FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

On Building Multidimensional Workflow Models for Complex Systems Modelling

DISSERTATION

zur Erlangung des akademischen Grades

Doktorin der technischen Wissenschaften

eingereicht von

Jasmine Malinao
Matrikelnummer e1429003

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: O.Univ.Prof. Dr. Walter G. Kropatsch

Diese Dissertation haben begutachtet:

(Prof. Jaime Caro, PhD)

(Prof. Shin-ya Nishizaki, PhD)

Wien, 25.01.2017

(Jasmine Malinao)

FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

On Building Multidimensional Workflow Models for Complex Systems Modelling

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktorin der technischen Wissenschaften

by

Jasmine Malinao
Registration Number e1429003

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: O.Univ.Prof. Dr. Walter G. Kropatsch

The dissertation has been reviewed by:

(Prof. Jaime Caro, PhD) (Prof. Shin-ya Nishizaki, PhD)

Wien, 25.01.2017

(Jasmine Malinao)

Erklärung zur Verfassung der Arbeit

Jasmine Malinao
Vorgartenstraße 67/41, 1200 Wien

(Ort, Datum)	(Unterschrift Verfasserin)

Acknowledgements

"The isolated man does not develop any intellectual power. It is necessary for him to be immersed in an environment of other men, whose techniques he absorbs during the first twenty years of his life. He may then perhaps do a little research of his own and make a very few discoveries which are passed on to other men. From this point of view the search for new techniques must be regarded as carried out by the human community as a whole, rather than by individuals."

– Alan Turing

To AIT Austrian Institute of Technology. I wholeheartedly express my gratitude for your very welcoming and supportive environment in doing research and development. Research is a space without boundaries – you are one of the institutions who have shown me that being a foreigner in Europe does not and cannot refute this as I go through bureaucratic and academic processes as a resident and scientist in this continent. I thank my supervisor in AIT, Dr. Florian Judex, for your wisdom, humor, trust, patience, and understanding in guiding me with research tasks in AIT and in my PhD. I thank Dr. Gerhard Zucker and DI Tim Selke for your help as I discovered profiles on energy systems in the few months I gave assistance in the ExtrACT project. I thank Najd Ouhajjou, Barbara Jandl, Maggie Schibor, Usman Habib, Ghazal Etminan, Bettina Brunnader, Anna-Maria Sumper, and Elvira Welzig for your kindness, friendship, and guidance.

To the Pattern Recognition and Image Processing Group(PRIP), Institute of Computer Graphics and Algorithms, Vienna University of Technology. I am deeply honored to have the opportunity to be a member of this research group, experience its practices, and perform my research with the generous advisorship of its lead. To my professor and PhD adviser, Dr. Walter Kropatsch, I am very lucky, honored, and grateful to have you as my supervisor in TU Wien. You had been so open and generous of your advice and time since I went through my admission until the completion of this dissertation. Thank you! I am also grateful to Elfriede "Elfi" Oberleitner, together with Barbara Wiesböck of the Office of the Dean of the Faculty of Informatics, for your kind assistance and coordination especially when documentary requirements were needed to be processed in the many stages of my academic life in the university.

To the Algorithms and Complexity Lab of University of the Philippines Diliman. Time and again, I always reflect how I am very blessed to have been developed as a scientist in this lab since I started graduate school. I owe it to you that I am able to thrive in research in whatever place and circumstance that I am in. I especially thank Sir Henry, Francis, Jhoirene, Jan, Nestine, and Rich for a lifetime of friendship and collaborations in research and development. I thank my former colleagues in the Computer Science Department of the university, Kuya Nestor and Ate Mila, for your kindness and undying support even when I am already abroad.

To my soul siblings in Vienna and Pinas. I thank Arlavinda Rezqita, Rasmiaditya Silasari, Ulfah Nurzannah, and Claudia Zanabria for your sisterhood and companionship during my stay and exploration of Europe. I appreciate all your help, advice, free-spiritedness, understanding, patience, and empathy in all experiences we went through these years. I thank Florian Miksch and Nelly Jean Forro for your care and huge help for my quest and adventures in my PhD in Vienna. I thank this bunch of the most hilarious women I have known – Kath *Pusa*, Kaysie *Kalaykay*, Tine *Tirintin*, Joyce *Aloca*, Apple *Mansanas*, Jel, Mona, Haniel. A huge part of me and my brain had been kept intact by all of you breaking it every now and then.

To Proficiency Evaluation Committee and Defense Panel. I thank Dr. Wolfgang Kastner, Dr. Thilo Sauter, Dr. Jaime Caro, and Dr. Shin-ya Nishizaki for accepting to be part of my Proficiency Evaluation Committee and/or Defense Panel. I thank you for taking time and effort in providing reviews for the development and accomplishment of the research goals in this dissertation.

Finally, I thank my family in the Philippines for the implicit trust, confidence, and eternal support of all my decisions in life. I also thank God for the strength and wisdom especially in my most trying times. I give all these back to you and to the world!

Abstract

From well-known, classical workflows such as Petri nets to one of the recent developments of modelling frameworks such as the Business Process Modelling Notation, the development of system representations has long been established and improved through the years. The common goal of such frameworks is to produce traceable, effective, and well-understood functional and nonfunctional specifications of business and scientific systems. This goal addresses issues from workflow design, verification, control and monitoring, and continual improvement as systems evolve. Through the years, these frameworks had been constructed, enabled, deployed and used under a singular or dual perspective of modelling and verification relating to workflow dimensions, i.e. process, resource, case. In literature, there is a huge gap for the support and enactment of all three dimensions into one model for system representation and verification. That is, these undertakings are mainly either process- or resource-centric. In terms of modelling with all three dimensions in place, some support is observed in the Robustness Diagram(RD) of the ICONIX framework. Because of its notational backbone, it was posed to serve as a bridge for requirements traceability when using other workflows that focus solely on either structure or behavior of system representations. It has potential in providing support from requirements capture to testing to redesign of models. However, this diagram has been underdeveloped with respect to these potentials in modelling and verification especially for complex systems.

In this research, the Robustness Diagram with Loop and Time Controls(RDLT) was introduced to support modelling and verification of complex systems. It is an extension based from RDs. Building on RDs, we propose formalizations of RDLTs with consideration for the use and representation of all three workflow dimensions in one model. Additionally, by accounting the requirements of volatility, persistence, multi-state configuration, and hierarchical structures and relationships present in complex systems, the concept of a *reset-bound subsystem(RBS)* in RDLTs was also formulated. RBS enacts capabilities of cancellation regions in well-known workflows in literature. However, RBS enforces topological and behavioral requirements to perform resets in values in models. Additionally, this research addressed the problems of explicitness and effectiveness of representing data, control flow patterns(e.g. sequential, parallel, splits, n-out-of-k joins, iteration, cancellation regions, etc.), and multi-level and multi-participant interactions. These problems were also dealt under the specifications of all three workflow dimensions, persistent and volatile structures and behavior in models.

In particular, this research introduced attribute-driven typing of vertices, arcs, and substructures to enforce structural and functional specifications in RDLTs. The values of the attributes and the resulting types of RDLT components directly influence their usage and groupings for the execution of activities in models. Furthermore, they also influence mechanisms for encapsulation of data and control flows in RDLTs. We proposed the concept of Points-of-Interests(POIs) in RDLTs that also rely on these information. POIs are then used to establish special regions in the model. We formally defined these regions and characterize their neighborhood structures. We established encapsulation rules for RBS and add its information to the characterized neighborhoods to determine metrics for the analysis of RDLTs. In this research, the metrics that were developed mainly focus on topological- and type-bound reachability, delays that cause bottlenecks, task synchronicity, and activity completion. They are computed with consideration of the presence of maximal substructures in RDLTs. Each substructure supports the execution of an activity profile for the completion of a case in a multi-activity RDLTs.

Among the model properties in literature for workflows, this research adopted and introduced soundness and free-choice for RDLTs. (We focus on these two properties because many other properties can be implied from them.) Noting the types of components and substructures and the presence of RBS in them, a behavioral profile of RDLTs that satisfy the first property is initially provided. This profile is produced by the use of the results from our proposed algorithm for activity extraction. However, this research devised two independent approaches to prove this property through structural views that account the types and the presence of RBS in RDLTs. The first approach was constraint-driven. RLDT attributes that pertain to the enforcement of splits and joins of type-alike components were used. We developed the concepts of an *extended RDLT* and a *vertex-simplified RDLT* to support this approach. Meanwhile, the second approach was component use-driven. Attributes that pertain to repeatability of task execution were checked in type-alike/mixed-type components and hierarchical structures and interactions in RDLTs. This approach used our proposed encapsulation rules for RBS. Collectively, both approaches proved soundness based on statically-verifiable information in RDLTs.

Meanwhile, the free-choice property for RDLTs was proposed and built from using the POI-driven metrics for reachability, boundedness, and synchronicity. In literature, this property focused on place-task relationships in workflows. In this research, it is extended by including combinations of topology-, constraint-, and type-driven free-choiceness in models. There are two RDLT configurations that were proposed for these combinations. One adopts the well-known view of free-choiceness while the latter its extension. However, both configurations consider all the workflow dimensions in modelling RDLTs.

In addition, this research provides efficient verification schemes for soundness, free-choiceness, and other proposed model properties for RDLTs. Real-world examples of RDLT models, including that of a complex energy system, was provided to illustrate how their structural and behavioral profiles were captured where these properties are checked. Finally, this research proved the relationships and hierarchies of RDLTs based on the properties that can be verified from them.

Contents

1 Introduction

In this chapter, a general background of classical and recent modelling frameworks are presented. The developmental issues pertaining to their notational construction, control and management schemes, transformations from design to deployment, current state of technological support, and their weaknesses in representation are provided and analyzed. From these information and the analysis, lists of generic and specific problems are collected. This chapter further provides assertions of possibilities for areas of improvement in modelling with respect to addressing these problems. It presents the foundation on the rationale and goals of this research. In particular, it focuses on presenting well-established graphical frameworks, such as Petri nets, whose theoretical backbone became a basis for the formulation of many other modelling and verification frameworks in literature. With the growing development and use of concepts and technologies of Business Process Modelling, this chapter also includes a discussion, brief survey, and analysis of one of its core language for design – the Business Process Model and Notation(BPMN). This chapter also includes the recent developments in diagramming standards of the Unified Modelling Language(UML). This standard had long been widely used for modelling especially in business and academic settings. From one lesser known diagramming support inside this standard, this chapter presents the UML's Robustness Diagram(RD) and its potentials to help address the different problems cited in this section. It also provides descriptions of the formalisms and schemes that enrich the current state and use of RDs for supporting the modelling and verification of complex systems.

1.1 Background of the Study

On Workflows

Workflows and Workflow Management Systems(WfMS) are undeniably a huge part of understanding, characterizing, analyzing, and improving different types of systems from the business to scientific domains. **Workflows**, as defined by the Workflow Management Coalition [1], is "the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules" [2]. **Workflow models** are "general models workflow structure, data, agents and execution policies according to user and application needs" [4]. Policies ensure correctness of workflows. Workflow models are generally used to aid in the analysis, verification, and eventual modifications of specifications in workflows when such are deemed necessary. Meanwhile, **WfMS** is "a software system that defines, creates, and manages the execution of workflows through the use of software, running on one or more workflow engines, which is able to interpret the process definition, interact with workflow participants, and, where required, invoke the use of information technology tools and applications" [2]. Although these definitions are mainly focused on businesses, they can also be adopted in the scientific domain. A scientific workflow is the assembly of sets of scientific data processing activities with data dependencies between them [5]. These processes can include data transformations, data mining processes, database queries, simulations done in high performance computers. In the same manner, scientific workflows are handled by scientific workflow management systems such as Kepler [6], Taverna [7], Triana [8], WOODSS(WOrkflOw-based spatial Decision Support System) [9], etc. More concretely, these workflows had been used for modelling ranging from simple systems, e.g. a recruitment process in human resource management [10], to complex ones such as those found in contract net protocols in bidding [11], flexible manufacturing systems [12], gas station servicing [15], biological systems [16, 17], etc. The complexity of systems mainly concern with the magnitude of the state space that can be generated from the system's structure and behavior. In [19, 20], a complex system can generate a huge state space because of its inherent processes which include "asynchronous and sequential and stochastic behavior, by high level of concurrency and conflict of its tasks and mutual exclusive resources. In general, a **complex system** is an organization made up of many interacting components where these interactions often lead to large-scale behaviors which are not easily predicted from a knowledge only of the behavior of its individual components [21]. The unpredictability often arises due to nonlinearity inherent in these behaviors. Nonetheless, such systems have the capability to withstand failures from among these interactions. This is in part to the hierarchical self-organization and its ability to withstand failures and perturbations by use of memory [22, 23]. **Persistence** in memory, relates to the system's capability to store and access information that is usable for system recovery when needed [24], e.g. when impending failures are expected. **Volatility**(or *ephemerality*) in such systems is the opposite of persistence, i.e. an attribute of possessing transience in their system and work specifications. Accounting all these aspects of complex systems, it is therefore not hard to see that certain problems that are computationally hard, i.e. NP-complete [18], arise in their modelling and verification with respect to both structure and behavior.

Workflow Dimensions and Existing Models and Issues

Workflows are formulated under the three workflow dimensions [25] namely (1) process, (2) resource, and (3) case, as seen in Figure 1.1.

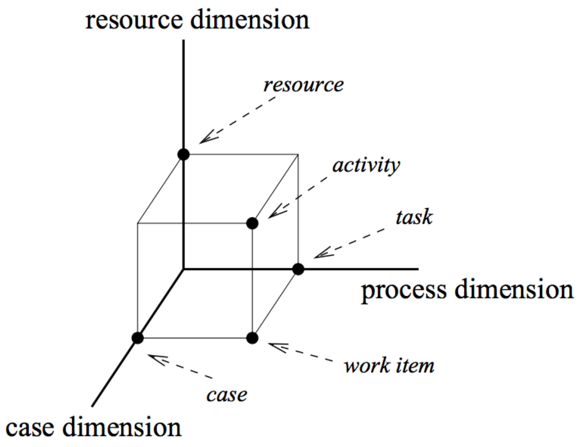


Figure 1.1: The three dimensions of a workflow. (Image source: [25])

Formally, the *process dimension* is a specification of processes, where a ***process*** is a partial ordering of a set of *tasks* performed by a system. This includes routing specifications of control schemes of these tasks like sequential, conditional, iteration, etc. A **task** is an atomic computation done by a system. **Atomic** means that, for modelling, the internal structure is not relevant [2]. One classic workflow model where the process dimension is being used is a Petri net [27, 86, 87]. Petri nets provide graphical notation showing step-by-step computations that are performed when conditions are satisfied given some initial input configuration in the net. These nets, among many others, have an exact mathematical definition of process executions with well-defined governing semantics for an effective process analysis.

The *resource dimension* is a specification of resources, where a ***resource*** is an object(e.g. user, database, component, etc.) with a determinable set of tasks to perform. This set of tasks define the role of the resource in the system and therefore a part of the functions in the organizational structure where a resource belongs in. In software engineering, this dimension is used in modelling some Unified Modelling Language(UML) [29] diagrams such as Class and Component Diagrams. These types of diagrams are used to model a system's static, mostly structural information – i.e. the list of resources that assist the accomplishment of the processes for some of the operations in a system.

The *case dimension* is a specification of cases, where a ***case*** represents an abstraction of a set of entities that is processed from some point of execution of its workflow until its corresponding output is produced. The concept of a case is equivalent to the concept of tokens [2] in Petri nets [87]. As described in [2], a case is represented by a set of tokens that are processed from an initial marking specification until one token arrives in an output place in the net. Consider the case where the initial configuration has every token in this set in their corresponding source place. By enabling and firing transitions in the net from this initial configuration to its corresponding final configuration, the behavioral profile of the net describes one complete functionality of a system. Note that the final configuration, in this sense, implies there is one token in a sink node which forces termination of process executions in the net. Under the assumption of this completeness of the behavioral profile from source/s to sink descriptions of a functionality in the net, it is further implied that there are no tokens in all intermediate places in the net. Intermediate places are places which are not a source or a sink in the net. In system representation, this means that there are no unnecessary components found in the model to execute such a case specification.

Cases can be illustrated by an example of having different scenarios that different students in a university face during an enrolment period in a semester. For this enrolment workflow, one case can mean a student is at his initial stage of enrollment, i.e. he is registered as a student in the semester he is enrolling in. For this case, a token in the source place of the net represents the condition that he is registered. Another case can mean another student is already registered and he has selected the minimum number of lecture units for the semester. For this second case, there are no tokens in the source place but one token in the place representing the check for the lecture units for the enrollment. Note that both cases can be embedded into one enrollment workflow and be processed simultaneously using process instances of the net with a WfMS that controls, manages, and distinguishes the two case implementations.

Work is a specification of a case with its relevant processes that enable an execution of a case. An **activity** is the actual performance of a resource on a work specification. Activities are either atomic or non-atomic [2]. **Tasks** are activities which are atomic. **Atomicity** refers to the attribute in a workflow where there is encapsulation of work and their enabling resources by means of a unitary graphical representation in its design. In *Business Process Model and Notation*(BPMN) [2], this type of encapsulation is achieved by using a rounded rectangle which pertains to the atomic activity itself. Because of the activity's atomicity, there is only one entry point and exit point in send and receive interactions. With this, activities can be implemented as nested activities to enable the creation of subprocesses within encapsulations. In Petri nets and workflow nets, tasks/activities are represented as transitions. In workflow nets, nesting is represented by task refinements [25] where subprocesses(known as subflows) are encapsulated in transitions. By a task refinement of a transition, the entire specification of the transition's subflow processes can be revealed and analyzed.

Workflow nets offer capabilities of distinguishing and tracking parallel executions of different cases across splits and joins of processes. This is done by providing graphical components in its framework. (In colored Petri nets [31], cases can be tracked by the use of colorings on the tokens. The colors are used for flow control and tracking although graphical representations to deal with these representations are not well supported in this type of Petri nets.) Moreover, one of the weaknesses of workflow nets is its absence of support in modelling resource-related information such as organizational structures and role assignments. Data handling with respect to role execution and management amongst resources in business processes are not fully and explicitly captured in workflows. Other types of process models would have to be created and associated to these workflows, e.g. human interaction workflows [2,32,33], to enable specification of this information.

Another type of workflow model, YAWL(Yet Another Workflow Language) [34], mainly relies on the framework of workflow nets. It provides enhancements of the latter by supporting some aspects of execution control. That is, multiple cases (in the form of process instances) are handled more effectively by associating each task with its own individual state transition diagram. Tokens representing multiple cases are coursed through the transition diagram's components. These components represent various states of case handling within the task. These states specify whether the instance execution is "enabled", "executing", "completed", and "active". The transition "add" is supplemented in YAWL for multiple instance handling. For handling multiple instances, each task is associated to a vector specifying the minimum, maximum, thresholds on number of completed cases, and whether the creation of new tokens for succeeding instances are made dynamically or not. The "add" transition is fired to put a token to the "enable" condition as soon as a new instance is introduced in the "active" condition when the dynamic setting in the task is provided [34].

Another enhancement is the nonlocal firing behavior in YAWL. This supports the enabling of processes in a workflow which are influenced by other processes that are not locally connected to the former processes. Nonlocal firing behavior in YAWL is supported by the use of a task t that performs cancellation on some predefined group G of tasks/activities in a workflow which are associated to t. **Cancellation** is the interference of the task t in the execution of the elements in G given some circumstance [46]. This interference can either be disabling or withdrawing the execution every task/activity in G, if it is already executing. Such a task t is referred to a **cancellation task** and G is called its *cancellation region*. That is, a **cancellation region** are components in workflows where cancellation occurs when its associated cancellation task is fired. A real-world example for this control scheme is a cancellation of an order that is performed by a customer in a workflow describing a product ordering system. When the cancellation task is enabled and fired due to a message sent from a customer to a seller, all other tasks such as product allocation, product delivery, invoice issuance, etc. are eliminated all at once. For such a case, mechanisms in the workflow to remove other tokens that belong to this case specification should be in place. In YAWL, this is achieved by associating a cancellation region to each task. Note that connectivity among the cancellation task and the members of its cancellation region is not required in a workflow.

Compared to other control flow languages, YAWL provides all but one control pattern, i.e. implicit termination. Apart from the two enhancements above, YAWL also provides effective support for OR-joins [46]. For models with both cancellation regions and OR-joins, verifying certain properties such as soundness [25] becomes undecidable [46,47]. The hardness in tackling this problem stems from the following: (a)the dependence on nonlocal information that is used to trigger cancellations, and (b)the semantics of OR-joins that is incapable of realizing such information when resolving the processes leading to those joins. In addition to this, the choice of which paths in splits or joins are still not explicit in YAWL. For instance, this choice made from using an “exclusive or split” cannot be formally specified in YAWL. The “k-out-of-n join” is not fully supported in YAWL. This type of join happens when multiple process specifications lead to this type of join wherein one of these processes is disregarded/ignored when k out of n of them has/have completed. This circumstance also implicitly brings up another shortcoming of YAWL on specifying subsets of these n processes wherein their completion is sufficient to proceed with the succeeding activities after the join component. This is also similar to the problem of resolving OR-joins given that there are cancellation regions along these components.

Two possible recommendations in tackling these problems in YAWL are the following (i)enable aggregation of workflow components which are involved in a cancellation region such that atomicity is enabled for these components, and (ii)provide explicitness of data-driven conditions by directly associating them to arcs leading to the joins at design-time. For (i), modellers should be able to abstract these components as one node in a graph while tackling and resolving multiple interactions with some other system components. That is, encapsulation is achieved while cancellations are still faithfully implemented in the model. An option to enable this mechanism is to have a class/object view for this node so it manages the cancellation region. With this view, those components will be viewed as members of the class, either as task or an attribute that is involved in data-driven conditions. This type of view, regardless if it is used on activities or resources that are part of systems, induces the use and support of all the workflow dimensions [25] in modelling. As a result, the explicitness of the structures and their relationships are imposed in models. This also aids in the efficient and unambiguous mapping from one type of model to another. In this research, this kind of abstraction is referred to as a **reset-bound subsystem(RBS)** of the model. (For more information on other reset mechanisms in process-centric modelling, see Reset Nets in [35, 36]) Implicitly, RBS can provide a platform to enable a multi-level analysis of the model itself. At a lower level, we enable analysis and verification of the cancellation region and its interaction as a subprocess with respect to its **environment**, i.e. those resources, tasks, or other systems that interact with the subsystem and are not part of the subsystem itself. Meanwhile, the decision on whether the OR-join is executed (and therefore, all other preceding processes leading to the join are aborted/ignored) is deterministic due to the explicitness of the workflow components mentioned in (ii).

On Business Process Modelling for Complex Systems

With the growth of the modelling and adoption of many process-centric system designs and analysis, the Business Process Management Initiative(BPMI) [37] started efforts in standardizing them for process design, deployment, execution, maintenance, and optimization in Business Process Management [38]. BPMI began to develop the Business Process Model and Notation(BPMN) that is now maintained by another consortium, i.e. the Object Management Group(OMG®)[1]. OMG® has since focused on technology

[1] The Website of OMG®: About OMG®, http://www.omg.org/gettingstarted/gettingstartedindex.htm

standards since the two groups merged in 2005. BPMN is considered as a graphical standard [48] in Business Process Management(BPM). The BPMN 2.0 version has been released by OMG® last 2011[2]. BPMN provides notations that is posed to be easily understandable for users coming from either the business or technical aspects of system design, management, and analysis. Furthermore, it also provides support for modelling the different control schemes as in other workflows as well as process choreographies [2]. Process choreographies show interactions across business entities. BPMN provides some support to efficiently enact designs by enforcing some readily mappable constructs for execution languages(e.g. Business Process Execution Language) to adopt. It is also notable that the Workflow Management Coalition had recognized the value of BPMN in modelling and has also enriched its support on executable aspects of the notation [39]. BPMN is claimed to be the de-facto standard for process modelling [10]. The UML's Activity Diagram and BPMN are considered the "two most expressive, easiest for integration with the interchange and execution level, and possibly the most influential in the near future" [48].

The backbone of BPMN in providing descriptions of processes and interactions in systems is the concept of functional decomposition [2]. By the use of its elements for describing flow, orchestrations, roles, and artefacts, the decomposition breaks down coarse-grained functional specifications into fine-grained ones, i.e. atomic activities themselves. In its attempt to perform this decomposition, the following criticisms and drawbacks had been observed by researchers in the field of modelling:

(1) **Concept excess.** Concept excess pertains to the possibility of representing the same semantics in multiple ways and has been found to negatively affect understandability [10]. The drawback of frameworks which induce concept excess is having modellers and/or nontechnical users to erroneously use and/or interpret another notation in place of the appropriate one. By doing so with less consideration of the significant effect of having minute alteration in representation, errors can be brought up both during design and run-time specifications of the models. With this, there is a gap that is created between the theoretical aspects and the actual usage in practice of the modelling language.

    With consideration to the latest release of BPMN, i.e. BPMN 2.0, a study in [10] analyzed 585 process models from six different companies with variations on the field of specialization, model sizes, and levels of experience of modellers in creating such designs. The study was conducted in response to the lack of research in analyzing the use of BPMN in actual practice. Their framework on analysis relies on the checking techniques for structure, labelling, and layout that are founded on 35 well-known BPMN guidelines and correctness rules [49–51]. The paper reported errors on the proper usage of muti-merges and the presence of deadlocks [25], i.e. 42% and 22%, respectively, in the models. Some elements of BPMN that are used for orchestrations and process interactions, i.e. throwing message events [2] and

[2] The Website of OMG®: Documents Associated With Business Process Model And Notation^{TM}(BPMN^{TM}) Version 2.0, http://www.omg.org/spec/BPMN/2.0/

main-to-subprocesses associations, have 48% and 86% errors in them, respectively. With these, the authors provided recommendations in avoiding these errors brought about by concept excess. One of these recommendations is the prohibition of the use of multiple arcs that compose implicit splits and joins in the model. They can, in fact, be sufficiently and more simplistically modelled by the use of gateways in BPMN [10].

Although BPMN attempts to provide notational convenience and clarity of control flows in model construction by use of gateways, this effort also becomes a nuisance for model size. For example, by not having activities used as starting points of splits and joins to avoid errors in process representation [10], it is imperative that these activities are limited to have one incoming and one outgoing arc. Resolving control flows, whether it is determined to be a join or split, are therefore performed on gateways. By this scheme of representation, it is unavoidable to have a huge size for the model.

(2) **Lack of support for explicit representation of data and rules in BPMN.**

   (a) methods and attributes that influence interaction flows in processes cannot be explicitly added to models using BPMN. The use of artefacts which lack support in the enactment of the design is insufficient to specify these objects[3] [2]. Through his online article[3], M. Pucher further states that because of this lack of support, the specification of business rules in BPMN is also unsupported. In fact, there is a problem of traceability with respect to the choice of process paths for splits and merges in BPMN. This is due to the lack of explicit information on the models themselves. Note that when the resource and process dimensions are both in place in the frameworks of graphical standards, the specification of these objects become natural and enforced in the design.

