability Distribution of the interarrival time	String	e.g. Exponential

Service

The Service element provides the service information for server or an operation. It is composed of a Distribution element to and also has mean rate and squared coefficient of variation as its sub-elements as listed in Table 5.6.

Table 5.6 Sub-Elements of Service

Element	Description	Data Type	Allowable Values
Mean	Mean service time	Double	$\mathbb{R}$
SCV	Squared coefficient of variation of service time	Double	$\mathbb{R}$
Distribution	Probability Distribution of the service time	Distribution	e.g. Exponential

Distribution

A Distribution element is used to specify probability distributions. Table 5.7 lists its sub-elements.

Table 5.7 Sub-Elements of Distribution

Element	Description	Data Type	Allowable Values
Name	Name of the distribution	String	e.g. Poisson
Parameters	Parameters associated with the distribution	ArrayList	

Flow

A Flow element corresponds to probabilistic routing between the nodes of a queueing network. It specifies the probability with which an entity is routed from a node to another node. Table 5.8 lists the sub-elements of a Flow element.

Table 5.8 Sub-Elements of Flow

Element	Description	Data Type	Allowable Values
OriginNode	Origin Node	ID Ref	ℝ
DestinationNode	Destination Node	ID Ref	ℝ
RoutingProbability	Probability of visting the destination node next	Double	[0,1]

Route

A Route element is an ordered collection of operations. A multi-class queueing network can be specified using a collection of routes. Table 5.9 lists the sub-elements of a Route element.

Table 5.9 Sub-Elements of Route

Element	Description	Data Type	Allowable Values
OperationCount	Number of operations in the route	Integer	ℕ
ArrivalRate	External arrival rate to the first node in the route	Double	ℝ
SCV	Squared coefficient of variation of the interarrival time	Double	ℝ
Operation	Node visits	Operation	

Graphics

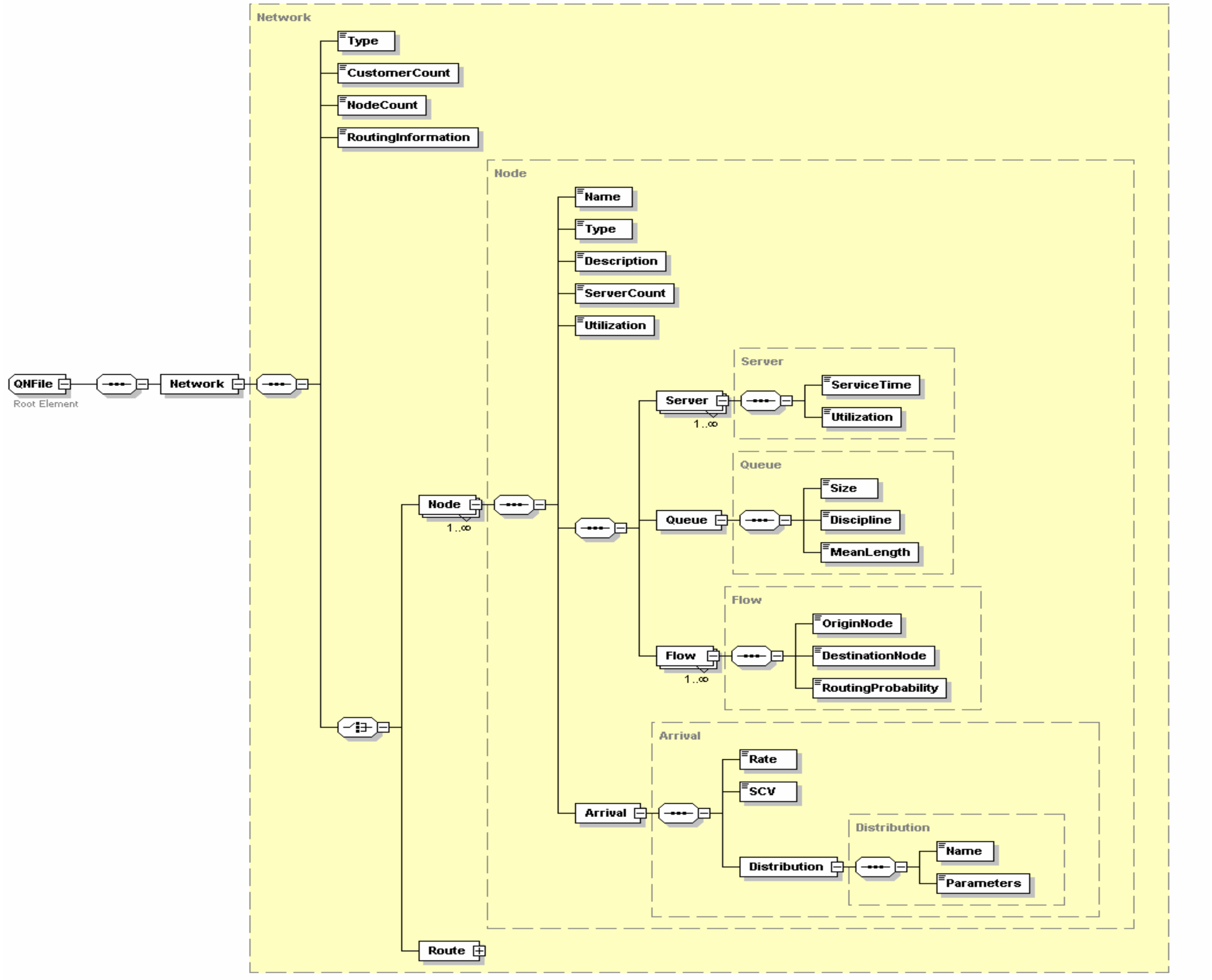
A Graphics element is provided to store the graphical information related to the pictorial representation of the queueing network model. This element as such does not add any extra detail to the specification of the queueing network, but could potentially be used for storing information related to the graphical representation of the queueing network. Since this information is tool specific, the details of this element are ignored in this discussion.

Tool/Solver Info

This element is defined to capture any tool specific information that is of significance. Potentially, this could be used to interpret the performance measures or account for any assumptions that are specific to the tool used. Ideally, if all the queueing analysis software tools report the same performance measures in the same format, then this element can be eliminated. The specific sub-elements cannot be determined at this stage as the QNML has not been used or tested with the available queueing analysis software tools.

5.3 QNML Schema

Based on the elements that are described in the previous section, the allowable values for each element are captured as part of the QNML schema defined in this section. Figures 5.2a and 5.2b show the design view of the schema developed for the QNML. Figure 5.2a shows the schema of the branch that uses Nodes and Flow elements (probabilistic routing) to describe a Queueing Network Model, whereas Figure 5.2b shows the schema of the branch that uses Routes (deterministic routing) to describe a Queueing Network Model. Please refer to Appendix B for the schema document of the QNML.



Generated with XMLSpy Schema Editor www.xmlspy.com

Figure 5.2a. QNML Schema for Probabilistic Routing

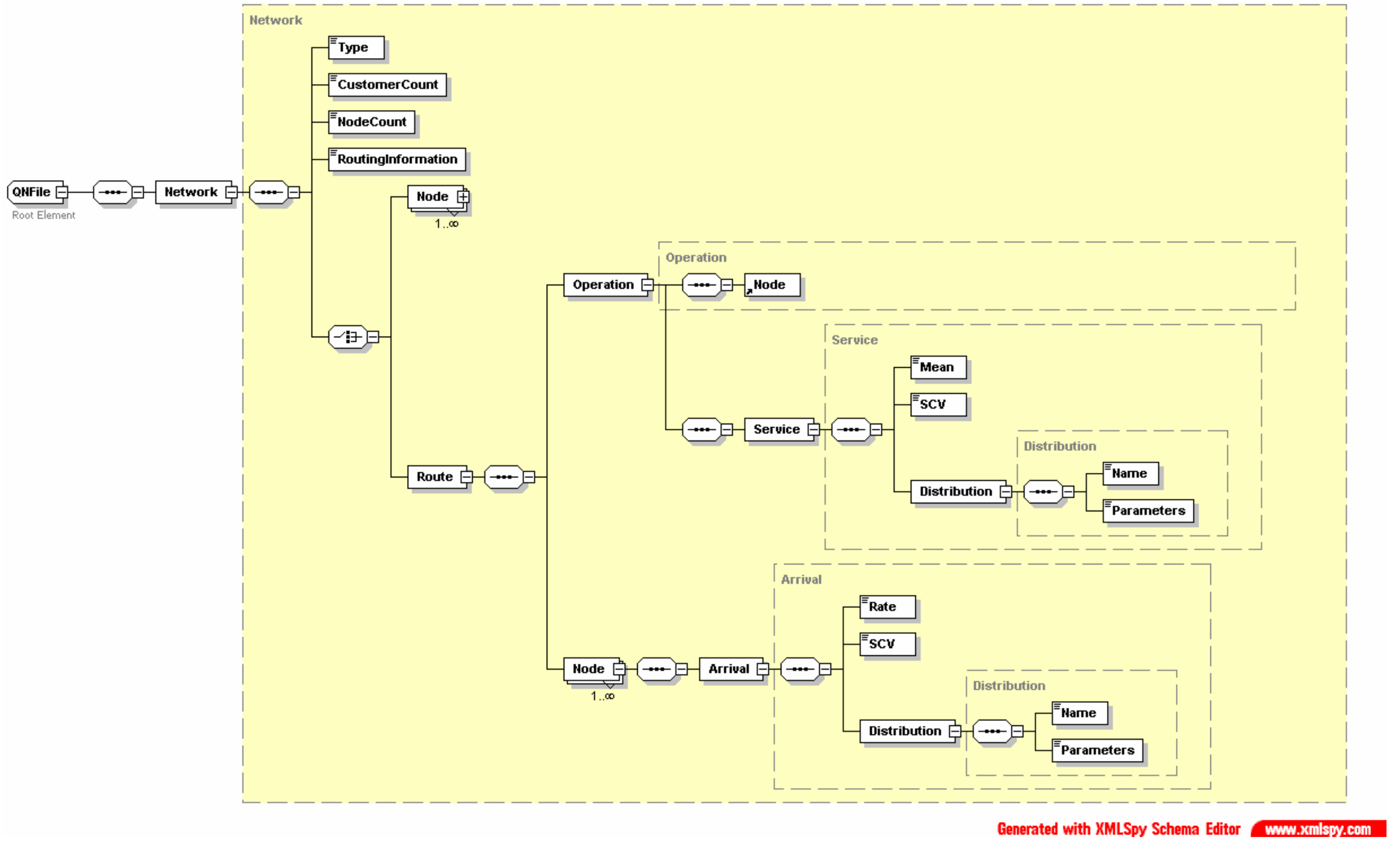


Figure 5.2b. QNML Schema for Deterministic Routing

Chapter 6

A Single-Step Approach for Automatic Configuration of a Queueing Network Model

This chapter presents an overview of the procedure developed and the mapping scheme required for the “single-step” approach (described in Section 3.3) to automatically configure a queueing network model from a business process description. Section 6.1 presents the definitions, formal notations and representations for both process models and queueing network models. The mappings that are required to translate a business process description, which is described by the DIME Markup Language (DML) to a queueing network model described by Queueing Network Markup Language (QNML), are presented in Section 6.2. Section 6.3 presents the algorithm that automates the translation of DML to QNML. Section 6.4 presents the implementation details of the transformation.

6.1 Introduction and Background

In the single-step approach a process model description is used to directly generate a queueing network model for analysis. The transformation is actually between the computer processable formats of the corresponding models. More specifically, a business process model, represented by a markup language like DML

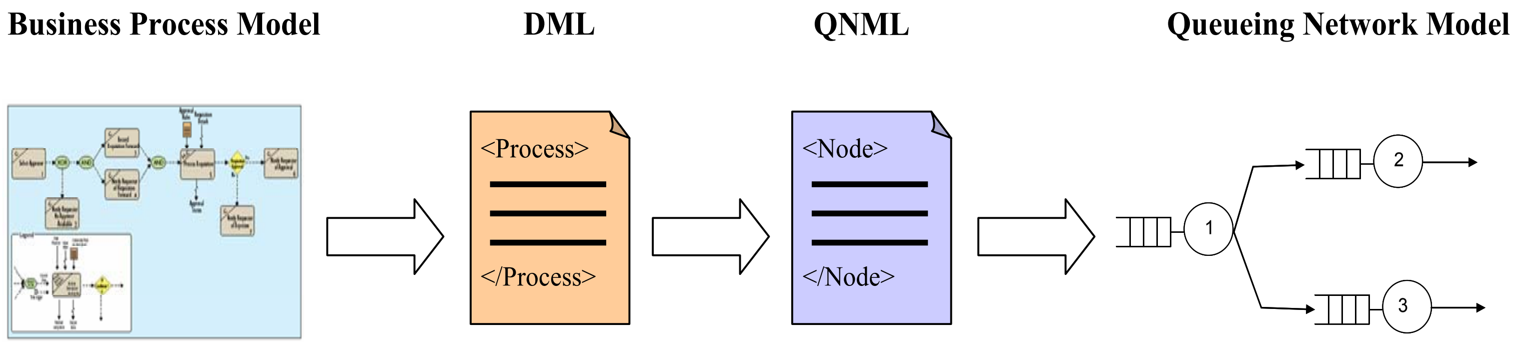


Figure 6.1. Pictorial Representation of the Single-Step Approach

is transformed to a queueing network model, represented by a markup language like QNML. To achieve this transformation it is important to identify the mappings between the various elements that make up each of the models. Section 6.1.1 presents the formal notations required to define both a process model as well as a queueing network model. Section 6.2 uses the formal notation of Section 6.1.1 to define the mappings between a business process model and a queueing network model. Figure 6.1 shows these ideas pictorially.

The definitions and formal notation suggested by [17] are used for defining a process model. The definitions from [23] are used for defining queueing network models.

6.1.1 Business Process Model Notation

A business process model is a collection of elements listed in Table 6.1.

Table 6.1 Elements of a Business Process Model

Element	Notation
Activities / Tasks	T
Resources	R
XOR – Splits	X_s
XOR – Joins	X_j
And – Splits	A_s
And – Joins	A_j

Task / Activity: An abstraction of either a unit activity or a composite description of a larger sub-process, embedded in the process's definition. It is graphically represented with a rounded rectangular symbol in accordance with DDL.

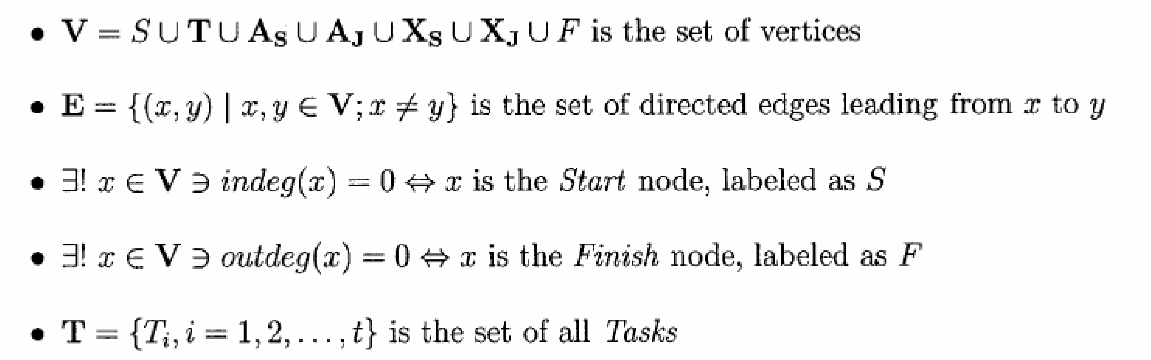
AND-Split: A logical operand that models the concurrent creation of several parallel threads of control from a single incoming flow.

AND-Join: A logical operand that models the asynchronous completion of several parallel sub-threads of execution, to be followed by a common outgoing flow.

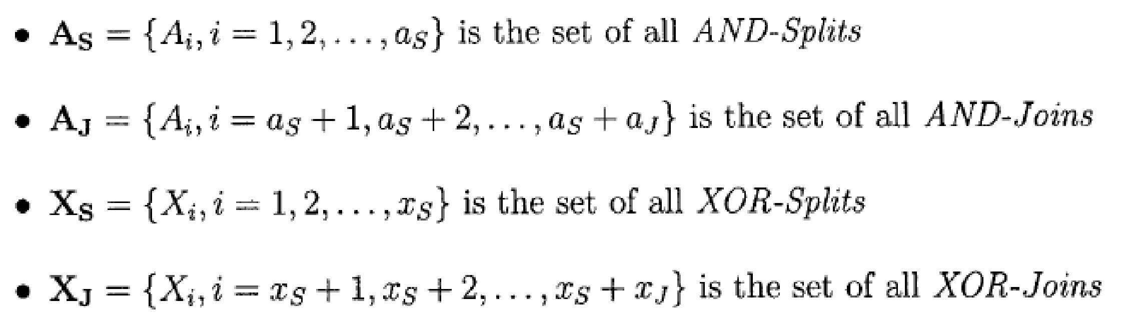
XOR-Split: A logical operand that depicts choice in the selection of exactly one of several possible outgoing control flows from a single incoming flow.

XOR-Join: A logical operand that merges several mutually exclusive, multiple sources of control, to create a common outgoing flow.

Formally a business process model is a directed graph G = (V, E) where



- $\mathbf{V} = S \cup \mathbf{T} \cup \mathbf{A_S} \cup \mathbf{A_J} \cup \mathbf{X_S} \cup \mathbf{X_J} \cup F$ is the set of vertices
- $\mathbf{E} = \{(x, y) \mid x, y \in \mathbf{V}; x \neq y\}$ is the set of directed edges leading from x to y
- $\exists!\ x \in \mathbf{V} \ni indeg(x) = 0 \Leftrightarrow x$ is the *Start* node, labeled as S
- $\exists!\ x \in \mathbf{V} \ni outdeg(x) = 0 \Leftrightarrow x$ is the *Finish* node, labeled as F
- $\mathbf{T} = \{T_i, i = 1, 2, \ldots, t\}$ is the set of all *Tasks*



- $\mathbf{A_S} = \{A_i, i = 1, 2, \ldots, a_S\}$ is the set of all *AND-Splits*
- $\mathbf{A_J} = \{A_i, i = a_S + 1, a_S + 2, \ldots, a_S + a_J\}$ is the set of all *AND-Joins*
- $\mathbf{X_S} = \{X_i, i = 1, 2, \ldots, x_S\}$ is the set of all *XOR-Splits*
- $\mathbf{X_J} = \{X_i, i = x_S + 1, x_S + 2, \ldots, x_S + x_J\}$ is the set of all *XOR-Joins*

The following additional definitions (not part of [17]) were developed.

Definition: **P** = {P_1, P_2, P_3, …, P_p}is the set of all **PATH**s from S to F.

A **PATH** from **S** to **F** is an ordered sequence of activities (S, T_1, T_2, T_3, …, T_n, F), where $T_i \in$ **T.** This definition of **PATH** is a simplified definition of the **PATH** contained in [17].

Definition: A function **Enumerate (G) = P,** returns the set of all **PATH**s of the business process model represented by **G**.