   (b) there are no mechanisms to enable flow redirection in real-time using explicit conditions that are based on data. Such redirections are essential for workflow management when maximal use of a component is reached due to bottlenecks in case executions. In real-world settings, resources are limited by nature (e.g. number of personnel, memory capacity of disks used in databases), therefore their corresponding BPMN designs must also reflect this and provide a mechanism for such explicit re-routes at run-time during the enactment of the business models. Mechanisms for resets, when effectively used, can help deal with the limited storage capacity at run-time of system implementations.

   (c) BPM resources are passive elements, i.e. they are not, by themselves, part of dynamic aspect of the model itself. BPMN restricts interaction in orchestrations between tasks/activities and/or events [2]. An attribute of a resource, e.g. memory capacity of a storage disk, imposes that resources are also modelled as with dynamic attributes whose values can be a reason for utility or non-utility

[3] Pucher, M: The Problem with BPM Flowcharts, https://isismjpucher.wordpress.com/2010/10/04/the-problem-with-bpm-flowcharts/ (2014)

of the components. Together with (a), BPMN inherently does not fully capture structure and relations of resources.

(d) Ad-hoc tasks [2, 52] in a subprocess in BPMN models do not have any form of association to this subprocess based on control flows. These tasks can only be executed upon an explicit specification of a choice given by a user through the subprocess they are contained in. This choice pertains to the selected task that is to be performed among the ad-hoc tasks in the subprocess. [52]. Real-world processes need such kind of procedural support but the lack of explicit representation of user-subprocess interactions induces the difficulty in the immediate enactment of ad-hoc tasks from design into their implementation. For the same reason, verification on model properties is hard to do for such kind of abstraction in workflows.

(3) **Problems in process orchestrations and information hiding.** Orchestrations are enabled by messages passed between two parties in a business system. Under the BPMN framework, these message interactions provide another difficulty in design and verification in models. Although multiple points of entries and exits can be ideal to handle and manage bottlenecks, the framework of BPMN contradicts its intent of process abstraction and information hiding. BPMN's framework on functional decomposition limits the view and representation of multi-level, organizational structures and their interactions in models. As stated in [2], "there are no formal investigations possible on the relationship between a business process and its externally visible behavior" for these interactions. In practice, message throwing and main to subprocess interactions lead to improper and/or incorrect orchestrations for 48% and 86%, respectively, in the BPMN models in [10]. BPMN's foundation, alongside the possibilities of concept excess(see (1)) and the lack of explicit representations for data and rules in modelling, easily induce confusion and overwhelms designers. Misaligned interactions [2] between parties in an orchestration is therefore unsurprising in practice. Moreover, it is also notable that full information hiding is unachievable whenever atomicity is not fully supported in process specifications and interactions within and between participants of a business process. Atomicity is eliminated when there exists multiple entries and exits of communication arcs between swimlanes and pools in BPMN models. Structure of processes which are internal to one party is partially revealed to the other. However, when this structure can be encapsulated as one atomic activity, this revelation is minimized or eliminated. Atomicity can also encourage, if not impose, proper nesting and control of main and subprocesses in models. Note that by integrating the resource, process, and case dimensions together in one design, atomicity and encapsulation can be achieved in designing and verification of structural components and their interactions.

In view of implementing multiple message interactions in multiple parties of an orchestration, a similar concept called workflow modules is found in workflow nets [25]. Such modules are disjoint in terms of process specifications and are only linked by messages passed through interfaces. These interfaces correspond to specially-marked places in workflow nets. An interface between two modules is represented as one place in a workflow net where structural compatibility [2] is observed between all interactions when these modules are merged as one net. However, despite this compatibility and the satisfaction of soundness in the individual nets, the integrated net may not be sound [25]. For this scenario, the resulting net contains token/s in its intermediate place/s although one token has already reached a sink that invokes an improper termination of the execution in the net. Additionally, in obtaining the integrated net, the exclusivity of the relation of the interfaces with their associated module is lost. This is another problem in identifying proper grouping of components into their corresponding submodules when decomposition of the net is performed.

(4) **Functional decomposition in BPMN fails against model complexity** The functional decomposition framework that supports BPMN modelling aims at describing complex systems with their atomic activities and interactions. By the decomposition, coarse-grained functionalities of complex systems are broken down into their finest-grained activities. Much of this reductionist approach in complex systems had been criticized by researchers in process modelling. A big part to this criticism stems from the concept of complexity being a systemic property. That is, the structure, behavior, and relationships of components with relation to others that are found at a higher resolutions of complex system descriptions may not entirely be determinable by those found at lower resolutions[3] [53, 54]. For this reason, every level of resolution must be accounted in the design and analysis of models of complex systems. In his online article[3], Pucher particularly pinpoints the lack of ability of BPMN to support the modelling of complex systems due to functional decomposition. Although it can be argued that the relationships of main and subprocesses can be used to model hierarchical structures, the support on effective schemes for design, implementation, and verification of interacting structures is lacking(see (3) above). (Despite the current limitations of BPMN modelling, there are still some process-centric verification frameworks for BPMN. They are implemented with the aid of transforming BPMN notations and structures to some classical models such as Petri nets [55–58], π-calculus [59], Timed Automata [60], agent-based representations [61], etc.).

(5) **Lack of support in (re)design and diagnosis of BPM models** From design, implementation, and management of BPM models, there is a huge consideration for an efficient and traceable transformation of representations throughout these stages. The BPM life cycle [2] covers design, configuration, execution and control, diagnosis, and intelligent redesign of models. Although it is well-known that workflow management is just a component of BPM, a survey in [48] of BPM standards and technologies in practice concludes another thing. [48] states that BPM standards and technologies today are, by their essence, largely WfMS. This claim is also consistent with the findings in [40]. In [40], there are gaps in the

standards and technologies that are available in practice today that make the support of the cycle partial. A few of these gaps are found in the diagnosis and intelligent redesign of BPM models. For example, redesign is still mainly powered by user judgement and limited to “what-if scenarios” in proposing updates to models. To pinpoint substructures(e.g. bottleneck sources and points, workload and resource allocation descriptions, routing probabilities in design and run-time) whose properties result to weaknesses in designs are still mainly dependent on the ability of the modellers to interpret results from data mining and/or verification. These gaps are even apparent in one of the leading commercial BPM systems, i.e. FileNet P8 BPM Suite Version 3.5[4]. The suite was also reinforced with the Process Mining Framework(ProM)[5], an open-source plug-in, that can perform process mining and verification on models that are expressed as Event-Process Chains, Petri nets, Colored Petri nets, etc. However, intelligent redesign of the BPM cycle has yet to be fully supported. As for BPMN, it is already obvious that there are still weaknesses even in the design and enactment stages as discussed in the above items (1)-(4).

Although many issues have yet to be addressed in full to support the entire BPM cycle, it is possible to address some of these issues by putting into models design-driven mechanisms which help manage points and substructures that bring about weaknesses such as bottlenecks. Process mining is mainly dependent on data. However, by enforcing explicitness in data structures and rules rather than have difficult-to-enact artefacts inside models, modellers can provide the workflows and WfMS more effective controls for process utility, routing possibilities, structural and behavioral associations at design and run time. When systems and system parameters still have unknown values such that having monitoring data is a must in defining and specifying workflows, it can be argued that their representations in an initial workflow design can be preset to their real-world capacities, e.g. maximum disk space of a storage device. When real-world data shows that actual processes utilize much more of this capacity, therefore the initial design would already ideally have a reroute mechanism at the onset since rules were already made explicit.

A View of the state of UML Notations

From these current problems in modelling, it is noticeable that a large portion of them come from the fact that modelling and analysis are mainly process- or data-centric. This is also due to the specificity of model construction that focus solely on describing systems with only one or two workflow dimensions considered. These modelling and analysis framework are akin to characterizing system structure and behavior by vertical and horizontal abstraction [2]. That is, for vertical abstraction, high-level definitions of

[4]The Website of IBM: FileNet P8 BPM Suite Version 3.5 Documentation, ftp://public.dhe.ibm.com/software/data/cm/filenet/docs/p8doc/35x/p8_35x_Release_notes.pdf (2011)

[5]The Website of Process Mining: ProM, Process Mining Group, Math&CS department, Eindhoven University of Technology, http://www.processmining.org (2016)

systems such that those expressed in textual, informal form are parsed first. Then the sets of information pertaining to process, resource, and cases are isolated individually and analyzed thereafter. Much of the characterization in literature focus on the process dimension such as those modelling and analysis done on Petri nets, workflow nets, π-calculus, UML, and BPMN. Examples to these undertakings focus on ensuring that designs in these models satisfy certain conditions of workflow properties such as soundness, deadlock-freeness, etc. [25]. (The formal definitions for these properties are given in Section 2.) Meanwhile, horizontal abstraction looks at systems from its concrete to the most abstract realizations of systems. That is, instances are identified such as those groups of data derived from every instance of the enactment of workflows to the workflow structure itself. By process and data mining, systems, subsystems, and their efficiency can be analyzed and described by the patterns from monitoring data which are gathered at different levels of design resolution.

Vertical abstraction is readily seen in the framework of UML modelling. In the latest version of UML released last June 2015, i.e. UML 2.5[6], there is still a clear categorization of models. This categorization captures either structure or behavior of systems as seen in Figure 1.2. Note that the items in blue in the figure are not considered a part of the official taxonomy of UML 2.5 diagrams. For example, there are no formal definitions for the Manifestation Diagrams in UML 2.5. Manifestation diagrams intend to show manifestations(i.e. implementations) of system components by artefacts(e.g. source files, binary executable files, text documents, etc.) and their internal structures. With the lack of this definition, the manifestation of components are represented by elements of component diagrams or deployment diagrams. Meanwhile, in building a Network Architecture Diagram, UML does not have a standard and specific elements which are related to networking and network architectures. This diagram, despite its lack of formal definitions in the release, are considered as Deployment Diagrams which are used to show logical or physical network architecture in systems.

It is emphasized in aforementioned UML release that mixing of different kinds of diagrams is allowed to combine both structure and behavior in modelling. This helps illustrate the execution of cases/functionalities that a system provides. However, mixing, in this sense, is more precisely described as a literal embedding of one UML diagram in another, e.g. a class diagram embedded in a state machine diagram such as in [41]. With the mixing of notations from different types of diagrams in one model, the problems on explicitness of interactions, rules, and verification of model properties naturally arise.

An Opportunity of Integrating Workflow Dimensions

First called as "Objectory Process-Specific Extensions", the Robustness Diagram(RD) was just partially included as an appendage in the UML standard [28]. However, the components and basic rules of construction of an RD provide a mechanism to partially support a unified modelling scheme that uses all three workflow dimensions. When the

[6] OMG® Unified Modeling Language™, The Website of the UML 2.5 Release Documentation: UML Diagrams Overview, http://www.uml-diagrams.org/uml-25-diagrams.html (2015)

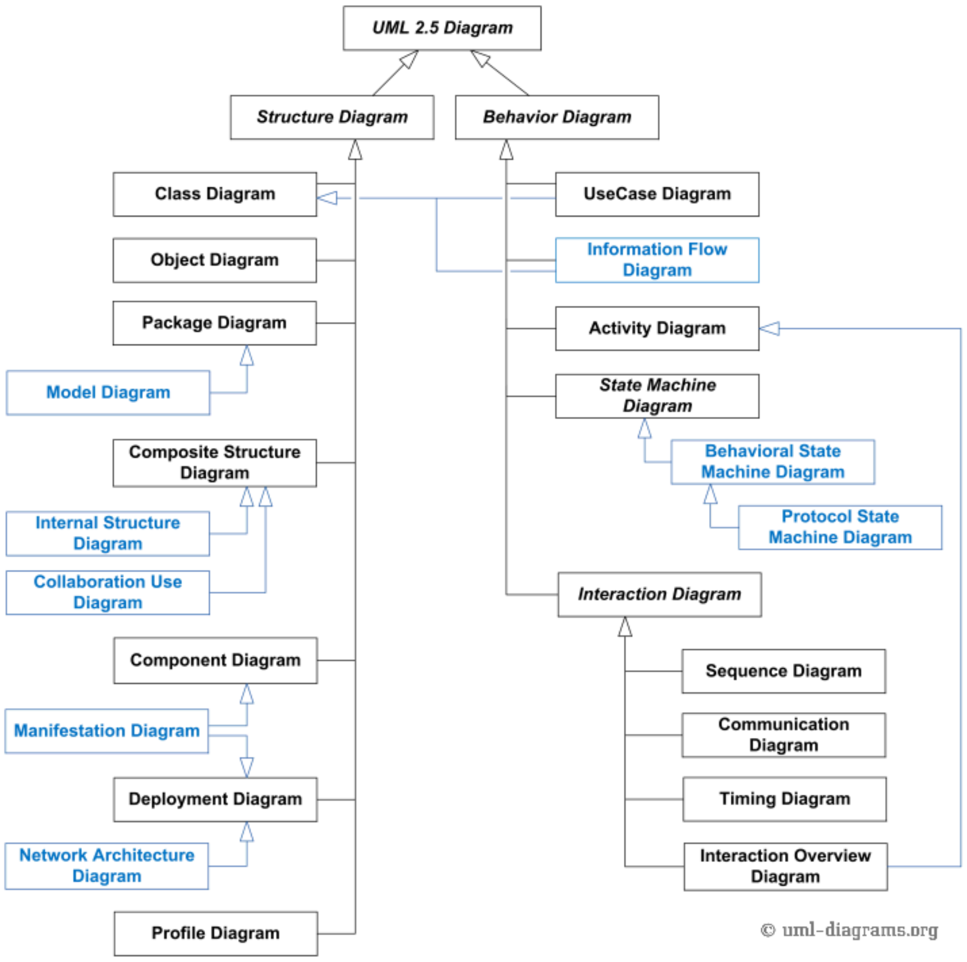


Figure 1.2: The UML 2.5 Diagrams and their Categories. (Image from the UML 2.5 Release Documentation.

ICONIX Framework was introduced [30], it highlighted on the use of Robustness Analysis. Its focus was on the use of RDs from design- to implementation-level representations of a system. Because of this streamlined modelling framework that the ICONIX provides, it is imperative that researchers look at its diagramming core, i.e. Robustness Diagram. This is done to identify some of the diagram's weaknesses and provide solutions to them as well as opportunities for holistic representations of complex systems where all three workflow dimensions are in place.

Developed by Doug Rosenberg, the ICONIX Framework [30] offers a minimalist, streamlined approach for Use Case-driven UML modelling that is centered at using Robustness Diagrams. It focuses on the use of a subset of the UML diagrams for object-oriented analysis and design. Its major component in accomplishing its task is the usage of robustness analysis. Robustness analysis provides capabilities to remove ambiguities in use case descriptions by ensuring that domain models are consistently represented both in structure and behaviour in the UML diagrams.

Shown in Figure 1.3 is the ICONIX Framework. This framework shows a Robustness Diagram bridging the gap between modelling the static information of domain models as represented by structural diagrams and their dynamic information as represented by behavioral diagrams.

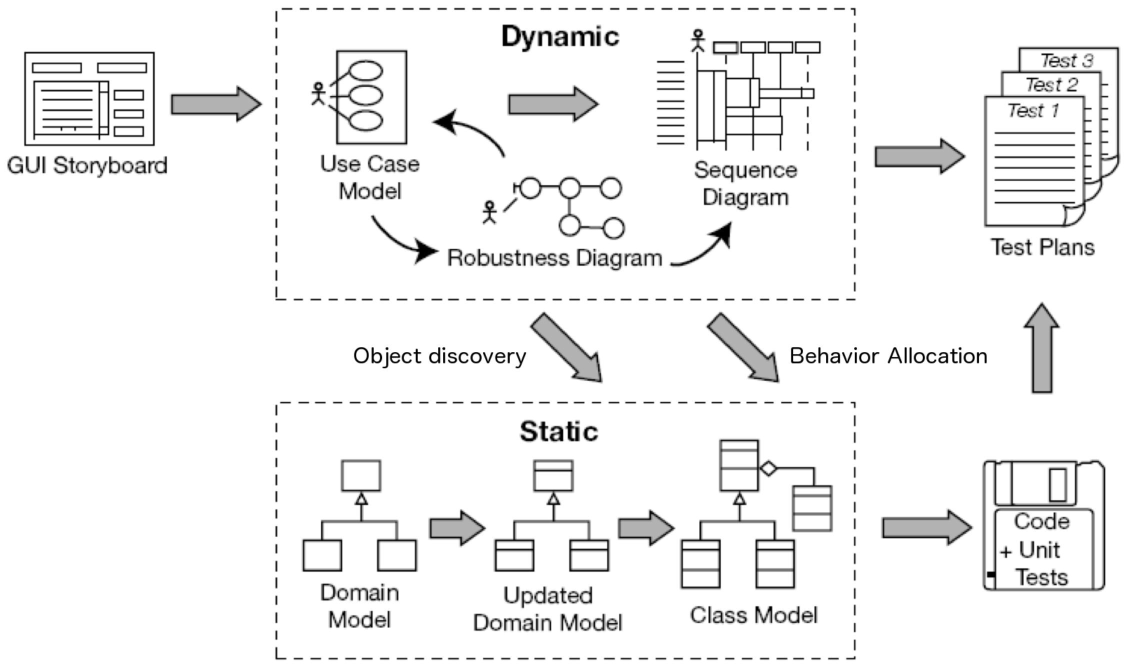


Figure 1.3: The Use Case Driven Object Modelling of the ICONIX Framework [30]. (Image from [29]).

Unlike other UML diagrams, a RD is capable of providing a holistic view of a domain model. The current notations available in RD diagramming can, at the least, be used to show domain objects, i.e. system resources, that can directly participate and support the execution of tasks/ activities. A domain object can be either a boundary or an entity object. Boundary objects represent resources which can interact with the system's environment and all other resources which are internal to the system, i.e. entity objects. Tasks/activities are represented by RD controllers. A controller can therefore abstract nested subprocesses where the controller itself acts as the subprocess' interface to the other components of the system. This means that the subprocess is only enacted when its controller is reached.

In this research, we take advantage of these different categories of RD components to establish structures, i.e. a class/object view, and therefore, roles and hierarchies. With these natural grouping, the concept of cancellation regions can easily be adopted with less utility of space and with distinguishable components for reset controls. This further creates opportunities to have explicit and well-defined controls in interactions of the group between its members and across components not belonging to the group but are part of the system. For such a special type of group with reset schemes that are invoked from system interactions, we call it as a **reset-bound subsystem(RBS)** of RDLTs. RBS themselves can, in essence, be abstracted further as one activity, i.e. a controller, in a RD. This can be used to include volatility in the RD diagramming framework.

In addition to the specification of the resource-process dimension in the RD diagramming, we propose to complete the specification of a system in RDs by putting explicit syntax and semantics for case execution and management in the designs. We do this by enacting actionable rules of execution. They will be based on settings/conditions of system parameters/input, bounding parameters for execution, system topologies, and typing of structures inside the system. For the first and second bases of execution, RDs provide opportunities to explicitly incorporate parameters/input information through system attributes. For the third and fourth bases, connectivities, attribute-driven constraints, and typing of components provide means to realize semantics of joins and splits. At the least, these information integrates the following execution control schemes in BPMN and other workflows: (1)topology-driven, (2)gateway-driven, (3)artefact- and user-driven, and (4)local and nonlocal information(cancellation region)-driven process flows. With complete specifications of systems using all three dimensions in the design, we further propose algorithms to aid in effective isolation of resources and tasks which execute a particular case. In effect, activity profiles that support the completion of their corresponding cases are isolated. Needless to say, we propose to provide effective construction and verification schemes for RDs containing multiple activities in one design.

With these support and opportunities for further development in RDs, various mathematical, scientific and business systems can be represented at different levels of resolutions of complexity. Because RD components are easily distinguishable as either a resource or process type, the interpretability of RDs can cater to technical or nontechnical users. For the same reason, RDs provide opportunities of efficient and effective requirements traceability from design specifications to their corresponding implementations. Despite these opportunities, the current state of RDs and RD modelling for both theory and practice is still pretty much in its infancy. For the meantime, it is still treated as a scratchboard diagram in the initial stages of requirements definition and specification. Their usage, however simplistic as of the moment and most of the time for illustrative purposes only, can be found in processes such as a reservation system[7], registration system [80], and human-computer interaction systems [74]. RDs in [74] were also further used to measure the traceability of requirements across structural and behavioral UML diagrams. With this significant absence of a pool of literature and use of RDs, there are no definitions and verification schemes of properties to check their quality and viability of system representation under a multidimensional view of workflows.

[7] The Website of IBM: Robustness Diagram or Ideal Object Diagram Version 11.4.3, IBM Knowledge Center. http://www.ibm.com/support/knowledgecenter/SS6RBX_11.4.2/com.ibm.sa.oomethod.doc/topics/c_Ideal_Object_Diagram.html (2016)

1.2 Problem Statement

In building models of complex systems [76], the recent releases of BPMN and other existing workflow standards and technologies still do not resolve many gaps in effective modelling of such systems. As discussed in Section 1.1, much of these gaps are brought about by the lack of support in modelling and verification of workflows that utilize all workflow dimensions [25,26], i.e. process, resource, and case, in one model. Even with the latest release of the UML 2.5 last June 2015[6], these gaps have yet to be addressed. The distinctive classifications of the UML diagrams, i.e. structural and behavioral diagrams particularly impose design restrictions in constructing designs with such integrated environment. For example, structural diagrams such as Class and Deployment diagrams can only show abstractions of the static information of systems. Meanwhile, behavioral diagrams such as Use Case and Activity Diagrams show abstractions of the behavior of the static elements of systems being modelled. The UML's Activity Diagram, for example, has been shown to be advantageous in the context of workflow specification against commercial Workflow Management Systems as it can effectively provide support for control and data specifications [42]. On the side of process modelling [44], Activity Diagrams were also shown to lack support for the design of resources and cases [42] in systems. For these concerns, the Robustness Diagrams of the Unified Modelling Language [29,30] are still highly underdeveloped in supporting such integration of the workflow dimensions for modelling complex systems. Although the ICONIX Framework [30] attempts to provide a bridge between the two diagramming classifications of UML through the use of Robustness Diagrams, formalisms are lacking for such kind of construction and for requirements traceability from requirements definition to implementation of complex systems. By this state of RD-based modelling, along with the existing foundations of BPMN and workflows which are either process- or data-centric, there is an absence of definitions of model properties and their corresponding verification schemes for such kind of multidimensional workflows. However, it is notable to mention that existing verification schemes are already well-established, albeit dimension-specific, and provide a ground and inspiration for developing similar workflow properties for RDs.

Moreover, in modelling complex systems, i.e. those which inherently possess a large number of states that is induced by chaos and bifurcations [76, 77] due to their nonlinear processes, there is a need for supporting designs of persistent and volatile components due to the said nature of these systems. Persistence and volatile structures support the need for memory – whether for state recovery and/or state modification due to irregular or regular behavior of systems. Needless to say, state enumeration in workflows poses problems for real-world complex systems. This is due to the exponential space explosion in enumerating all of the states as in Continuous Petri nets [79] used for modelling manufacturing systems [81]. For example, a model checking tool known as SPIN [82] is limited to enumerating at most 1 million states [83] from workflows. Although the number of states that is derivable from a model is unquestionably high, the presence of bounding values and other control mechanisms can reduce this statistic significantly. This can be achieved through aggregation of values and/or states and by explicitly imposing reroutes using this mechanism when necessary(e.g. under the presence of bottlecks). These bounds and control mechanisms can be embedded in persistent and volatile components in workflows either in their structural profile or in the attributes associated to these components. Because these structures provide means for storage of data regarding the execution of processes and workload distribution for resources in case handling, they can be used to lessen or remove certain weaknesses such as bottlenecks at run-time in models. In the current results of literature, there is an absence of formalisms in effective construction, implementation, and model verification of workflows wherein all three workflow dimensions are in place and whose structures contain persistent and volatile components.

1.3 Aim of the Work

The general aim of this research is the formalization of effective modelling and verification of complex system representations which use all three workflow dimensions, i.e. process, case, resource, under the Robustness Diagram with Loop and Time Controls(RDLT) specifications. RDLTs are extensions proposed for Robustness Diagrams. Due to the integration of the workflow dimensions into one design, multiple functional specifications arise in RDLTs. Each of this functional specification imply a set of tasks constituting one activity profile in the system. This profile describes the execution of a case through the support of a set of resource and process specifications in a model. We establish a framework that would support designs and model verification of multi-input, multi-output, and multi-activity RDLTs. Activities in complex systems might include requirements for persistent and/or volatile structures in their representations. This research addresses the lack of literature to address both the integration of all workflow dimensions and the persistence/volatility requirements for modelling complex systems. We aim to establish mechanisms so as to support the creation and handling of these persistent and volatile components in RDLTs. We provide design specifications that illustrate effective embedding of such structures in the designs. These aims are supported by the the concept of explicit typing of arcs and vertices that will impose restrictions on reachability and process flow controls for substructures and components in the models. We propose metrics for reachability and synchronicity of task executions in RDLTs with considerations of the typing and connectivity profiles of components. One key difference of this research with the results in literature with respect to establishing means for synching of tasks is that synchronizations are not just imposed by connectivity. For this research, we collectively account connectivity, typing of arcs and vertices and their attribute values, and dealing with these information either individually or collectively to identify and measure aspects of synching of task execution. We propose algorithms to isolate individual activities from RDLTs by accounting all of these information during activity extraction for each output vertices in the models. We propose instances of an RDLTs for real-world complex systems and illustrate activity extraction from them.

Meanwhile, we propose different model properties to be used for modelling and model verification for RDLTs with consideration of the integration of all workflow dimensions in the designs. We focus on adopting and extending the properties that are established in literature, e.g. deadlock-freeness, boundedness, free-choiceness, soundness [25], to the proposed RDLTs. (The formal definitions for these properties are given in Section 2.) Because of the introduction of persistence and volatility in our proposed modelling framework, we formulate these properties in RDLTs to address these two aspects of representing complex systems. The formulation of these properties mainly rely on identifying different types of a **Point-Of-Interest**(POI) in the models. A ***POI*** is a vertex in a models wherein possibilities of deadlocks, delays in reachability, task repetitions and synchronizations are present. We propose formalisms to identify these points and relate their structural placements, connectivities, and their relevant graph attributes with other components in RDLTs. Furthermore, we establish different types of substructures and neighborhood relative to the presence of these POIs. We prove and verify model properties in these subgraphs and relations and establish generalizations of the properties for the entire model. The introduction of an extended RDLT is proposed in this research to establish these generalizations of the properties. Finally, we also aim to establish hierarchies amongst these properties and provide polynomial time and space verification of such relations in RDLT substructures and the whole model itself.

More specifically, this research aims to achieve the following,

1. Propose a framework for designing and handling multidimensional RDLTs with support on persistence and volatility in structure and behavior. This framework shall account abstractions, control schemes, and metrics for effective complex systems modelling. Collectively, they will address modelling and verification under the construction of an integrated representation of all workflow dimensions, i.e. process, resource, and case. More specifically, the proposed framework shall address the following,

   a) **Information Abstraction.**

      i. support for modelling static and modifiable components in RDLTs for isolating activities embedded in RDLTs. Apart from connectivities, the model shall support the use and modelling of (a) multi-type vertices, arcs, and subgraphs, (b) time- and constraint-bound vertices, arcs, and subgraphs, (c) system parameters and constraints imposed on them, e.g. bounds for temperature, pressure, etc., (d) combinational relationships between two or more constraints, e.g. disjunctions(i.e. splits), conjunctions(i.e. joins), that are bound to the typing of RDLT components,

      ii. support the modelling of explicit associations and controls of the system components and the tasks that they execute. By these associations, subgraphs in the model which correspond to volatile structures can be discriminated from the others, and

      iii. support the modelling of hierarchical relations of structures and processes with the identification of system components and their associations with the tasks they execute.

b) **Control Abstraction.**

   i. enable the modelling of control schemes for sequential, parallel, conditional(joins and splits), iterative, and time-bound resets in the RDLT attributes,

   ii. support sharing and exclusivity of information that are embedded in modifiable RDLT attributes. This shall enable or disallow the execution of processes despite the connectivities and type-related relations of the static components in RDLTs, and

   iii. enable embedding of multiple activities through the support of information sharing, exclusivity, and time-bound attributes in RDLTs.

c) **Persistence and Volatility.** Provide mechanisms to maintain persistence and enable volatile structures and behavior in RDLT modelling such as

   i. storage for information that control vertex reachability and synching of task executions in RDLTs, and

   ii. definitions of types of RDLT vertices, arcs, and substructures(referred to as “reset-bound subsystems”) that impose resets on the values of RDLT attributes when they are used in some time steps during the execution of activities.

d) **Computable Metrics for Reachability, Delays, and Synchronizations.** Formulation of different types of a **Point-of-Interest(POI)**, i.e. vertices in the models wherein possibilities of deadlocks, delays in reachability, task repetitions and synchronizations are present, as well as types of neighborhood that are defined over them. More specifically,

   i. definition of a POI, called a **Point-of-Delays(POD)**, which is a vertex in RDLTs where there is a delay of its reachability/use that is caused by the settings of the attribute values in the model,

   ii. definition of a POI, called a **Point-Of-Reentry(POR)**, which is a vertex in RDLTs that is the first point of reuse of components/substructures/tasks,

   iii. definition of a POI, called a **Point-of-Synching(POS)**, which is a vertex in RDLTs where synching can be imposed for succeeding tasks accounting topological restrictions and/or constraints based on RDLT attribute settings,

   iv. formulation of definitions for metrics on reachability, delays, and synchronicity for the aforementioned POIs in RDLTs,

   v. provide computations of the metrics in polynomial time and space complexity

   vi. establish types of structures and neighborhood relations using these metrics and attribute settings, e.g. nonself-controlling structures which relate to the presence of deadlocks. These deadlocks can be brought about by multiple constraints associated to type-alike components of RDLTs.

2. **Real-world application.** Provide instances of RDLT models of real-world complex systems to illustrate designing and handling of RDLTs for such systems. We shall focus on building RDLTs for energy systems, e.g. adsorption chiller. We model its reactor chambers, condenser, evaporator, valve system, and different modes of operations and illustrate how one of its activities relating to one of its modes, i.e. adsorption, is extracted using our proposed algorithms. We describe substructures in the model with relation to different model properties proposed in this research.

3. **Model Properties and Verification.** Propose formalisms of model properties and verification schemes for RDLTs with persistence and volatility. We adopt and extend model properties found in literature to the construction and semantics of designing and handling of multi-activity RDLTs. In particular, we establish properties, relationships, and computability of metrics supporting their identification in RDLTs with certain focus on the following:

    a) reachability of vertices bound to topology and constraints derivable from RDLT attribute settings,

    b) boundedness and delays of reachability for PODs that are influenced by the presence of POR and POS vertices and with different types of structures induced by these points of interests in RDLTs

    c) free-choiceness for different types of structures mentioned above. In addition to focusing on topology to define free-choice structures for PODs and the vertices whom they share a common parent, we establish a relation of these vertices with respect to their POS ancestors to define another aspect of free-choice structures. This aspect considers the types defined over arcs, vertices, and time-bound attributes in RDLTs.

    d) soundness in multi-activity workflow designs that includes persistence and volatility. We propose ancillary properties that can verify for soundness by using information on the structure of RDLTs and by algorithms based on graph reduction operations. These operations can be done on substructures involved in individual activities or on the extension of the multi-activity RDLT itself.

    e) hierarchies and other relations between properties in RDLTs, and

    f) proposing means of verification of model properties in substructures and the entire model itself within polynomial amount of time and space.

1.4 Methodological Approach and Structure of Work

In this section, a layout of the research activities that are used in the accomplishment of the research objectives is provided. They are enumerated based on the usage of the proposed concepts and schemes from design to verification of RDLT models for complex systems. Furthermore, the structure in discussing these activities is given by citing the corresponding sections in this manuscript where they are provided. An activity diagram that shows this ordering is given in Figure 1.4 as a graphical guide to the readers.

To create RDLT models for real-world complex systems, we propose RDLT constructs and control schemes for effective and traceable designs in multidimensional workflows with a supplementary task of model verification. Based on the constructs themselves, static information such as task specifications, objects/resources, parameters, constraints, subgroups supporting the execution of some cases are readily identifiable from the model itself. Meanwhile, based on the control schemes, modellers are given rules and samples of design specifications to handle the creation of process flows. These shall effectively support multi-activity, -input, and -output RDLT despite the presence of persistent and volatile structures in the design requirements. The formalisms for the construction of multi-activity RDLTs and the specifications of embedding the different process flows and controls are provided in Section 3.1.

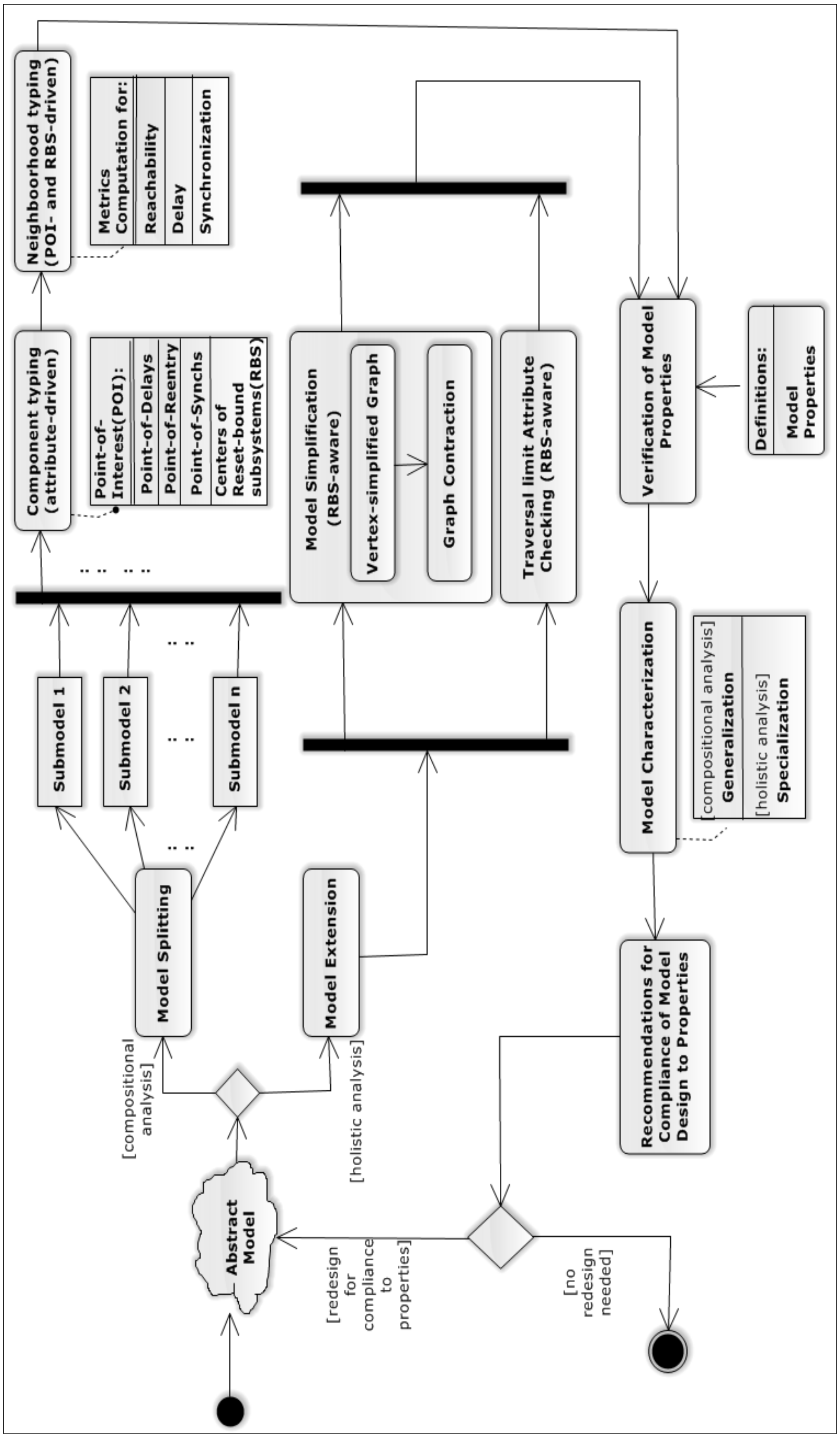
Given an RDLT model of a real-world complex system, the proposed framework of model analysis and verification comes in two approaches, i.e. *holistic* and *compositional*. These approaches are represented as two independent flows emanating from the *Abstract Model* node in Figure 1.4 and only merging at a time when the RDLT and/or its submodels are verified. These approaches are detailed as follows:

Firstly, the **holistic approach** in designing and verifying RDLTs takes in an entire specification of a system. An RDLT, possibly designed with multiple activities and time-bound attributes, is analyzed using its entirety. In this approach, an *extended RDLT* is created from the given model. Using the extended RDLT of a model, the framework produces a *vertex-simplified RDLT* that only uses the information of the constraints that are bound to system parameters. The model simplification also isolates each RDLT subgraph wherein time-dependent and reset-bound attributes are present. This subgraph is proposed in Section 3.1 as the *reset-bound subsystem*(RBS). The simplification encapsulates every RBS with another representation, however, preserving pertinent information with regard to interactions. Representation, for this case, would account to the connectivities established between persistent components of the RDLT and some vertices belonging to every RBS. Furthermore, the representation abstracts all volatile components in the subgraph while summarizing the RBS's topological information. Summarization discounts constraints imposed by attribute values of the arcs in every RBS. We refer to this simplification of RDLTs as a level-1 simplification of the model. A level-2 simplification of the model uses each RBS as input to the vertex-simplification process. The formalisms of extended and vertex-simplified RDLTs, the simplification process and rules, and the resulting multi-level perspectives of RDLT models and derivations thereof are formally defined in Section 8. Using graph contraction rules on the simplified models, the proposed framework assesses whether the static information in the model does not impede with the continuous execution of processes that support the accomplishment of every activity in the RDLT. Whenever this is the case, the contraction reduces the entire model itself into a single vertex. The governing rules and the operations in graph contractions in RDLTs are proposed in Section 3.5.

Meanwhile, because the bounded repeatability of tasks that influences the modification of the time-dependent attributes in the design, the framework performs a separate analysis for them. Note that this separation of analysis can be done since we can view repeatability of tasks separate from the satisfiability of the constraints. The latter are examined throughout the process of model simplification and contraction. For the former analysis, models are assessed such that the total number of times the persistent/volatile components are used in an activity corresponds to the total number of times their consequent persistent/volatile components are used in an activity. Graphically speaking, this analysis accounts indegrees and outdegrees of vertices, their associated arcs, and the types that are defined over them based on the attributes in the model. Section 3.5 provides the definitions and steps that support these analyses on RDLTs. Collectively, the results in this holistic approach can be used to verify the compliance of the entire model with RDLT with respect to the properties that are proposed in this research. However, it is notable that the results from the holistic approach is usable to determine the compliance of substructures of RDLTs with respect to the workflow properties that are formulated in this research. Moreover, they can also be used to produce lists of substructures or aspects of the RDLT which can be targets of modification in the model. They can be used in redesign whenever users want specific properties to be maintained in these substructures.

On another hand, the **compositional approach** proposed in this research for model construction, analysis and verification follows a bottom-up scheme. In contrast with the holistic one, it splits an entire RDLT into its maximal substructures. Each maximal substructure corresponds to a set of RDLT components that support at least one activity corresponding to some case execution in the model. Each of these maximal substructures is analyzed and verified. This is done by identifying every **Point-of-interest**(POI) and studying the different types of substructures around this point. These POIs are vertices where constraints associated to their incident arcs are present. These constraints affect reachability of the POIs such that delays can be induced by them. Another consequence of these constraints is synchronicity in using the tasks associated to these vertices, whether desired by modellers or not. Moreover, the attributes in vertices and arcs in an RBS are analyzed. Because of the possibilities for modifications on graph attributes induced by such RBS, samples of design specifications to effectively designing and maintaining them in RDLTs are also given in this research. Compared to the holistic approach, the compositional approach checks POI- and RBS-based neighborhoods rather than dealing with graph components based solely on topology and interactions. The second approach also provides metrics from the information that are gathered from these local neighborhood to establish a characterization of the models. These metrics provide bounds for reachability, delays, and synchronizations whenever they exist in RDLTs. Each of the maximal substructures can be verified and these substructures can be collectively used to conclude generalizations for the entire model with respect to compliance of RDLT properties. The formalisms of maximal substructures and how they are efficiently determined from multi-activity RDLTs are provided in Section 3.5. The definition of POIs, RBS, the neighborhood relations, and typing of components that they establish are given in Section 3.2.

For both holistic and compositional approaches, their analyses rely on the pool of model properties that are defined in this research for RDLTs as shown in Figure 1.4. These properties are discussed in Chapter 3. Hierarchies and relationships among these properties are proved in this research. The summarization of the hierarchies and other relations of the proposed model properties can be seen in Section 3.7. We provide instances of RDLT models for a real-world complex system, i.e. an energy system, and illustrate activity extraction from it. Throughout the discussions of properties for RDLTs, these instances are used to illustrate the verification process of these properties. Finally, the computational complexity in extracting activities and model verification is also given.



Abstract Model
[compositional analysis]
[holistic analysis]
Model Splitting
Model Extension
Submodel 1
Submodel 2
Submodel n
Component typing (attribute-driven)
Point-of-Interest(POI):
Point-of-Delays
Point-of-Reentry
Point-of-Synchs
Centers of Reset-bound subsystems(RBS)
Neighboorhood typing (POI- and RBS-driven)
Metrics Computation for:
Reachability
Delay
Synchronization
Model Simplification (RBS-aware)
Vertex-simplified Graph
Graph Contraction
Traversal limit Attribute Checking (RBS-aware)
Verification of Model Properties
Definitions:
Model Properties
Model Characterization
[compositional analysis] Generalization
[holistic analysis] Specialization
Recommendations for Compliance of Model Design to Properties
[redesign for compliance to properties]
[no redesign needed]

Figure 1.4: Compositional and Holistic Analysis and Model Verification of RDLTs

2 Review of Related Literature

In this chapter, a detailed look at the specifications of different classical workflows, Business Process Model and Notation(BPMN), the Unified Modelling Language with emphasis on Robustness Diagrams, and their supporting platforms and technologies are given. These are additionally reviewed and analyzed for their capabilities, benefits, and weaknesses in terms of complex system representation. Model properties for a process-centric verification on models, particularly on workflow nets, are highlighted in this chapter. A discussion about resource- and process-centric verification in BPMN is also shown. The relationship of BPMN models and workflow nets are given to show how this verification is performed on the former set of models. This chapter also looks into modelling under the perspective of the approaches model abstractions, i.e. vertical and horizontal. For the first approach, one illustrative discussion is given by the use of the categorization of models/diagrams in the Unified Modelling Language. This shall highlight the innate difficulty in providing modelling frameworks that have an integrated environment for embedding all three workflow dimensions in one model. Under the second approach, a detailed look at the modelling and management of process instances in workflows are provided. Lastly, a discussion of the current formalisms and recent literature for the UML's Robustness Diagram are provided. The challenges on modelling and verification in this diagramming framework are given with consideration of its potential to support an integrated platform for all three workflow dimensions.

2.1 Workflows and Model Properties

On Petri Nets and Workflow Nets

Petri nets are often quoted to be the basis of the basic workflow patterns which many modelling frameworks are based and/or inspired from. From graphical to execution standards in modelling, the influence of Petri nets can be found in Business Process Modelling technologies, UML modelling, YAWL, event-driven process chains, among others [48]. Therefore, it is only fitting that the theory of modelling and verification for systems as represented by their Petri nets model is discussed in this research. The definitions in this section are mainly taken from [25, 84] with additional descriptions supplemented for explanatory purposes. Although Petri nets and Workflow nets only highlight the process and case dimensions, they provide a comprehensive theoretical foundation that is needed in this research. In this research, we use this foundation to help build up theories on modelling and verification of models wherein all three workflow dimensions are used. For this section, soundness and free-choice properties are the main foci because there are many other properties that are implicated by them [25]. Furthermore, these two properties also have relations to each other. That is, the latter is usable to verify in efficient time the former in workflow models [84]. This research takes inspiration from these relations to build hierarchies and relationships among RDLT properties under the integration of all workflow dimensions in one model.

Definition 1 (Petri nets and Markings) *A **Petri net** is a directed bipartite graph where its nodes represent **transitions**(i.e. tasks) and **places**(i.e. conditions for task execution) such that a place p is considered a pre-condition(post-condition) for a task t if there is a directed arc from p(t) to t(p). p is called an **input place(output place)** of a transition t if p is a pre-condition(post-condition) for t. The notations •t and t• refer to the sets of input and output places of t, respectively. Similarly, p• and •p refer to the sets of transitions that have p as an input and output place, respectively.*

*A place(condition) p is satisfied if there is at least one token in p. A transition t is **enabled** if every input place of t has at least one token. If t is enabled, then it may **fire**. Firing t means that every output place of t gets one token. Furthermore, every input place of t loses one token. A distribution of tokens in all the places is called a **marking**(i.e. state). A **dead marking** means that the marking does not enable any transition. A **dead transition** t means that t can never be fired.*

Formally, a Petri net is a 4-tuple (P, T, F, M_0) where P and T are finite sets of places and transitions such that $P \cap T = \emptyset$, $F \subseteq (P \times T) \cup (T \times P)$ is a set of flow relations, and $M_0 : P \to \mathbb{N}$ is its initial marking. A marking is defined by the function $M : P \to \mathbb{N}$ where $M(p)$ denotes the number of tokens in $p \in P$. Transitions and places are graphically drawn as rectangles and circles, respectively. Meanwhile, tokens are drawn as black dots inside the circles. When a transition t is fired at a marking M, the firing produces a successor marking M' of M (written as $M \xrightarrow{t} M'$) defined for every place $p \in P$ where

$$M'(p) = \begin{cases} M(p), & \text{if } p \notin \bullet t \text{ and } p \notin t\bullet, \text{ or } p \in \bullet t \text{ and } p \in t\bullet, \\ M(p) - 1, & \text{if } p \in \bullet t \text{ and } p \notin t\bullet, \\ M(p) + 1, & \text{if } p \notin \bullet t \text{ and } p \in t\bullet. \end{cases}$$

There are four routing schemes in Petri nets [87], namely, (a)sequential, (b)parallel, (c)conditional, and (d)iteration as seen in Figure 2.1. From these schemes, different types of joins and splits can be enabled.

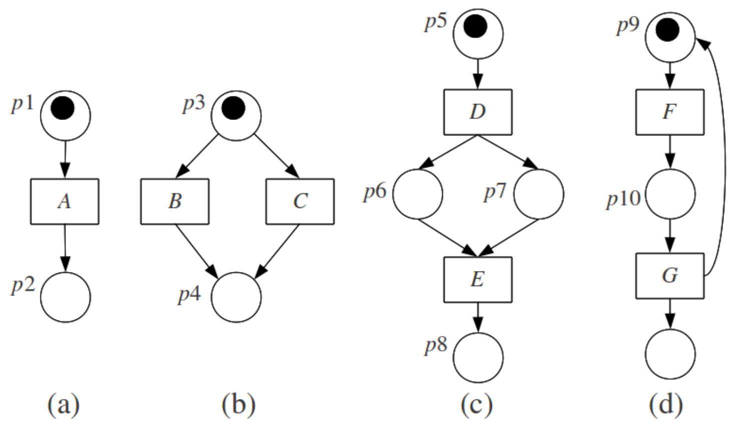


Figure 2.1: Routing schemes: (a)sequential, (b)conditional, (c)parallel, (d)iteration (Image source: [87])

Figure 2.1.(a) shows a sequential flow from using the token in $p1$ to enable A. A fires thereby removing the token from $p1$ and producing one in $p2$.

Figure 2.1.(b) shows a conditional routing control whereby a choice for firing is made on either B or C which are both enabled from the token in $p3$. This structure forms an instance of an OR-split. With this, only one of the transition fires and then a token is produced in $p4$ and the token in $p3$ is removed. Note that the structure of the transitions B and C with respect to $p4$ is an OR-join. Only one of the process flows from $p3$ to $p4$ is executed to put one token in $p4$ at the outset.

Figure 2.1.(c) shows the AND-split where a token in $p5$ enables and causes D to fire such that both $p6$ and $p7$ contain one token each. Furthermore, the presence of tokens in every input place of E enables E. Note that the structure of $p6$ and $p7$ with relation to E is an instance of an AND-join. That is, all conditions(represented by $p6$ and $p7$) to fire E must be satisfied(represented by the presence of at least one token in $p6$ and $p7$).

Meanwhile, Figure 2.1.(d) shows an iteration of process execution as one token is repeatedly produced in $p9$ by a series of firing of F and G. For this Petri net, the output place of G accumulates tokens indefinitely per iteration of the execution. This structure can be used to simulate counting of the number of executions of tasks. There will always be exactly one token that is removed or present/produced in $p9$ and $p10$ throughout the series of firing of F and G. However, this will not be true if the output place of G is assigned to be the sink place such that the execution of the processes stops at the first firing of G. Upon this termination, there is also one token in $p9$ and one in the output place of G.

For brevity, the following notations are used for the marking M_i,

(1) $M_1 \xrightarrow{t} M_2$: M_2 is produced from M_1 with the firing of the transition t,

(2) $M_1 \rightarrow M_2$: there exists a transition that produced M_2 from M_1,

(3) $M_1 \xrightarrow{\sigma} M_n$: the firing sequence $\sigma = t_1 t_2 \dots t_{n-1}$ that is used to produce M_n from M_1 by a series of firings $M_1 \xrightarrow{t_1} M_2$, $M_2 \xrightarrow{t_2} M_3$, …, $M_{n-1} \xrightarrow{t_{n-1}} M_n$.

The notation $M_1 \geq M_2 (M_1 > M_2)$ means that $M_1(p) \geq M_2(p)(M_1(p) > M_2(p))$ for every element $p \in P$.

Soundness in Workflows

Definition 2 (Reachable markings) *A marking M_n is **reachable** from M_1 in a Petri net, denoted as $M_1 \xrightarrow{*} M_n$, if and only if there is a firing sequence σ such that $M_1 \xrightarrow{\sigma} M_n$. Note that the empty firing sequence is allowed, i.e. $M_1 \xrightarrow{*} M_1$.*

Definition 3 (Deadlock-free) *A Petri net is **deadlock-free** if every reachable marking enables some transition, i.e. if no dead marking can be reached from the initial marking.*

Definition 4 ((Elementary) Paths, Conflict-freeness) *A path $C = x_1 x_2 \dots x_n$ is a series of nodes in a Petri net from x_1 to x_n such that $(x_i, x_{i+1}) \in F$ for $1 \leq i \leq n-1$. The set of nodes that is found in C is denoted as $\alpha(C)$. C is elementary if and only if $x_i \neq x_j$ for every $x_i, x_j \in \alpha(C)$ where $i \neq j$. C is conflict-free if and only if for any transition $x_i \in \alpha(C)$ and $j \neq i-1$, $x_j \notin \bullet x_i$.*

Definition 5 (Strongly connected) *A Petri net is strongly connected if and only if, for every pair of nodes x and y in the net, there is a path from x to y.*

Definition 6 (Workflow nets) *A Petri net PN is a Workflow net if and only if the following hold,*

(1) PN has two special places, i.e. a source place i where $\bullet i = \emptyset$, and a sink place o where $o\bullet = \emptyset$,

(2) if a transition t^* is added to PN and an arc connects o to t^* and t^* to i, i.e. $\bullet t^* = \{o\}$ and $t^* \bullet = \{i\}$, then the resulting Petri net is strongly connected.

The notations $\boldsymbol{i}$ and $\boldsymbol{o}$ will be used to denote the initial and final markings of the workflow net, respectively, where $\boldsymbol{i}$ has the configuration $M(i) > 0$(with $M(i) = 1$ for workflows with one case being processed) and $M(p) = 0$ for $p \in P \backslash \{i\}$, and $\boldsymbol{o}$ has $M(o) > 0$ and $M(p') = 0$ for $p' \in P \backslash \{o\}$.

Definition 7 (Soundness) *A Workflow net $PN = (P, T, F, \boldsymbol{i})$ is* ***sound*** *if and only if the following hold,*

(1) if $\boldsymbol{i} \xrightarrow{*} M$, then $M \xrightarrow{*} \boldsymbol{o}$, i.e. for every reachable marking M from $\boldsymbol{i}$, there is a firing sequence usable to reach $\boldsymbol{o}$ from M.

This condition assures that all reachable markings in the net would also lead to the final marking $\boldsymbol{o}$. This means that the execution of the net will eventually terminate.

(2) if $\boldsymbol{i} \xrightarrow{*} M$ and $M \geq \boldsymbol{o}$, then $M = \boldsymbol{o}$, i.e. $\boldsymbol{o}$ is the only marking reachable from $\boldsymbol{i}$ with one token in place o.

This condition assures that when a token reaches o, all other places in the net has no token inside. This corresponds to proper termination of all processes in the net such that there are no computations that are halted when the sink place is reached.

(3) for every $t \in T$, there are M and M' satisfying that $i \xrightarrow{*} M$ and $M \xrightarrow{t} M'$, i.e. there are no dead transitions in the net.

This condition assures that every transition is involved in at least one process specification that the net executes.

Definition 7 was eventually referred to as “classical soundness” when variations of this property were introduced [55, 59] for verification of models with weaker or stronger notions of this property.