This enumeration for a business process model to generate all paths can be achieved using an algorithm like the one described in [17]. Developing such an algorithm was not within the scope of the current effort. In the example shown in the Figure 6.1, the two possible **PATH**s are:

$P_1 = \{T_1, T_2, T_3, T_4\}$ and $P_2 = \{T_1, T_2, T_5, T_6\}$.

Hence, Enumerate(G) should return $\{P_1, P_2\}$. The "Start (S)" and "Finish (F)" nodes have been omitted here for simplicity.

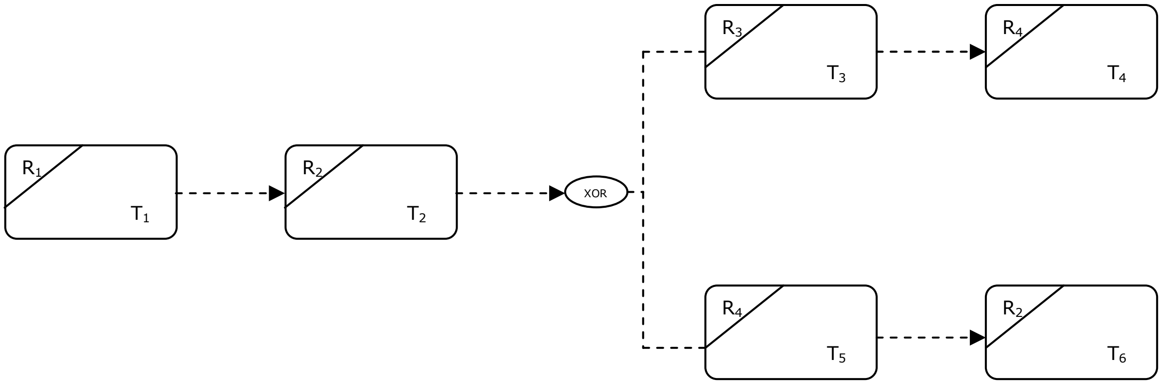
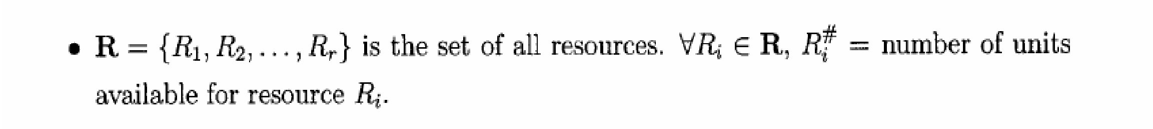


Figure 6.2. A Business Process Model with Multiple Paths

Resources

A business process in the course of its execution, will use some resources – more specifically, tasks in a process will often require the use of resources (e.g., machines, people, and instruments) that the tasks capture (i.e., access and exclusively use), which are then released by either the tasks that captured them, or by subsequently executed tasks. In specifying the resource requirements of a process, the focus is on re-usable, non-perishable, non-depleteable physical or informational entities that are accessed or captured by tasks, which are then subsequently released wholly, without loss or detriment in their size, quantity or operational ability [17]. A resource is graphically represented at the top left corner of the activity block in accordance with DDL. More formally, the resource requirements for the tasks in a process are specified as defined in [17].



- $\mathbf{R} = \{R_1, R_2, \ldots, R_r\}$ is the set of all resources. $\forall R_i \in \mathbf{R}$, $R_i^{\#}$ = number of units available for resource R_i.

The following additional definition (not part of [17]) is used.

Definition: A function **Resource(**T_i**)** = {R_k ∈ **R**}, returns the set of resource(s) required by activity T_i.

6.1.2 Queueing Network Model Notation

At the outset, a queueing network model is a network of nodes and directed arcs. It is important to note that the nodes represent service facilities and the arcs represent movement of customers, jobs, or data packets. Customers enter the network at any of the nodes, move from node to node along the directed arcs, and eventually leave the system from a node. To model such a system, information regarding the nodes and the routing information is critical. The routing is either deterministic or probabilistic in nature. Deterministic routing is specified by a set of routes and probabilistic routing by a routing matrix. Deterministic routing is typically used when the specific routes can be clearly identified, i.e., ordered sequences of node visits that different customer types (or classes) follow. Probabilistic routing is used when the number of possible routes becomes very large, and hence, difficult to specify. The customer flow in this case is specified by routing probabilities, i.e., probabilities of going to different destination nodes from a given origin node. The routing probabilities are often arranged in the form of a routing matrix.

Since deterministic routing using routes has a closer resemblance to actual process executions, it was chosen as the mode of specification in all the models (discussed in Section 6.2) that would allow such a specification. The required elements required to specify a queueing network model are listed in Table 6.2.

Table 6.2. Elements of a Queueing Network Model.

Element	Notation
Node	**N**
Routing Matrix	**Q**
Route	**r'**

Node: A node is a service facility, where customers come for some service or processing. A node consists of servers (resources), which provide service, to the customers. If all servers at a node are busy when a customer arrives, then the customer joins a queue and waits until a server is free.

Routing Matrix: The flow of customers from one node to another is specified by a routing matrix. It is (n × n) matrix, where n is the number of nodes in the network. Each element in the matrix is a probability with which a customer moves from a node to another node in the network. For example, q_{ij} the element in row i and column j, corresponds to the probability with which a customer visiting node i visits node j next for service.

Route: A route specifies the sequence of the nodes visited by a class of customers. The flow of customers specified by routes is deterministic in that the sequence is ordered and given.

Formally, N = {Ni, i= 1, 2, 3, …, n} is a set of nodes in the network.

Also for each node N_j,

m_j = number of servers at node N_j.

λ_{0j} = external arrival rate at node N_j.

c_{0j}^2 = squared coefficient of variation of external arrival process at node N_j.

τ_j = mean service time at node N_j.

c_{sj}^2 = squared coefficient of variation of service time distribution at node N_j.

Q is a (n × n) routing matrix.

$Q \equiv [q_{ij}]$; i, j = 1, 2, 3, …, n.

r' = {$r_{i\,,}$ i = 1, 2, 3, …, r} is the set of all possible routes in the network,

Where, a route $r_1 = \{n_1, \lambda_1, c_1^2; N_{11}, \tau_{11,} c_{s11}^2; N_{12}, \tau_{12,} c_{s12}^2; \ldots; N_{1n_1}, \tau_{1n_1}, c_{s1n1}^2\}$

where, n_i = number of nodes on route k

λ_i = external arrival rate for class k represented by route r_i

c_i^2 = variability parameter of external arrival process for route r_i.

n_{ij} = the *j*th node visited by class on route r_i

τ_{ij} = the mean service time of class i on route r_i at the j^{th} node on route r_i.

c_{si}^2 = the squared coefficient of variation of the service time of class i on route r_i at the j^{th} node.

The above set of notations was chosen based on the notation defined in [23] which is considered to be a defacto standard framework for queueing network modeling and solution.

6.2 Mapping Schemes

A study of various examples of business process models revealed the following types of business process model configurations.

Type 1. A sequential model with a series of activities where each activity requires a different resource.

Type 2. A sequential model with a series of activities, where some of the activities share (common) resources.

Type 3. A model with a choice of paths. Each path is a sequential model with a series of activities where each activity requires a different resource.

Type 4. A model with a choice of paths. Each path is a sequential model with a series of activities where some of the activities share (common) resources.

Type 5. A model with branching into concurrent set of activities that merge later, where some of the activities may share (common) resources.

By using examples of each type of the above process model configurations and their equivalent queueing models, it was observed that models of types 1, 2, 3 and 4 are typical and give rise to queueing network models that can be solved at least approximately. However, there is no satisfactory means of solving models with concurrency (type 5) using existing queueing network theory. So type 5 was determined to be beyond the scope of the thesis effort and hence, not included for further consideration. The mapping schemes for types 1, 2, 3 and 4 are explained in Section 6.2. For the sake of simplicity, following assumptions were made while modeling resource requirements.

- An activity requires only one unit of an available resource.
- A resource that is seized by an activity is released by the same activity.
- A resource that is seized by an activity is released by the same activity before the control is transferred to the succeeding activity, if any.

Type 1: A sequential process model with a series of activities where each activity requires a different resource.

Case 1a: No feedback is involved.

Model

Figure 6.3a is a pictorial representation of a sequential process model. Note that there could be multiple instances of the process depicted that are active at any given time.

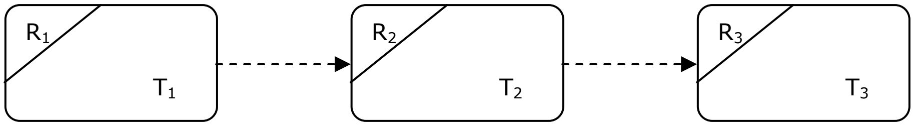


Figure 6.3a. A Sequential Process Model

Additional Assumptions

- Without any loss of generality it can assumed that the activity T_j requires resource $R_{j\,,}$ j = 1, 2, …, t.
- There is no feedback. A process instance executes an activity only once.

Mappings

- The queueing network model consists of a single class of customers. For every resource R_j in resource set **R,** there exists a corresponding node N_j in the queueing network model. The number of nodes in the network, n = |R| = t.
- The number of servers at node $N_{j,}$ m_j is equal to $R_j^{\#}$, the number of available units of resource $R_{j.}$
- Because of the sequential nature, there exists only one path $P_1 = \{T_{i,}\ i=1, 2, \ldots, t\}$ specified by the control flow in the process model. This would correspond to one route, r_1 in a queueing network.
    - This implies that **r'** = $\{r_1\}$
    - Where

      $$r_1 = \{n_1, \lambda_1, c_1^2; N_{11}, \tau_{11,} c_{s11}^2; N_{12}, \tau_{12,} c_{s12}^2; \ldots; N_{1n_1}, \tau_{1n_1}, c_{s1n1}^2\}$$

      Here n_1 = t, as the number of operations is equal to the number of activities.
- The service time parameters of a node (operation) N_j on route r_1 correspond to the activity duration parameters of activity T_j that requires resource R_j. For example, the mean service time, $\tau_{j,}$ of node N_j is the mean duration of T_j that requires R_j.
- The external arrival rate parameters λ_1 and c_1^2 are obtained from the process initiation logic.

Formally the mappings can be summarized in the following steps:

1. $\forall$ R$_j$ $\in$ **R** $\exists$ N$_j$ $\in$ **N**
2. $\forall$ N$_j$ $\in$ **N**, m$_j$ = R$_j^{\#}$
3. **P** = {P$_i$, i = 1}

   P$_1$ = {T$_1$, T$_2$, ... , T$_t$}
4. $\forall$ T$_i$ $\in$ P$_1$

   Resource(T$_i$) = R$_{k_i}$ and node N$_{k_i}$ corresponds to R$_{k_i}$

   τ_i = Duration(T$_i$)

   c_{si}^2 = SCV(T$_i$)
5. $\forall$ P$_i$ $\in$ **P** $\exists$ r$_i$ $\in$ **r′**

   Here, r$_1$ corresponds to P$_1$.

   r$_1$ = (t, λ_1, $c_1{}^2$; N$_{k_1}$, τ_1, c_{s1}^2 ; ... ;N$_{k_t}$, τ_t, c_{st}^2)

Case 1b: Feedback is involved.

Model

The pictorial representation of a sequential process model with feedback is shown in Figure 6.3b. The feedback is after activity T$_2$ in the model represented in Figure 6.3b. In a general model the feedback can transfer the control to any of the activities preceding the XOR junction used for feedback. Note that there could be multiple instances of the process depicted that are active at any given time.

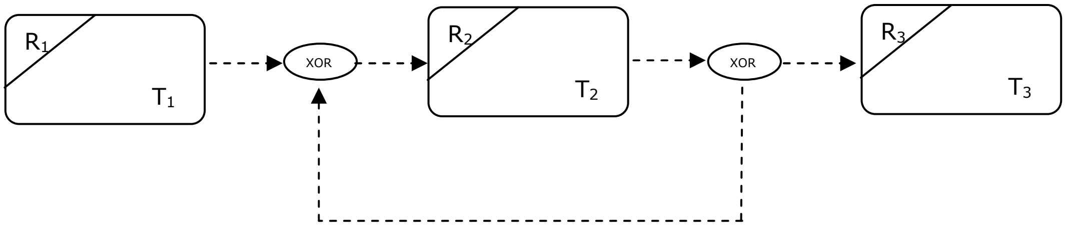


Figure 6.3b. A Sequential Process Model with Feedback

Additional Assumptions

- Without any loss of generality we can assume the activity T_j requires resource $R_{j\,,}$ j = 1, 2, …, t.

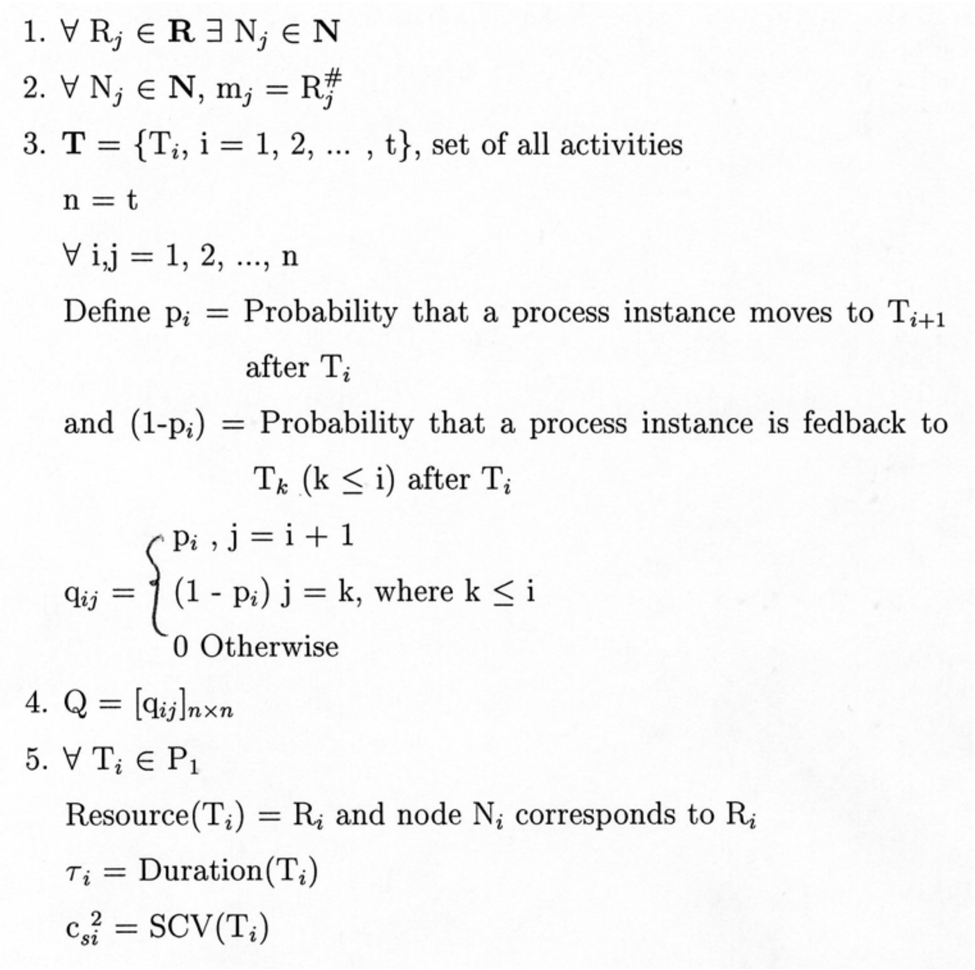
Mappings

- For every resource R_j in resource set **R,** there exists a corresponding node N_j in the queueing network model. The number of nodes in the network, n = |R| = t.
- The number of servers at node $N_{j,}$ m_j is equal to $R_j^{\#}$, the number of available units of resource $R_{j.}$
- Because of feedback, one or more nodes can be visited a random number of times. Hence, it is not possible to specify a deterministic routing in this case. The routing probabilities are specified in the routing matrix.
- The routing matrix Q = [q_{ij}], is given by where

$$q_{ij} = \begin{cases} p_i \text{ , } j = i + 1 \\ (1 - p_i) \text{ } j = k, \text{ where } k \leq i \\ 0 \text{ Otherwise} \end{cases}$$

- In the example given in Figure 6.3b, $q_{12} = 1$, $q_{23} = p_2$, $q_{22} = (1 - p_2)$.
- The service time parameters of node N_j correspond to the activity duration parameters of activity T_j that requires resource R_j. The mean service time τ_j of node N_j is the mean duration of T_j that requires R_j.
- The external arrival rate parameters λ_1 and c_1^2 are obtained from the process initiation logic.

Formally the mappings can be summarized in the following steps:



1. $\forall$ $R_j \in \mathbf{R} \; \exists \; N_j \in \mathbf{N}$
2. $\forall$ $N_j \in \mathbf{N}$, $m_j = R_j^{\#}$
3. $\mathbf{T} = \{T_i, \text{ i} = 1, 2, \ldots, t\}$, set of all activities

   $n = t$

   $\forall$ i,j = 1, 2, ..., n

   Define p_i = Probability that a process instance moves to T_{i+1} after T_i

   and (1-p_i) = Probability that a process instance is fedback to T_k ($k \leq i$) after T_i

   $$q_{ij} = \begin{cases} p_i \text{ , } j = i + 1 \\ (1 \text{ - } p_i) \text{ } j = k, \text{ where } k \leq i \\ 0 \text{ Otherwise} \end{cases}$$

4. $Q = [q_{ij}]_{n \times n}$
5. $\forall$ $T_i \in P_1$

   Resource(T_i) = R_i and node N_i corresponds to R_i

   τ_i = Duration(T_i)

   c_{si}^2 = SCV(T_i)

T*ype 2: A sequential model with a series of activities, where some of the activities share (common) resources.*

Model

The pictorial representation of a sequential process model where some of the activities share resources is shown in Figure 6.4. The resource R_1 is shared by activities T_1 and T_3 in the model represented in Figure 6.4. Note that there could be multiple instances of the process depicted that are active at any given time.

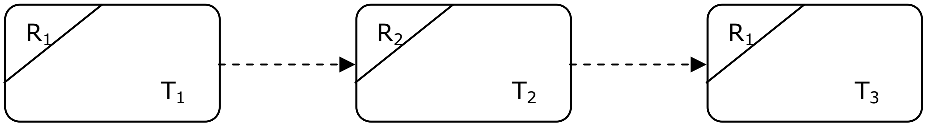


Figure 6.4. A Sequential Process Model (with Resource Sharing)

Additional Assumptions

- There is no feedback. A process instance executes an activity only once.

Mappings

- The queueing network model consists of a single class of customers.
- For every resource R_j in resource set **R,** there exists a corresponding node N_j in the queueing network model. The number of nodes in the network, n = |R|.
- The number of servers at node $N_{j,}$ m_j is equal to $R_j^{\#}$, the number of available units of resource $R_{j.}$
- Because of the sequential nature, there exists only one path P_1 specified by the control flow in the process model. This would correspond to one route, r_1 in the queueing network.