Definition 7 offers a very strict criterion because it imposes that all process instances/cases are completed with no tasks pending, cancelled, or withdrawn when one token reaches the sink. For systems that can allow modelling and implementation wherein at least every transition is involved in a case that is properly completed, i.e. the marking $\boldsymbol{o}$ is reached, other less restrictive notions of soundness is investigated in [55, 59] for Workflow and Petri nets. Some of these notions of soundness in these investigations are as follows,

Definition 8 (Sound firing sequence) *Let σ be a firing sequence and let M be a marking in a workflow net $PN = (P, T, F, \boldsymbol{i})$. σ is a* ***sound firing sequence*** *if $\boldsymbol{i} \xrightarrow{\sigma} M$ and $M \xrightarrow{\sigma'} \boldsymbol{o}$ for some firing sequence σ'.*

Definition 9 (Relaxed Soundness) *A workflow net PN is* ***relaxed sound*** *if and only if each of its transition is an element of some sound firing sequence, i.e. $\forall t \in T$, $\exists M, M'$ such that $\boldsymbol{i} \xrightarrow{*} M \xrightarrow{t} M' \xrightarrow{*} \boldsymbol{o}$.*

Relaxed soundness requires at least one case that completes with exactly one token in the sink o and all other places have no tokens for each transition in the net. Implicitly, transitions can participate in the execution of cases without a proper termination of all the tasks in place.

Definition 10 (Weak Soundness) *A workflow net PN is* ***weak sound*** *if and only if the following hold,*

(1) for every marking M where $\boldsymbol{i} \xrightarrow{*} M$, $M \xrightarrow{*} \boldsymbol{o}$,

(2) for every marking M where $\boldsymbol{i} \xrightarrow{*} M$ and $M \geq \boldsymbol{o}$, $M = \boldsymbol{o}$.

Definition 10 removes the requirement that all transitions be involved in the execution of cases as long as proper completion, i.e. $\boldsymbol{o}$ is reached, for every reachable marking from $\boldsymbol{i}$. Note that deadlocks can never occur in weak sound workflows. When workflows of different functionalities are collected together and made to interact and coordinate together by some interfacing specifications to serve another purpose, some of their built-in tasks may not be needed in the latter job. For this, checking for weak soundness in the aggregated workflows may be sufficient in checking whether they would indeed support the expected functionality that they collectively provide.

Definition 11 (Lazy Soundness) *A workflow net PN is* ***lazy sound*** *if and only if the following hold,*

(1) for every M where $\boldsymbol{i} \xrightarrow{*} M$, there exists M' such that $M \xrightarrow{*} M'$ and $M'(o) = 1$,

(2) for every M where $\boldsymbol{i} \xrightarrow{*} M$, $M(o) \leq 1$.

Lazy soundness imposes that every reachable marking from $\boldsymbol{i}$ will eventually lead to a completion of a case that it helps to execute, thereby, putting one token in o. However, when one token reaches o, no other tokens are allowed to be added to it by some process instance that has yet to complete. Note that [2] emphasizes the difference between "completion" and "termination" of process instances for this definition. That is, o is loosely used in Definition 11 to mean the final node that is defined in the workflow. Nonetheless, o does not have any outgoing arc yet putting one token in o(in Definition 11.(1)) only implies the execution of its corresponding case is completed. Termination happens when every process execution that is running in the net are terminated. Therefore, completion is not allowed when a termination event has already been executed in the net.

Definition 12 (Easy soundness) *A workflow net PN is* ***easy sound*** *if and only if $\boldsymbol{i} \xrightarrow{*} \boldsymbol{o}$.*

For the succeeding definitions from [55], the notations $\boldsymbol{i}^k$ and $\boldsymbol{o}^k$ mean that the marking has $M(i) = k$ and $M(o) = k$, $k \in \mathbb{N}$, respectively, and all other places have 0 tokens.

Definition 13 (k-soundness) *A workflow net PN with $\boldsymbol{i}^k$ is* ***k-sound*** *if and only if for every M where $\boldsymbol{i} \xrightarrow{*} M$, $M \xrightarrow{*} \boldsymbol{o}^k$.*

The definition of classical soundness(Definition 7.(1)) anticipates exactly one token when process executions in workflows terminate. Therefore, Definition 7).(1) implies 1-soundness. Weak soundness is also 1-soundness.

Definition 14 (up-to-k-soundness) *A workflow net PN is **up-to-k-sound** if and only if PN is l-sound for all $0 \leq l \leq k$, $k \in \mathbb{N}$.*

Definition 15 (Generalized soundness) *A workflow net PN is **generalized sound** if and only if for all $k \in \mathbb{N}$, PN is k-sound.*

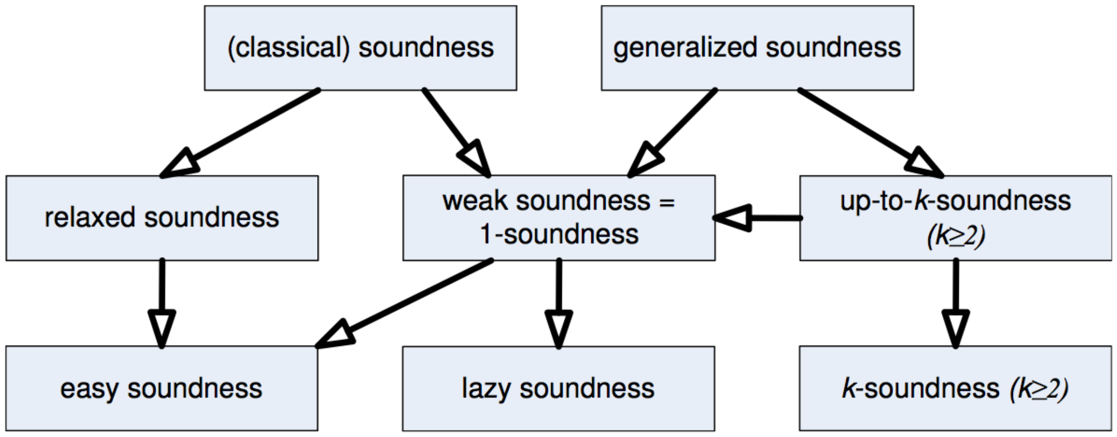


Figure 2.2: The relationships of the different notions of soundness in workflows. Arrows refer to the "implies" relationship of one property to another, e.g. (classical) soundness implies weak soundness. (Image source: [55])

Free-choice Petri nets

At a certain marking of the net, when two transitions are enabled and can be fired simultaneously they are said to be *concurrent*. On another hand, for two transitions that are enabled by a common input place, they are in *conflict* with each other whenever the choice of firing one disables the other. That is, the transition which was not fired cannot execute its computation as well as its succeeding ones which rely on its firing. For certain business processes, the freedom to make this choice between executing the tasks must not be impeded by the rest of the system. That is, every choice is *free*. A workflow that satisfies this condition is said to be *free-choice*. The free-choice property can be enforced by structural compositions as defined below,

Definition 16 (Free-choice) *A Petri net is **free-choice** if and only if, for every two transitions t_1 and t_2, $\bullet t_1 \cap \bullet t_2 \neq \emptyset$ implies $\bullet t_1 = \bullet t_2$.*

Free-choice and (classical) sound workflows are closely related to each other. The latter is verifiable in efficient time by using the former. By this way, the need for enumeration of all reachable markings in Definition 7 is avoided. The size of the state space of arbitrary workflow nets can easily become huge which poses the problem in enumeration. Deciding the soundness of workflow nets may be intractable [84] and checking for model properties that relates to it, i.e. liveness and boundedness in Definitions 17 and 18, respectively, is EXPSPACE-hard [85].

Definition 17 (Live) *A Petri net $PN = (P, T, F, M_0)$ is **live** if and only if, $\forall M'$ where $M_0 \xrightarrow{*} M'$ and every transition $t \in T$, there exists M'' where $M' \xrightarrow{*} M''$ which enables t.*

Definition 18 (Bounded) *A Petri net $PN = (P, T, F, M_0)$ is **bounded** if and only if, $\forall M'$ where $M_0 \xrightarrow{*} M'$ and $\forall p \in P$, there exists a $b \in \mathbb{N}$ such that $M'(p) \leq b$.*

In [25], the following lemma was proven,

Lemma 1 *Let PN be a Petri net and $\overline{PN}$ be its workflow net with the initial marking **i**. A Petri net PN is sound if and only if $\overline{PN}$ is live and bounded.*

By the use of the Rank Theorem [84], the relation of free-choice to liveness and boundedness is established in workflows. This further establishes a relation of free-choiceness to classical soundness in workflows(by Lemma 1). This way of proving soundness by the free-choice property can be performed in efficient time(see Corollary 1). (The Rank Theorem can help show liveness and boundedness in a net by characterizing a net's classes, i.e. S-systems and T-systems, and invariants thereof. S-systems are nets where each transition has only one input place and one output place. By this way, concurrency is absent in the net. Meanwhile, T-systems have every place with one input transition and one output transition. Through this, conflicts never happen. The reader is directed to [84] for a more detailed discussion on the proof of Corollary 1.)

Corollary 1 *It is decidable in polynomial time whether a free-choice workflow net is sound.*

The reader is referred to [55] for the schemes and results with respect to the verification of the different notions of soundness, as well as the computational complexity in proving them.

Although Petri nets and workflow nets are mainly concerned with representations using the process dimension, this research adopts the same kind of structurally-driven perspectives of models to prove properties in efficient time. This adoption will now consider models that have all three workflow dimensions, i.e. process, resource, and case, in place.

2.2 The Business Process Model and Notation

Along with the Class Diagram of the UML, Business Process Model and Notation(BPMN) is considered to be one of the two most expressive and easiest for integration for the interchange and execution level of business process management as discussed in Section 1. With this view, this section provides the technical information of the BPMN and its framework as well as a critical perspective on its advantages and weaknesses for systems modelling in theory and practice.

The Building Blocks in BPMN

The components of BPMN are divided into four categories [2, 3]: (1)Flow objects, (2)Connecting objects, (3)Artefacts, and (4)Swimlanes. Figure 2.3 presents the notational elements in each of the categories and their corresponding graphical representations for modelling.

Flow objects are the building blocks of business processes. An event is a stage of execution of an activity, i.e. its start or termination, or an intermediate stage – an activity momentarily suspends its termination while interactions between other activities are performed. An activity/task encapsulates the process specifications that are performed for some case with or without a set of resources that enact these processes. A rounded rectangle, representing a task/activity, abstracts these information, and therefore, upholds atomicity in its representation of the task/activity. The labels on each task/activity can use a verb-case combination where the verb pertains to the type of enactment of the task for some case, e.g. "Place Order". Activities can be annotated graphically to signify nesting of activities, i.e. subprocesses, and/or the presence of the execution of multiple cases/process instances. Gateways represent joins and splits of processes that lead to or from them, respectively. Symbols inside the graphical notation of the gateways specify the type of split or join that they implement. For example, the '+' signifies an AND split or join of processes that are linked to the gateway.

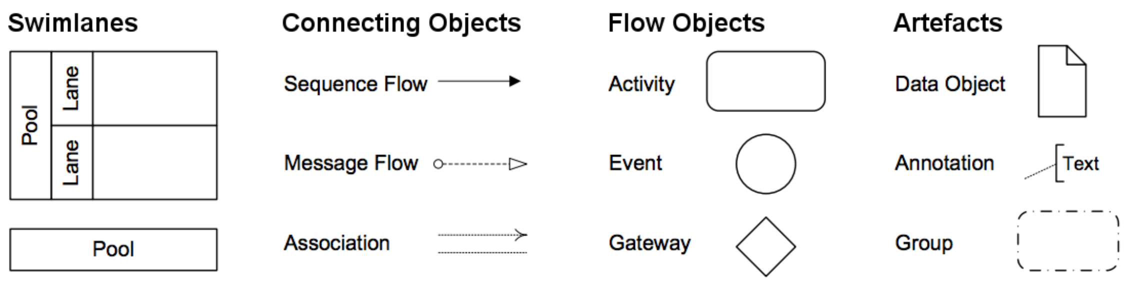


Figure 2.3: Categories of elements in BPMN.(Image source: [3])

Artefacts are nonactionable information in BPMN models. That is, they do not directly influence execution of process flows or choices thereof. They act as means of support for model interpretability. They may be expressed in informal, textual or graphical form for documentation purposes.

Connecting objects connect the components of BPMN models. Sequence and message flow objects specify the order of the components' execution and the type of the interaction, i.e. sequential execution of the two components linked by the sequence flow or an indication of a requirement in receiving/sending a message that triggers an execution of an event.

Swimlanes represent organizational/participant information, i.e. (1) a pool that represents the name of an organization/participant, and (2) lanes that represent the entities found in the organization, e.g. departments. Each lane can contain flow and connecting objects that show some processes that are performed inside the business entity.

By the use of swimlanes and connecting objects, interactions can be done between and within swimlanes of BPMN models.

Shown in Figure 2.4 is a MBPN model of a recruiting process illustrated in [10]. This model illustrates one specification of employee recruitment from the time the applicant submits his application to the HR Department until a decision is made by the latter regarding his acceptance or rejection of his application. The interactions between the applicant and the HR Department can be seen by the messages sent along the communication arcs that cross between swimlanes in the model. An XOR gateway joins two process flows regarding acceptance or rejection of the applicant. The "Application assessment" activity contains a subprocess which evaluates some requirements pertaining to the qualifications of the applicant.

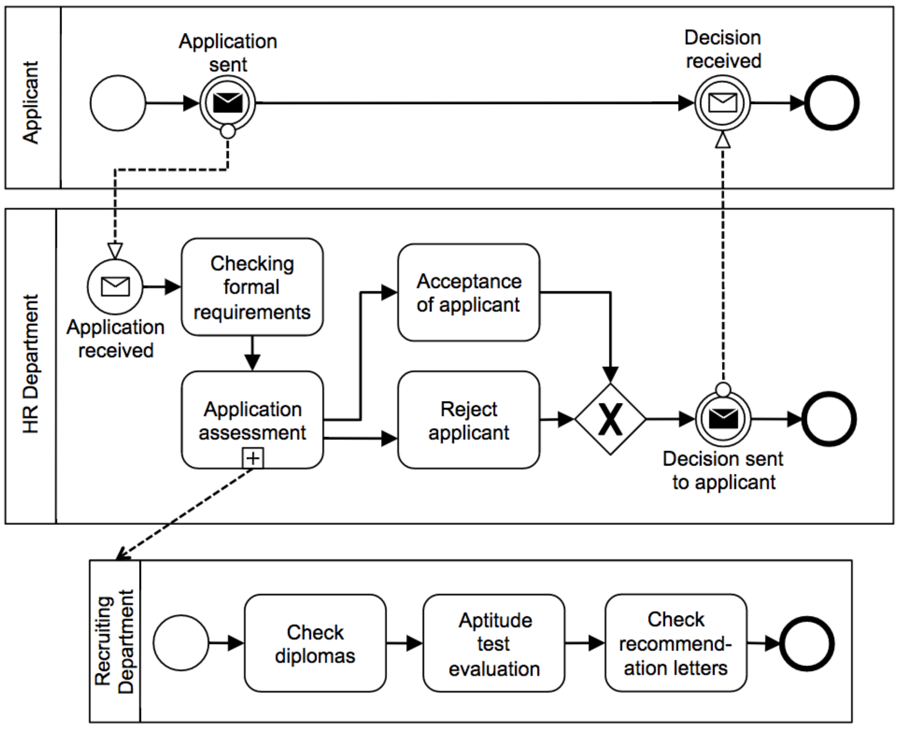


Figure 2.4: A BPMN model of a recruitment process in an organization.(Image source: [10])

BPMN and the process dimension

Figure 2.4 can help show some of the immediate drawbacks in using implicit splits or joins and the use of gateways in models. Apart from these drawbacks in the sample BPMN model in this figure, the following are some other notational limitations of BPMN,

(i) BPMN provides freedom to connect components of the same category which can lead to ambiguity in modelling. For example, such connection is seen in Figure 2.4 which involves three activities, namely, “Application assessment”, “Acceptance of applicant”, and “Reject applicant”. This example shows an implicit split of process flows from the first to the second and third activities. This split semantically means a parallel execution of the latter set of activities. Conceptually, this parallelism contradicts the intended business process of having exactly one of the processes to be executed, i.e. either accept or reject an applicant. From this sample, it can be seen that this freedom on notational convenience and concept excess can add ambiguity and/or errors in representing process flows.

(ii) The splits and joins, either coming from activities themselves or from gateways, do not explicitly provide information as to the basis of the choice of process execution. For example, the implicit split such as the one mentioned in (i) do not state which parameter is used to make the decision of acceptance or rejection of an applicant. Annotations can be added however they are not readily actionable. They may induce ambiguity when formalisms are absent in interpreting and transforming them to executable entities in WfMS.

(iii) Implicit splits for process flows lead to mismatched split-join combinations in designs. Note how the model uses an implicit split in (i) which induces two parallel process that eventually converge to an XOR join gateway – a gateway that expects exactly one incoming process flow to be executed.

(iv) The subprocess of the “Application assessment” activity in Figure 2.4 show a syntactically erroneous nesting of activities. This error is induced by labelling the subprocess’ pool as “Recruiting Department” albeit it is just semantically a part of the “Application assessment” activity which is executed in the HR Department.

(v) Note that there is no explicit representation of rules that control the execution of the subprocess in (iv) when its main process is reached in the flow.

The problems of implicit joins and splits in (i) and nesting in (iv) are mentioned in [10]. This paper mentions that aforementioned errors, although preventable, happen in actual BPMN models of real-world systems. These models were developed using BPMN 2.0 and represent 585 business processes from six companies. They vary in size as well as they come from differing industries. They are analyzed and checked using 35 well-known BPMN guidelines and correctness rules that were proposed/recommended in [49–51].

One of the recommendations in [10] that help avoid the errors relevant to (i) is to avoid implicit splits and joins. Gateways can substitute implicit splits and joins while still able to capture the same relations that are intended to be modelled using the latter but without ambiguity in notational representations. This duplication in intent among these control constructs is known as “concept excess”. In particular, concept excess refers to the possibility to represent the same semantics in multiple graphical ways [10]. It has been shown to negatively affect understandability [62] of models. Furthermore, the use

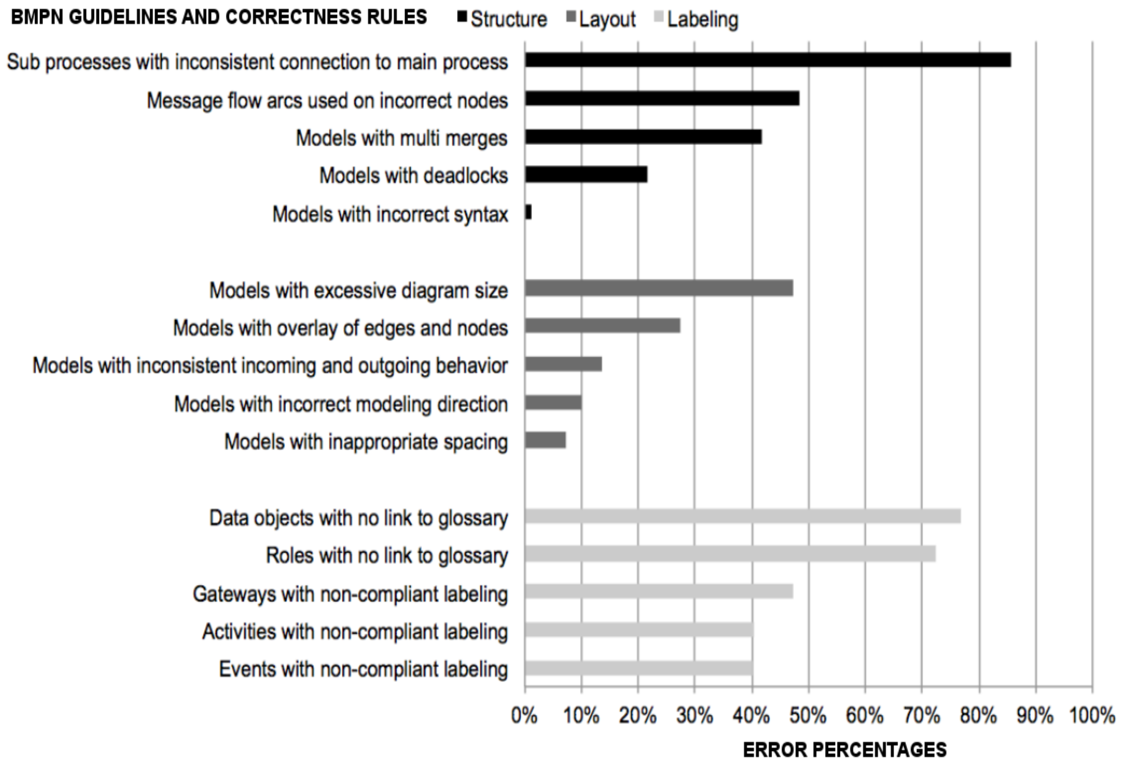


Figure 2.5: Error percentages in modelling guidelines and correctness rules [49–51] determined from real-world BPMN models as computed in [10].(Image source: [10])

of gateways enforces modellers to be explicit in the type of split or join that is required in managing the process flows.

Model verification for process-centric workflows such as Petri nets and workflow nets [25, 27] are well-known. In literature, this model verification is adopted for some aspects in BPMN modelling which is mainly powered by BPMN to classic workflow transformations. The issue in (iii) can also be addressed when these transformations are checked for being well-structured nets. A hindrance to this adoption is the lack of explicitness of BPMN models such as in (i), (ii), and (v), as well as lack of formalisms for effective and rule-based interactions between interacting participants.

Another control scheme that is enacted using gateways are the k out of n process executions in joins. For this, a complex gateway of BPMN is used with n preceding process flows(and k of them are executed where k is less than or equal to n) that joins at the gateway. This scheme is also offered as a means to enable representations of fault tolerant systems wherein k out of n process need to be executed before activities after the join are performed [63]. Implicitly, this means that there are at least n process specifications which are actually performed and which ends at the join gateway. They can correspond to different process flows that are taken by considering different process flows from preceding split gateways in the designs.

Figure 2.6 illustrates the use of a complex gateway in a representation of a fault tolerant system [63]. This gateway can aid in performing the "n out of k process" pattern in a BPMN model for such real-world systems. In this example, all n process specifications lead from a parallel gateway. Note that activities themselves may encapsulate subprocesses such that their execution time may differ relative to each other. That is, some activities might take longer than others to complete. A complex gateway waits for k of them to finish and thereby execute Activity C. At this level of model resolution, it is not easily determinable which subset of the processes are considered as the k processes that the gateway anticipates for completion. Furthermore, there is also no means of imposing certain subsets of process executions to be waited for by the complex gateway. Note that the same problem of the lack of support for explicit rules are still seen in the split and join in Figure 2.6.

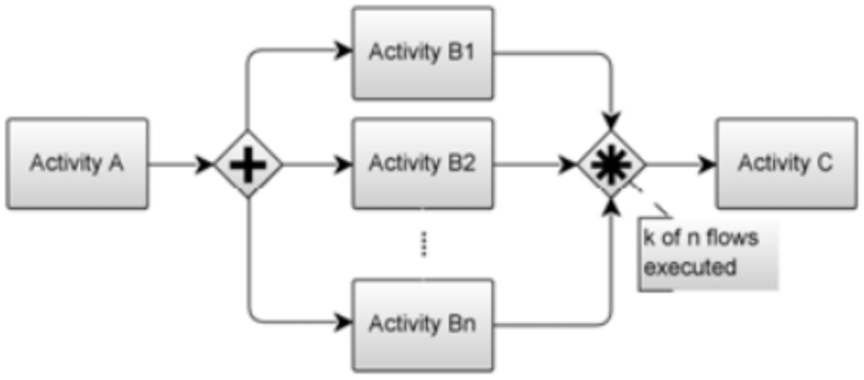


Figure 2.6: A BPMN model of a fault tolerant system using a complex gateway for the "k out n process" pattern.(Image source: [63])

Whenever there are still $n - k$ activities that have not completed and a terminal event is reached in the execution of the model, then the entire process instance is considered completed. For a workflow net representation with this design involving such type of join gateway, there will be tokens that are left out in non-terminal places in the net. This implies that soundness will not hold for such kind of design.

In proving for properties in models that contain complicated process specifications, e.g.n out of k process pattern, discriminator, multiple instances without synchronization, there are efforts such as in [59, 65] that introduced different notions of well-known model properties in workflows. An example of these properties is soundness, namely, (1)classical soundness, (2)generalized soundness, (3)relaxed soundness, (4)weak soundness, (5)up-to-k-soundness($k \geq 2$), (6)easy soundness, (7)lazy soundness, and (8)k-soundness($k \geq 2$)(see Section 2.1). The results in these efforts also included specifications on the verification process of these notions of soundness. The results on decidability of workflow nets that contain different types of such control flow structures with respect to these notions of soundness are detailed in [65].

On Process Interactions in BPMN modelling

The same problems for explicitness in designs and control schemes mentioned in Section 2.2 for BPMN modelling can still be found in interactions between swimlanes of its models. Added to these problems are the lack of formalisms to establish effective and explicit rules in interactions between process participants, e.g. message throws [10]. Note on the high percentage of errors with respect to message flows seen in Figure 2.5.

Figure 2.7 shows a process interaction in an auctioning system [2, 3] with three participants, namely, a Bidder, an Auctioning Service, and a Seller. In this auction system, every bidder must ask an auctioning service to participate by sending a request. As a response, the service may immediately send an acceptance or a rejection to the bidder's request. The service can alternatively forward such request to its seller. With this, the seller can decide whether the request is accepted or rejected and then informs the service of the decision. Thereafter, the service can send back this decision to the requesting bidder. Notice how the exclusive nature of whether an acceptance or rejection for the two interactions is not immediately apparent in the model. Furthermore, by occluding their corresponding process specifications, it cannot be revealed that they indeed are exclusive and for which conditions they are enacted upon.

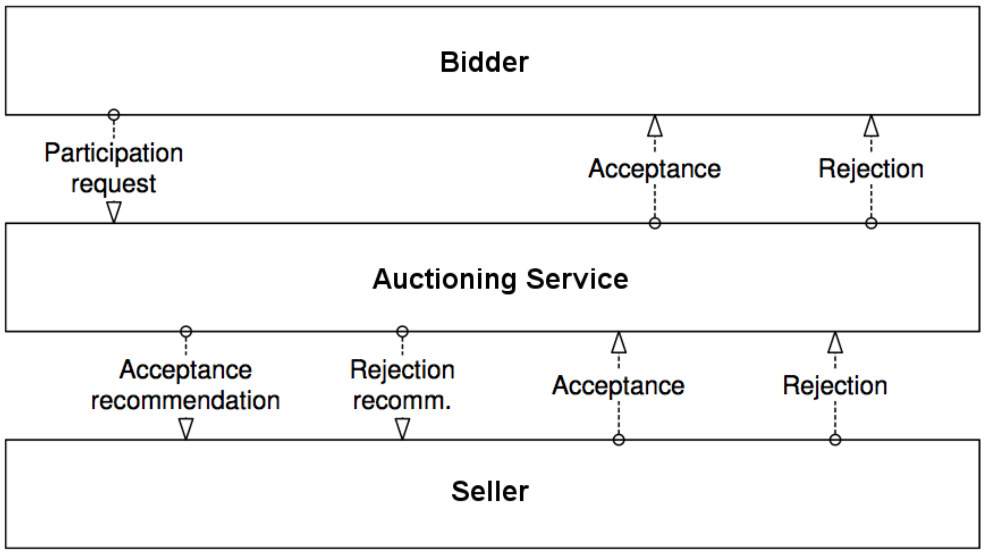


Figure 2.7: The auctioning system viewed at the level of message passing in interactions of process participants.(Image adopted from [3].)