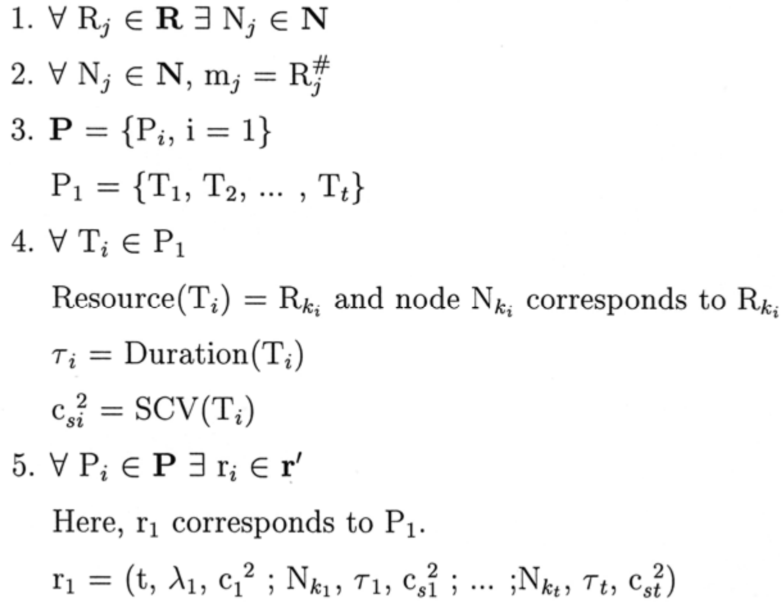
- Where $r_1 = \{n_1, \lambda_1, c_1^2; N_{11}, \tau_{11,} c_{s11}^2; N_{12}, \tau_{12,} c_{s12}^2; ...; N_{1n_1}, \tau_{1n_1}, c_{s1n_1}^2\}$

  The number of operations n_1 in the route r_1 will be equal to t, the total number of activities. Note that t > n because of resource sharing.

- The service time parameters of node N_j will have to be computed as part of network solution procedure and is not part of the network model specification.

- The external arrival rate parameters λ_1 and c_1^2 are obtained from the process initiation logic.

Formally the mappings can be summarized in the following steps:



1. $\forall\, \mathrm{R}_j \in \mathbf{R}\ \exists\, \mathrm{N}_j \in \mathbf{N}$
2. $\forall\, \mathrm{N}_j \in \mathbf{N}$, $\mathrm{m}_j = \mathrm{R}_j^{\#}$
3. $\mathbf{P} = \{\mathrm{P}_i,\ \mathrm{i} = 1\}$

   $\mathrm{P}_1 = \{\mathrm{T}_1, \mathrm{T}_2, \ldots, \mathrm{T}_t\}$
4. $\forall\, \mathrm{T}_i \in \mathrm{P}_1$

   $\mathrm{Resource}(\mathrm{T}_i) = \mathrm{R}_{k_i}$ and node N_{k_i} corresponds to R_{k_i}

   $\tau_i = \mathrm{Duration}(\mathrm{T}_i)$

   $\mathrm{c}_{si}^2 = \mathrm{SCV}(\mathrm{T}_i)$
5. $\forall\, \mathrm{P}_i \in \mathbf{P}\ \exists\, \mathrm{r}_i \in \mathbf{r'}$

   Here, r_1 corresponds to P_1.

   $\mathrm{r}_1 = (\mathrm{t},\ \lambda_1,\ \mathrm{c}_1{}^2\ ;\ \mathrm{N}_{k_1},\ \tau_1,\ \mathrm{c}_{s1}^2\ ;\ \ldots\ ;\mathrm{N}_{k_t},\ \tau_t,\ \mathrm{c}_{st}^2)$

Type 3: A model with a choice of paths. Each path is a sequential model with a series of activities where each activity requires a different resource.

Model

The pictorial representation of a process model with choice is shown in Figure 6.5. The choice is between the path $P_1 = \{T_1, T_2, T_3, T_4\}$ and path $P_2 = \{T_1, T_2, T_5, T_6\}$ in the model represented in Figure 6.5. Note that there could be multiple instances of the process depicted that are active at any given time.

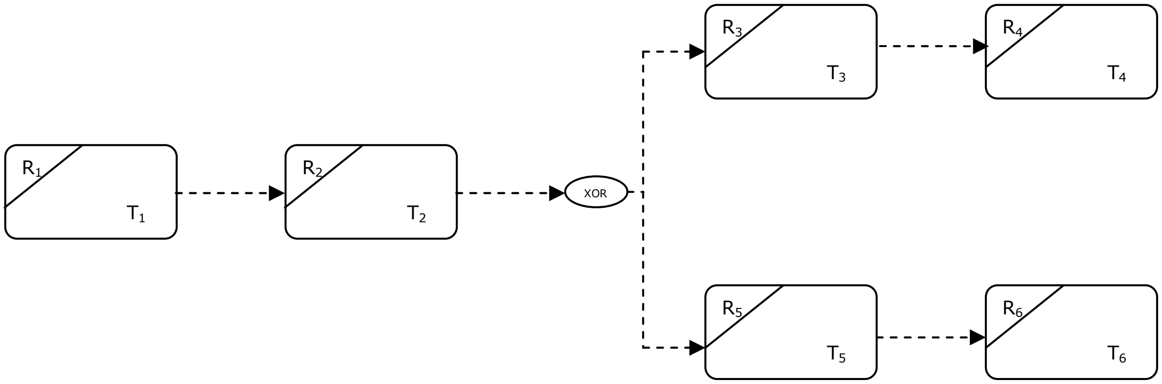


Figure 6.5. A Process Model with Choice (No Resource Sharing)

Additional Assumptions

- Without any loss of generality it can assumed that the activity T_j requires resource $R_{j\,,}$ j = 1, 2, …, t.

Mappings

- For every resource R_j in resource set **R,** there exists a corresponding node N_j in the queueing network model. The number of nodes in the network, n = |R|.
- The model is not sequential, hence there exists a set of paths **P** (defined earlier), specified by the control flow in the process model. The set of paths **P** defines a set of routes **r'** = $\{r_k$, k = 1, 2, 3, …., $n_r\}$ in the queueing network

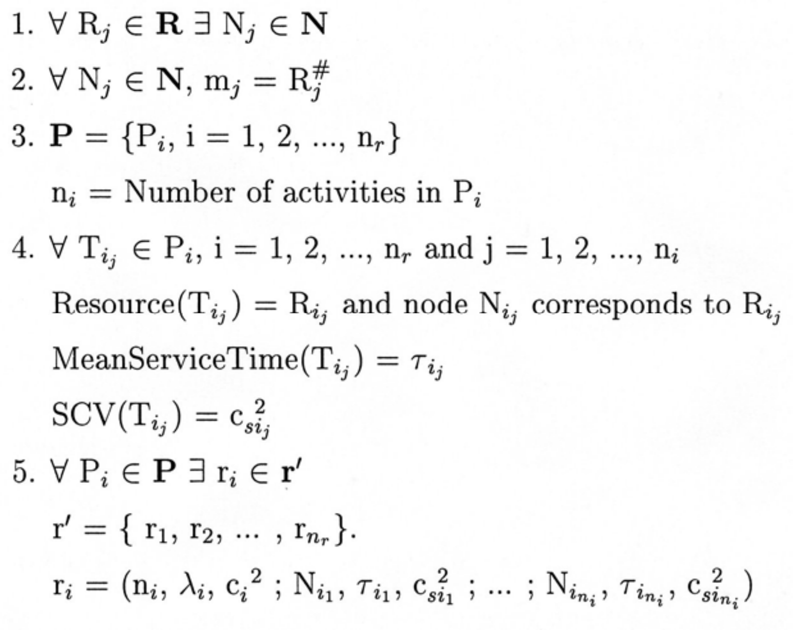
model, where n_r, the number of routes in the queueing network model is the number of paths in **P**.

- Also for each path $P_i \in$ **P,** there exists a $r_i \in$ **r'**. All valid paths can be obtained using an enumeration algorithm (refer [17] for example).

On each route r_k (where k = 1, 2, 3, …., n_r)

- The number of nodes is equal to the number of activities in path $P_{k.}$
- The j^{th} node visited is the node corresponding to the resource required by the j^{th} activity in path P_k
- The service time parameters of node visit j on route r_k corresponds to the activity duration parameters of the j^{th} activity of path P_k. For example, the mean service time of 3^{rd} node on path 2 is the duration of T_5 that requires resource R_5.

Formally the mappings can be summarized in the following steps:



1. $\forall\ \mathrm{R}_j \in \mathbf{R}\ \exists\ \mathrm{N}_j \in \mathbf{N}$
2. $\forall\ \mathrm{N}_j \in \mathbf{N},\ \mathrm{m}_j = \mathrm{R}_j^{\#}$
3. $\mathbf{P} = \{\mathrm{P}_i,\ \mathrm{i} = 1,\ 2,\ ...,\ \mathrm{n}_r\}$

   n_i = Number of activities in P_i
4. $\forall\ \mathrm{T}_{i_j} \in \mathrm{P}_i,\ \mathrm{i} = 1,\ 2,\ ...,\ \mathrm{n}_r$ and $\mathrm{j} = 1,\ 2,\ ...,\ \mathrm{n}_i$

   $\mathrm{Resource}(\mathrm{T}_{i_j}) = \mathrm{R}_{i_j}$ and node N_{i_j} corresponds to R_{i_j}

   $\mathrm{MeanServiceTime}(\mathrm{T}_{i_j}) = \tau_{i_j}$

   $\mathrm{SCV}(\mathrm{T}_{i_j}) = \mathrm{c}^2_{si_j}$
5. $\forall\ \mathrm{P}_i \in \mathbf{P}\ \exists\ \mathrm{r}_i \in \mathbf{r}'$

   $\mathrm{r}' = \{\ \mathrm{r}_1,\ \mathrm{r}_2,\ \dots\ ,\ \mathrm{r}_{n_r}\}.$

   $\mathrm{r}_i = (\mathrm{n}_i,\ \lambda_i,\ \mathrm{c}_i{}^2\ ;\ \mathrm{N}_{i_1},\ \tau_{i_1},\ \mathrm{c}^2_{si_1}\ ;\ \dots\ ;\ \mathrm{N}_{in_i},\ \tau_{in_i},\ \mathrm{c}^2_{sin_i})$

Type 4: A model with choice of paths. Each path is a sequential model with a series of activities where some of the activities share (common) resources.

Model

The pictorial representation of a process model with choice and some of the activities that share resources is shown in Figure 6.5. In the model represented in Figure 6.6, the choice is between the path $P_1 = \{T_1, T_2, T_3, T_4\}$ and path $P_2 = \{T_1, T_2, T_5, T_6\}$. Also resource R_2 is shared by activities T_2 and T_6, and the resource R_4 is shared by activities T_4 and T_5. Note that there could be multiple instances of the process depicted that are active at any given time.

Additional Assumptions

- A process instance executes an activity only once.

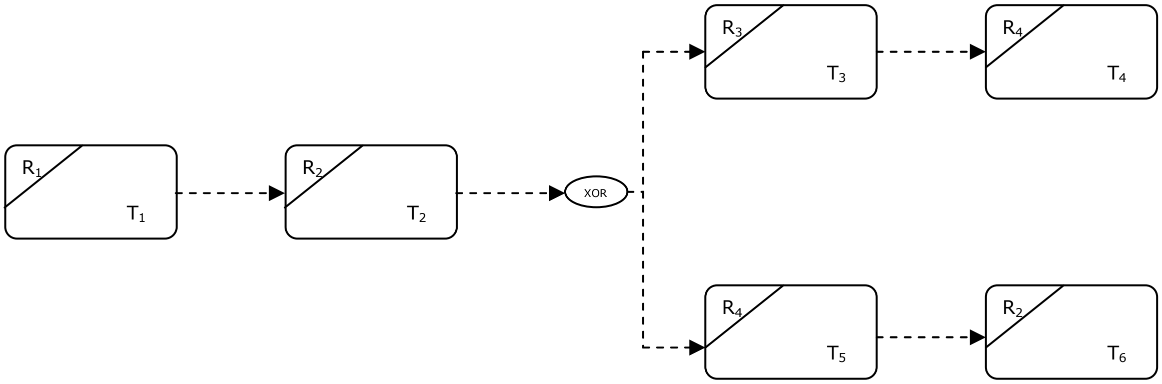


Figure 6.6. A Process Model with Choice and Resource Sharing

Mappings (same as in type 3)

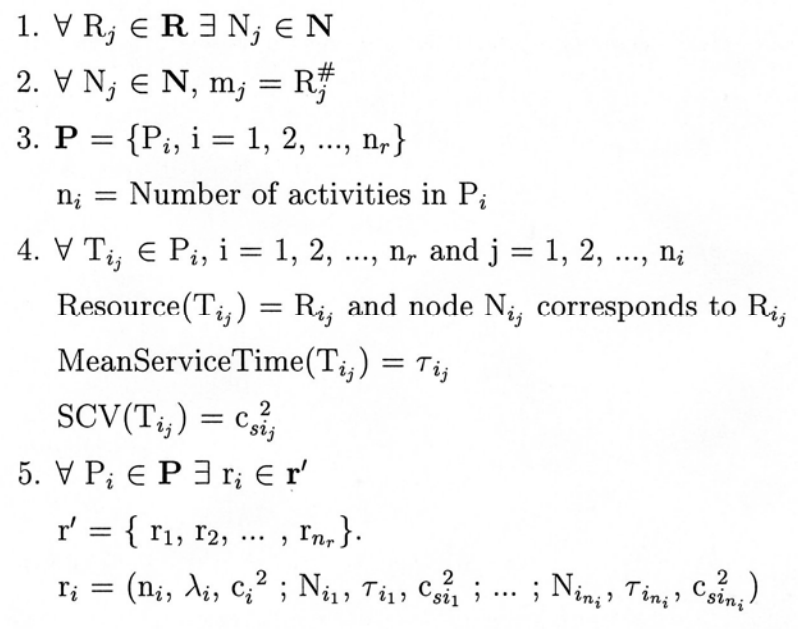
- For every resource R_j in resource set **R,** there exists a corresponding node N_j in the queueing network model. The number of nodes in the network, n = |R|.

- The model is not sequential, hence there exists a set of paths **P** (defined earlier), specified by the control flow in the process model. The set of paths **P** defines a set of routes **r'** = $\{r_k, k = 1, 2, 3, ..., n_r\}$ in the queueing network model, where n_r, the number of routes in the queueing network model is the number of paths in **P**.
- Also for each path $P_i \in$ **P,** there exists a $r_i \in$ **r'**. All valid paths can be obtained using an enumeration algorithm (refer [17] for example).

On each route r_k (where k = 1, 2, 3, …., n_r)

- The number of nodes is equal to the number of activities in path $P_{k.}$
- The j^{th} node visited is the node corresponding to the resource required by the j^{th} activity in path P_k
- The service time parameters of node visit j on route r_k corresponds to the activity duration parameters of the j^{th} activity of path P_k. For example, the mean service time of 3^{rd} node on path 2 is the duration of T_5 that requires resource $R_{5.}$
- The external arrival rate parameters λ_k and c_k^2 are obtained from the process initiation logic.

Formally the mappings can be summarized in the following steps:



1. $\forall\ \mathrm{R}_j \in \mathbf{R}\ \exists\ \mathrm{N}_j \in \mathbf{N}$
2. $\forall\ \mathrm{N}_j \in \mathbf{N}$, $\mathrm{m}_j = \mathrm{R}_j^{\#}$
3. $\mathbf{P} = \{\mathrm{P}_i,\ \mathrm{i} = 1, 2, ..., \mathrm{n}_r\}$

   n_i = Number of activities in P_i
4. $\forall\ \mathrm{T}_{i_j} \in \mathrm{P}_i$, $\mathrm{i} = 1, 2, ..., \mathrm{n}_r$ and $\mathrm{j} = 1, 2, ..., \mathrm{n}_i$

   $\mathrm{Resource}(\mathrm{T}_{i_j}) = \mathrm{R}_{i_j}$ and node N_{i_j} corresponds to R_{i_j}

   $\mathrm{MeanServiceTime}(\mathrm{T}_{i_j}) = \tau_{i_j}$

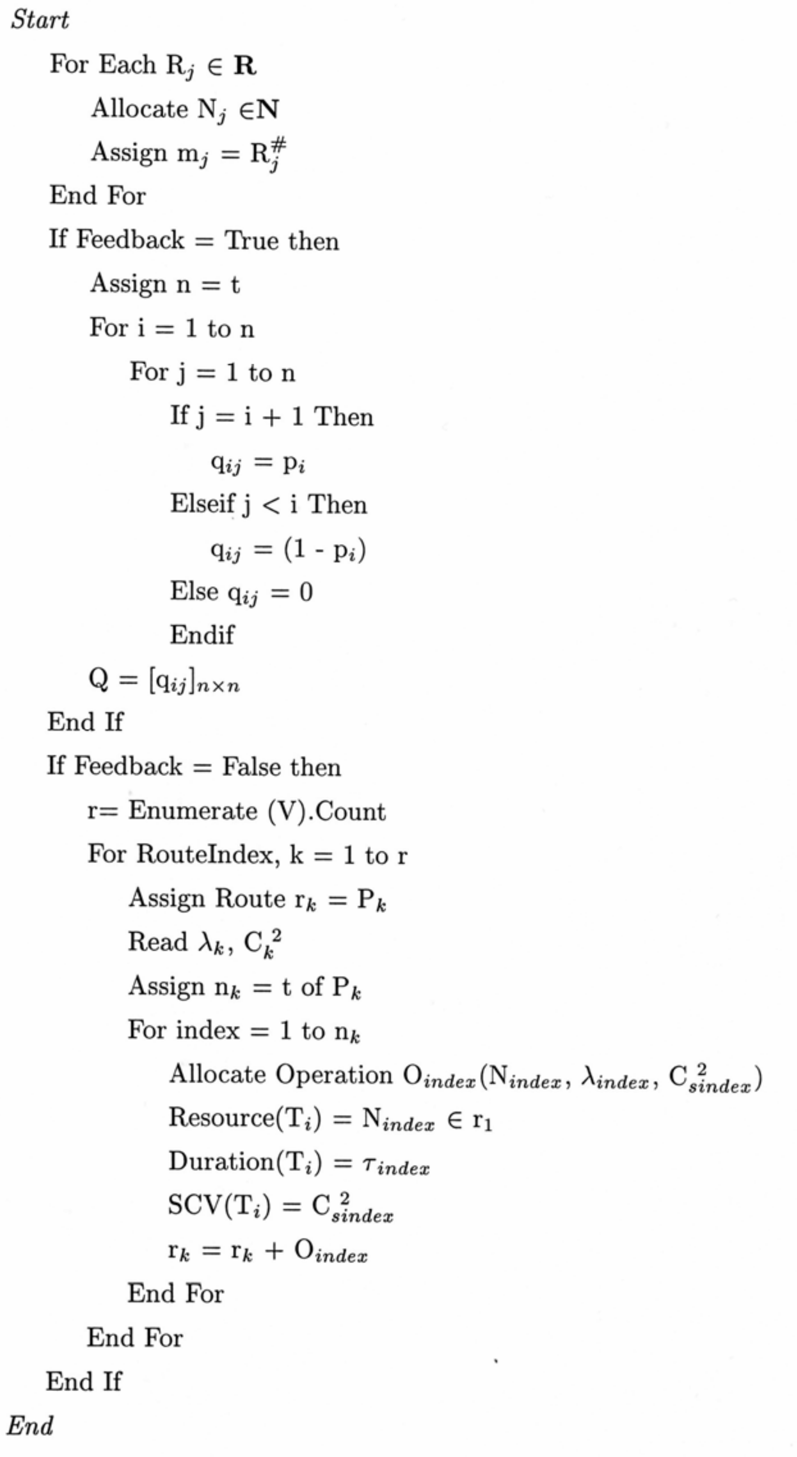
   $\mathrm{SCV}(\mathrm{T}_{i_j}) = \mathrm{c}_{si_j}^2$
5. $\forall\ \mathrm{P}_i \in \mathbf{P}\ \exists\ \mathrm{r}_i \in \mathbf{r}'$

   $\mathrm{r}' = \{\ \mathrm{r}_1, \mathrm{r}_2, \ldots\ , \mathrm{r}_{n_r}\}.$

   $\mathrm{r}_i = (\mathrm{n}_i,\ \lambda_i,\ \mathrm{c}_i{}^2\ ;\ \mathrm{N}_{i_1},\ \tau_{i_1},\ \mathrm{c}_{si_1}^2\ ;\ \ldots\ ;\ \mathrm{N}_{in_i},\ \tau_{in_i},\ \mathrm{c}_{sin_i}^2)$

6.3 Transformation Algorithm

Following algorithm was developed to automatically configure a queueing network model from a business process description, based on the mappings from Section 6.2.



```
Start
    For Each $R_j \in \mathbf{R}$
        Allocate $N_j \in \mathbf{N}$
        Assign $m_j = R_j^{\#}$
    End For
    If Feedback = True then
        Assign n = t
        For i = 1 to n
            For j = 1 to n
                If j = i + 1 Then
                    $q_{ij} = p_i$
                Elseif j < i Then
                    $q_{ij} = (1 - p_i)$
                Else $q_{ij} = 0$
                Endif
        $Q = [q_{ij}]_{n \times n}$
    End If
    If Feedback = False then
        r= Enumerate (V).Count
        For RouteIndex, k = 1 to r
            Assign Route $r_k = P_k$
            Read $\lambda_k$, $C_k^2$
            Assign $n_k$ = t of $P_k$
            For index = 1 to $n_k$
                Allocate Operation $O_{index}(N_{index}, \lambda_{index}, C_{sindex}^2)$
                $Resource(T_i) = N_{index} \in r_1$
                $Duration(T_i) = \tau_{index}$
                $SCV(T_i) = C_{sindex}^2$
                $r_k = r_k + O_{index}$
            End For
        End For
    End If
End
```

6.4 Implementation

As a proof-of-concept implementation for the algorithm presented in Section 6.3, a prototype function was programmed using the VB.NET language. The function that was developed is DML2QNML. This function takes a DML file as input and generates a corresponding QNML file. On successful generation the function returns a boolean true, otherwise it return a boolean false. The prototype of the function is:

Function DML2QNML (DML As XMLDocument) As Boolean

The code for this function is presented in Appendix C. This function is implemented as part of a simple Windows based environment, which would allow the user to specify the required inputs. The screenshots for this environment are provided in Appendix E.

Chapter 7

A Multi-Step Approach for Automatic Configuration of a Queueing Network Model

This chapter presents an overview of the technique developed and the mapping scheme required for the multi-step approach (described in Section 3.3) to automatically configure a queueing network model from a Petri net representation of a business process. Section 7.1 presents the definitions, formal notations and representations for Petri nets. The mappings that are required to translate a Petri net based business process representation, which is described using the Petri Net Markup Language (PNML), to a queueing network model described by Queueing Network Markup Language (QNML), are identified in Section 7.2. Section 7.3 presents an algorithm that automates the translation of PNML to QNML. Section 7.4 presents the implementation details of the transformation.

7.1 Introduction and Background

This approach is basically a two-step approach, wherein a business process model is first transformed to a Petri net representation which is then transformed to a queueing network model for analysis. The transformation is between the computer processable formats of the corresponding models. More specifically, a business process model,

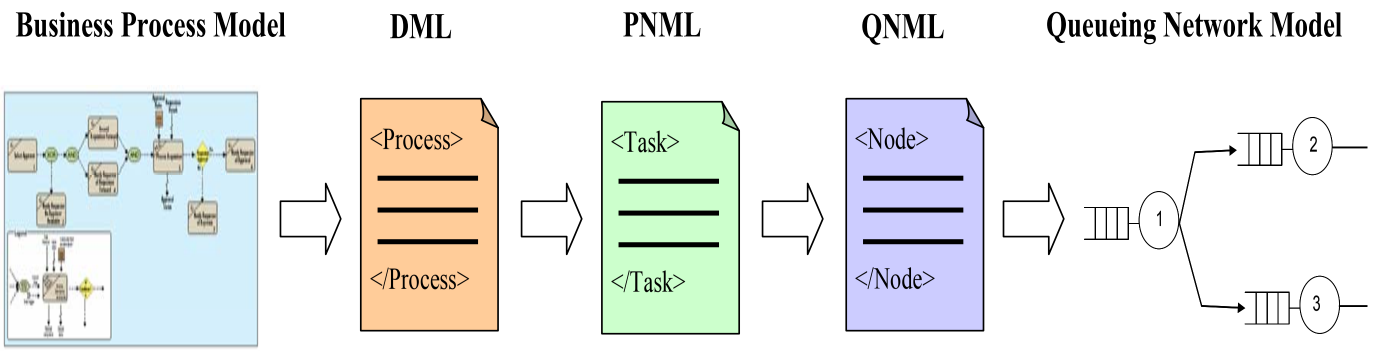


Figure 7.1. Pictorial Representation of the Multi-Step Approach

represented by a markup language like DML is transformed to a Petri-net model, represented by a mark up language like PNML. The next step is to transform the Petri-net model, represented by a markup language like PNML to a queueing network model, represented by a mark up language like QNML. To achieve the transformation it is important to identify the mappings between the various elements that make up each of the models. Section 7.1.1 presents the formal notations required to define a Petri net based process model. The formal notation defined in Section 6.1.2 is used for specification of the queueing network model. Section 7.2 uses the formal notation of Sections 7.1.1 and 6.1.2 to define the mappings between the Petri-net based process model and a queueing network model. Figure 7.1 shows these ideas pictorially.

Notation and Definitions

Petri nets

Petri nets or place-transition nets are classical models of concurrency, non determinism, and control flow, first proposed in 1962 by Carl Adam Petri. Petri nets provide an elegant and mathematically rigorous modeling framework for discrete event dynamical systems. In this section an overview of Petri nets is presented with the aid of several definitions [20]. For a detailed treatment of Petri nets the reader is referred to [20].

Definition: A Petri net is a four-tuple (P, T, IN, OUT) where

$P = \{p_1, p_2, p_3, \ldots, p_n\}$ is a set of places

$T = \{t_1, t_2, t_3, \ldots, t_n\}$ is a set of transitions

$P \cup T \neq \Phi, P \cap T = \Phi$

IN: (P **X** T) → N is an input function that defines directed arcs from places to transitions, and OUT: (P **X** T) → N is an output function that defines directed arcs from transitions to places.

Pictorially, places are represented by circles and transitions by horizontal or vertical bars. If IN $(p_i, t_j) = k$, where $k > 1$ is an integer, a directed arc from place p_i to transition t_j is drawn with a label k. If IN $(p_i, t_j) = 1$, we include an unlabeled directed arc. If IN $(p_i, t_j) = 0$ then no arc is drawn from p_i to $t_{j.}$ Similarly OUT (p_i, t_j) results in a directed arc from transition t_j to place p_i if OUT $(p_i, t_j) > 0$.

Places of Petri nets usually represent conditions or resources in the system while transitions model the activities in the system.

Definition: The set of input places of transition t_j , denoted by $IP(t_j)$, and the set of output places of $t_{j,}$ denoted by $OP(t_j)$ are defined by

$$IP(t_j) = \{p_i \in P: IN(p_i, t_j) \neq 0\}$$

$$OP(t_j) = \{p_i \in P: OUT(p_i, t_j) \neq 0\}$$

Definition: A marking M of a Petri net is a function M: P → N. A marked Petri net is a Petri net with an associated marking. If $M(p_i) = m_i > 0$ then the marking is represented by m_i black dots inside place p_i.

Definition: A transition t_j of a Petri net is said to be *enabled* in a marking M if

$$M(p_i) \geq IN(p_i, t_j) \quad \forall \; p_i \in IP(t_j)$$

An enabled transition t_j can *fire* at any time. When a transition t_j enabled in a marking M fires, a new marking M' is reached according to the equation

$$M'(p_i) = M(p_i) + OUT(p_i, t_j) - IN(p_i, t_j) \quad \forall \; p_i \in P$$

Stochastic Petri nets

Classical Petri nets are useful in investigating qualitative or logical properties of concurrent systems. However, for quantitative performance evaluation, the concept of time needs to be added to the definition of the Petri nets. Time is associated with transitions, indicating that they can fire some time after they become enabled. The association of deterministic time led to the development of timed Petri nets. However to associate random time durations with the firing of transitions, Stochastic Petri nets (SPNs) are used [20].

Definition: A SPN is a sex-tuple (P, T, IN, OUT, M_0, F) where (P, T, IN, OUT, M_0) is a marked Petri net and F is a function with domain ($R[M_0]$ **X** T), which associates with each transition in each reachable marking, a random variable. The function F is called the firing function and the random variable F (M, t) for $M \in R[M_0]$ and $t \in T$ as the firing time of transition t in the marking M.

The following additional definitions are needed.

Definition: A function **Resource (t_i) =** $\{R_{ki} \subset IP(t_i)\}$, returns the set of resource(s) required by transition t_i.

Definition: Let PN be a Petri net. A function **Enumerate (PN)** returns the set of all transition firing sequences that lead to process termination.

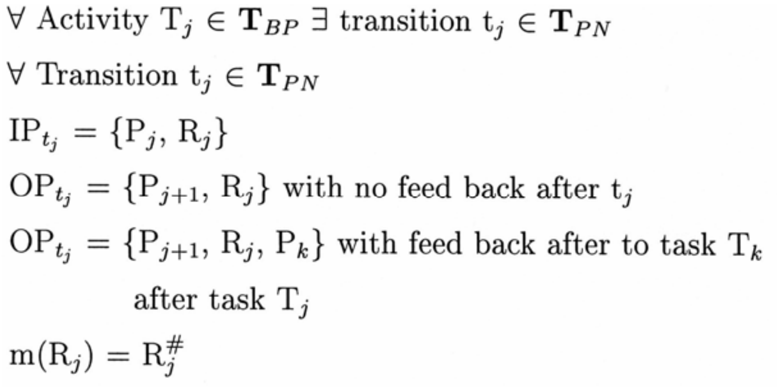
$$\text{Enumerate (PN)} = TSF = \{TSF_{i,}\ i = 1, 2, \ldots, n_r\}$$

$$\mathrm{TSF}_i = \{\mathrm{t}_{i_1}, \mathrm{t}_{i_2}, \ldots , \mathrm{t}_{i_{n_i}}\}$$

7.2 Mapping Schemes

Petri nets are process-centric views of business process models. The correspondence between the business process model elements and the Petri nets has been identified within the DIME framework. The Figure 7.1 shows this mapping between business process model elements and Petri nets. An extension to this mapping would be to include resource requirements for an activity and correspondingly for a task in Petri nets. A resource required by an activity in a business process model would transform into an input place for the task corresponding to the activity. Also the number of units of a resource available would translate to the number of tokens in the input place.

Formally,



$$\begin{aligned}
&\forall \text{ Activity } \mathrm{T}_j \in \mathbf{T}_{BP} \; \exists \text{ transition } \mathrm{t}_j \in \mathbf{T}_{PN}\\
&\forall \text{ Transition } \mathrm{t}_j \in \mathbf{T}_{PN}\\
&\mathrm{IP}_{t_j} = \{\mathrm{P}_j, \mathrm{R}_j\}\\
&\mathrm{OP}_{t_j} = \{\mathrm{P}_{j+1}, \mathrm{R}_j\} \text{ with no feed back after } \mathrm{t}_j\\
&\mathrm{OP}_{t_j} = \{\mathrm{P}_{j+1}, \mathrm{R}_j, \mathrm{P}_k\} \text{ with feed back after to task } \mathrm{T}_k\\
&\qquad\quad \text{after task } \mathrm{T}_j\\
&\mathrm{m}(\mathrm{R}_j) = \mathrm{R}_j^{\#}
\end{aligned}$$

where BP stands for business process model and PN stands for Petri net model.

With this background, the transformations between Petri nets and queueing models can be achieved. To do so, the types of process models defined in Section 6.2 are used here also. However, only models of types 1b and 4 models from Section 6.2 are discussed in this section. Models of types 1a, 2 and 3 are special cases of type 4. To arrive at a generalized transformation scheme it is enough to identify the mapping schemes for types 1b and 4.

Notation	Description
t_i	Transition i
P_i	Place i
R_i	Resource i

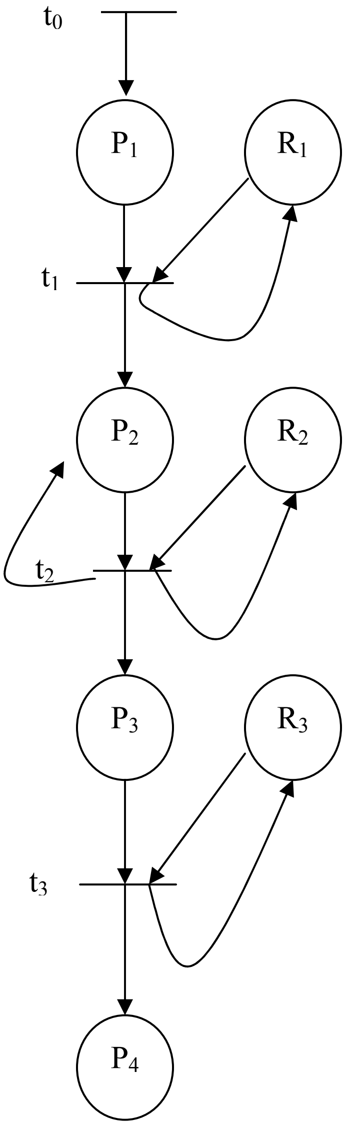


Figure 7.2: Petri net Representation of a Process Model with Feedback

(*Refer to Figure 6.1b for the corresponding process model)*

Type 1b: A sequential model with a series of activities where each task requires a different resource. Feedback allowed.

Model

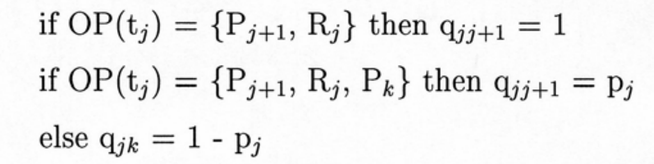
The pictorial representation of the business process model is shown in Figure 6.3b in Section 6.2 and the corresponding Petri net model is shown here in Figure 7.2. Note that there could be multiple instances of the process depicted that are active at any given time.

Additional Assumptions

- Without any loss of generality we can assume the activity T_j requires resource $R_{j\,,}$ j = 1, 2, …, t.

Mappings

- For every resource $R_j \in$ IP (t_j)**,** there exists a corresponding node N_j in the queueing network model. The number of nodes in the network, n = |IP| = t.
- The number of servers at node $N_{j,}$ m_j is equal to M_0 (R_j), the number of available units of resource R_j in the initial marking.
- Because of feedback, one or more nodes can be visited a random number of times. Hence, it is not possible to specify a deterministic routing in this case. The routing probabilities are specified in the routing matrix.
- The routing matrix Q = [q_{ij}], is given by

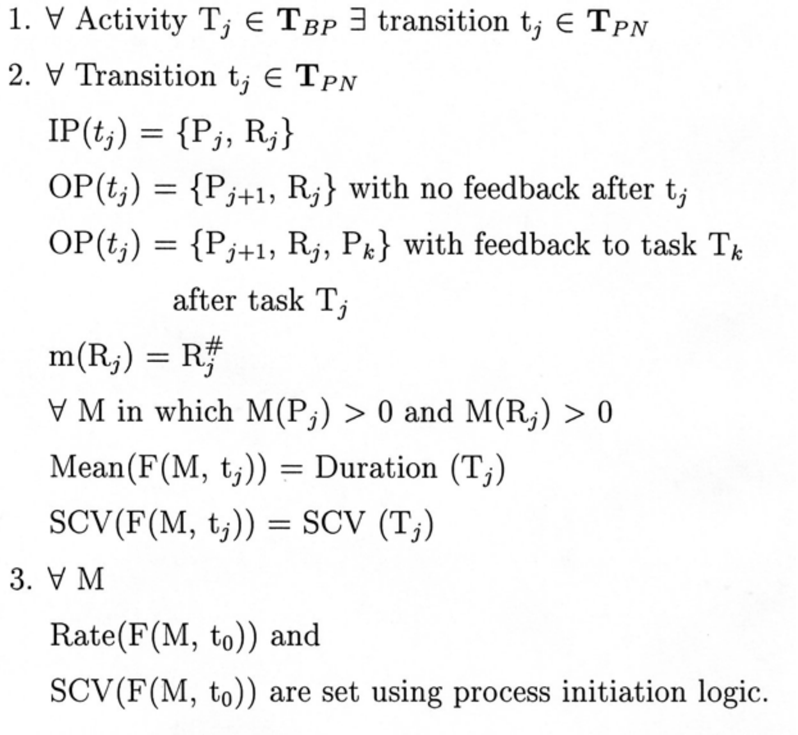


if $\mathrm{OP}(\mathrm{t}_j) = \{\mathrm{P}_{j+1}, \mathrm{R}_j\}$ then $\mathrm{q}_{jj+1} = 1$
if $\mathrm{OP}(\mathrm{t}_j) = \{\mathrm{P}_{j+1}, \mathrm{R}_j, \mathrm{P}_k\}$ then $\mathrm{q}_{jj+1} = \mathrm{p}_j$
else $\mathrm{q}_{jk} = 1$ - p_j

- The service time parameters of node N_j correspond to the mean firing rate of transition t_j that requires resource R_j, denoted by Mean (F(M, t_j)).

- The external arrival rate parameters λ_{01} and c_{01}^2 are obtained by the functions Rate (F (M, t_0)) and SCV (F (M, t_0)) respectively.

For completeness, the mappings to translate a business process model to a Petri net model are presented here.