The interactions and abstractions thereof in Figure 2.7 can be modelled by the swimlanes of BPMN. The interactions are modelled by communication arcs that are linked to events in the lanes for message passing. In order to prove properties and correctness of these interactions and abstractions, some aspects of BPMN models can be transformed to workflow modules [64] to help in verifying the effectiveness of interactions among the participants. Workflow modules are essentially Petri nets with some places which are identified as interfaces [64]. These interfaces serve as sending or receiving places of tokens in the net. The net's tokens correspond to messages sent between interacting modules. The concept of structural compatibility [2] and the varying levels thereof for interactions are used for verification of the models. For each of these levels, modules are checked for their number of interfaces and how these interfaces are used in the module. Furthermore, compatibility between modules is measured by the sufficiency of the number of sending/receiving places and whether every message sent(received) from one module is received(sent) by another. Pairs of interacting interfaces are identified and will be merged as one when their interacting modules are integrated as a workflow net.

Figure 2.8 shows two workflow modules corresponding to the Auctioning Service and Seller participants from Figure 2.7. Note that their interfaces (places) have been given the same names to resolve the integration of of the two nets. These modules only show a part of the interactions of the participants in Figure 2.7.

The interactions shown in the modules in Figure 2.8 correspond to their messaging for acceptance and rejection recommendations in the auctioning process. The labelling of the transitions/task imply the type of messaging involved, i.e. a prefix of '!' is a task corresponding to a sending of a message while '?' a receiving message. By firing a transition, a token can be distributed from one module to another when this token reaches an interface place. Note that the integration of the modules is supported by creating a workflow net with an added input place 'i', output place 'o', two transitions 't1' and 't2', and arcs(drawn using dashed lines) that connect these additional components with the components in the two modules as shown in the figure. Lastly, pairs of interacting interfaces(i.e. places with the same names) are merged to create a unified workflow.

Note that from the process of integrating workflow modules, it is not hard to see that there can be certain properties in the structure and behavior of the modules which hold for each of them but not for their workflow net. For example, deadlock-freeness and soundness may not hold in the integrated workflow [25].

BPMN and the resource dimension

As discussed in Section 2.2, the passiveness of resources/participants with respect to the interactions in BPMN models(such as in Figure 2.7) is carried on through its workflow net representation(such as in Figure 2.8). In design frameworks wherein resources are essentially designed to act as containers and/or grouping symbols which cannot be explicitly and/or readily usable throughout model transformations from their design to implementation, the following issues arise,

- Resources do not and cannot directly interact with model components. They are merely limited to being graphical guides to imply roles that are supported by a set of events and activities in the workflow.

- Resources do not and cannot have modifiable attributes that can be used to control and manage its set of model components.

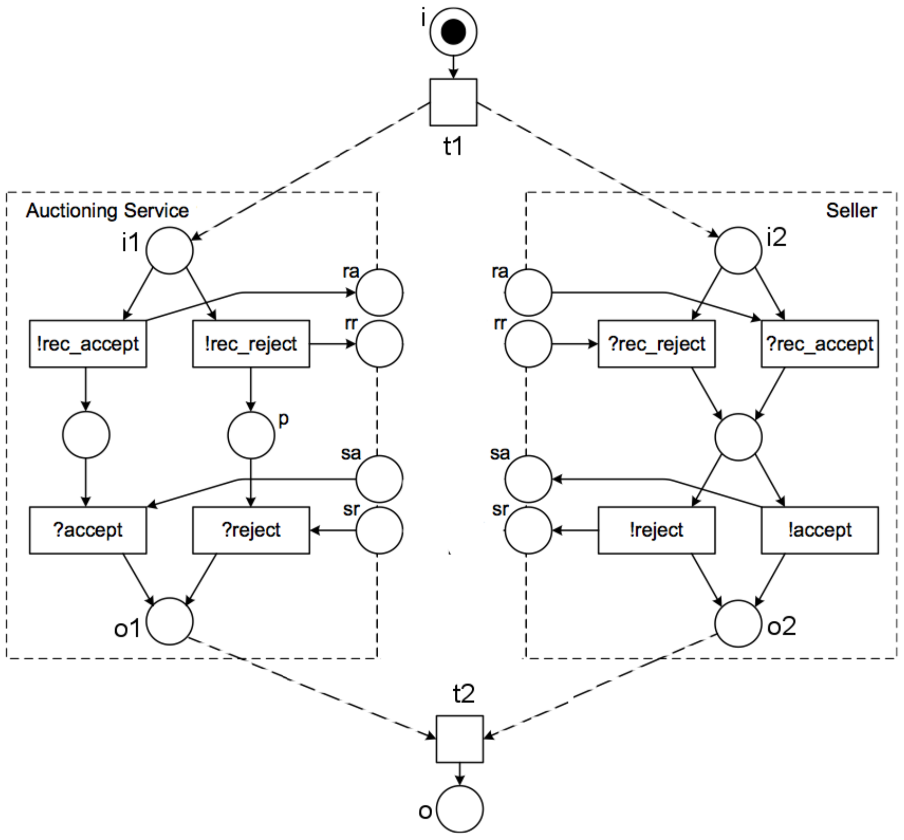


Figure 2.8: Workflow modules that show the interaction between Auctioning Service and Seller participants with respect to their messaging scheme of acceptance and rejection recommendations in Figure 2.7. These modules can be integrated by using an additional input and output places('i' and 'o') and two transitions('t1' and 't2') as well as representing each pair of interfaces with the same labels with one place and establishing the arcs(drawn with dashed lines) between them to form a workflow net. (Image adopted from [2, 3])

- Resources do not and cannot directly influence in the execution of activities and contribute to (run-time) process flow specifications.

- Problems in *requirements traceability*, i.e. "the ability to describe and follow the life of a requirement, in both a forwards and backwards direction" [66] would arise with respect to the requirements engineering in modelling.

- Structural relations and dependencies are not readily determinable.

- Controls that affect entire subsystems are not fully enabled and supported from design to enactment of models. A task-driven control of a group of graph components can be seen in the form of cancellation regions[8](see Section 1) in workflows. Groupings are explicitly defined for each task in the model. When a task with a cancellation region is executed, all tasks in its region is disabled and, when they are already executing, withdrawn. Whenever this task is a part of its cancellation region, it is also cancelled upon its completion.

Consider the following model in Figure 2.9 of a holiday scenario experienced by students who are planning for their holidays after taking their exams. In this model, the place **c4**(**c5**) contains a token if the student passes(fails) his exam. Furthermore, the *"Resit exam"* task has a cancellation region that includes the places **c2** and **c3** and the task *"Book flight"*(all annotated by the region's enclosing dashed lines). Therefore, the student will take a holiday(*"Take holiday"*) if he passes his exam and had booked his flight(*"Book flight"*). On another hand, if he fails his exam, he resits on it(*"Resit exam"*) and must stop planning his holidays(not booking a flight if he has not done so when concurrency is not imposed in the workflow implementation for *"Take exam"* and *"Book flight"*). With this, the *"Book flight"* task is disabled/withdrawn and any token in **c2** or **c3** is removed. Meanwhile, the student might need to cancel his flight(*"Cancel flight"*) if he had previously made the booking. With the completion of any of these process flows, the execution ends at an XOR-join with the *"Finalize plans"* task performed by the student, and thereafter a token is finally placed at the model's output place.

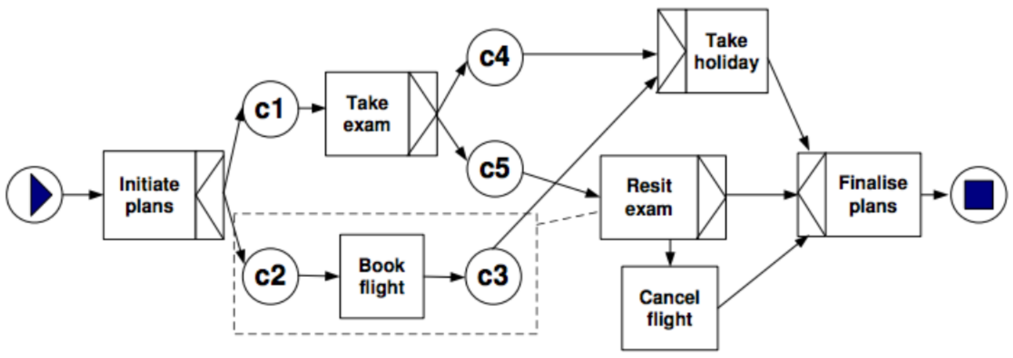


Figure 2.9: Holiday scenario for students. (Image source: [47])

It is assumed that every task in a cancellation region has a corresponding "bypass" task. The latter is executed instead of the former when a cancellation is triggered using the task that is associated to the region.

With regard to the implementation of cancellation regions in modelling frameworks and their supporting technologies, the Workflow Patterns Initiative[8] lists some of the following specifications,

– Staffware [67] supports withdrawal of tasks only if they have not been initiated.

[8]The Website of the Workflow Patterns Initiative: Pattern 25 (Cancel Region), http://www.workflowpatterns.com/patterns/control/ new/wcp25.php (2011)

  - FLOWer[9] supports skipping of individual tasks rather than cancelling groups of them.
  - UML 2.0 Activity Diagrams [29] fully support the use of cancellation regions. Cancellations can be accomplished by using interruption edges(i.e. arcs whose line segment is rendered as lighting bolts) which are linked to the components of the region.
  - BPMN and XML Process Definition Language(XPDL) [1] partially support cancellation regions through the use of subprocesses. These subprocesses would have to be associated with an error event to trigger cancellations.

  The semantics of cancellation regions shows that groupings are mainly (or ideally) established with considerations of behavioral aspects of systems. This also means that groupings can even be established without regard of structural/topological aspects of systems. Due to the possibilities of disruptions in process executions which are triggered by a non-local(topology-based) cancellations, model verification is also a problematic undertaking. For instance, the verification of soundness in models which contain both cancellation regions and OR-joins become undecidable [46, 47]. Furthermore, by the allowance of the grouping of topologically-unrelated tasks in cancellation regions, it is difficult to establish and support encapsulation in many aspects of workflow designs and implementation.

- Encapsulation of work specifications and details cannot be fully supported from design to enactment of models. Note how alignments in interactions impose the revelation of some aspects of the structural relations in process specifications in pools and lanes such as in Figure 2.7. The lack of rules on how resources are to be dealt as a part of the model itself and how to manage the interactions in messaging become problems in BPMN modelling. These can easily be seen in the high percentage of errors in linking the correct nodes found inside the pools [10] as shown in Figure 2.5.

- With respect to handling a work specification in BPMN models where cases are executed using multiple resources with their associated tasks, Figure 2.10 shows a design recommendation from an online article[10] of G. Polančič. Note that Figure 2.10.(a) is marked as an erroneous design. This is because activity notations cannot be aligned between two lanes. This is due to the exclusivity in assigning them to one resource in a pool. The intent of this design is to show that Task 3 is executed for cases wherein Person A and Person B are both responsible for its execution. The recommendation in Figure 2.10.(b) shows that Task 3 can be duplicated in both lanes to establish the intended design.

  As seen in Figure 2.10, the simple duplications of a task/activity to illustrate a joint responsibility of resources may create ambiguity in role distribution and accounting.

[9]The Website of FLOwer: FLOwer, http://www.workflowpatterns.com/vendors/flower.php (2011)

[10]Polančič, G.: Common BPMN Modeling Mistakes - Activities, http://blog.goodelearning.com/bpmn/common-bpmn-modeling-mistakes-activities/

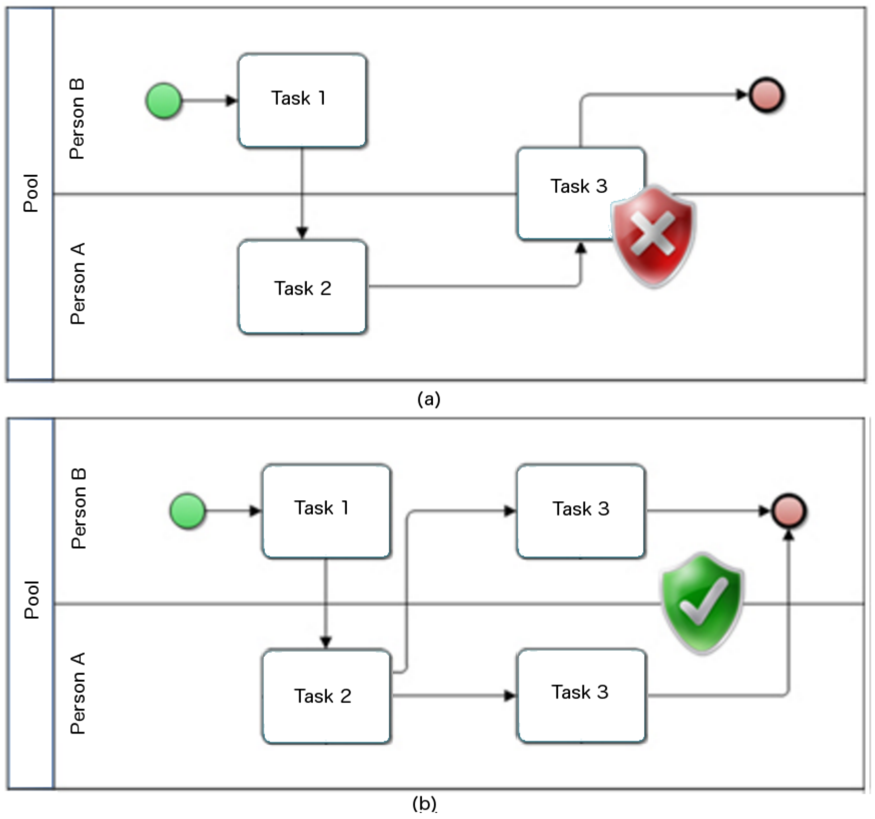


Figure 2.10: Design recommendations[10] in handling a task that is assigned to be executed by multiple resources to support the processing of cases in BPMN models.

> The duplication itself may incur inconsistencies in the intended business rules when the number of interactions and requirements increase and become more complicated.

BPMN and the case dimension

Case handling in BPMN is mainly supported with view of horizontal abstractions [2] of business process models. That is, different levels of models are considered in their design and analysis. They are the (1)instance level, (2)model level, and (3)metamodel level. In view of the instance level, tasks take in cases either singly or multiple, sequential or parallel, and/or with inclusions of looping under some design- or run-time specification. Figure 2.11 shows the graphical notations that are used in the BPMN representation of a tasks/activitity. The loop symbol/curved arrow in Figure 2.11.(a) is used when a task is repeated. The number of repetitions can be added as an attribute to the task. Thick vertical and horizontal lines in the notation Figure 2.11.(b) and .(c) mean that there can be multiple cases that are handled in the task executions in parallel or sequential manner, respectively. Note that the boxed plus sign indicate subprocesses that are associated to activity they are represented by in Figure 2.11.(d)-(f).

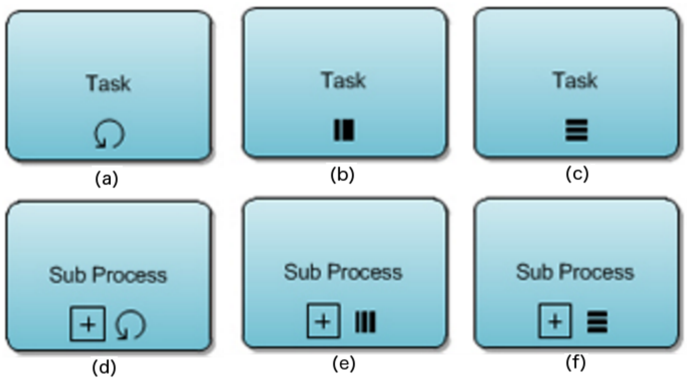


Figure 2.11: Variations of notations for task/activities in handling cases in BPMN modelling.(Image source[10])

In using a process specification that supports the execution of multiple cases, there are different control flow patterns which include the following: (1)Multiple Instances Without Synchronization, (2)Multiple Instances With A Priori Design Time Knowledge, (3)Multiple Instances With A Priori Run Time Knowledge, (4)Multiple Instances Without A Priori Run Time Knowledge, (5)Sequential Execution without A Priori Design Time Knowledge. (Note that instances mean cases in this context.) For these patterns of control flow, each of the cases may have design- and/or run-time specifications with respect to (1) the time it is dealt with, and/or (the support for its implicit or explicit (non)termination whenever their succeeding component in the model is a termination event or a task.

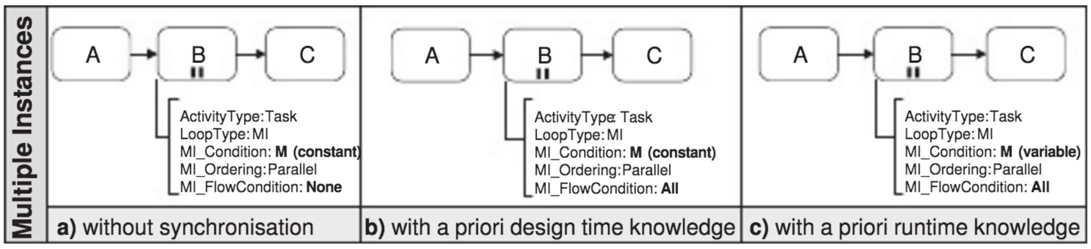


Figure 2.12: Annotated BPMN models for some control flow patterns used in dealing multiple instances(Image source: [70])

From the aforementioned control flow patterns for handling multiple cases, the first three patterns can be captured using BPMN [70], however, with the use of texts pertaining to coded attributes and their values as shown in Figure 2.12. Meanwhile, the last two patterns have no direct support and/or cannot be explicitly represented using BPMN [2,70]. In [2], a similar BPMN model is drawn for the Multiple Instances Without A Priori Run Time Knowledge pattern. That is, using three activity nodes A, B, and C with the same connectivities as in Figure 2.12 but without the texts. This is because the number of instances for B cannot be explicitly set at design time nor is it known at any stage during run time before the instances of B are enabled. A rough workaround in [2] in dealing with this pattern is to create a management activity ‘b’ that can be linked to B in the BPMN model. That is, this management activity ‘b’ controls the initiation of the instances of the activity B when B is executed. Furthermore, ‘b’ suspends the execution of Activity C until all instances of B has already completed.

For case handling in BPMN 2.0 and its supporting technologies, the following can be observed:

- An activity with an internal loop does not allow time-, activity-, and/or requirement-related executions to modify the execution of the loop itself, and vice-versa.
- Management activity tasks, such as ‘b’ in for managing the activities A, B, and C in the Multiple Instances Without A Priori Run Time Knowledge pattern (as discussed above), that control the initiation of task executions in multiple activities cannot be explicitly represented in BPMN.
- Accumulated workloads in BPMN models and in their components cannot be described fully at run time. This is because case executions are monitored and handled individually at run-time by external workflow management tools. These workload information is not explicitly represented within the models themselves. Relying on external management systems and/or knowledge workers themselves to continually monitor and control workloads at run time is not ideal for complex systems with huge amount of state space. The resources and their associated components in model representations should also be given explicitly-represented attributes. These attributes can be used for automatic reroutes when bottlenecks arise. This can aid in handling multiple instances in run-time while providing means to check for static flow controls. Because this information is known at design-time, it is readily usable for verification of models. Furthermore, resources themselves should be able to use those attributes to perform automatic resets. These resets are done when some control flow schemes are detected to have been used to trigger them. Resets that are performed in subgroups of components in models can be used to indicate availability for usage of system components or their transient behavior in real-world systems. Essentially, atomicity and encapsulation in system components and relationships can be effectively implemented using these schemes. Another benefit that results from these schemes is the capability to perform localized controls within well-defined substructures that are grouped together by topological connectivities and bound by a common functional role inside the system.
- With the presence of problems regarding sufficient support for representations of joint responsibilities among resources in task execution, a resulting problem arises regarding the management of multiple cases in these shared tasks. Furthermore, since resources act as passive containers of process specifications, they cannot be used to enact subsystem-wide controls for explicit reroutes of process flows,

bottleneck-handling, and/or task delegation in a multi-case setting in design and implementations of models.

- With respect to the representation and execution of ad-hoc tasks [2] in BPMN, including aspects of which subsets of ad-hoc tasks to be executed together, cases would have to be enumerated to cover all their possible combinations before any further analysis can be done on the model. This problem on enumeration is further complicated and the state space is further enlarged when gateways, such as the k-out-of-n join, are included in the process flow where these tasks are involved in. These problems implicitly lessen the possibilities to effectively describe, validate, and verify models and real-world systems.

2.3 Robustness Diagrams

The building blocks of a Robustness Diagram(RD) are Nouns and Verbs representing resources and tasks/atomic activities - i.e. boundary and entity objects; and controllers, respectively, as seen in Figure 2.13. These components interact with each other through arcs that connect them. Boundary objects support interaction of users with the system being modelled. Entity objects are conceptual representations of the internal components of a system. They have specific set of attributes and operations associated to them. Controllers serve as the “glue” between boundary and entity objects. They are used to implement the logic required to manipulate the nouns, their information, and interactions. As a rule of thumb, no arcs connect the nouns to each other. Furthermore, arcs can be made between a verb and a noun.

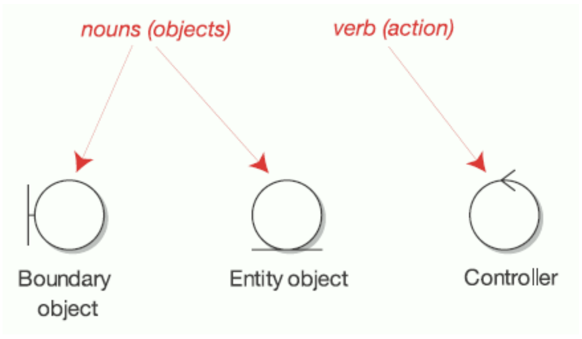


Figure 2.13: Robustness Diagram components. (Image source: [29])

From the time RDs were introduced as an appendage to the UML standard [28] until the ICONIX Framework highlights its use in Robustness Analysis [30], the literature for effective RD modelling and verification has remained lacking. They had remained to be used as rudimentary guides and illustrative visuals in the first few stages of system modelling[7] [80]. A study in [74] provides a preliminary view of modelling and activity extraction using RDs. An algorithm *Alg* was also introduced therein that uses arc attributes which enforce condition-dependent joins and splits in RDs. This is implemented by supplementing these attributes to the arcs in the model. The algorithm is based on Depth-First Search(DFS) for graph traversals. These activities are represented as sequences that list the order of the vertices which are reached by traversals. An attribute of time is added to RDs to mark the time of traversals of arcs, and therefore, the time that tasks are accomplished in the model. More formally, a Robustness Diagram in [74] is defined as follows,

Definition 19 *A Robustness Diagram(RD) is a graph representation R of a system that is defined as $R = (V_1, V_2, V_3, E, C, T)$ where:*

- *V_1 and V_2 are sets of boundary and entity objects, respectively, and V_3 is a set of controllers where V_1, V_2, and V_3 are disjoint with each other. A boundary object represents a system component that receive(send) values from(to) a system's environment. An entity object represents an internal system component that interacts with other components but not with the environment. A controller represents a task that is executed by the system and is associated to some component in the system. (Refer to Figure 2.13)*
- *$E \subset ((V_1 \cup V_2) \times V_3) \cup (V_3 \times (V_1 \cup V_2 \cup V_3))$ is a set of arcs,*
- *$C : E \to \Sigma \cup \{\epsilon\}$ where Σ is a non-empty finite set of symbols and ϵ is the empty string. Furthermore, $C((x, y)) \in \Sigma$ represents a constraint to be satisfied to reach y from x. This constraint can represent either an input requirement or a parameter $C((x, y))$ which needs to be satisfied to proceed from using the component/task x to y. $C((x, y)) = \epsilon$ represents a constraint-free process flow to reach y from x or a self-loop when $x = y$.*
- *$T : E \to \mathbb{N}$ assigns an integer to an arc marking the time of traversal of some algorithm's walk on R.*

An algorithm proposed in [74] extracts an activity from a RD using the rules for traversal as shown below. Note that for brevity, we shall denote this algorithm as *Alg* in the entirety of this research. *Alg* initially uses $s \in V_1 \cup V_2$ until an arc $f \in E$ is reached by using the following traversal-backtracking conditions,

(1) from $x \in V_1 \cup V_2 \cup V_3$, select $(x, y) \in E$ where $T((x, y)) = 0$ for possible traversal of *Alg*. (*Alg* starts with $x = s$.) If $|\{(x, z) \in E \mid z \in V_1 \cup V_2 \cup V_3\}| > 1$, arbitrarily choose any outgoing arc of x in this set for a possible traversal. We call any arc (x, y) as **unexplored** if $T((x, y)) = 0$.

(2) From the chosen outgoing arc (x, y) in (1) for possible traversal, set $T((x, y)) = k$ where $k = 1 + \max_{(u,x)\in E}\{T((u, x))\}$.

(3) If (x, y) is **unrestricted**, i.e. $\nexists v, v \in V_1 \cup V_2 \cup V_3$, $(v, y) \in E$, s.t. $C((x, y)) \neq C((v, y)) \wedge C((x, y)) \in \Sigma \wedge T((v, y)) = 0$, traverse (x, y) to reach y and execute its

process. Upon traversing (x,y), if $C((x,y)) \in \Sigma$, update $T((v,y)) = T((x,y))$ for every $(v,y) \in E$.

When no unexplored outgoing arcs are found in x and f is still unexplored, *Alg* backtracks to $u \in V_1 \cup V_2 \cup V_3$ such that $(u,x) \in E$ and there is an unexplored arc (u,x') for some $x' \in V_1 \cup V_2 \cup V_3$ and *Alg* proceeds from there. *Alg* stops when f is explored. Finally, it outputs the sequence of vertices, the arcs traversed and their corresponding times of traversals.

The nondeterministic choice of an outgoing arc (x,y) of x for *Alg* to traverse in step (1) results to having equivalent time of traversals that are assigned to all outgoing arcs of x. This equivalence will eventually not hold in case of updates to $T(x,y)$ and $T(v,y)$ in step (3) of the algorithm. Such an update on (x,y) and every (v,y) in the RD means that a join on y has already been resolved, i.e. all conditions imposed by every incoming arc of y are already satisfied. Note that before *Alg* performs traversals in the RD, $T(.) = 0$.

In [74], a sample Use Case Text(UCT) was given with its corresponding Robustness Diagram. This UCT represents one of the basic functionalities of Fujitsu Ten's **C**omputer Aided Multi-Analysis System **A**uto **T**est **T**ool (CATT) [75] that aids in analyzing embedded systems in automobiles developed by Fujitsu Ten Limited - Japan in collaboration with Fujitsu Ten Solutions Philippines.