1. $\forall$ Activity $\mathrm{T}_j \in \mathbf{T}_{BP}$ $\exists$ transition $\mathrm{t}_j \in \mathbf{T}_{PN}$
2. $\forall$ Transition $\mathrm{t}_j \in \mathbf{T}_{PN}$

   $\mathrm{IP}(t_j) = \{\mathrm{P}_j, \mathrm{R}_j\}$

   $\mathrm{OP}(t_j) = \{\mathrm{P}_{j+1}, \mathrm{R}_j\}$ with no feedback after t_j

   $\mathrm{OP}(t_j) = \{\mathrm{P}_{j+1}, \mathrm{R}_j, \mathrm{P}_k\}$ with feedback to task T_k
   after task T_j

   $\mathrm{m}(\mathrm{R}_j) = \mathrm{R}_j^{\#}$

   $\forall$ M in which $\mathrm{M}(\mathrm{P}_j) > 0$ and $\mathrm{M}(\mathrm{R}_j) > 0$

   $\mathrm{Mean}(\mathrm{F}(\mathrm{M}, \mathrm{t}_j))$ = Duration (T_j)

   $\mathrm{SCV}(\mathrm{F}(\mathrm{M}, \mathrm{t}_j))$ = SCV (T_j)
3. $\forall$ M

   $\mathrm{Rate}(\mathrm{F}(\mathrm{M}, \mathrm{t}_0))$ and

   $\mathrm{SCV}(\mathrm{F}(\mathrm{M}, \mathrm{t}_0))$ are set using process initiation logic.

Formally the mappings to translate a Petri net to a queueing network model can be summarized in the following steps

1. $\forall$ $R_j \in \mathbf{IP(t}_j\mathbf{)}$ $\exists$ node N_j in the queueing network model.

   $m_j = M_0(R_j)$

2. For j = 1, 2, ..., t-1

   if $OP(t_j) = \{P_{j+1}, R_j\}$ then $q_{jj+1} = 1$

   if $OP(t_j) = \{P_{j+1}, R_j, P_k\}$ then $q_{jj+1} = p_j$

   else $q_{jk} = 1 - p_j$

   Routing matrix $Q = [q_{ij}]_{n \times n}$

3. Rate and SCV of external arrivals at Node 1

   are given by Rate(F(M, t_0)) and SCV(F(M, t_0))

   $\lambda_{01} = \text{Rate}(F(M, t_0))$

   $c_{01}^2 = \text{SCV}(F(M, t_0))$

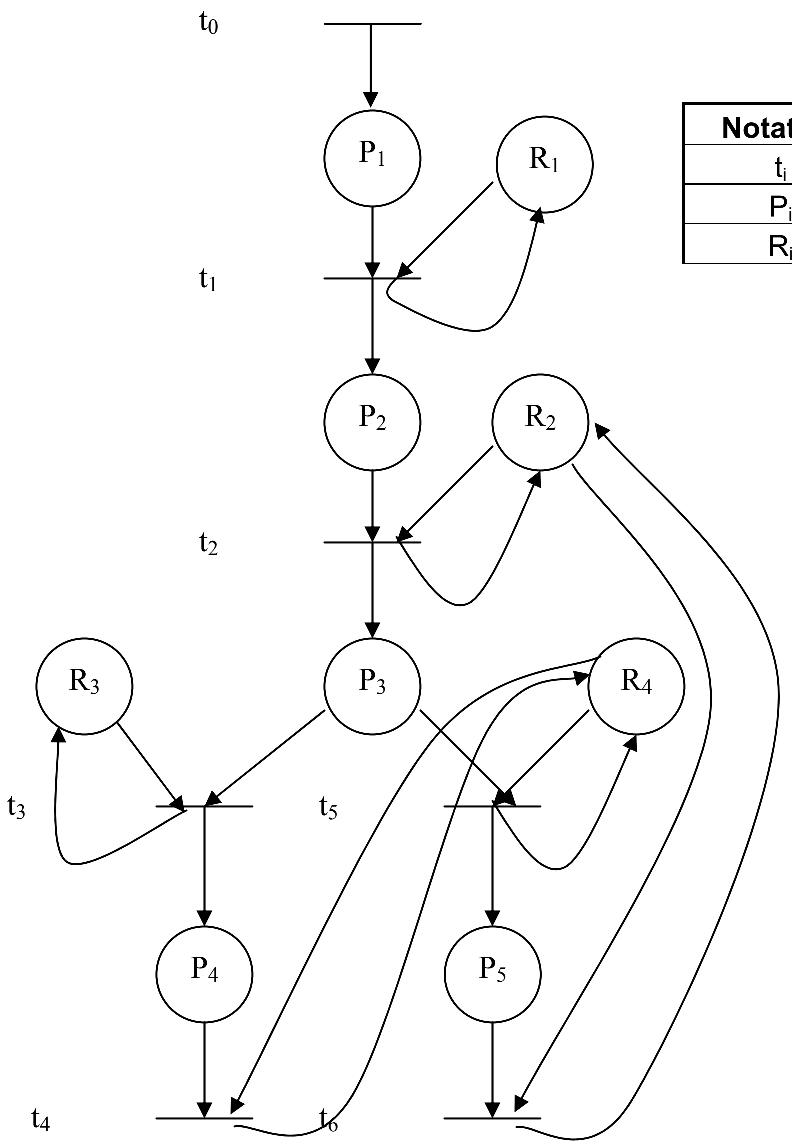
4. $\forall$ $t_j \in T_{PN}$

   Resource(t_j) = R_j and node N_j corresponds to Resource R_j.

   $\tau_j = \text{Mean}(F(M, t_j))$

   $c_{s_j}^2 = \text{SCV}(F(M, t_j))$

   where $M \in R[M_0]$, $M(P_j) > 0$, $M(R_j) > 0$



Notation	Description
t_i	Transition i
P_i	Place i
R_i	Resource i

Figure 7.3: Petri net Representation of a Process Model with Choice

(Refer to Figure 6.6 for the corresponding process model)

Type 4: A model with a choice of paths. Each path is a sequential model with a series of activities where some of the activities share (common) resources.

Model

The pictorial representation of a business process model with choice and the corresponding Petri net is shown in Figure 7.3. Note that there could be multiple instances of the process depicted that are active at any given time.

Additional Assumptions

- There is no feedback. An entity flows through an activity only once.

Mappings

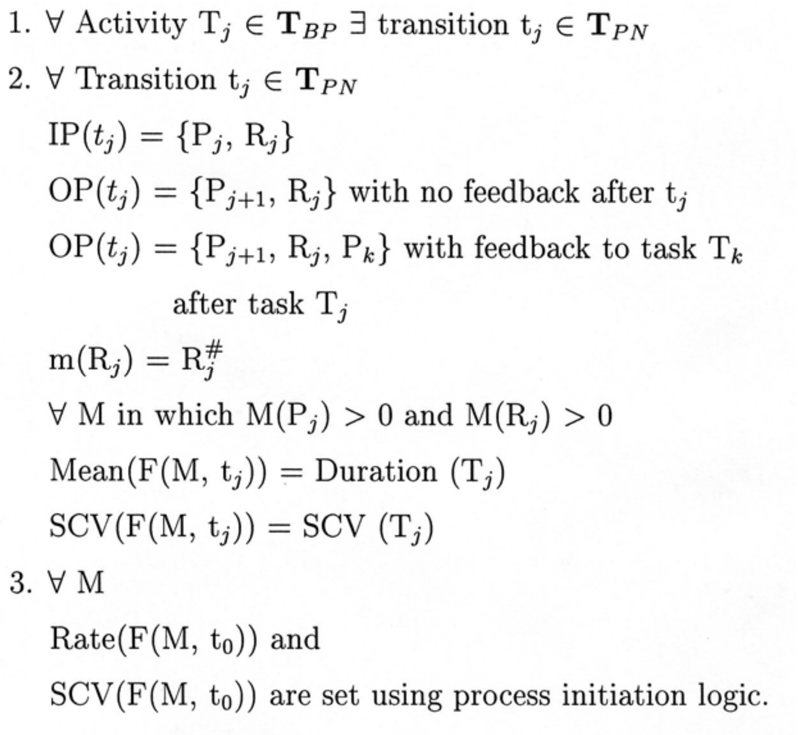
- For every resource $R_j \in IP(t_j)$**,** there exists a corresponding node Nk_j in the queueing network model. The number of nodes in the network, n = |IP| = t.
- The number of servers at node $N_{j,}$ m_j is equal to M_0 (R_j), the number of available units of resource R_j in the initial marking.
- The model is not sequential, hence there exists a set of transition firing sequences (defined earlier), specified by the control flow in the process model. The set of transition firing sequences **TSF** corresponds to the set of routes **r'** = {r_k , k = 1, 2, 3, …., n_r} in the queueing network model, where n_r, the number of routes in the queueing network model is the number of sequences in **TSF**.
- Also for each path $TSF_i \in$ **TSF,** there exists a $r_i \in$ **r'**. All valid firing sequences can be obtained using an enumeration algorithm (defined earlier).

On each route r_k (where k = 1, 2, 3, …., n_r)

- The number of nodes is equal to the number of transitions in transition firing sequence $TSF_{k.}$

- The j^{th} node visited is the node corresponding to the resource required by the j^{th} transition in transition firing sequence $TSF_{k.}$
- The service time parameters of node visit j on route r_k correspond to the mean rate of firing of the j^{th} transition in path TSF_k.
- The external arrival rate parameters λ_{01} and c_{01}^2 are obtained by the functions Rate (F (M, t_0)) and SCV (F (M, t_0)) respectively.

For completeness, the mappings to translate a Business process model to a Petri net model are presented here.



1. $\forall$ Activity $T_j \in \mathbf{T}_{BP} \ \exists$ transition $t_j \in \mathbf{T}_{PN}$
2. $\forall$ Transition $t_j \in \mathbf{T}_{PN}$

   $IP(t_j) = \{P_j, R_j\}$

   $OP(t_j) = \{P_{j+1}, R_j\}$ with no feedback after t_j

   $OP(t_j) = \{P_{j+1}, R_j, P_k\}$ with feedback to task T_k
   after task T_j

   $m(R_j) = R_j^{\#}$

   $\forall$ M in which $M(P_j) > 0$ and $M(R_j) > 0$

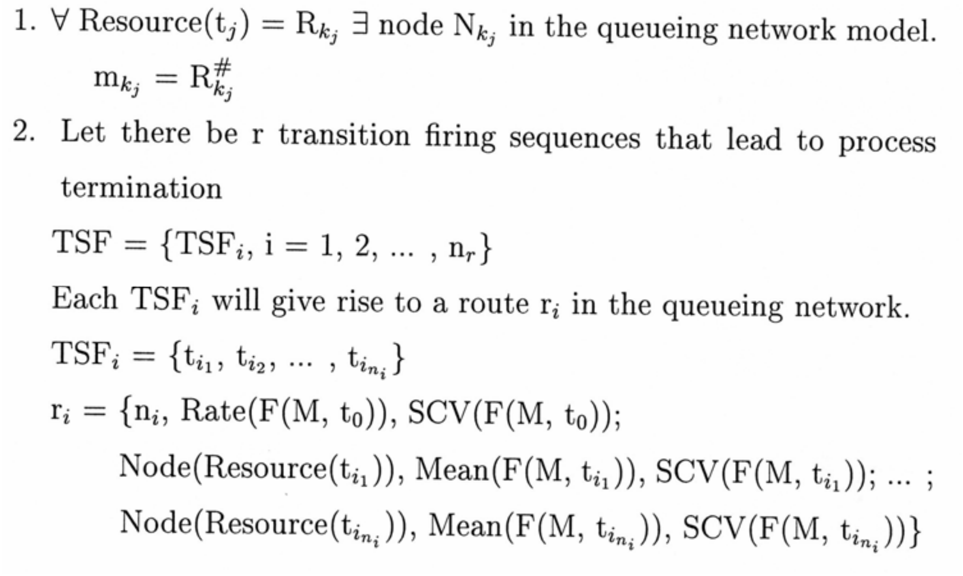
   Mean(F(M, t_j)) = Duration (T_j)

   SCV(F(M, t_j)) = SCV (T_j)
3. $\forall$ M

   Rate(F(M, t_0)) and

   SCV(F(M, t_0)) are set using process initiation logic.

The mappings to translate form a Petri net model to a queueing network model can be summarized in the following steps.



1. $\forall$ Resource(t_j) = R_{k_j} $\exists$ node N_{k_j} in the queueing network model.

   $m_{k_j} = R^{\#}_{k_j}$

2. Let there be r transition firing sequences that lead to process termination

   TSF = $\{TSF_i,\ i = 1,\ 2,\ \ldots\ ,\ n_r\}$

   Each TSF_i will give rise to a route r_i in the queueing network.

   $TSF_i = \{t_{i_1},\ t_{i_2},\ \ldots\ ,\ t_{in_i}\}$

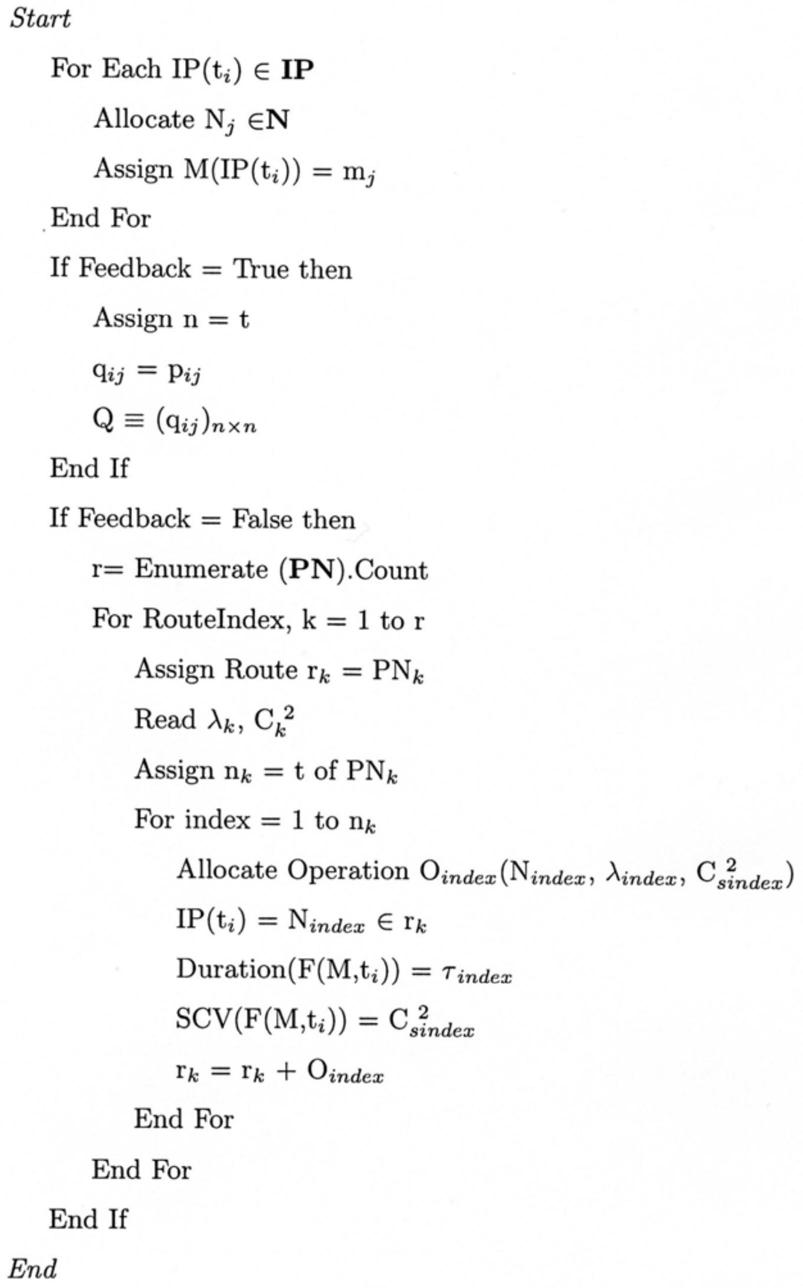
   $r_i = \{n_i$, Rate(F(M, t_0)), SCV(F(M, t_0));

   Node(Resource(t_{i_1})), Mean(F(M, t_{i_1})), SCV(F(M, t_{i_1})); ... ;

   Node(Resource(t_{in_i})), Mean(F(M, t_{in_i})), SCV(F(M, t_{in_i}))$\}$

7.3 Transformation Algorithm

Based on the mappings identified in Section 7.2, the following algorithm was developed to automatically configure a queueing network model from a Petri net representation of the business process description.



```
Start
    For Each $\mathrm{IP}(\mathrm{t}_i) \in \mathbf{IP}$
        Allocate $\mathrm{N}_j \in\mathbf{N}$
        Assign $\mathrm{M}(\mathrm{IP}(\mathrm{t}_i)) = \mathrm{m}_j$
    End For
    If Feedback = True then
        Assign $\mathrm{n} = \mathrm{t}$
        $\mathrm{q}_{ij} = \mathrm{p}_{ij}$
        $\mathrm{Q} \equiv (\mathrm{q}_{ij})_{n\times n}$
    End If
    If Feedback = False then
        r= Enumerate ($\mathbf{PN}$).Count
        For RouteIndex, k = 1 to r
            Assign Route $\mathrm{r}_k = \mathrm{PN}_k$
            Read $\lambda_k$, $\mathrm{C}_k^2$
            Assign $\mathrm{n}_k = \mathrm{t}$ of $\mathrm{PN}_k$
            For index = 1 to $\mathrm{n}_k$
                Allocate Operation $\mathrm{O}_{index}(\mathrm{N}_{index}, \lambda_{index}, \mathrm{C}^2_{sindex})$
                $\mathrm{IP}(\mathrm{t}_i) = \mathrm{N}_{index} \in \mathrm{r}_k$
                $\mathrm{Duration}(\mathrm{F}(\mathrm{M},\mathrm{t}_i)) = \tau_{index}$
                $\mathrm{SCV}(\mathrm{F}(\mathrm{M},\mathrm{t}_i)) = \mathrm{C}^2_{sindex}$
                $\mathrm{r}_k = \mathrm{r}_k + \mathrm{O}_{index}$
            End For
        End For
    End If
End
```

7.4 Implementation

As a proof-of-concept for the algorithm presented in section 7.3, a prototype function was programmed using VB.NET language. Two functions were developed, namely DML2PNML and PNML2QNML. The function DML2PNML takes a DML file as input and generates a PNML file. This PNML file would then serve as input to the function PNML2QNML, which generates a QNML file. On successful generation both the functions return a boolean true, otherwise a boolean false is returned. The prototypes of the functions are as follows:

Function DML2PNML (DML As XMLDocument) As Boolean

Function PNML2QNML (PNML As XMLDocument) As Boolean

The code for these functions is presented in Appendix D. These functions are implemented as part of a simple Windows based environment, which would allow the user to specify the required inputs. The screenshots for this environment are provided in Appendix E.

Chapter 8

Evaluation of the Single-Step Approach and the Multi-Step Approach

This chapter presents the strengths and limitations of the single-step and multi-step approaches discussed in Chapters 6 and 7. Also, a qualitative evaluation of both the single-step and the multiple-step approaches is presented here. Section 8.1 summarizes the strength and limitations, and Section 8.2 is devoted to the discussion of the qualitative evaluation. The focus of this evaluation is to bring out the similarities, differences and limitations of the abovementioned approaches. This evaluation was based on the following criteria: Feasibility of the Approach, Complexity in Modeling, Representational Capability, Formality, and the Complexity in Retrieving Performance Measures. Section 8.3 presents the conclusions drawn from the evaluation in Section 8.2.

8.1 Strengths and Limitations

Single-Step Approach

The main strengths of this approach are due to the feasibility and simplicity of this approach. The strengths that are evident at this point of time are as follows.

- The translation from the process domain to the queueing analysis domain is direct and involves only a single step.

- As there are no intermediate steps, in general, there is less potential for loss of information between the translations.
- The mappings suggested using this approach are general and are extensible to other general process modeling frameworks.
- The complexity involved in retrieving the results from the queueing analysis domain to the process domain is less in this approach.

Despite the simplicity of this approach there are some limitations that are evident at this point of time:

- Lack of a theory base in the process domain is a limitation to extensibility of this approach.
- The analysis capability is limited to the analysis capability that could be obtained via the existing queueing theory. Some qualitative analysis that could be achieved using a theory base is missing in this approach.
- The analysis capability is limited to the configurations discussed in Section 6.2, though some extensions are possible as suggested in Section 9.3. Hence to analyze a real world process situation, some assumptions must be made.
- The actual strength of this approach is in the representational capability provided by the graphical modeling language. The ability to capture the resource requirements in the process modeling language could itself be a limitation for this approach.

Multi-Step Approach

The main strengths of this approach are due to the feasibility and formality associated with the intermediate Petri net representation. Strengths that are evident at this point of time are summarized below.

- The translation from process domain to the queueing analysis domain is feasible.
- The mappings suggested using this approach are general and are extensible to other process modeling frameworks.
- Though, there is an additional step in translating to Petri nets from the process description, the formality provided by the Petri nets serve as a theory base for further translations.
- This approach is more desirable in frameworks that provide the multi-analysis capability, as the intermediate process representation using the Petri nets forms the base format.
- This approach also provides insight into the process domain, as analysis using Petri nets could reveal some logical aspects of the process descriptions.

Despite the feasibility of this approach there are some limitations that are evident at this point of time:

- As there is an intermediate step of translation, there is a greater possibility of loss of some information.
- The analysis capability includes the analysis using queueing models and qualitative analysis using the Petri net representation.

- The analysis capability is limited to the configurations discussed in Section 6.2, though some extensions are possible as suggested in Section 9.3. To analyze a real world process situation, some assumptions have to be made. Since there is an additional step of translating to the Petri nets, some additional assumptions may have to be made.
- The actual strength of this approach is in the formality provided by the Petri nets. The ability to capture the process domain requirements using Petri nets could itself be a limitation for this approach.
- Existing Petri net representation does not provide means to store the analysis results which are to be transferred back to the process model. This also contributes to the limitations of the multi-step approach.

8.2 Evaluation Criteria

8.2.1 Feasibility

The mappings identified for both the single-step approach and the multi-step approach described in chapters 6 and 7 suggest that both the approaches are feasible. Essentially, in both the approaches, a process-centric view is translated to a resource centric view. In the single-step approach, the graphical process model is directly translated to a queueing model. The mappings that are required for this translation are specified in Section 6.2 of Chapter 6. In the second case the graphical process model is converted to a Petri net, which in turn is translated into a queueing model. Although the queueing model is being generated from a Petri net, the underlying methodology is still the same: generating the queueing network model from a process-centric view of the system. In this case the Petri net model provides the process-centric view. The translations from the Petri net representation to the queueing model are specified in Section 7.2 of the Chapter 7. There is no conceptual difference in either of the approaches as the same methodology is followed in identifying the mappings.

8.2.2 Complexity in Modeling

The complexity in either of the approaches can be attributed to the difficulty in arriving at the mappings for each intermediate step. The level of complexity increases with the change of views at each intermediate step. For instance, consider the two step process where the first step involves conversion to a process-centric view and the second step involves conversion to a resource centric view. As the views are different, a one-to-one mapping may not exist between them. This in turn adds to the complexity of the entire process. Further, the number of such intermediate translations complicates to the modeling effort.

Since the single-step approach involves only a direct conversion, the complexity is relatively less. This additional step in the multi-step approach is expected to add some complexity to the overall process. However, in the intermediate step, the translation is between the same views. As there is a one-to-one correspondence between the process model and a Petri net representation as identified in [12], this added step of mappings does not contribute significantly to the complexity of the overall process. However, this is an extra step compared to the single-step approach. At this juncture, it is difficult to arrive at a conclusion as to which of these approaches is better.

8.2.3 Representational Capability

The graphical languages existing in the literature encompass different dimensions of the business process life cycle. The amount of detail that could be captured using the graphical language constructs constrains the specification of any model. A considerable amount of detail in the process models is required for specification purposes. However, for analyses, all the details may not be required. Depending upon the analysis technique chosen, some detail needs to be ignored or some extra detail needs to be added to the existing model. This extra information may not be readily captured by the modeling constructs. For example, if queueing is chosen as the analysis technique, details regarding the resource requirements are important and the information regarding the other entities could be ignored. In the process models the resource availability is not explicitly provided. The modeling language chosen should be capable of capturing the resource requirements.

In both the approaches discussed in this thesis, the same modeling language's (DDL) constructs are used. The DDL provides the capability of storing the resource requirements assuming that all the necessary details are provided by the user. In the first approach, the translation is from these constructs directly, hence there is no difficulty in identifying the resources and the corresponding mappings. In the second approach, when the process model is translated to a Petri net, these resource requirements are captured as input places. All the other required entities are also captured as input places. So there needs to be a way to identify or differentiate an input place that corresponds to a resource from the other input places that are a result of other required entities. Otherwise, it would be difficult to achieve the mappings as suggested earlier. One feasible solution to this is to attach a special label to identify an input place that corresponds to a resource. This could be viewed as a representational constraint in this approach.

Also, in the second approach though a Petri net is suggested as the base model, depending on the analysis technique chosen specialized Petri nets may have to be used instead of high level Petri nets. For example, to configure a queueing model stochastic Petri nets need to be used as suggested in Chapter 7. The required specialization may be a potential constraint in this approach.

8.2.4 Formality

Various modeling languages have been suggested in the literature. It is difficult to find a formal language that can capture all the details from a process model developed using different modeling languages. The formality suggested by [17] is used in this thesis for both the approaches. As DDL fits the formality used here, the suggested mappings hold in the single-step approach where, the mappings are shown between these formal elements. However, if a different modeling language is used, then there is a possibility that it may not fit the formality suggested. In such a case the first approach may not be feasible as is. However, such a case may not be encountered in the literature. In the multi-step approach, the base model is a Petri net, which has a standard formality associated with it. This standard gives us an added confidence in the validity of this approach.

8.2.5 Complexity in Retrieving Performance Measures

One of the desired functionalities of these approaches is to interpret and pass the performance measures obtained as a result of the analysis back to the modeler. Both the approaches have some limitations in this regard. However, this desired functionality is more practical in the single-step approach, where the graphical modeling language can store some if not all of the measures from the queueing model. As suggested by [5], DDL allows for holding some of these performance measures. However, if the modeler chooses a language other than the DDL, the ease of achieving this functionality depends on the capability of that language.

In the second approach, since Petri nets are used as the intermediate representation, there is a need to augment the Petri nets to hold these measures and pass them back to the process model. Since the computer processable format of Petri nets, the PNML, is still in the development stage, addition of such functionality may not be a daunting task.

Also it should be noted that, in order to achieve the functionality of interpreting the performance measures, the entire process needs to be guided by an intelligent support system like the DISS within the DIME framework.

8.3 Conclusions

In Section 8.2, the pros and cons of the single-step and the multi-step approaches were evaluated with respect to the criteria suggested. It should be noted that both the approaches are feasible, but complexity varies based on the criterion evaluated. Also, it is evident that both the single-step and multi-step approaches have their strengths and limitations of different dimensions. At this juncture, there is no clear winner as both the approaches suggest complexities in their own respect. Also these approaches were evaluated only with the limited criteria suggested in the previous section. The pros and cons of the respective approaches were assessed mainly with queueing analysis in view. At the outset, the single-step approach is feasible and simple but lacks the much needed theory base to drive the translation from a business process description to a performance model. However, as suggested in the DIME framework, the need for an established theory base such as Petri nets for driving the analysis in a distributed and integrated framework, favors the choice of the multi-step approach. However, it has to be noted that, if the multi-step approach is used, the downside would be the complexity in passing the performance measures back to the process model. But as suggested in Section 8.2.5, this drawback could be addressed with more research in this area. With this limited knowledge, it would be too early to recommend an approach that could potentially be used as the approach to generate performance models from business process descriptions.

Chapter 9

Conclusions and Future Work

This final chapter is divided into three sections. The first section summarizes the research completed. The second section lists the contributions of this thesis and the third section outlines areas for future work.

9.1 Research Summary

Focus of Research

The purpose of this research was to develop formal representations within the DIME framework to support performance analysis using queueing theory. The central idea was to develop a general purpose representation which could be used either as a part of an integrated environment or used in isolation. The existing framework is designed to use Petri net representation as the back-end to generate other formal representations. As part of the current effort, two-way mappings schemes between DIME Descriptive Modeling Language and queueing models were explored in addition to the mappings between the Petri nets and queueing models.

Methodology Employed

The existing modeling approaches were studied and explored in detail to gain an understanding of their purpose, strengths and limitations. The next step was to study and identify the linkages between modeling constructs, their representations and corresponding queueing models. Specifically, linkages between the DIME Modeling Language, corresponding Petri net representation and the queueing models were studied. This was followed by the development of a set of elements that are required for the specification of queueing network models. Based on the elements, a meta model was developed to identify the relationships between these elements. A markup language (QNML) for queueing networks was then specified. This step was followed by the development of two-way mapping schemes between DDL and QNML, and two-way mapping schemes between PNML and QNML. Transformations were accomplished through the transformation algorithms. Next, the pros and cons of the approaches from the previous step were evaluated for suggesting an approach to generate formal representations for queueing analysis within the DIME framework.

Results

The outcomes of this thesis are two XML based approaches to automatically configure a queueing Network model from a business process description. Also a standard XML-based interchange format, called the Queueing Network Markup Language (QNML), is developed as part of this research effort to store and exchange queueing network descriptions.

9. 2 Contributions

The main contribution of this research is enabling the performance analysis using queueing models within DIME Framework. Also an approach to support performance analysis (through queueing models) of business processes is suggested. Though this approach has been developed with DIME framework in view, it could also be used in isolation. During the course of this research, an XML-based markup language, called Queueing Network Markup Language has been developed to describe queueing network models. Also this thesis provides an insight for translation of business process descriptions to other quantitative models like simulation models.

9.3 Future Work

This thesis effort is a first step towards enabling performance analysis of business process models in an integrated and distributed environment. The main purpose here was to develop an approach to enable queueing analysis, in doing so many simplifying assumptions have been made. To model a real world system and analyze it, the developed methodology may not be adequate. However, using the developed methodology as base, extensions can be made to achieve analysis of a realistic system. To make this effort complete, the following considerations need to be made in future.

1. Though the resource sharing issue was addressed in the business process models studied as part of this thesis, it is assumed that any activity that captures a resource is also the activity that releases it. However, this is not usually the case

in many real world systems. This dimension of modeling needs to be explored as part of the future work.

2. In all the process models it was assumed that an activity requires a single unit of a resource of the same type. However, this could be extended to accommodate multiple units of multiple resources.
3. Since this effort was mainly focused on business process modeling and the main contributions are towards the field of enterprise modeling, no effort was made to explore solutions of any special cases in queueing models. For example, concurrency in business process models could be modeled using a fork-join queue with some assumptions. However, arriving at such models was not the focus of this research and is left for future work.
4. This research was mainly focused on taking a business process description to a queueing model and analyzing it. However, in an integrated and distributed environment, interpreting the analysis results and carrying them back to the modeler is also a significant step. Though some ideas have been presented in the initial sections, the actual implementation needs a methodology to drive the thrust. This dimension also needs to be explored in future. The complications in this case are that the results reported are tool specific. So there needs to be a standard methodology through which some uniformity is achieved.
5. One of the outcomes of this thesis is the identification of the need for a Queueing Network Markup Language. This thesis effort has led to an initial version of QNML. However, this needs to be distributed among the queueing community for review and acceptance. Also there may be many changes that need to be made

during the course of the suggested review. The ultimate goal of QNML would be to become the standard input for all queueing analysis tools. This would be one of the areas where some research may also be conducted in future.

References

1. Arkin, A., 2003. "Business Process Modeling Language – BPMI Proposed Recommendation," Retrieved from http://www.bpmi.org/bpml-spec.esp. Last accessed July 7, 2004.
2. Billington, J., et al., 2003. "*The Petri Net Markup Language: Concepts, Technology, and Tools,"* Retrieved from http://www.informatik.hu-berlin.de/top/pnml/. Last accessed July 23, 2004.
3. Birbeck, M., et al., 2001. *Professional XML*. Wrox Press Inc. Chicago, IL.
4. Bray, T., Paoli, J., Sperberg-McQueen, C., and Maler, E., Yergeau, F., 2000, "eXtensible Markup Language (XML) 1.0 (Second Edition)," Retrieved from World Wide Web Consortium (W3C) website: http://www.w3.org/TR/REC-xml. Last accessed July 7, 2004.
5. Chaugule, A., 2001. *A User-Oriented Enterprise Process Modeling Language*. Masters Thesis, Oklahoma State University, Stillwater, OK.
6. Dalal, N., Kamath, M., Kolarik, W., and Sivaraman, E., 2004. "Toward an Integrated Framework for Modeling Enterprise Processes," *Communications of the ACM*, Vol. 47, Iss. 3, pg. 83-87.
7. Electronic Business using eXtensible Markup Language (ebXML), 1999. Retrieved from *http://www.ebxml.org*. Last accessed July 7, 2004.

8. Choi, I., Song, M., Park, C., and Park, N., 2003. "An XML Based Process Definition Language for Integrated Process Management," *Computers in Industry*. Vol. 50, Iss. 1; pg. 85 –102.

9. Jungel, M., Kindler, E., and Weber, M., 2000. "*The Petri Net Markup Language,"* Petri Net Newsletter, 59:24-29.

10. Kamath, M., Dalal, N., Kolarik, W., Chaugule, A., Sivaraman, E., and Lau, A., 2001, "Process Modeling Techniques for Enterprise Analysis and Design – A Comparative Evaluation," *Proceedings of the 10th Industrial Engineering Research Conference,* IIE, Norcross, GA.

11. Kamath, M., Dalal, N., Kolarik, W., Lau, A., Sivaraman, E., Chaugule, A., Choudhury, S., Gupta, A., and Channahalli, R., 2002, "An Integrated Framework for Process and Performance Modeling of Next Generation Enterprise Systems: Design and Development Issues," *Proceedings of the University Synergy Program (USP) Conferenc.*

12. Kamath, M., Dalal, N., Chaugule, A., Sivaraman, E., and Kolarik, W., 2003, "A Review of Enterprise Process Modeling Techniques," in *Scalable Enterprise Systems: An Introduction to Recent Advances*, V. Prabhu, S. Kumara, and M. Kamath, (Eds)., Kluwer Academic Publishers, Boston, MA, 1–32.

13. Kamath, M., Dalal, N., Kolarik, W., 2004, "The Distributed Integrated Process Modeling of Next Generation Enterprises Framework," *Working Paper,* Center for Computer Integrated Enterprises, Oklahoma State University, Stillwater, OK.

14. Kelton, D., Sadowski, R., and Sadowski, D., 2002, *Simulation with Arena,* Second Edition. McGraw-Hill, Inc., New York, NY.

15. Organization for the Advancement of Structured Information Standards (OASIS), 2000, Retrieved from the "XML.org: The XML Industry Portal," website: *http://www.xml.org*. Last accessed June 30, 2004.

16. Fishwick, P., 2002, "Using XML for Simulation Modeling," *Proceedings of the 2002 Winter Simulation Conference,* Vol. 1, pg. 616-622.

17. Sivaraman, E., 2003. *Formal Techniques for Analyzing Business Process Models*. PhD Dissertation, Oklahoma State University, Stillwater, OK.

18. Sivaraman, E. and Kamath, M., 2002, "On the use of Petri Nets for Business Process Modeling", *Proceedings of the 11th Industrial Engineering Research Conference.,* IIE, Orlando, FL.

19. Unified Enterprise Modeling Language (UEML), 2002, Retrieved from "The UEML Portal," website: *http://www.ueml.org*. Last accessed July 25, 2004.

20. Viswanadham, N., and Narahari, Y., 1998, *Performance Modeling of Automated Manufacturing Systems*. Prentice-Hall, Inc., Englewood Cliffs, NJ.

21. Van der Aalst, W.M.P., and Kumar, A., 2003, "XML Based Schema Definition for Support of Inter-organizational Workflow," *Information Systems Research*, Vol. 14, Iss. 1, pg.23-47.

22. Weidmann T, 2002, "Next Generation Simulation Environments Founded on Open Source Software And XML Based Standard Interfaces," *Proceedings of the 2002 Winter Simulation Conference*.

23. Whitt, W. (1983), "The Queueing Network Analyzer," *Bell Systems Technical Journal,* 62, No 9, 2779-2815.

24. World Wide Web Consortium (W3C), 2001, "XML Schema," Retrieved from *http://www.w3.org/XML/Schema.html*. Last accessed July 23, 2004.