Example 1 (A Sample UCT)
Name: *Create Analog Signal*
Precondition*:* *User adds an analog series*

1. *User selects single/multiple empty cell(s) in the Chart worksheet*
2. *User creates an analog signal*
   - *Pop Up Menu option*
     - a) *User right clicks on the selected cell(s)*
     - b) *System displays the pop up menu*
     - c) *User selects one of the following options:*
       - – *"Fixed Value Input"*
       - – *"Slope Input"*
   - *Toolbar button*
     - a) *User clicks one of the ff. toolbar buttons:*
       - – *"Fixed Value Input"*
       - – *"Slope Input"*
3. *System gets the selected cell(s)*
4. *System gets the parent Row of one cell in the cell collection*

5. *System gets the boundary*
6. *Repeat steps 3-4 until all selected cells are traversed*
    a) *System determine Min/Max boundary*
7. *System displays an input dialog box and asks the input value(s) from the user*
    a) *System displays input dialog box*
        - *If User selects "Fixed Value Input", System asks for a single value from the user*
        - *Else If User selects "Slope Input", System asks for start and end values from the user*

    b) *User inputs values in the dialog box*
8. *System restricts user's entered values to Max/Min boundary values and Max/Min time values*
    - *If user input/s is/are out-of-range, System disables OK button.*
    - *ELSE*
        - *System enables OK button.*
        - *User clicks "OK" or "CANCEL"*
            - *If User Selects "OK", perform steps 9-14*
            - *Else, skip steps 9-14*
9. *System gets the user's input*
10. *System gets the parent row and the parent column of once cell in the collection of cells*
11. *System gets the time interval of the cells column*
12. *User creates an analog signal in that cell*
    a) *setting Value(s) and Time(s) by calling Add Point*
13. *System plots an analog signal in that cell by performing createNewPlot() method*
14. *Repeat steps 10-13 until all selected cells are traversed*

Alternate Methods: None

In Figure 2.14, the values highlighted in red and yellow font on the arcs correspond to the time of a backtrack and the final traversal, respectively, that were done by the algorithm on the RD for activity extraction. For this example, $x0$ is the start vertex and

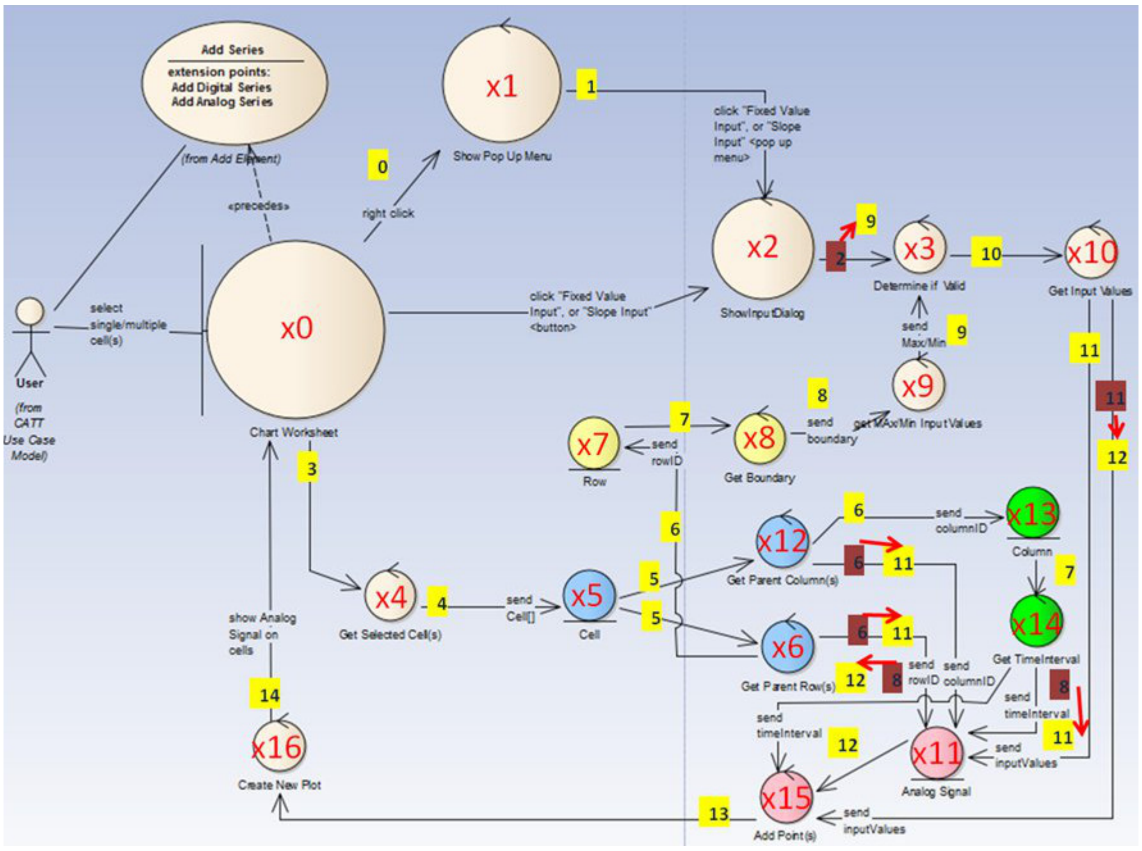


Figure 2.14: The Robustness Diagram representing the process specified in the UCT in Example 1.(Image source: [74])

the final arc is $(x16, x0)$. For this activity extraction, the activity can be described using the sequence

$$x0\ x1\ x2\ x0\ x4\ x5\ [x12\ x13\ x14][x6\ x7\ x8[x2\ x9]x3\ x10]x11\ x15\ x16\ x0.$$

The symbol [ ] enclosing the nodes of the graph corresponds to tasks that may be performed in parallel manner. The execution of such tasks do not depend on each other. These are shown by the equality of the time that the algorithm traverses their associated arcs.

From the specifications of modelling and activity extraction in RDs in [74], the following problems can be identified,

(i) When the input RD of the algorithm for activity extraction contains multiple activities, there might be components and conditions that are included in the extraction which do not contribute to the execution of the intended case. This happens due to the nondeterminism of the choice of outdegree in step 1 of *Alg*. The traversals could still lead to an unintended output node and/or including a subsystem which are entirely unnecessary for the reachability of the intended output/s. This can lead to errors in the profile of the structure and behavior of the subsystem being associated to the extracted activity.

With the aforementioned issue, the existing algorithm also fails to achieve the correct temporal profile of the extracted components. That is, the time of traversals(which implies the time of task executions) is wrongly set when components that are unnecessary for the intended activity are included in the description of the activity.

(ii) The time of traversals for arcs leading to a join for which their arc attributes are marked ϵ are not discriminated against those marked with Σ during updates of T in step (3) of *Alg*. This discrimination is important because the former set of arcs must not be affected by the arc that causes the update because they should not affect the reachability of nodes in R. They, in fact, correspond to sequential executions of two successive tasks which do not have input requirements/parameter-based conditions.

(iii) Although the resource and process dimensions are inherently used in RDs(apart from the case dimension) by the presence of objects and controllers, structural relations of related components are still undefined. Therefore, the definition of role-related subsystems are still undefined.

(iv) An immediate result to (iii) is the absence of support for subsystem-based controls in RDs. The concept of resets or cancellation regions in workflows to support volatility for complex systems modelling is unsupported.

(v) Explicit controls for effective loop mechanisms are absent in the current modelling and activity extraction in RDs.

(vi) There are still no model properties for describing different aspects of RDs with respect to structure and behavior with consideration of the integration of the three workflow dimensions in one RD model. With this, there are also no verification schemes for checking such properties in RDs.

3 Robustness Diagrams with Loop and Time Controls

This chapter presents the core theoretical contributions of this research. It presents the details of the proposed concepts and methods which are mentioned in the Methodological Approach in Chapter 1. These concepts and methods shall address the problems and achieve the goals that are established in this research for modelling and verification of complex systems. In particular, the UML's Robustness Diagram is extended to support the integration of all three workflow dimensions into one modelling framework. This extension additionally includes the consideration of modelling complex systems with requirements of multi-activity construction, persistence, and volatility for structure and behavioral profiles. Such an extension is referred to in this research as the *Robustness Diagram with Loop and Time Controls*(RDLT). This chapter discusses the structural and behavioral profiles and the methodologies that are used to extract them in RDLTs. These profiles are either captured through a holistic or a compositional approach in model analysis for RDLTs. The holistic approach takes in a multi-activity RDLT and creates an extended RDLT thereof. From its extended RDLT, conclusions to each substructure that supports the execution of an activity profile for a completion of a case are derived. Meanwhile, the compositional approach decomposes the input RDLT into these substructures. Each of them are analyzed to establish generalizations for the entire model with respect to compliance of properties. These approaches are detailed in this chapter. A set of these model properties are proposed to help in the verification of specific aspects pertaining to these RDLT profiles. Lastly, this chapter provides proofs that establishes the relationships and hierarchies of RDLTs with respect to their compliance of the proposed properties in modelling.

3.1 RDs with Loop and Time Controls

Definitions and Basic Notations

Definition 20 (Robustness Diagram with Loop and Time Controls(RDLT)) *A Robustness Diagram with Loop and Time Controls(RDLT) is a graph representation R of a system that is defined as $R = (V, E, T, M)$ where*

- *V is a finite set of vertices where every vertex has a type $V_{type} : V \to \{`b`, `e`, `c`\}$, where $`b`, `e`$, and $`c`$ means the vertex is either a "boundary object", "entity object", or a "controller", respectively.*

- *A finite set of arcs $E \subseteq (V \times V)\backslash E'$ where $E' = \{(x, y)|x, y \in V, V_{type}(x) \in \{`b`, `e`\}, V_{type}(y) \in \{`b`, `e`\}\}$ with the following attributes with user-defined values,*

  - *$C : E \to \Sigma \cup \{\epsilon\}$ where Σ is a finite non-empty set of symbols and ϵ is the empty string. Note that for real-world systems, a task $v \in V$, i.e. $V_{type}(v) = `c`$, is executed by a component $u \in V$, $V_{type}(u) \in \{`b`, `e`\}$. This component-task association is represented by the arc $(u, v) \in E$ where $C((u, v)) = \epsilon$. Furthermore, $C((x, y)) \in \Sigma$ represents a constraint to be satisfied to reach y from x. This constraint can represent either an input requirement or a parameter $C((x, y))$ which needs to be satisfied to proceed from using the component/task x to y. $C((x, y)) = \epsilon$ represents a constraint-free process flow to reach y from x or a self-loop when $x = y$.*
  - *$L : E \to \mathbb{Z}^+$ is the maximum number of traversals allowed on the arc.*

- *Let T be a mapping such that $T((x, y)) = (t_1, \ldots, t_n)$ for every $(x, y) \in E$ where $n = L((x, y))$ and $t_i \in \mathbb{N}$ is the time a check or traversal is done on (x, y) by some algorithm's walk on R.*

- *$M : V \to \{0, 1\}$ indicates whether $u \in V$ and every $v \in V$ where $(u, v) \in E$ and $C((u, v)) = \epsilon$ induce a subgraph G_u of R known as a **reset-bound subsystem(RBS)**. The RBS G_u is induced with the said vertices when $M(u) = 1$. In this case, u is referred to as the **center** of the RBS G_u. G_u's vertex set V_{G_u} contains u and every such v, and its arc set E_{G_u} has $(x, y) \in E$ if $x, y \in V_{G_u}$.*

  *Finally, $(a, b) \in E$ is called an **in-bridge** of b if $a \notin V_{G_u}$, $b \in V_{G_u}$. Meanwhile, $(b, a) \in E$ is called an **out-bridge** of b if $b \in V_{G_u}$ and $a \notin V_{G_u}$. Arcs $(a, b), (c, d) \in E$ are **type-alike** if $\exists y \in V$ where $(a, b), (c, d) \in Bridges(y)$ with $Bridges(y) = \{(r, s) \in E|(r, s)$ is either an in-bridge or out-bridge of $y\}$ or if $\forall y \in V$, $(a, b), (c, d) \notin Bridges(y)$.*

Figure 3.1 shows an example of an RDLT R having 2 boundary and 1 entity objects and an RBS(marked with a box) with center **x1**. The component-task relation is emphasized by the sameness in vertex coloring. Note that (**u4**, **x1**) and (**u5**, **x1**) are type-alike since they are in-bridges of **x1**. Also, (**x6**, **x1**) is not type-alike with (**u4**, **x1**) and (**u5**, **x1**).

(**x6**, **y**) is an out-bridge of **x6**. The arc attributes are shown as $L((x, y)) : C((x, y))$ for every $(x, y) \in E$ in the figure.

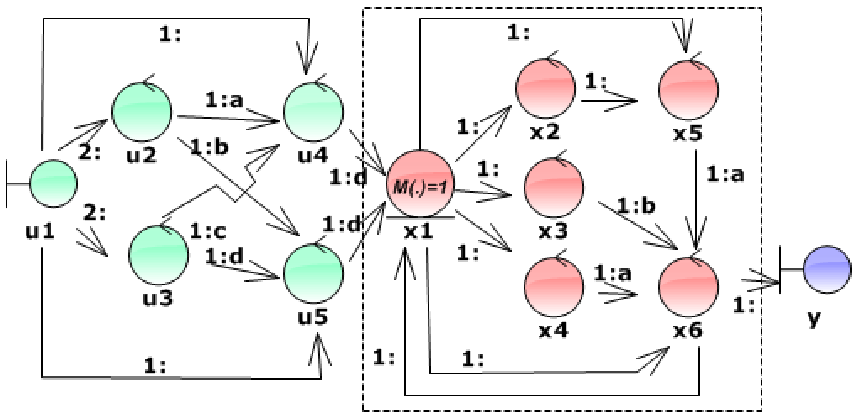


Figure 3.1: RLDT R with 1 entity and 2 boundary objects with an RBS with center **x1**.

Note that the default value of $T((x, y)) = \mathbf{0}$, where $(x, y) \in E$, $x, y \in V$, and $\mathbf{0}$ is the zero vector, before any traversal algorithm is implemented on R. Moreover, an RBS will have the values of $T(.)$ for all its arcs reset to $\mathbf{0}$ when some of its arcs are traversed. This reset process is formalized in the succeeding subsection below.

Activity Extraction in RDLTs

We propose an algorithm $\mathcal{A}$ to perform activity extraction using RDLT R where R contains (1)multiple activities by the integration of all three workflow dimensions, (2)typed arcs and substructures by virtue of C and M, (3)attribute-dependent (i.e. C-, T- and L-based) paths, loops, joins, and splits, and (4)persistent and volatile components in regions that are defined to be a part of reset-bound subsystems. The algorithm $\mathcal{A}$ is an improvement to the activity extraction algorithm from [74](see Section 2.3). $\mathcal{A}$ can generate an activity profile with the four aforementioned specifications of structure, topology, and groupings using a start vertex s and a goal vertex f. From this input pair, $\mathcal{A}$ identifies a set of vertices which are used to execute one activity in R. Note that more than one activity can be extracted from R.

One approach to perform extraction of multiple activities in R is implementing copies of $\mathcal{A}$ in R with varying combinations of the said input pair. By using a parallelized design, two copies will choose two different vertices in R when splits of process flows are observed. However, this parallelized implementation is only effective when there are no joins that are found in the set of succeeding tasks in R. A split is observed in Figure 3.1 at **u2** where two process flows are generated from **u2** to **u4** and from **u2** to **u5**. Meanwhile, a merging point is seen at **x6**. However, **x1** is not a merging point of the flows. This is because splits and joins of process flows are bound to the values of arc attributes $C(.)$ and $M(.)$ in R. In the succeeding subsection, different types of process flows are illustrated where the role of attributes of R that affects these flows is shown. In essence, prior checking on the connectivities and these arc attributes of R is required before performing this parallel design. On this note, another approach to achieve an efficient multi-activity extraction is to derive maximal subgraphs in R whose vertices, arcs, and RDLT attributes are relevant to the given input pair. This approach can adopt extraction of activities either one at a time using the entire R or by the parallelized design using the maximal subgraphs of R. Therefore, we propose another algorithm P as preprocessing step to discover these subgraphs accounting connectivities, arc attributes, splits and merging of process flows. P uses the arc attribute $C(.)$ to generate maximal subgraphs in RDLTs containing multiple activities, inputs, and outputs. We shall discuss how to deal with such RDLTs in activity extraction and identification of maximal subgraphs and how model properties are checked for them in Section 3.5. For the meantime, an algorithm that performs activity extraction using s and f in R is proposed in the subsequent subsection. Consequently, the time and space complexity for this algorithm is also provided.

Proposed Algorithm for Activity Extraction

To perform a step of an activity extraction from a vertex x, where $x = s$ at the start of the algorithm $\mathcal{A}$, $\mathcal{A}$ performs a **check** on x and any other vertex $w \in V$ where $(x, w) \in E$. A check means that $\mathcal{A}$ arbitrarily selects $(x, y) \in E$ where the number of traversals done on (x, y) has not reached $L((x, y))$, if such y exists in V. If y exists, $\mathcal{A}$ gets the largest value $maxV$ from all the time of traversals $T((u, x))$ for every $(u, x) \in E$. If such $(u, x) \notin E$, $maxV = 0$. The process of checking ends when $\mathcal{A}$ updates the leftmost zero of the vector $T((x, y)) = maxV + 1$. Note that two successive checks on (x, y) are not allowed in Algorithm 3.1(*Check*(x)) by using the indicator vector $CTInd_{(x,y)}[.]$ associated to (x, y). The values 0, 1, and 2 in an element of $CTInd_{(x,y)}[.]$ mean that (x, y) has been (a)neither checked nor traversed, (b)checked, and (c)traversed, respectively, by $\mathcal{A}$.

After the check, $\mathcal{A}$ evaluates whether every $(v, y) \in E$, $v \in V$, does not prevent traversal on (x, y), where (x, y) and (v, y) are type-alike. Such an arc (v, y) does not prevent traversal on (x, y) if either the following holds, (a) both arcs have the same value of C or $C((v, y)) = \epsilon$, (b) the number of traversals done on (x, y) has not exceeded that of (v, y)'s (and $L((v, y))$), or (c) there is no unsatisfied constraint $C((v, y))$ (imposed by (v, y)) to reach y from v since (v, y) was previously explored by $\mathcal{A}$. The second constraint implies that $C((x, y)), C((v, y)) \in \Sigma$, $C((x, y)) \neq C((v, y))$, while the third indicates a loop where v, y, and x are present. Here, y is reached by $\mathcal{A}$ before x. We call (x, y) **unconstrained** if any of these constraints are satisfied. On another hand, if (x, y) and (v, y) are not type-alike, (v, y) will not prevent traversal of (x, y), i.e. (x, y) remains unconstrained if it is such with respect to to other type-alike arcs incident to y. Formally,

Definition 21 (Unconstrained Arc) *Let $R = (V, E, T, M)$ be an RDLT(with the arc attributes C and L). An arc $(x, y) \in E$ is* ***unconstrained*** *if $\forall (v, y) \in E$ where (x, y) and (v, y) are type-alike, any of the following holds,*

1. $C((v, y)) \in \{\epsilon, C((x, y))\}$,

Algorithm 3.1: *Check*(x): Determines if there exists some $y \in V$ where $(x, y) \in E$ and its attribute $L((x, y))$ allows that at least one traversal on the arc. If y exists, the algorithm updates $T((x, y))$ and returns y, otherwise it returns $\emptyset$.

```
Input: $x \in V$
Output: $y \in V$ if the arc attribute $L((x, y))$ allows that at least one traversal on
        the arc, $\emptyset$ otherwise
1  $y \leftarrow w$ where $w \in V$, $(x, w) \in E$ and for $1 \leq i \leq |L(x, w)|$ such that either
     $CTInd_{(x,y)}[i-1] = 2$ and $CTInd_{(x,y)}[i] = 0$, or $CTInd_{(x,y)}[i = 1] = 0$;
2  if $y \neq \emptyset$ then
3      if $\exists u \in V$ such that $(u, x) \in E$ then
4          $t_i \in T((x, y)) : t_i \leftarrow maxV + 1$ where
           $maxV = \max_{(u,x) \in E} \{\max\{t_k \mid 1 \leq k \leq L((u, x)), t_k \in T((u, x))\}\} + 1$;
5      else
6          $t_i \in T((x, y)) : t_i \leftarrow 1$;
7      end
8      return $y$;
9  else
10     return $\emptyset$;
11 end
```

2. $|\{t_i \in T((x, y))|t_i \geq 1\}| \leq |\{t_j \in T((v, y))|t_j \geq 1\}| \leq L((v, y))$,

3. $C((v, y)) \in \Sigma \wedge C((x, y)) = \epsilon \wedge T(v, y) \neq \mathbf{0}$.

If (x, y) is unconstrained, $\mathcal{A}$ traverses it. $\mathcal{A}$ either retains or updates $T((v, y))$ for every $(v, y) \in E$ where (x, y) and (v, y) are type-alike, depending on $C((v, y))$ and $T((v, y))$. For every such $(v, y) \in E$, $T((v, y))$ is updated if either (1) $C((v, y)) \in \Sigma$, or (2) $v \in V$ is either an entity or a boundary object and y is a controller and $C((v, y)) = \epsilon$ and $|\{t_k \in T((v, y))|t_k \geq 1\}| = 0$. For every $(v, y) \in E$ that satisfies the first condition, $\mathcal{A}$ updates its last value in $T((v, y))$ where the last check was done on (v, y). On another hand, for every $(v, y) \in E$ that satisfies the second condition, $\mathcal{A}$ updates its first value in $T((v, y))$. This update signals the first use of the task represented by y as its object v has already been previously reached by $\mathcal{A}$. The value that is assigned by $\mathcal{A}$ to every such component of $T((v, y))$ is $MAX + 1$, where MAX is the maximum value of all $T(v', y)$, $\forall v' \in V$ where (v', y) and (v', y) and (x, y) are type-alike. With this, y is considered as **reachable** at time $MAX + 1$. In real world systems, this means that constraints/preliminary tasks with respect to y had already been satisfied/executed by this time step. Note that the updates on $T((v, y))$ using MAX considers that $\mathcal{A}$ might have previously explored vertices which generates bigger values of $T((v, y))$ during the last check on (v, y) as compared with (x, y)'s.

Note that two successive checks on (x, y) by $\mathcal{A}$ is not allowed, i.e. (x, y) needs to be traversed by $\mathcal{A}$ first before it can be selected again by the check process. When (x, y) is traversed and (x, y) is an out-bridge of x, every $T((a, b))$ is reset to $\mathbf{0}$ where (a, b) is in the arc set of the RBS where x belongs to in its vertex set. The process of checks and traversals are repeated until the terminal vertex f is reached by $\mathcal{A}$.

Meanwhile, if (x, y) is not unconstrained and checks are not allowed on any other $(x, y') \in E$, $\mathcal{A}$ backtracks to $a \in V$ where $(a, x) \in E$ and a was previously used by $\mathcal{A}$ to reach x. Then, $\mathcal{A}$ proceeds with checking and traversals using a.

Profiles of Reachability in RDLTs

Definition 22 *A* ***reachability configuration*** $S(t)$ *in* $R = (V, E, T, M)$ *contains the arcs traversed by* $\mathcal{A}$ *at time step* $t \in \mathbb{N}$. *We call a set* $S = \{S(1), S(2), \ldots, S(d)\}$ *of reachibility configurations,* $d \in \mathbb{N}$*, as an* ***activity profile*** *in* R *where* $\exists (u, v) \in S(1)$ *and* $(x, y) \in S(d)$ *such that* $\nexists w, z \in V$ *where* $(w, u), (y, z) \in E$.

Definition 23 *Let* $R = (V, E, T, M)$ *be an RDLT. A vertex* $v_n \in V$ *is* ***reachable*** *from any vertex* $v_1 \in V$ *if there exists an activity profile* $S = \{S(1), S(2), \ldots, S(d)\}$ *of* R*,* $d \in \mathbb{N}$*, and a sequence of vertices* $p = v_1 \ldots v_n$*,* $v_i \in V$ *where for some* $t \in \{1, \ldots, d\}$ *and* $i = 1, \ldots, n-1$ *such that* $1 \leq n-1 \leq d-t+1$*,* $(v_i, v_{i+1}) \in S(t + (i-1))$.

Algorithm 3.2(*TraverseAndUpdate*$((x, y))$) shows the process of updating an activity profile S and all the arc attributes T of the arcs used in an activity where $(x, y) \in E$, $x, y \in V$.

Applying $\mathcal{A}$ on the RDLT in Figure 3.1 using **u1** and **y** as a start and goal vertices, resp, we obtain one activity profile $S = \{S(1), S(2), \ldots, S(6)\}$ where $S(1) = \{$(**u1**, **u2**), (**u1**, **u3**)$\}$, $S(2) = \{$(**u2**, **u4**), (**u3**, **u4**), (**u1**, **u4**),$\}$, $S(3) = \{$(**u4**, **x1**)$\}$, $S(4) = \{$(**x1**, **x3**), (**x1**, **x4**)$\}$, $S(5) = \{$(**x3**, **x6**), (**x4**, **x6**), (**x1**, **x6**)$\}$, and $S(6) = \{$(**x6**, **y**)$\}$. Note that **x1** is immediately reached from **u4** because (**u4**, **x1**) is unconstrained with respect to all of its type-alike arcs, for this case, (**u5**, **x1**). This is because the first condition of Definition 21 to perform the traversal is satisfied. Also, (**x6**, **x1**) does not affect the traversal of (**u4**, **x1**) because it is not type-alike with the latter. Furthermore, when (**x3**, **x6**) and (**x4**, **x6**) at time $t = 5$, the latter becomes an unconstrained arc by satisfying the first and second conditions of Definition 21. Therefore, (**x4**, **x6**) is traversed together with (**x3**, **x6**) with updates to their arc attributes $T($(**x4**, **x6**)$) = T($(**x3**, **x6**)$) = 5$. Therefore, **x6** is reachable at $t = 5$. Finally, when (**x6**, **y**) is traversed at $t = 6$, the values of T will be reset to 0 for the following arcs: (**x1**, **x2**), (**x1**, **x3**), (**x1**, **x4**), (**x1**, **x5**), (**x1**, **x6**), (**x2**, **x5**), (**x3**, **x6**), (**x4**, **x6**). The values of T for the in-bridge (**u4**, **x1**) and out-bridge (**x6**, **y**) are not affected by this reset.

Meanwhile, another activity can be generated using another reachability configuration from **u1** to **y**, i.e. $S = \{S(1), S(2), \ldots, S(7)\}$ where $S(1) = \{$(**u1**, **u2**), (**u1**, **u3**)$\}$, $S(2) = \{$(**u2**, **u5**), (**u3**, **u5**), (**u1**, **u5**)$\}$, $S(3) = \{$(**u5**, **x1**)$\}$, $S(4) = \{$(**x1**, **x2**),(**x1**, **x3**)$\}$, $S(5) = \{$(**x2**, **x5**), (**x1**, **x5**)$\}$, $S(6) = \{$(**x3**, **x6**), (**x5**, **x6**), (**x1**, **x6**)$\}$, and $S(7) = \{$(**x6**, **y**)$\}$.

Shown below is the entire algorithm $\mathcal{A}$ which uses Algorithms 3.1(*Check*()) and 3.2(*TraverseAndUpdate*()) to extract an activity profile from R.

Algorithm 3.2: *TraverseAndUpdate*$((x,y))$ Evaluating if (x,y) unconstrained. If it is unconstrained, the algorithm updates $T(v,y)$ and an activity profile S which uses (x,y) accordingly, and then returns y, otherwise, it returns $\emptyset$.

```
   Input: Checked $(x,y)$
   Output: $y$ if $(x,y)$ is unconstrained, $\emptyset$ otherwise
1  if $\forall (v,y) \in E$, $v \in V$, where $(v,y),(x,y)$ are type-alike, $(C((v,y)) \in \{\epsilon, C((x,y))\})$
     or $(|\{t_i \in T((x,y)) | t_i \geq 1\}| \leq |\{t_j \in T((v,y)) | t_j \geq 1\}| \leq L((v,y)))$ or
     $(C((v,y)) \in \Sigma \wedge C((x,y)) = \epsilon \wedge T(v,y) \neq \mathbf{0})$ then
2      $MAX \leftarrow \max_{\text{type-alike } (x,y),(v',y) \in E} \{ \max_{t_k \in T((v',y))} \{t_k\}\}$;
3      $CTInd_{(x,y)}[i] \leftarrow 2$ where either $CTInd_{(x,y)}[i] = 1$ and $CTInd_{(x,y)}[i-1] = 2$, or
         $CTInd_{(x,y)}[i = 1] = 1$;
4      $t_i \in T((x,y)) : t_i \leftarrow MAX + 1$;
5      $S(MAX + 1) \leftarrow (x,y)$;
6      for each $(v,y) \in E$ where $(v,y)$ and $(x,y)$ are type-alike, such that either
         (1) $C((v,y)) \in \Sigma$, or (2) $v \in V$ is either an entity or a boundary object and
         $y$ is a controller and $C((v,y)) = \epsilon$ and $|\{t_k \in T((v,y)) | t_k \geq 1\}| = 0$ do
7          $CTInd_{(v,y)}[j] \leftarrow 2$ where either $CTInd_{(v,y)}[j] = 1$ and
             $CTInd_{(v,y)}[j-1] = 2$, or $CTInd_{(v,y)}[j = 1] = 1$;
8          $t_j \in T((v,y)) : t_j \leftarrow MAX + 1$;
9          $S(MAX + 1) \leftarrow (v,y)$;
10     end
11     if $(x,y)$ is an out-bridge of $x \in V_{G_u}$ of the RBS $G_u$ (with arc set $E_{G_u}$), with
         center $u \in V$ then
12         $T((a,b)) \leftarrow \mathbf{0}$, for every $(a,b) \in E_{G_u}$;
13         $CTInd_{(a,b)}[i] \leftarrow \mathbf{0}$, for every $(a,b) \in E_{G_u}$;
14     end
15     return $y$;
16 else
17     return $\emptyset$;
18 end
```

Algorithm 3.3: Algorithm $\mathcal{A}$ to extract an activity profile from R using s and f.

```
Input: RDLT R, and start $s$ and goal $f$ vertices, $s, f \in V$, where $\nexists w, z \in V$ where
       $(w, s), (f, z) \in E$
Output: Activity profile $S$ if an activity is extracted by $\mathcal{A}$ for $s$ and $f$, otherwise
        it returns $\emptyset$.
$S \leftarrow \{S(1), S(2), \ldots, S(d)\}$, $d \in \mathbb{N}$;
$S(t) \leftarrow \emptyset$, $t = 1, 2, \ldots, d$;
$a \leftarrow \emptyset$;
$x \leftarrow s$;
while $x \neq f$ do
    $x_0 \leftarrow x$;
    $y \leftarrow Check(x)$;
    while $y \neq \emptyset$ do
        $y \leftarrow TraverseAndUpdate(x, y)$;
        if $y \neq \emptyset$ then
            $a \leftarrow x$;
            //traverse $(x, y)$
            $x \leftarrow y$;
        end
        $y \leftarrow Check(x)$;
    end
    if $x_0 = x$ then
        //i.e. no traversal was done by $\mathcal{A}$ from $x$, $\mathcal{A}$ backtracks
        if $x \neq s$ then
            //backtrack from $x$ to $a \in V$ is possible
            $x \leftarrow a$;
        else
            //activity extraction from $s$ to $f$ fails
            return $\emptyset$;
        end
    end
end
return $S$;
```

Remark 1 *The time and space complexity of $\mathcal{A}$ is $O(|E| * l)$, $l = \max_{(x,y)\in E}\{L((x,y))\}$ and $O(|V|^2)$, respectively.*

Control Flow Designs in RDLTs and Notes in Construction

The checks and traversals of $\mathcal{A}$ are tied with the constraints imposed by R in terms of its arc attributes T, C, and the structural composition of the vertices and their connectivities. Traversals of some pair of arcs (u,v) and (w,v) in R, for instance, happen simultaneously because of connectivities or the setting of their arc attributes C to a value in Σ or both. Bearing in mind the check and traversal conditions in Definition 21, the following discussions show the consequential connectivities and relationships of the components and attributes with respect to designing control flows in RDLTs. These discussions also include how these results for designing control flow schemes in RDLT relate to the ones known in literature for well-known workflows.

Note that for simplicity, arcs with no annotations for T and L in the following figures are understood to be set to $\mathbf{0}$ and 1, respectively. (When the reader desires to consider process flows in the RDLT R' where $T(.) \neq \mathbf{0}$ and $L(.) \neq 1$, the list of conditions for traversals shown above can be easily applied.)

- **Sequential Flow**. Sequential flows from one task u to v in R, $u, v \in V$ are modelled by $(u,v) \in E$ where for every $w \in V$ where $(w,v) \in E$, $C((w,v)) \in \{\epsilon, C((u,v))\}$.

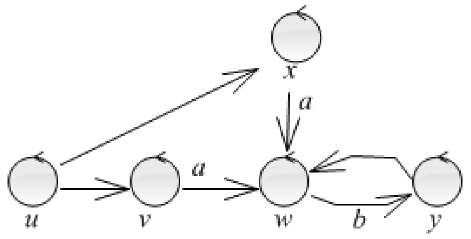


Figure 3.2: Unconstrained arcs of sequential process flows for RDLTs. (Note that $L(.) = 1$ for this RDLT instance.)

  Figure 3.2 illustrates sequential flows in the arcs (u,v), (u,w), (w,y). Similarly, the same type of flow holds for $(u,x), (x,w), (w,y)$. Note that at any arc $(p,q) \in E$ in this RDLT can be traversed because all of them are unconstrained arcs.

- **Conditional Flow and Parallelism**. Conditional flows pertain to process flow specifications of splits and joins in RDLTs. In this research, they are determined by the attributes for both vertices and arcs in the RDLT models.

  A *conditional split* is a structural relationship involving the vertices $u, v, w \in V$ where $(u,v), (u,w) \in E$ and $C((u,v)) \neq C((u,w))$ with $C((u,v)), C((u,w)) \in \Sigma$. They imply two different constraints $C((u,v))$ and $C((u,w))$ that lead to two different flows wherein each flow is executed when its corresponding constraint is satisfied. Note that the inequality of the values of C on these arcs do not imply mutual exclusivity of the two process flows that begin from u to v and from u to w. That is, the execution of one does not imply the nonexecution of the other. However, they imply that these process flows can induce the existence of at least two activity profiles in the system. This happens when these process flows are designed such that either of the two settings hold: (a)they meet at a vertex $z \in V$ using their corresponding flows' arcs (v', z) and (w', z) where $C((v', z)) = C((w', z)) \in \{\epsilon, \Sigma\}$, or (b) they do not meet at all. Disregarding the concept of reset-bound subsystems in this research, the first setting is analogous to having a structure that is composed of a *well-structured* OR-split-OR-join pattern [25] in workflows. This setting is also shown in Figure 3.3.(i). Needless to say, to put a structure having the conditional split and then having their process flows meet at z where none of these two settings are satisfied, i.e. $C((v', z)) \neq C((w', z))$ and $C((v', z)), C((w', z)) \in \Sigma$ defeat the purpose of having the constraints $C((u, v)) \neq C((u, w))$ at the onset. However, by not imposing mutual exclusivity on the process flows, there is freedom in giving designers multiple capabilities for condition-driven flows. The problem of implicit splits in BPMN is avoided because actionable attributes are expressed explicitly in the arcs. Together with the mentioned settings, it is apparent in the model whether the split induces process flows which can either be mutually exclusive or not given the attributes in RDLTs.

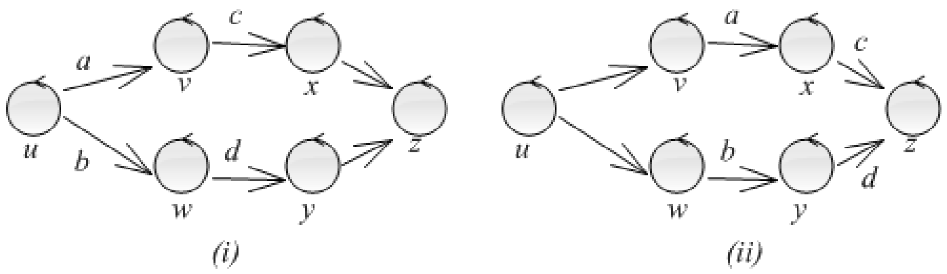


Figure 3.3: Control flow structures showing instances of splits and a join in RDLTs. (Note that $L(.) = 1$ for this RDLT instance.)

On another hand, a *parallel split* is a structural relationship involving $u, v, w \in V$ where $C((u, v)) = C((u, w)) = \epsilon$. Roughly speaking, the arcs simply represent two disjoint flows that are executed altogether and do not have constraints imposed in their execution. Figure 3.3.(ii) shows this relationship for u, v, and w. Furthermore, the two process flows that are induced from these relationships might support the execution of one activity profile as can be seen in Figure 3.3.(ii).

A *conditional join* is a structural relationship between the vertices $x, y, z \in V$ where (x, z) and (y, z) are type-alike arcs in E and $C((x, z)) \neq C((y, z))$ with $C((x, z)), C((y, z)) \in \Sigma$. This type of join imply the satisfaction of both requirements before z may be executed. (Note that the conditions for traversals still consider L and T for the reachability of z.) Furthermore, this relationship also imply that x, y, and z will altogether be present in every activity profile where z is involved in. A structure that is composed of a parallel split where all its induced process flow meet using conditional join can cover the modelling of the well-structured nets with respect to AND-split-AND-join pattern [25]. Coverage, in this sense, imply that the AND-split-AND-join pattern [25] is not semantically equivalent to the parallel-split-parallel-join combination. This inequivalence is apparent because the AND-split-AND-join pattern [25] cannot fully model parallel-split-parallel-join combination because of the additional restrictions present when $M(z) = 1$, i.e. z is a vertex in a reset-bound subsystem in the RDLT.

Remark 2 *By simply looking at M of the vertices and C involved in process flows that end up at a common vertex z, a k-out-of-n join can be enacted at z. Furthermore, subsets of these flows with a subset size k can be explicitly represented to discriminate which subsets are sufficient so z, and its succeeding tasks, are executed. Both of these are possible because of the traversal conditions of the proposed algorithm $\mathcal{A}$ for activity extraction(see Section 3.1). These mechanisms for join specifications remain robust despite looped structures because the traversal conditions also account looping in models.*

- **Iteration**. Iteration is a process flow derivable from a structure in R where $(x_1, x_2), (x_2, x_3), \ldots, (x_{n-1}, x_n), (x_n, x_1) \in E$, $x_i \in V$, and the arc (x_n, x_1) models a sequential flow from x_n to x_1. In this case, the task x_1 is executed again as well as every (x_i, x_{i+1}) if the values of $L((x_i, x_{i+1}))$ support repeated checks and traversals on (x_i, x_{i+1}), $i = 1, 2, \ldots, n-1$.

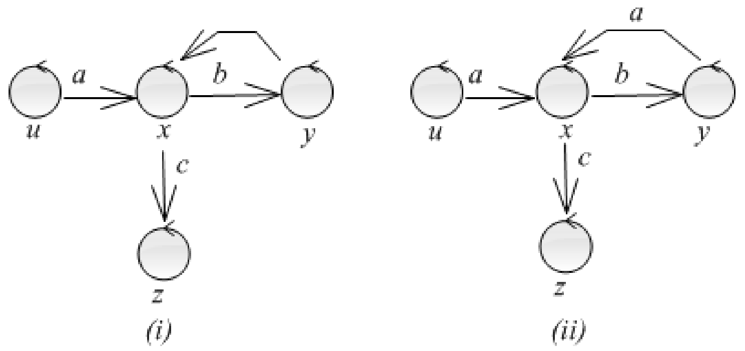


Figure 3.4: Iteration-conditional process flow for RDLTs. (Note that $L(.) = 1$ for this RDLT instance.)

In Figure 3.4.(i), a precondition $C((u, x)) = a$ is satisfied before an iteration structure is encountered between x and y. When y is reached by the algorithm and a check on (y) yields x, (y, x) will be marked as unconstrained because it satisfies the third condition in Definition 21. In this case, the task x is reached/executed again. When x is checked again, z is the sole output of *Check*(x) because the number of check/traversals of (x, y) has reached $L((x, y))$. In this case, only x is iterated for the structure and (x, z) is traversed instead. On another hand, a similar control flow can be enforced for iteration as shown in Figure 3.4.(ii). For this design, $C((y, x)) = a$ is tested again after executing the task y. This design can be used to evaluate whether a parameter considered in the constraint a has reached a specific value for iteration purposes. Note that both (u, x) and (y, x) are unconstrained when u and y are reached, respectively, since they satisfy the first condition of Definition 21.

- **Reset-bound subsystems**(see Definition 20). The use of RBS in designs of RDLTs can pose challenges for reachability and use of components and tasks. However, due to its definition that induces a group-based control for constraint-checking and resets of attribute values, these challenges are contained and easily isolated from other substructures in RDLT models. Encapsulation and atomicity in representation are therefore natural in their design. The following discussions show scenarios of RBS designs in RDLTs and their implications to the execution of activities when they are involved in processes such as conditional and parallel flows.

Scenario 1 (RBS in Parallel Flows)

Input Configuration: *A pair of parallel flows that start at $x \in V$ and merge at $q \in V$ which use the structures $Q_1 = \{(x, y_1), (y_1, y_2), \ldots, (y_{n-1}, y_n), (y_n, q)\}$ and $Q_2 = \{(x, z_1), (z_1, z_2), \ldots, (z_{n-1}, z_n), (z_n, q)\}$ where $Q_1 \cap Q_2 = \emptyset$, $Q_1 \cup Q_2 \subseteq E$, $Q_1 \cap E_{V_{G_u}} \neq \emptyset$, $Q_2 \cap E_{V_{G_u}} \neq \emptyset$, $C((x, y_1)) = C((x, z_1)) = \epsilon$, with x, y_1, and z_1 being distinct from each other, and $r \neq s$ for every $(r, s) \in Q_1 \cup Q_2$, $x_i \neq y_j$ for all vertices used in the arcs of Q_1 and Q_2*

Design at Merging Point: $C((y_n, q)) \neq C((z_n, q))$, $C((y_n, q)), C((z_n, q)) \in \Sigma$

Design of Out-bridges:

- $(x_i, x_{i+1}) \in Q_1$ *is an out-bridge of* $x_i \in V_{G_u}$ *iff* $(y_i, y_{i+1}) \in Q_2$ *is an out-bridge of* $y_i \in V_{G_u}$
- (x, y_1) *is an out-bridge of* $x \in V_{G_u}$ *iff* $(x, z_1) \in Q_2$ *is an out-bridge of* x
- (y_n, q) *is an out-bridge of* $y_n \in V_{G_u}$ *iff* $(z_n, q) \in Q_2$ *is an out-bridge of* z_n *and* $\nexists(w, q) \in E$, $w \in V$, $C((w, q)) \in \Sigma$, *where* $C((w, q)) \in \{C((y_n, q)), C((z_n, q))\}$ *and* $\{(w, q), (y_n, q), (z_n, q)\}$ *are type-alike*

Implication of Designs: *These designs provide that no resets to $T((a, b))$, for every $(a, b) \in E_{G_u}$, are done at any time step of the check-traversal operations of $\mathcal{A}$ that results to the loss of information of the time of reachability of all other vertices in V. This loss induces erroneous values assigned to $T(.)$ for some vertices that are subsequently reached by using some arcs in $Q_1 \cup Q_2$. Furthermore, the design specification at the merging point q of the two flows using Q_1 and Q_2 imposes that they are completed simultaneously when q is executed by $\mathcal{A}$.*

Note that the equality in the number of arcs in Q_1 and Q_2 assures that for any traversal made by $\mathcal{A}$ using any out-bridge $(w, v) \in E$ of $w \in V_{G_u}$, $v \in V$, resets of $T((x_i, x_{i+1}))$ and $T((y_i, y_{i+1}))$ happen simultaneously, as well as for (x, y_1) with respect to (x, z_1) and (y_n, q) with respect to (z_n, q). This equality is particularly necessary when every step of the checking of $\mathcal{A}$ on the arcs in Q_1 and Q_2 determines that any of these selected arcs are unconstrained, therefore, no backtracks are necessary.

Figure 3.5 provides an instance of Scenario 1. That is, $u \in V_{G_u}$ and $z \in V$ are the vertices which start and end two parallel flows using $Q_1 = \{(u, v), (v, x), (x, z)\}$ and $Q_2 = \{(u, w), (w, y), (y, z)\}$.

Note that these designs in Scenario 1 can be extended to parallel flows using Q_1 and Q_2 where $|Q_1| \neq |Q_2|$ and/or $|Q_1 \cap Q_2| \geq 1$. This is seen by taking different pairs of the parallel flows in Figure 3.5 and verifying that out-bridges for Q_1 and Q_2 are traversed simultaneously when $\mathcal{A}$ uses all the arcs Q_1 and Q_2.

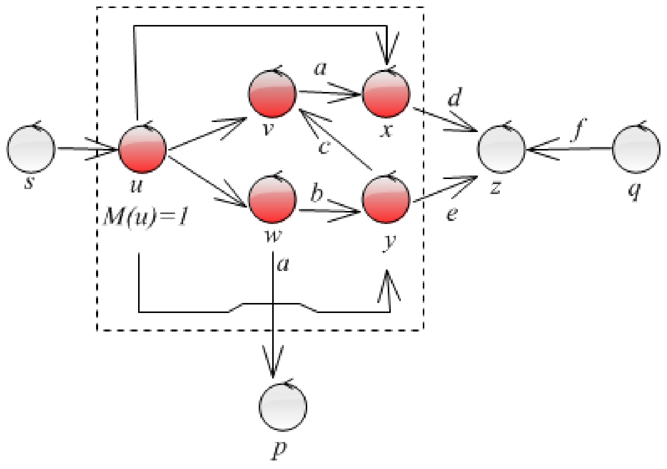


Figure 3.5: RBS process flow for RDLTs.

Scenario 2 (RBS in Conditional Flows)

Input Configuration: *A pair of conditional flows that start at $x \in V$ and merge at $q \in V$ which use the structures $Q_1 = \{(x, y_1), (y_1, y_2), \ldots, (y_{n-1}, y_n), (y_m, q)\}$ and $Q_2 = \{(x, z_1), (z_1, z_2), \ldots, (z_{n-1}, z_n), (z_n, q)\}$, where $m \neq n$, $Q_1 \cup Q_2 \subseteq E$, $Q_1 \cap E_{V_{G_u}} \neq \emptyset$, $Q_2 \cap E_{V_{G_u}} \neq \emptyset$, and $C((x, y_1)) \neq C((x, z_1))$, $C((x, y_1)), C((x, z_1)) \in \Sigma$*

Design at Merging Point: $C((y_m, q)) = C((z_n, q)) = \epsilon$

Implication of Designs: *With the design specification at the merging point $q \in V$ for the two conditional flows that use Q_1 and Q_2, resets caused by the traversal of an out-bridge $(r, s) \in Q_1(Q_2)$ will not affect the reachability of q when $Q_1(Q_2)$*

completes, i.e. $(y_m, q)((z_n, q))$ is traversed and q is executed by $\mathcal{A}$. That is, q is reached by using either Q_1 or Q_2.

The disjointness in conditional flows and the resets caused by a traversal of an out-bridge, whether these flows eventually merge at such $q \in V$ in Scenario 2, can be seen in Figure 3.5. Note the conditional flows that start at $w \in V$ in the said figure. If (w, p) is checked and traversed by $\mathcal{A}$, the attribute $T(.)$ in the arcs of the RBS G_u are reset to $\mathbf{0}$. The execution then proceeds from task p and to all of its descendants while the flows that start at the other choice from w, i.e. y, are not executed.

3.2 Reachability Profiles and Boundedness in RDLTs

The reachability of vertices in R does not depend only on arc connectivities. It also considers the arc attributes C and L and the different types of control flows that use them. In this section, we define reachability in terms of discoverable paths for pairs of vertices in R that account these static, structural information and constraints. We define bounds to the composition of paths, their lengths, the time of reachability of vertices, and bounds for the entire RDLT itself. We define points of delay of reachability of vertices and measure this delay when splits and merges of process flows are present. By identifying these points, we can scrutinize substructures in R and verify whether these points and the vertices which are connected to them induce unreachable vertices/subgraphs in R. We subsequently use these measures in R to identify structural relations and synchronization of the times of reachability for vertices which share a common structure in the process flows they are involved in. We establish structural characterizations for subgraphs of R under the presence of these point of delays, definable bounds for vertices and the entire model itself, and sharing of structures for supporting activities in R. In this section, we adopt and extend the free-choice property of workflows in RDLTs by using the aforementioned characterizations on R.

On Constraint-dependent Paths and Diameters

Definition 24 (Path and Path Lengths) *A **path** $p = x_1x_2 \ldots x_n$ from x_1 to x_n is a sequence of vertices $x_i \in V$ of R where $(x_i, x_{i+1}) \in E$ and $x_i \neq x_1$ and $x_i \neq x_n$ $(2 \leq i \leq n-1)$.*

*If a path exists from x_1 to x_n, then x_1 is called an **ancestor** of x_n, and x_n is called a **descendant** of x_1. Any vertex without an ancestor(descendant) and having at least one descendant(ancestor) is called a **source(sink)**. Any x and y in V are called **siblings** if $\exists w \in V$ where $(w, x), (w, y) \in E$. Furthermore, with $(w, x) \in E$, w is called the **parent** of x and x is the **child** of w.*

Finally, the maximum number of occurrences of x_i *in* p $(2 \leq i \leq n-1)$, $n \in \mathbb{N}$, *is equal to* N *where*

$$N = \prod_{l=1}^{2}\{\max\{1, \{\{\sum_{(a,b)\in In(x_i,l), C((a,b))=\epsilon} L((a,b))\} - \beta(x_i) + Sum_{\Sigma}(x_i)\} + RBS(x_i,l)\}\}$$

where

(a) $In(x_i,l) = \{(q,x_i) \in E | (q,x_i)$ *is not an in-bridge of* $x_i\}$ *if* $l = 1$, *otherwise* $\{(v,q) \in E|\ (v,q)$ *is an in-bridge of* $q \in V_{G_u}$, $v \in V$, *where* V_{G_u} *is the vertex set of an RBS* G_u *of* R *with center at* $u \in V_{G_u}$, *and* q *is an ancestor of* $x_i \in V_{G_u}$ *or* $q = x_i\}$,

(b) $\beta(x_i) = \begin{cases} 1, & \textit{if } \exists (a,b),(a',b') \in In(x_i,l), C((a,b)) = \epsilon, C((a',b')) \in \Sigma, \\ 0, & \textit{otherwise}, \end{cases}$

(c) $Sum_{\Sigma}(x_i) = \begin{cases} \sum_{(a,b)\in In(x_i,l), C((a,b))\in\Sigma} L(a,b), & \textit{if } \forall (a,b),(a',b') \in In(x_i,l) \\ & \quad \textit{where} (a,b) \neq (a',b'), \\ & \quad C((a,b)) = C((a',b')), \\ & \quad C((a,b)), C((a',b')) \in \Sigma \\ \min_{(a,b)\in In(x_i,l), C((a,b))\in\Sigma}\{L(a,b)\}, & \textit{if } \exists (a,b),(a',b') \in In(x_i,l) \\ & \quad \textit{where } (a,b) \neq (a',b'), \\ & \quad C((a,b)), C((a',b')) \in \Sigma, \\ & \quad C((a,b)) \neq C((a',b')) \\ 0, & \textit{otherwise}, \end{cases}$

(d) $RBS(x_i,l) = \begin{cases} 1, & \textit{if } \exists (q,x_i) \in E \textit{ that is an in-bridge of } x_i \in V_{G_u}, L((q,x_i)) \geq 1, \\ & q \in V, l = 1, \textit{and } \exists (v,x_i) \in E_{G_u}, \\ 0, & \textit{otherwise}. \end{cases}$

The ***length*** *of* p, *denoted as* $|p|$, *is the number of arcs used in* p, *i.e.* $|p| = n - 1$.

The ***alphabet*** *of* p, *denoted as* $Lit(p)$, *is the set* $Lit(p) = \{x_i \in V | x_i$ *occurs in* $p\}$.

Remark 3 *In contrast to the definition of paths in literature(such as paths in Petri nets in Section 2), connectivity is not the sole concern in constructing paths for RDLTs. In addition to connectivity, the preset values of the maximum traversal of arcs, i.e. values of* $L(.)$, *and the combination of the dynamic values of the time of traversals of arcs, i.e.* $T(.)$ *and the definition of reset-bound subsystems by virtue of the preset value of* $M(v)$ *of some vertex* $v \in V$ *and the typing of arcs induced by these values of* $M(.)$ *influence the construction of paths in RDLTs. By Definition 24, the maximum number that a vertex is used in a path is determined by these values in these attributes in RDLTs.*

For Definition 24.(a), the arcs considered when $l = 1$ are from x_i to its parent q where $(q,x_i) \notin E \cap E_{G_u}$. Since any $(q,x_i) \in V$ where $C((q,x_i)) = \epsilon$ does not depend on the traversal of (v, x_i), $v \in V$ after (v, x_i) has been traversed at least once, the number of occurrence of x_i can reach the sum of all $L(q, x_i)$ using every such arc (q, x_i). Their sum is used to anticipate modelling of systems having non-parallelizable tasks that are relevant to the execution of x_i. Definition 24.(b) subtracts 1 from this sum because every (v, x_i) where $C((v, x_i)) \in \Sigma$, if such exists, must be traversed at least once so (q, x_i) becomes unconstrained by Definition 21 and traversed thereafter. Furthermore, $Sum_\Sigma(x_i)$ in Definition 24.(c) considers every (q, x_i) where $C((q, x_i)) \in \Sigma$. Because of Definition 21, any pair of parents $q, q' \in V$ where $C((q, x_i)), C((q', x_i)) \in \Sigma$ and $C((q, x_i)) \neq C((q', x_i))$, the arcs (q, x_i) and (q', x_i) are traversed simultaneously. Since the values of the arc attributes $L(.)$(i.e. the maximum number of traversals allowed on an arc) are preset by user-designers themselves, these values can be differently set for (q, x_i) and (q', x_i). In this case, $Sum_\Sigma(x_i)$ is set to the minimum $L(.)$ of any of these pairs because of imposed synchronicity of the traversals by Definition 21. However, $C((w, x)) = C((q', x))$ and $C((q, x)), C((q', x)) \in \Sigma$ (for each q and q') would imply this imposed synchronicity to be invalidated by Definition 21, i.e. (q, x) and/or (q', x) are unconstrained upon reaching q and/or q'. This induces the same case as having every $C((q, x)) = C((q', x)) = \epsilon$ mentioned above. Because of the resets done on G_u by $\mathcal{A}$, an in-bridge (q, x_i) of x_i is viewed to add to the indegree of x_i brought by the arcs $(v, x_i) \in E_{G_u}$. Therefore, we add 1 to the first factor of N in Definition 24.(d) when at there is at least one $(v, x_i) \in E_{G_u}$ in the RBS G_u and $l = 1$.