Appendix A

UML Notation:

□	Abstract Class
——◆	Composition
——▶	Generalization
——	Association

Appendix B

QNML Schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
    <xs:complexType name="QNFile">
        <xs:annotation>
            <xs:documentation>Root Element</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element name="Network">
                <xs:complexType>
                    <xs:complexContent>
                        <xs:extension base="Network"/>
                    </xs:complexContent>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="Network">
        <xs:sequence>
            <xs:element name="Type" type="xs:string"/>
            <xs:element name="CustomerCount" type="xs:integer"/>
            <xs:element name="NodeCount" type="xs:integer"/>
            <xs:element name="RoutingInformation" type="xs:string"/>
            <xs:choice>
                <xs:element name="Node" maxOccurs="unbounded">
                    <xs:complexType>
                        <xs:complexContent>
                            <xs:extension base="Node"/>
                        </xs:complexContent>
                    </xs:complexType>
                </xs:element>
                <xs:element name="Route">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="Operation">
                                <xs:complexType>
                                    <xs:complexContent>
                                        <xs:extension base="Operation">
                                            <xs:sequence>
                                                <xs:element name="Service">
                                                    <xs:complexType>
                                                        <xs:complexContent>
                                                            <xs:extension base="Service"/>
                                                        </xs:complexContent>
                                                    </xs:complexType>
                                                </xs:element>
                                            </xs:sequence>
                                        </xs:extension>
                                    </xs:complexContent>
                                </xs:complexType>
                            </xs:element>
                            <xs:element name="Node" maxOccurs="unbounded">
                                <xs:complexType>
                                    <xs:sequence>
                                        <xs:element name="Arrival" type="Arrival"/>
                                    </xs:sequence>
                                </xs:complexType>
                            </xs:element>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:choice>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="Node">
        <xs:sequence>
            <xs:element name="Name" type="xs:string"/>
            <xs:element name="Type" type="xs:string"/>
            <xs:element name="Description" type="xs:string"/>
            <xs:element name="ServerCount" type="xs:integer"/>
            <xs:element name="Utilization" type="xs:double"/>
            <xs:sequence>
                <xs:element name="Server" type="Server" maxOccurs="unbounded"/>
                <xs:element name="Queue" type="Queue"/>
                <xs:element name="Flow" type="Flow" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:element name="Arrival" type="Arrival"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="Flow">
        <xs:sequence>
            <xs:element name="OriginNode" type="xs:IDREF"/>
            <xs:element name="DestinationNode" type="xs:IDREF"/>
            <xs:element name="RoutingProbability" type="xs:double"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="Queue">
        <xs:sequence>
            <xs:element name="Size" type="xs:integer"/>
            <xs:element name="Discipline" type="xs:string"/>
            <xs:element name="MeanLength" type="xs:integer"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="Server">
        <xs:sequence>
            <xs:element name="ServiceTime" type="xs:double"/>
            <xs:element name="Utilization" type="xs:double"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="Operation">
        <xs:sequence>
            <xs:element ref="Node"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="Arrival">
        <xs:sequence>
            <xs:element name="Rate" type="xs:double"/>
            <xs:element name="SCV" type="xs:double"/>
            <xs:element name="Distribution" type="Distribution"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="Service">
        <xs:sequence>
            <xs:element name="Mean" type="xs:double"/>
            <xs:element name="SCV" type="xs:double"/>
            <xs:element name="Distribution" type="Distribution"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="Distribution">
```

```
        <xs:sequence>
            <xs:element name="Name" type="xs:string"/>
            <xs:element name="Parameters" type="xs:ENTITIES"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="Route">
        <xs:sequence>
            <xs:element name="OperationCount" type="xs:integer"/>
            <xs:element name="ArrivalRate" type="xs:double"/>
            <xs:element name="SCV" type="xs:double"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="Node"/>
</xs:schema>
```

Appendix C

Function DML2QNML

```
    '*******************************************************
    'This function takes a DMLFile as input and returns a
    'boolean true after the QNML File is written. If any
    'exception is encountered it return a false.
    '
    'This function uses the following other class objects
    '1.Activity
    '2.Resource
    '3.ResourceRequired
    '*******************************************************

    Function DML2QNML(ByVal DMLFile As XmlDocument) As Boolean
        Dim ResourcePool As New ArrayList
        Dim ActivityPool As New ArrayList

        Try
            'Reading Part of the DML
            Dim reader As XmlNodeReader = New XmlNodeReader(DMLFile)
            While reader.Read()
                'This loop reads all the resource elements and buils a
resource object list
                If reader.NodeType = XmlNodeType.Element And
reader.Name = "Resource" Then
                    Dim ResObj As New Resource
                    ResObj.id = reader.GetAttribute("id")
                    Do
                        reader.Read()
                        If reader.NodeType = XmlNodeType.Element And
reader.Name = "Name" Then
                            ResObj.Name = reader.ReadString
                        ElseIf reader.NodeType = XmlNodeType.Element
And reader.Name = "Type" Then
                            ResObj.Type = reader.ReadString
                        ElseIf reader.NodeType = XmlNodeType.Element
And reader.Name = "Description" Then
                            ResObj.Description = reader.ReadString
                        ElseIf reader.NodeType = XmlNodeType.Element
And reader.Name = "Count" Then
                            ResObj.Count =
Convert.ToInt16(reader.ReadString)
                        ElseIf reader.NodeType = XmlNodeType.Element
And reader.Name = "Utilization" Then
                            ResObj.Utilization =
Convert.ToDouble(reader.ReadString)
                        End If
                    Loop Until (reader.NodeType =
XmlNodeType.EndElement) And (reader.Name = "Resource")
```

```
                    ResourcePool.Add(ResObj)
                End If

                'This loop reads all the Activity elements and builds a
Activity object list
                If reader.NodeType = XmlNodeType.Element And
reader.Name = "Activity" Then
                    Dim ActObj As New Activity
                    ActObj.id = reader.GetAttribute("id")
                    Do
                        reader.Read()
                        If reader.NodeType = XmlNodeType.Element And
reader.Name = "ActivityName" Then
                            ActObj.Name = reader.ReadString
                        ElseIf reader.NodeType = XmlNodeType.Element
And reader.Name = "Type" Then
                            ActObj.Type = reader.ReadString
                        ElseIf reader.NodeType = XmlNodeType.Element
And reader.Name = "Classification" Then
                            ActObj.Classification = reader.ReadString
                        ElseIf reader.NodeType = XmlNodeType.Element
And reader.Name = "ActivityDuration" Then
                            ActObj.ActivityDuration =
Convert.ToInt16(reader.ReadString)
                        ElseIf reader.NodeType = XmlNodeType.Element
And reader.Name = "SCV" Then
                            ActObj.SCV =
Convert.ToDouble(reader.ReadString)
                        ElseIf reader.NodeType = XmlNodeType.Element
And reader.Name = "INPUT" Then
                            Do
                                reader.Read()
                                'This loop reads all the Resources
required by an activity
                                If reader.NodeType =
XmlNodeType.Element And reader.Name = "RESOURCES" Then
                                    Do
                                        reader.Read()
                                        If reader.NodeType =
XmlNodeType.Element And reader.Name = "Resource" Then
                                            ActObj.ResourceReq.id =
reader.GetAttribute("ResID")
                                            Do
                                                reader.Read()
                                                If reader.NodeType =
XmlNodeType.Element And reader.Name = "UnitsAvailable" Then

ActObj.ResourceReq.UnitsRequired = Convert.ToInt16(reader.ReadString)
                                                End If
                                            Loop Until (reader.NodeType
= XmlNodeType.EndElement) And (reader.Name = "Resource")
                                        End If
                                    Loop Until (reader.NodeType =
XmlNodeType.EndElement) And (reader.Name = "RESOURCES")
                                End If
                            Loop Until (reader.NodeType =
XmlNodeType.EndElement) And (reader.Name = "INPUT")

                        End If
                    Loop Until (reader.NodeType =
XmlNodeType.EndElement) And (reader.Name = "Activity")
                    ActivityPool.Add(ActObj)
                End If
            End While



            'Writing Part of the PNML from DML
            Dim NodeCount, OperationCount, i, j, k As Integer
            Dim textWriter As XmlTextWriter = New
XmlTextWriter(OutputFile, Nothing)
            textWriter.WriteStartDocument()
            textWriter.WriteStartElement("QNFile")
            textWriter.WriteStartElement("QNML")



            textWriter.Formatting = Formatting.Indented
            'Following lines add general information about the network
to the QNFile
            'Only open Queueing Networks with Deterministic routing
were considered here
            textWriter.WriteStartElement("Network")
            textWriter.WriteElementString("Type", "Open")
            textWriter.WriteElementString("CustomerCount", "Infinity")
            textWriter.WriteElementString("NodeCount",
ResourcePool.Count().ToString)
            textWriter.WriteElementString("RoutingInformation",
"Deterministic")

            NodeCount = ResourcePool.Count()

            'This loop writes the Node elements using the Resource list
            For i = 1 To NodeCount
                Dim ResObj As Resource
                ResObj = ResourcePool.Item(i - 1)

                Dim id As String = "n" & i.ToString()
                textWriter.WriteStartElement("Node")

                'Following lines add general information about the node
                textWriter.WriteAttributeString("id", ResObj.id)
                textWriter.WriteElementString("Name", ResObj.Name)
                textWriter.WriteElementString("Type", ResObj.Type)
                textWriter.WriteElementString("Description",
ResObj.Description)
                textWriter.WriteElementString("ServerCount",
ResObj.Count)
                textWriter.WriteElementString("Utilization",
ResObj.Utilization)
                'Following lines add general information about the
Server to the node element
```

```
                'Only Single Server systems were considered here
                textWriter.WriteStartElement("Server")
                textWriter.WriteElementString("ServiceTime", "")
                textWriter.WriteElementString("Utilization", "")
                textWriter.WriteEndElement() ' End Server
                'Following lines add Queue information to the node
element
                textWriter.WriteStartElement("Queue")
                textWriter.WriteElementString("Size", "Infinity")
                textWriter.WriteElementString("Discipline", "FCFS")
                textWriter.WriteElementString("MeanLength", "")
                textWriter.WriteEndElement() ' End Queue
                'Following lines add Arrival information to the node
element
                textWriter.WriteStartElement("Arrival")
                textWriter.WriteElementString("Rate", "")
                textWriter.WriteElementString("SCV", "")
                textWriter.WriteElementString("Distribution", "--")
                textWriter.WriteEndElement() ' End Arrival


                textWriter.WriteEndElement() ' End Node

            Next


            'Based on the path information provided, the Route elements
are wriiteen in this loop
            For j = 1 To PathCount

                OperationCount = EnumPathObj(j).Operations.GetLength(0)

                textWriter.WriteStartElement("Route")
                textWriter.WriteAttributeString("id", "R" & j)

                'On each path, the Activities correspond to the
Operations
                'This loop builds the Activities on each path and a
corresponding operation is added
                'to the Route
                For i = 1 To OperationCount


                    Dim ActObj As Activity
                    For k = 1 To ActivityPool.Count()
                        ActObj = ActivityPool(k - 1)
                        If ActObj.id = EnumPathObj(j).Operations(i - 1)
Then
                            Exit For
                        End If
                    Next

                    textWriter.WriteStartElement("Operation")

                    textWriter.WriteAttributeString("id", ActObj.id)
                    textWriter.WriteElementString("Name", ActObj.Name)
                    textWriter.WriteElementString("Node",
ActObj.ResourceReq.id)
                    textWriter.WriteElementString("Description",
ActObj.Description)

                    textWriter.WriteStartElement("Service")
                    textWriter.WriteElementString("Mean",
ActObj.ActivityDuration)
                    textWriter.WriteElementString("SCV", ActObj.SCV)
                    textWriter.WriteElementString("Distribution", "--")
                    textWriter.WriteEndElement() ' End Service


                    textWriter.WriteEndElement() ' End Operation
                Next
                textWriter.WriteEndElement() ' End Route
            Next


            'Following lines Close all the open tags from the beginning
            textWriter.WriteEndElement() ' End Network
            textWriter.WriteEndElement() 'End QNML
            textWriter.WriteEndElement() 'End QNFile
            textWriter.WriteEndDocument() 'End XMLDocument
            textWriter.Close()

            Return True
        Catch ex As Exception

            MessageBox.Show(ex.Message, "Error", MessageBoxButtons.OK,
MessageBoxIcon.Error)
            Return False

        End Try

    End Function
```

Appendix D

Function DML2PNML

```
'*******************************************************
    'This function takes a DMLFile as input and returns a
    'boolean true after the PNML File is written. If any
    'exception is encountered it return a false.
    '
    'This function uses the following other class objects
    '1.Transition
    '2.Resource
    '3.ResourceRequired
    '
    'This function takes transitiona and resource input places
    'into consideration. Output Places are ignored.
    '*******************************************************

    Function DML2PNML(ByVal DMLFile As XmlDocument) As Boolean

        Dim ResourcePool As New ArrayList
        Dim ActivityPool As New ArrayList

        Try
            'Reading Part of the DML
            Dim reader As XmlNodeReader = New XmlNodeReader(DMLFile)
            While reader.Read()
                'This loop reads all the resource elements and buils a
resource object list
                If reader.NodeType = XmlNodeType.Element And
reader.Name = "Resource" Then
                    Dim ResObj As New Resource
                    ResObj.id = reader.GetAttribute("id")
                    Do
                        reader.Read()
                        If reader.NodeType = XmlNodeType.Element And
_reader.Name = "Name" Then
                            ResObj.Name = reader.ReadString
                        ElseIf reader.NodeType = XmlNodeType.Element
And reader.Name = "Type" Then
                            ResObj.Type = reader.ReadString
                        ElseIf reader.NodeType = XmlNodeType.Element
And reader.Name = "Description" Then
                            ResObj.Description = reader.ReadString
                        ElseIf reader.NodeType = XmlNodeType.Element
And reader.Name = "ServerCount" Then
                            ResObj.Count =
Convert.ToInt16(reader.ReadString)
```

```
                        ElseIf reader.NodeType = XmlNodeType.Element
And reader.Name = "Utilization" Then
                            ResObj.Utilization =
Convert.ToDouble(reader.ReadString)
                        End If
                    Loop Until (reader.NodeType =
XmlNodeType.EndElement) And (reader.Name = "Resource")
                    ResourcePool.Add(ResObj)
                End If
                'This loop reads all the Activity elements and buils a
Activity object list
                If reader.NodeType = XmlNodeType.Element And
reader.Name = "Activity" Then
                    Dim ActObj As New Activity
                    ActObj.id = reader.GetAttribute("id")
                    Do
                        reader.Read()
                        If reader.NodeType = XmlNodeType.Element And
reader.Name = "ActivityName" Then
                            ActObj.Name = reader.ReadString
                        ElseIf reader.NodeType = XmlNodeType.Element
And reader.Name = "Type" Then
                            ActObj.Type = reader.ReadString
                        ElseIf reader.NodeType = XmlNodeType.Element
And reader.Name = "Classification" Then
                            ActObj.Classification = reader.ReadString
                        ElseIf reader.NodeType = XmlNodeType.Element
And reader.Name = "ActivityDuration" Then
                            ActObj.ActivityDuration =
Convert.ToInt16(reader.ReadString)
                        ElseIf reader.NodeType = XmlNodeType.Element
And reader.Name = "SCV" Then
                            ActObj.SCV =
Convert.ToDouble(reader.ReadString)
                        ElseIf reader.NodeType = XmlNodeType.Element
And reader.Name = "INPUT" Then
                            'This loop reads all the Resources required
by an activity
                            Do
                                reader.Read()
                                If reader.NodeType =
XmlNodeType.Element And reader.Name = "RESOURCES" Then
                                    Do
                                        reader.Read()
                                        If reader.NodeType =
XmlNodeType.Element And reader.Name = "Resource" Then
                                            ActObj.ResourceReq.id =
reader.GetAttribute("ResID")
                                            Do
                                                reader.Read()
                                                If reader.NodeType =
XmlNodeType.Element And reader.Name = "UnitsAvailable" Then

ActObj.ResourceReq.UnitsRequired = Convert.ToInt16(reader.ReadString)
                                                End If
                                            Loop Until (reader.NodeType
= XmlNodeType.EndElement) And (reader.Name = "Resource")
```

```
                                        End If
                                    Loop Until (reader.NodeType =
XmlNodeType.EndElement) And (reader.Name = "RESOURCES")
                                End If

                            Loop Until (reader.NodeType =
XmlNodeType.EndElement) And (reader.Name = "INPUT")

                        End If
                    Loop Until (reader.NodeType =
XmlNodeType.EndElement) And (reader.Name = "Activity")
                    ActivityPool.Add(ActObj)
                End If
            End While


            'Writing Part of the PNML from DML
            Dim IPlaceCount, TransitionCount, i, j, k As Integer
            Dim textWriter As XmlTextWriter = New
XmlTextWriter(OutputFile, Nothing)
            textWriter.WriteStartDocument()
            textWriter.WriteStartElement("PNML")

            textWriter.Formatting = Formatting.Indented
            'Following lines add general information about the Petri
net
            'Only open Stochastic Petri nets were considered here
            textWriter.WriteStartElement("PetriNet")
            textWriter.WriteElementString("Type", "Stochastic")

            IPlaceCount = ResourcePool.Count()
            'This loop writes the Input Places using the Resource list
            textWriter.WriteStartElement("InputPlaces")
            For i = 1 To IPlaceCount
                Dim ResObj As Resource
                ResObj = ResourcePool.Item(i - 1)

                Dim id As String = "n" & i.ToString()
                'Following lines add general information about the
InputPlace of type Resource
                textWriter.WriteStartElement("Resource")
                textWriter.WriteAttributeString("id", ResObj.id)
                textWriter.WriteElementString("Name", ResObj.Name)
                textWriter.WriteElementString("Type", ResObj.Type)
                textWriter.WriteElementString("Description",
ResObj.Description)
                textWriter.WriteElementString("TokenCount",
ResObj.Count)
            Next
            textWriter.WriteEndElement() ' End InputPlaces

            'This loop writes the Trasitions using the Activity list
            For i = 1 To TransitionCount
                Dim TransObj As Transition
                TransObj = CType(ActivityPool.Item(i - 1), Transition)

                Dim id As String = "t" & i.ToString()
                textWriter.WriteStartElement("Transition")
                textWriter.WriteAttributeString("id", TransObj.id)
                textWriter.WriteElementString("TransitionName",
TransObj.Name)
                textWriter.WriteElementString("Type", TransObj.Type)
                textWriter.WriteElementString("MeanFiringRate",
TransObj.MeanFiringRate)
                textWriter.WriteElementString("SCV", TransObj.SCV)


                textWriter.WriteStartElement("PLACES")
                textWriter.WriteStartElement("Resource")

                textWriter.WriteAttributeString("id",
TransObj.ResourceReq.id)
                textWriter.WriteStartElement("UnitsRequired",
TransObj.ResourceReq.UnitsRequired)

                textWriter.WriteEndElement() 'End Resource
                textWriter.WriteEndElement() 'End Places
                textWriter.WriteEndElement() 'End Transition

            Next

            textWriter.WriteEndElement() ' End PetriNet

            textWriter.WriteEndElement() 'End PNML
            textWriter.WriteEndDocument() 'End XMLDocument
            textWriter.Close()

            Return True

        Catch ex As Exception

            MessageBox.Show(ex.Message, "Error", MessageBoxButtons.OK,
MessageBoxIcon.Error)
            Return False

        End Try
```

Function PNML2QNML

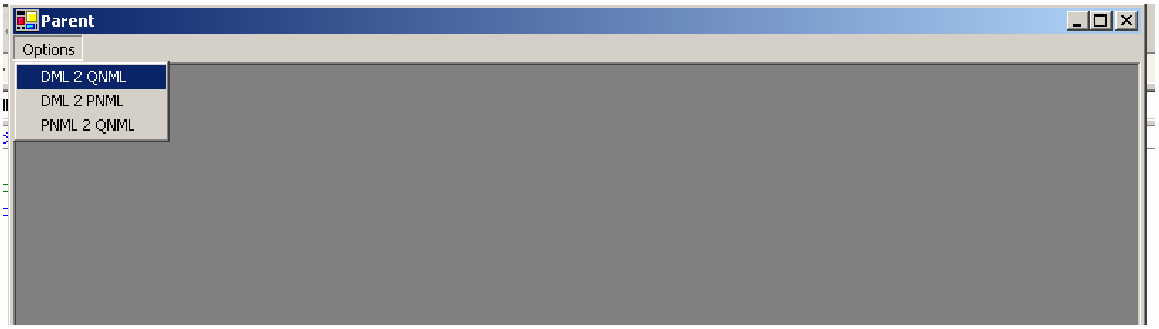
```
'********************************************************
    'This function takes a PNMLFile as input and returns a
    'boolean true after the QNML File is written. If any
    'exception is encountered it return a false.
    '
    'This function uses the following other class objects
    '1.Transition
    '2.Resource
    '3.ResourceRequired
    '********************************************************

    Function PNML2QNML(ByVal PNMLFile As XmlDocument) As Boolean
        Dim ResourcePool As New ArrayList
        Dim TransitionPool As New ArrayList
        Try
            'Reading Part of the PNML
            Dim reader As XmlNodeReader = New XmlNodeReader(PNMLFile)
            While reader.Read()
                'This loop reads all the Input Places and buils a
resource object list from it
                If reader.NodeType = XmlNodeType.Element And
reader.Name = "InputPlaces" Then
                    reader.Read()
                    If reader.NodeType = XmlNodeType.Element And
reader.Name = "Resource" Then
                        Dim ResObj As New Resource
                        ResObj.id = reader.GetAttribute("id")
                        Do
                            reader.Read()
                            If reader.NodeType = XmlNodeType.Element
And reader.Name = "Name" Then
                                ResObj.Name = reader.ReadString
                            ElseIf reader.NodeType =
XmlNodeType.Element And reader.Name = "Type" Then
                                ResObj.Type = reader.ReadString
                            ElseIf reader.NodeType =
XmlNodeType.Element And reader.Name = "Description" Then
                                ResObj.Description = reader.ReadString
                            ElseIf reader.NodeType =
XmlNodeType.Element And reader.Name = "TokenCount" Then
                                ResObj.Count =
Convert.ToInt16(reader.ReadString)
                            End If
                        Loop Until (reader.NodeType =
XmlNodeType.EndElement) And (reader.Name = "Resource")
                        ResourcePool.Add(ResObj)
                    End If
                End If
                'This loop reads all the Transition elements and buils
a transaction object list
                If reader.NodeType = XmlNodeType.Element And
reader.Name = "Transition" Then
                    Dim TransObj As New Transition
                    TransObj.id = reader.GetAttribute("id")
                    Do
                        reader.Read()
                        If reader.NodeType = XmlNodeType.Element And
reader.Name = "TransitionName" Then
                            TransObj.Name = reader.ReadString
                        ElseIf reader.NodeType = XmlNodeType.Element
And reader.Name = "Type" Then
                            TransObj.Type = reader.ReadString
                        ElseIf reader.NodeType = XmlNodeType.Element
And reader.Name = "MeanFiringRate" Then
                            TransObj.MeanFiringRate =
Convert.ToInt16(reader.ReadString)
                        ElseIf reader.NodeType = XmlNodeType.Element
And reader.Name = "SCV" Then
                            TransObj.SCV =
Convert.ToDouble(reader.ReadString)
                        ElseIf reader.NodeType = XmlNodeType.Element
And reader.Name = "INPUT" Then
                            Do
                                reader.Read()
                                If reader.NodeType =
XmlNodeType.Element And reader.Name = "PLACES" Then
                                    Do
                                        reader.Read()
                                        If reader.NodeType =
XmlNodeType.Element And reader.Name = "Resource" Then
                                            TransObj.ResourceReq.id =
reader.GetAttribute("ResID")
                                            Do
                                                reader.Read()
                                                If reader.NodeType =
XmlNodeType.Element And reader.Name = "TokensRequired" Then

TransObj.ResourceReq.UnitsRequired = Convert.ToInt16(reader.ReadString)
                                                End If
                                            Loop Until (reader.NodeType
= XmlNodeType.EndElement) And (reader.Name = "Resource")
                                        End If
                                    Loop Until (reader.NodeType =
XmlNodeType.EndElement) And (reader.Name = "PLACES")
                                End If

                            Loop Until (reader.NodeType =
XmlNodeType.EndElement) And (reader.Name = "INPUT")

                        End If
                    Loop Until (reader.NodeType =
XmlNodeType.EndElement) And (reader.Name = "Transition")
                    TransitionPool.Add(TransObj)
                End If
```

```
            End While



            'Writing Part of the QNML from PNML
            Dim NodeCount, OperationCount, i, j, k As Integer
            Dim textWriter As XmlTextWriter = New
XmlTextWriter(OutputFile, Nothing)
            textWriter.WriteStartDocument()
            textWriter.WriteStartElement("QNFile")
            textWriter.WriteStartElement("QNML")
            textWriter.Formatting = Formatting.Indented
            'Following lines add general information about the network
to the QNFile
            'Only open Queueing Networks with Deterministic routing
were considered here
            textWriter.WriteStartElement("Network")
            textWriter.WriteElementString("Type", "Open")
            textWriter.WriteElementString("CustomerCount", "Infinity")
            textWriter.WriteElementString("NodeCount",
ResourcePool.Count().ToString())
            textWriter.WriteElementString("RoutingInformation",
"Deterministic")

            NodeCount = ResourcePool.Count()
            'This loop writes the Node elements using the Resource list

            For i = 1 To NodeCount
                Dim ResObj As Resource
                ResObj = ResourcePool.Item(i - 1)

                Dim id As String = "n" & i.ToString()
                textWriter.WriteStartElement("Node")

                textWriter.WriteAttributeString("id", ResObj.id)
                textWriter.WriteElementString("Name", ResObj.Name)
                textWriter.WriteElementString("Type", ResObj.Type)
                textWriter.WriteElementString("Description",
ResObj.Description)
                textWriter.WriteElementString("ServerCount",
ResObj.Count)
                textWriter.WriteElementString("Utilization",
ResObj.Utilization)
                'Following lines add general information about the
Server to the node element
                'Only Single Server systems were considered here
                textWriter.WriteStartElement("Server")
                textWriter.WriteElementString("ServiceTime", "")
                textWriter.WriteElementString("Utilization", "")
                textWriter.WriteEndElement() ' End Server
                'Following lines add Queue information to the node
element
                textWriter.WriteStartElement("Queue")
                textWriter.WriteElementString("Size", "Infinity")
                textWriter.WriteElementString("Discipline", "FCFS")
                textWriter.WriteElementString("MeanLength", "")
                textWriter.WriteEndElement() ' End Queue
                'Following lines add Arrival information to the node
element
                textWriter.WriteStartElement("Arrival")
                textWriter.WriteElementString("Rate", "")
                textWriter.WriteElementString("SCV", "")
                textWriter.WriteElementString("Distribution", "--")
                textWriter.WriteEndElement() ' End Arrival

                textWriter.WriteEndElement() ' End Node
            Next


            'Based on the Transition firing Sequences provided, the
Route elements are writteen in this loop
            For j = 1 To TSFCount

                OperationCount =
EnumFiringSeqObj(j).Operations.GetLength(0)


                textWriter.WriteStartElement("Route")
                textWriter.WriteAttributeString("id", "R" & j)
                'On each sequence, the Transitions correspond to the
Operations
                'This loop builds the Transitions on each path and a
corresponding operation is added
                'to the Route
                For i = 1 To OperationCount


                    Dim TransObj As Transition
                    For k = 1 To TransitionPool.Count()
                        TransObj = TransitionPool(k - 1)
                        If TransObj.id =
EnumFiringSeqObj(j).Operations(i - 1) Then
                            Exit For
                        End If
                    Next

                    textWriter.WriteStartElement("Operation")

                    textWriter.WriteAttributeString("id", TransObj.id)
                    textWriter.WriteElementString("Name",
TransObj.Name)
                    textWriter.WriteElementString("Node",
TransObj.ResourceReq.id)
                    textWriter.WriteElementString("Description",
TransObj.Description)

                    textWriter.WriteStartElement("Service")
                    textWriter.WriteElementString("Mean",
TransObj.MeanFiringRate)
                    textWriter.WriteElementString("SCV", TransObj.SCV)
                    textWriter.WriteElementString("Distribution", "--")
                    textWriter.WriteEndElement() ' End Service
                    textWriter.WriteEndElement() ' End Operation
                Next
                textWriter.WriteEndElement() ' End Route
            Next

            'Following lines Close all the open tags from the beginning
            textWriter.WriteEndElement() ' End Network

            textWriter.WriteEndElement() 'End QNML
            textWriter.WriteEndElement() 'End QNFile
            textWriter.WriteEndDocument() 'End XMLDocument
            textWriter.Close()
            Return True
        Catch ex As Exception

            MessageBox.Show(ex.Message, "Error", MessageBoxButtons.OK,
MessageBoxIcon.Error)
            Return False

        End Try
    End Function
```

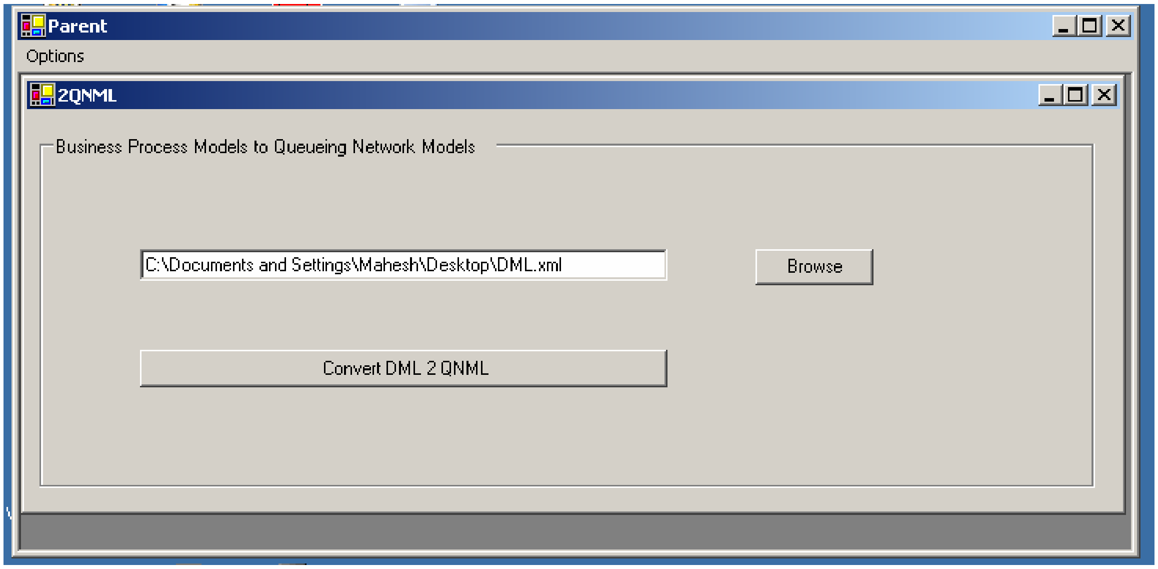
Appendix E

Screenshots of the Windows based environment for QNML Configuration

1. Choosing the function to execute

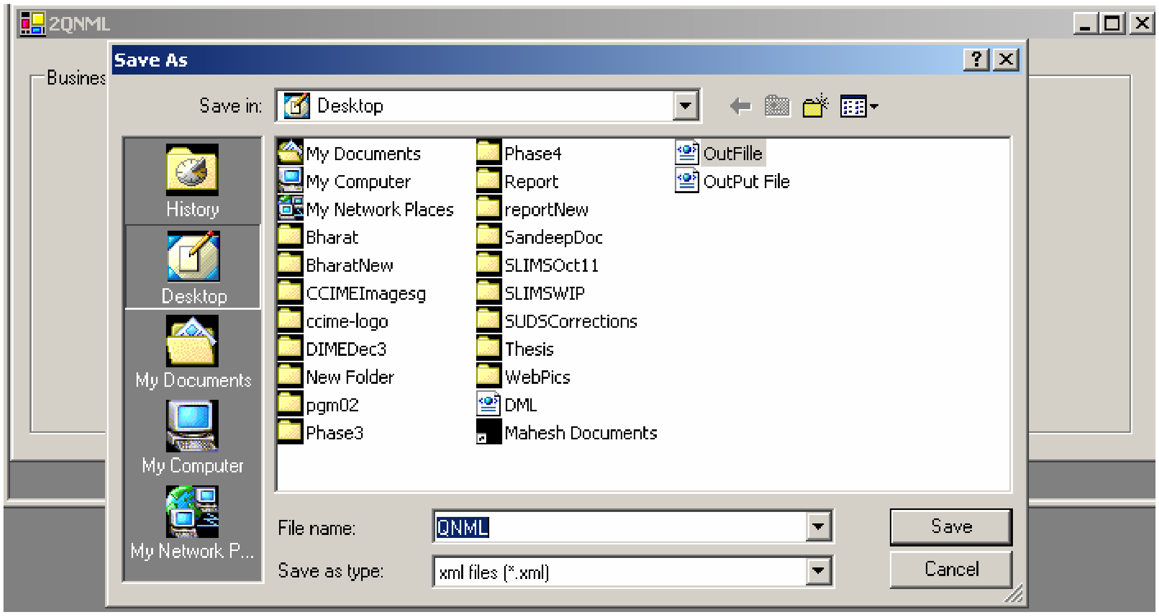


2. Screen shot browsing the input DML file



C:\Documents and Settings\Mahesh\Desktop\DML.xml

3. Providing the location to save the output QNML File

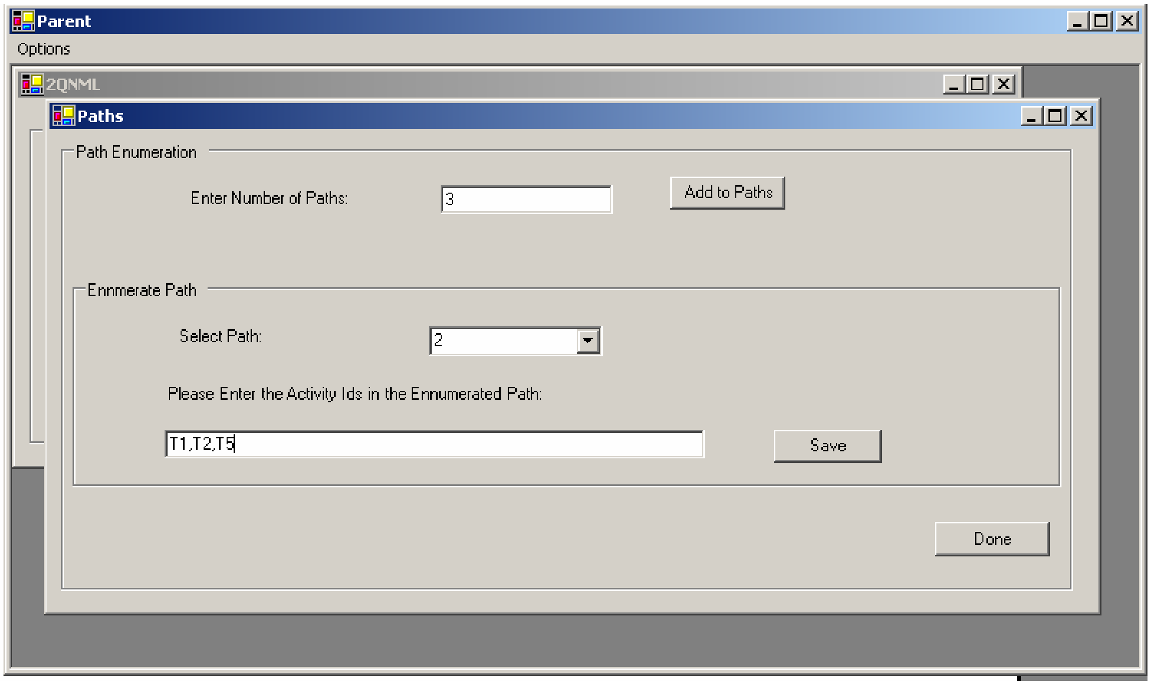


Save in: Desktop

File name: QNML

Save as type: xml files (*.xml)

4. Providing the Enumerated Paths as input



Enter Number of Paths: 3

Select Path: 2

Please Enter the Activity Ids in the Ennumerated Path:

T1,T2,T5

VITA

Uma Maheshwar Chalavadi

Candidate for the Degree of

Master of Science

Thesis: AUTOMATIC CONFIGURATION OF QUEUEING NETWORK MODELS FROM BUSINESS PROCESS DESCRIPTIONS

Major Field: Industrial Engineering and Management

Biographical:

Personal Data: Born in Hyderabad, India, on July 28, 1979, the son of Chandra Sekhar and Shashikala Chalavadi.

Education: Received the Bachelor of Engineering Degree in Mechanical Engineering from Birla Institute of Technology & Science, Pilani in May, 2002; graduated with first class honors. Completed the requirements for the Master of Science degree in Industrial Engineering and Management at Oklahoma State University in December, 2004.

Experience: Currently working as a Software Developer in the Department of Bio-Systems at Oklahoma State University. Earlier worked as a graduate research assistant in the same department. Also, worked as Teaching Assistant in the School of Industrial Engineering and Management, at Oklahoma State University from August 2003 to May 2004.

Professional Memberships: Alpha Pi Mu (National Honor Society for Industrial Engineers)

Name: Uma Maheshwar Chalavadi Date of Degree: December, 2004

Institution: Oklahoma State University Location: Stillwater, Oklahoma

Title of Study: AUTOMATIC CONFIGURATION OF QUEUEING NETWORK MODELS FROM BUSINESS PROCESS DESCRIPTIONS

Pages in Study: 112 Candidate for the Degree of Master of Science

Major Field: Industrial Engineering and Management

Scope and Method of Study: The purpose of this study was to develop a transformation scheme for automatic configuration of queueing network models from business process descriptions. Such a transformation scheme would enable queueing analysis of process models within an enterprise modeling framework, which is theory driven and links business process descriptions with formal qualitative and quantitative analyses in an integrated manner. A framework of this kind is suitable for next generation enterprises such as e-businesses, virtual enterprises, and global supply chains. The aim of this research was to extend the analysis capability of such an enterprise modeling framework. The transformation scheme uses the process control flow and task resource requirements to create a view where activities belonging to one or more process instances flow through a network of resources. Two alternative approaches were explored to automatically configure a queueing network model from a business process description. The first approach called the single-step approach generates a queueing network model from a business process markup language description. The second approach called the multi-step approach generates a queueing network model from a formal Petri-net based business process representation, which is described using the Petri Net Markup Language (PNML). An XML-based interchange format, called the Queueing Network Markup Language (QNML), was developed as part of this research effort to store queueing network descriptions. As a proof of concept, the transformation scheme was implemented as part of the DIME framework, developed at the Center for Computer Integrated Manufacturing Enterprises, Oklahoma State University. The DIME framework was a result of research funded by the National Science Foundation through grant # DMI-0075588, under the Scalable Enterprise Systems Initiative.

Findings and Conclusions: Two feasible approaches were developed to automatically configure queueing network models from business process descriptions. This research also led to the development of QNML, a Queueing Network Markup Language, which is a XML-based format to describe queueing network models. This work has also enabled the queueing analysis capability within the DIME framework.

Advisor's approval: Dr. Manjunath Kamath