In Figure 3.6, $In(\boldsymbol{y1}, l = 1) = \emptyset$, $\beta(\boldsymbol{y1}) = 0$, $Sum_\Sigma(\boldsymbol{y1}) = 0$, and $RBS(\boldsymbol{y1}, l = 1) = 0$.

On another hand, when $l = 2$, the arcs and vertices considered are in-bridges and out-bridges and every vertex $x_i \in V_{G_u}$ for some RBS G_u in R where x_i at least one ancestor $q \in V_{G_u}$ having at least one in-bridge and/or $q = x_i$ itself has at least one in-bridge. Since all the values of $T((a, b))$ for all $(a, b) \in E \cap E_{G_u}$ are reset to $\mathbf{0}$ for each time an out-bridge in G_u is traversed, (a, b) can be repeatedly checked and/or traversed whenever Definition 24 allow traversals of in-bridges and ancestors of a and b in G_u. Therefore, the values of L of every $(v, q) \in E$ for $v \in V$ where (v, q) is an in-bridge of q also influences the number of occurrences of x_i in p the same way as v with respect to q.

In Figure 3.6, $In(\boldsymbol{y1}, l = 2) = \{(\boldsymbol{x7}, \boldsymbol{y1}), (\boldsymbol{x8}, \boldsymbol{y1})\}$, $\beta(\boldsymbol{y1}) = 0$, $Sum_\Sigma(\boldsymbol{y1}) = 2$ where $Sum_\Sigma(\boldsymbol{y1}) = L((\boldsymbol{x7}, \boldsymbol{y1})) + L((\boldsymbol{x8}, \boldsymbol{y1}))$ because every value of $C(.)$ of all the arcs in $In(\boldsymbol{y1})$ are equal with respect to to each other, and $RBS(\boldsymbol{y1}, l = 2) = 0$.

Finally, when all four addends – $In(x_i, l = 1)$, $\beta(x_i)$, $Sum_\Sigma(x_i)$, and $RBS(x_i, l = 1)$ sum up to 0, the first factor of N is therefore 1. These values are derived when every parent q of x_i establishes the in-bridge $(q, x_i) \in E$ or x_i is a source in R. Similarly, the sum of these addends are 0 when $l = 2$ (and therefore the second factor of N is equal to 1) implies that $x_i \notin V \cap V_{G_u}$ or x_i is a source. The second factor of N reflects the multiplier effect of the repeatable reachability of x_i and the resets performed on $(w, x_i) \in E$, when these resets are applicable, using the information of L and the presence of RBS in R where x_i can be part of.

Shown in Figure 3.6 is an instance of an RDLT model of the reactor and valve systems of an adsorption chiller as described in [68]. More specifically, this shows a detailed specification of the reactors, their modes of operation, i.e. desorption/adsorption and idle(refrigerant and heat recovery), and the valve subsystem of the chiller. This subsystem is a reset-bound subsystem whose center is $\boldsymbol{y1}$, i.e. $M(\boldsymbol{y1}) = 1$. Furthermore, it shows one instance of labelling and settings of L and C. These labels and settings can illustrate one cycle of operation of the chiller where the refrigerant is coursed through the reactor chamber where it is adsorbed/desorbed and then passed to the condenser. We shall discuss the details of these operations of the reactor and valve systems of this adsorption chiller in Section 4.1. We shall also provide the model of the entire adsorption chiller in the same section.

Note that the labelling $l : c$ on an every arc (x, y) sets $L((x, y)) = l$ and $C((x, y)) = c$ where $c \in \Sigma \cup \{\epsilon\}$, respectively. Furthermore, the label $l :$ signifies that $c = \epsilon$.

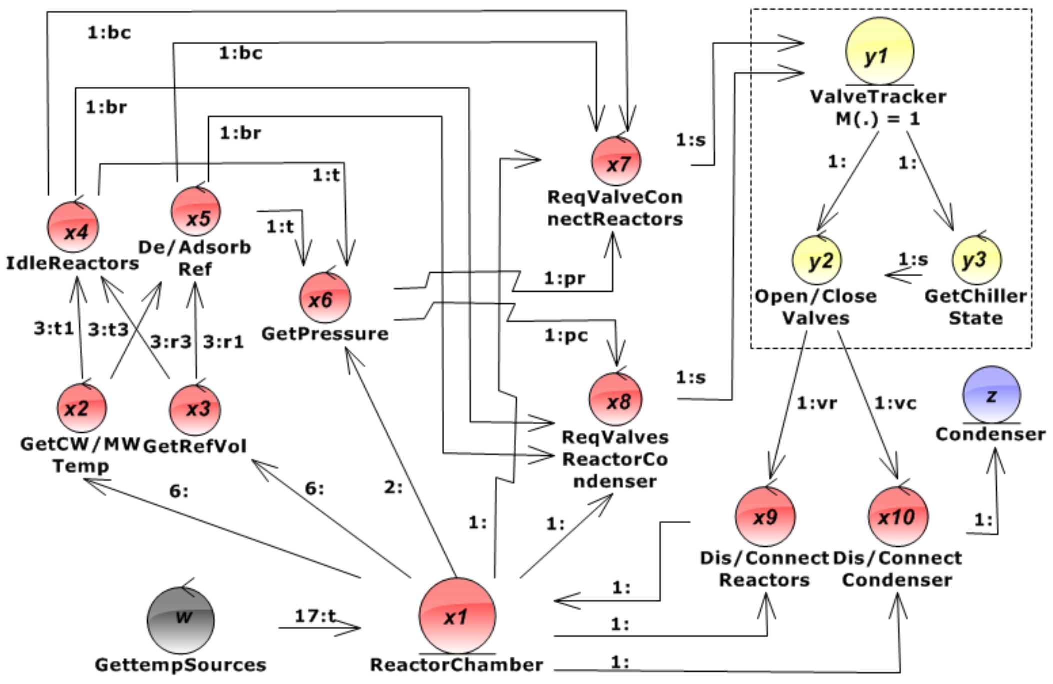


Figure 3.6: RDLT model of the reactors and valve subsystem of an adsorption chiller [68].

In Figure 3.6, the maximum number of occurrences of $\boldsymbol{y1}$ for any path from w to z in R is $N = 2$, i.e. $N = (\max\{1, \{\{0\} - 0 + \{0\}\} + 0\}) * (\max\{1, \{\{0\} - 0 + \{1 + 1\}\} + 0\}) = (1) * (2) = 2$. Meanwhile, $\boldsymbol{x8}$ can occur in at most $N = (\max\{1, \{\{1\} - 1 + \min\{1, 1, 1\}\} + 0\}) * (\max\{1, \{\{0\} - 0 + \{0\}\} + 0\}) = (1) * (1) = 1$ because $C((\boldsymbol{x4}, \boldsymbol{x8})) \neq C((\boldsymbol{x6}, \boldsymbol{x8}))$ with $C((\boldsymbol{x4}, \boldsymbol{x8})), C((\boldsymbol{x6}, \boldsymbol{x8})) \in \Sigma$.

Definition 25 (Diameter between Two Vertices in R) *Let $R = (V, E, T, M)$ be an RDLT. The* ***diameter between two vertices*** *of $x, y \in V$, denoted as $diam(x, y)$, is the largest length of a path, if such path exists, from x to y in R. If no such path exists, then $diam(x, y) = \infty$.*

Definition 26 (Diameter of RDLT) *The **diameter of an RDLT** $R = (V, E, T, M)$, denoted as $diam(R)$, is the maximum diameter derivable from all pairs $\{x, y\} \in V$ of R.*

In the RDLT R in Figure 3.6 of the reactors and valve system of the adsorption chiller in Figure 3.6, its diameter is computable between w and z, i.e. $diam(w, z) = diam(R)$. Furthermore, $diam(\boldsymbol{w}, \boldsymbol{x8}) = 14$ by the path $p =$ ***wx1 x2 x4 x6 x7 y1 y3 y2 x9 x1 x3 x5 x6 x8***. Note that ***x6*** can appear in any path in R at most two times based in Definition 24.

Remark 4 *The identifier/abbreviations of the name of the task/vertex in the adsorption chiller model is fully described in Tables 4.1-4.3 in Chapter 4.*

Definition 27 (Elementary Path) *A path $p = x_1x_2\ldots x_n$ is **elementary** if for every pair $x_i, x_j \in Lit(p)$ where $i \neq j$, $x_i \neq x_j$.*

Definition 28 (Connected RDLT) *An RDLT is **connected** if for every $x \in V$ and a source $w \in V$ of R, $x \neq w$, $diam(w, x) < \infty$.*

Points Of Interests in RDLTs

In this section, the concept and types of a **Point-of-Interest(POI)** are formally defined. As discussed in Section 1, **POIs** are vertices in RDLT models wherein possibilities of deadlocks, delays in reachability, task repetitions and synchronizations are present. The different types of POIs are *Point-Of-Reentry(POR)*, *Point-of-Synching(POS)*, and *Point-of-Delays(POD)*. From each of these types, different types of relationships can be established from them and their neighborhood. These relationships are described and further investigated in the succeeding discussions, as follows,

Definition 29 (Antecedent set) *An **antecedent set** of $x \in V$ in an RDLT is a set $\alpha_x = \{v \in V | v \in Lit(p)$ for every elementary path p from a source $w \in V$ to $x \in V\}$. The **maximal antecedent set** of x, denoted as $\overline{\alpha}_x$, is an antecedent set of x where there is no antecedent set α_x such that $\overline{\alpha}_x \subset \alpha_x$.*

Definition 30 (Consequent set) *A **consequent set** of $x \in V$ in an RDLT is a set $\Omega_x = \{v \in V | v \in Lit(p)$ for every path p from x to $u \in V \cap \overline{\alpha}_x$ where $(u, x) \in E\} \backslash \{\overline{\alpha_x}\}$. The **maximal consequent set** of x, denoted as $\overline{\Omega}_x$, is a consequent set of x where there is no consequent set Ω_x such that $\overline{\Omega}_x \subset \Omega_x$.*

Definition 31 (Looping arc in R) *An arc $(a, b) \in E$ is a **looping arc** used by $x \in V$ where $a, b \in V$ if any of the following holds,*

- $b \in \overline{\Omega}_x$ *and* $a \in \overline{\alpha}_x$
- $(a, b) = (x, u)$ *where* $(x, u) \in E$, $u \in \overline{\alpha}_x$, *and* $\overline{\Omega}_x = \emptyset$

Definition 32 (Point-of-Reentry) *A **Point-of-Reentry(POR)** $b \in V$ for $x \in V$ is the child vertex in the looping arc $(a, b) \in E$ used by x.*

Definition 33 (Point-of-Synching) *A **Point-of-Synching(POS)** $v \in V \cap \overline{\alpha}_x$ for some $x \in V$ where v is either a source or a POR for x in R.*

In Figure 3.6, the maximal antecedent set of ***x8*** is $\overline{\alpha}_{\boldsymbol{x8}} = \{\boldsymbol{w}, \boldsymbol{x1}, \boldsymbol{x2}, \boldsymbol{x3}, \boldsymbol{x4}, \boldsymbol{x5}, \boldsymbol{x6}, \boldsymbol{x8}\}$. Note that no elementary path can be derived from the source ***w*** to ***x8*** that uses ***x9***(***x7***). Therefore, $\{\boldsymbol{x7}, \boldsymbol{x9}\} \notin \overline{\alpha}_{\boldsymbol{x8}}$. Meanwhile, the maximal consequent set of ***x8*** is $\overline{\Omega}_{\boldsymbol{x8}} = \{\boldsymbol{y1}, \boldsymbol{y2}, \boldsymbol{y3}, \boldsymbol{x9}\}$. The looping arc used by ***x8*** is $(\boldsymbol{x9}, \boldsymbol{x1})$. Therefore the POR for ***x8*** is ***x1*** and its POS are ***w*** and ***x1***.

Notice that the looping arc can be used to discriminate the vertices of R that can be used to reach ***x8*** from the source ***w*** against those that may not be necessary for ***x8***'s reachability. However, the presence of looping arcs can affect the reachability of ***x8*** (or any other vertex in R) if the value of their arc attribute $C(.)$ are not set properly, i.e. an arc $(v, \boldsymbol{x1}) \in E$ of the POR ***x1***, where $v \in \overline{\alpha}_{\boldsymbol{x8}}$ and $(\boldsymbol{x9}, \boldsymbol{x1})$ and $(v, \boldsymbol{x1})$ are type-alike, never becomes unconstrained when $\mathcal{A}$ starts its checks and traversal operations from ***w*** to ***x8***. When this happens, ***x1*** and every descendant of ***x1***(including ***x8***) are never reachable from ***w***. With this, we propose in Section 3.2 the non-self controlling property for RDLTs which user-designers can use in building RDLTs and/or verifying them to avoid such problems in reachability. Furthermore, we also propose a measure of reachability of vertices accounting connectivities as well as the graph attributes $C(.)$, $L(.)$ and $M(.)$.

Non-self controlling Structures and Bounds for Reachability

Although paths and path lengths can be computable from R, it still important to note that these do not by themselves determine the time vertices are reached from some ancestors. In addition to these, the values of the arc attribute C of parents also determine this time. This section shall discuss how this attribute influences reachability of vertices and substructures in R. Additionally, these values also help in identifying vertices which can cause delays in reachability of their descendants in R. Since C is a static information in the design of R, user-designers can already check whether the values of C induce non-reachability by verifying if a substructure in R(and therefore R as well) is non-self controlling or not. This property is presented below and measures of delays in reachability for models which satisfy this property are given.

Definition 34 (Point-of-Delay) *A vertex $x \in V$ is called a **Point-of-Delay(POD)** if $\exists u, w \in V$ where $(u, x), (w, x) \in E$ are type-alike and $C((u, x)) \neq C((w, x))$.*

A POD $x \in V$ of R is the main cause that the algorithm $\mathcal{A}$ suspends traversals from x to y where $(x, y) \in E$. In essence, they act as points of bottleneck where the next set of tasks consequent of the task x. In terms of structure with respect to R, they are a consequence of parallel process flows that merge at x. However, note that the delay is imposed by merges that are induced by type-alike arcs (w, x) and (v, x) in E of R. For some designs of RDLTs, when a mix of two (disjoint) sets of type-alike arcs are encountered at x, the traversal and delays(if such exist) would depend on the

diameter of the most-recently checked arc and all its type-alike arcs. For example, if we had set $C((\textbf{u4, x1})) \neq C((\textbf{u5, x1}))$ and $C((\textbf{u4, x1}))$, $C((\textbf{u5, x1})) \in \Sigma$ of the sample RDLT model in Figure 3.1, these sets are $\{(\textbf{u4, x1}), (\textbf{u5, x1})\}$ and $\{(\textbf{u5, x1})\}$. In this setting, **x1** becomes a POD with respect to the first set but not for the second. The delay in reaching **x1** would depend on the diameters of **u4** and **u5** with respect to **u1**. Furthermore, by this setting, the algorithm $\mathcal{A}$ would have to reach the vertices **u1**, **u2**, **u3**, **u4**, and **u5** before **x1** is reached. The other PODs in this sample RDLT are **u4**, **u5**, and **x6**.
To characterize the influence of ancestors and descendants of every POD in RDLTs with respect to its reachability, Definitions 35 - 36 are first introduced.

Definition 35 (Antecedent set with Internal Loops) *An **antecedent set with internal loops**(ASIL) of $q \in V$ is a set $\Gamma_q = \{v \in V | v \in Lit(p)$ for every path p from a source $w \in V$ to $q \in V\} \backslash \overline{\Omega}_q$. The **maximal ASIL** of q, denoted as $\overline{\Gamma}_q$, is an ASIL of q where there is no ASIL Γ_q such that $\overline{\Gamma}_q \subset \Gamma_q$.*

Definition 36 ***($POS_{\overline{\Gamma}_q}$-path)*** *A $POS_{\overline{\Gamma}_q}$-path p is a path from $w \in V$ to q where w is a POS for q and $Lit(p) \subseteq \overline{\Gamma}_q$.*

Definition 37 ***($POS_{\overline{\Gamma}_q}$-diameter)*** *The $POS_{\overline{\Gamma}_q}$-diameter from a POS $w \in V$ to $q \in V$, denoted as $diam_{POS;\overline{\Gamma}}(w, q)$, is the largest length of all $POS_{\overline{\Gamma}_q}$-paths from w to q.*

Definition 38 (Non-self controlling) *An RDLT R is **Non-Self Controlling(NSC)** if R is connected and for every POD $x \in V$ and every looping arc $(a, b) \in E$ used by x where $C((a, b)) \in \Sigma$, $a \in \overline{\Omega}_x$, $b \in \overline{\alpha}_x$, there exists an arc $(q, b) \in E$ such that $C((a, b)) = C((q, b))$, (a, b) and (q, b) are type-alike, and $q \in \overline{\alpha}_x$.*

Theorem 1 *For an RDLT R that is not NSC, there exists a POD $x \in V$ that is not reachable from a source $w \in V$ of R.*

Proof 1 *Suppose every POD $x \in V$ is reachable in R where R is not NSC. By Definition 23, there exists an activity profile $S = \{S(1), S(2), \ldots, S(d)\}$, $d \in \mathbb{N}$ where $(w, x) \in S(1)$ and the following holds,*

(i) $\exists t < d$, $(u, x) \in S(t)$ *where* $u \in \overline{\Gamma}_x$,

(ii) $\exists t' < t$, $(b, y) \in S(t')$ *where* $b, y \in \overline{\Gamma}_x$ *and y is a POS for x, and*

(iii) $\exists d > t'' > t$, $(q, y) \in S(t'')$ *where* $q \in \overline{\Omega}_x$ *and* $C((q, y)) \in \Sigma$. *Moreover, since R is not NSC, then* $C((q, y)) \neq C((b, y))$, *i.e. y is a POD of R where* $C((b, y)) \in \Sigma$.

By Definition 21, $(b, y), (q, y) \in S(t''')$ for every $t''' < d$ for S. That is, both arcs are traversed simultaneously at every time step t'''. Therefore, we arrive at a contradiction. ■

Lemma 2 *If every vertex $v \in V$ of R is reachable from a source $w \in V$ then R is NSC.*

Proof 2 *By Definition 23, there exists an activity profile* $S = \{S(1), S(2), \ldots, S(d)\}$, $d \in \mathbb{N}$, *where* $(u, v) \in S(t)$, $1 \leq t \leq d$, $u \in V$, *for every vertex* $v \in V$ *of* R *that is reachable from a source* $w \in V$, *and* $(w, x) \in S(1)$, $x \in V$. *Since* $d \in O(\max\{L(.)\} * |E|)$, *therefore* $diam(w, v) \leq d \leq diam(R)$. *This proves that* R *is a connected RDLT. Since* $(u, v) \in S(t)$ *in at least one activity profile* S *where* $S(t) \in S$, (u, v) *is an unconstrained arc with respect to any of its type-alike vertex* $(q, v) \in E$. *By Definition 21, any* (u, v) *where* v *is a POD can only be traversed at time step* t *in the reachability configuration* $S(t)$ *if any such* (q, v) *has either an equivalent value of its arc attribute* $C(.) \in \Sigma$ *with respect to* (u, v)*'s or* (q, v) *has been checked at a previous/same time step. For any parent* $u' \in V$ *where* $(u', v) \in E$, $u' \notin \overline{\alpha}_v$ *and* $(u, v) \in S(t)$ *with* $C((u, v)) \in \Sigma$, (u, v) *and* (u', v) *are type-alike,* (u', v) *is traversed at a later time* t', $t < t' < d \leq diam(R)$, *since* $u' \in \overline{\Omega}_u$ *and* $u' \in \overline{\Omega}_v$, *and* $u' \notin \overline{\alpha}_u$ *and* $u' \notin \overline{\alpha}_v$. *That is,* u' *is reached after* u *and* v *had been reached, in that order, beforehand. This traversal also implies that* $C((u', v))$ *allows* (u, v) *to be unconstrained at* t. *Furthermore, this shows that* (u', v) *is a looping arc used by* u *and* v *in* R. *It is notable that looping arcs are differentiable from other arcs which are involved in structures where iterations happen where such* u *and* v *exist. This is because of the usage of a reference vertex, i.e. the source* $w \in V$, *and* $\overline{\alpha}_v$ *and* $\overline{\Omega}_v$. *These prove that* R *is NSC if every vertex* $v \in V$ *of* R *is reachable from the source* $w \in V$.
■

In Figure 3.6, the RDLT model of the adsorption chiller contains the PODs ***x1***, ***x4***, ***x5***, ***x6***, ***x7***, ***x8***, ***x9***, ***x10***, and ***y2***. The POD ***x8*** has its parents ***x4***, ***x5***, ***x6*** with the values of their arc attributes $\{C((\boldsymbol{x4}, \boldsymbol{x8})) = C((\boldsymbol{x5}, \boldsymbol{x8}))\} \neq C((\boldsymbol{x6}, \boldsymbol{x8})) \neq C((\boldsymbol{x1}, \boldsymbol{x8}))$. For this example, the maximal antecedent set of ***x8*** is $\overline{\alpha}_{\boldsymbol{x8}} = \{\boldsymbol{w}, \boldsymbol{x1}, \boldsymbol{x2}, \boldsymbol{x3}, \boldsymbol{x4}, \boldsymbol{x5}, \boldsymbol{x6}, \boldsymbol{x8}\}$. Meanwhile, the maximal consequent set of ***x8*** is $\overline{\Omega}_{\boldsymbol{x8}} = \{\boldsymbol{y1}, \boldsymbol{y2}, \boldsymbol{y3}, \boldsymbol{x9}\}$. The looping arc used by ***x8*** is $(\boldsymbol{x9}, \boldsymbol{x1})$. This looping arc allows $(\boldsymbol{w}, \boldsymbol{x1})$ to be unconstrained at $t = 1$(by Definition 21). ***x1*** is reachable from ***w*** at $t = 1$ in executing $\mathcal{A}$ on R. Note that there is no looping arc $(a, b) \in E$ in R for any pair of its vertices where $C((a, b)) \in \Sigma$ that causes $(q, b) \in E$ to be not unconstrained. In this case, the RDLT in Figure 3.6 is an non-self controlling R.

Remark 5 *The time and space complexity to check if* R *is NSC is* $O(|V|^3)$ *and* $O(|V|^2)$, *respectively.*

Upon knowing that any vertex $x \in V$ is reachable from a source $w \in V$ in R, a bound for reachability of x when it is a POD is computed with respect to the time of reachability of all its parents whose arcs towards x are type-alike. This bound describes how long it takes for a POD to be reached by $\mathcal{A}$ from the time at least one of its parents are reached until every such parent y where $C((y, x)) \in \Sigma$ becomes unconstrained. This measures how much delay there is to reach x when traversals of every (y, x) are suspended due to nonsatisfiability of any of the conditions in Definition 21.

Definition 39 (Δ-delayed POD) *A POD $x \in V$ of a NSC R is Δ_x-**delayed** with respect to its parent $q \in V$ and a POS $w \in V$ for x such that $0 \leq \Delta_x \leq diam(R)$ where*

$$\Delta_x = \max_{l \in \{1,2\}} \left\{ \max_{P(l:(q,x)\in E,\Sigma)} \{diam_{POS;\overline{\Gamma}}(w,q)\} - \min_{P(l:(q,x)\in E,\Sigma\cup\{\epsilon\})} \{diam_{POS;\overline{\Gamma}}(w,q)\} \right\} + 1$$

where $P(1:(q,x)\in E, \mathscr{A}) = \{q \in V | (q,x) \in E,\ (q,x) \notin E_{G_u}$ and $C((q,x)) \in \mathscr{A}\}$ and $P(2:(q,x)\in E, \mathscr{A}) = \{q \in V | (q,x) \in E,\ (q,x) \in E_{G_u}$ and $C((q,x)) \in \mathscr{A}\}$ for some arc set E_{G_u} of an RBS G_u with center $u \in V \cap V_{G_u}$ of R, $\mathscr{A} \subseteq \Sigma \cup \{\epsilon\}$,

and if for every $(a,b) \in E$ where $a,b \in \overline{\Gamma}_x$,

$$L((a,b)) \geq L_{reqMinimum}((a,b))$$

where $\forall v \in V$ where v is a POS of some $g \in V \cap \overline{\Gamma}_x$ and there exists a path p from v to x such that $v,a,b,x \in Lit(p)$,

$$L_{reqMinimum}((a,b)) = \begin{cases} |\{POS\ v \in V_{G_u}\}| & \text{if } (a,b) \in E_{G_u}, \\ |\{POS\ v \notin V_{G_r}\}| & \text{if } a,b \notin V_{G_r} \text{ for any RBS } G_r \text{ in } R, r \in V. \end{cases}$$

Definition 39 can be explained as follows,

(1) $\max_{P(l:(q,x)\in E,\Sigma)} \{diam_{POS;\overline{\Gamma}}(w,q)\}$ accounts the longest $POS_{\overline{\Gamma_q}}$ path from a POS w to q where q is a parent of x with $C((q,x)) \in \Sigma$. The parent q (of x) that yields the maximum distance with respect to w, accounting all loops in the antecedent set of x, imposes the time of reachability of x. Accounting the worst-case time of reachability q, whether by topological or constraint-based reachability, this time is also used as an upper bound to reach x. Roughly speaking, this minuend gives the *longest path of the farthest parent of x with respect to w with the constraint $C((q,x))$.*

(2) $\min_{P(l:(q,x)\in E,\Sigma\cup\{\epsilon\})} \{diam_{POS;\overline{\Gamma}}(w,q)\}$ accounts the shortest $POS_{\overline{\Gamma_q}}$ path from a POS w to q where q is a parent of x with $C((q,x)) \in \Sigma$ or $C((q,x)) = \epsilon$. For this, the best-case time of reachability of a parent q (of x) from w are accounted for from using the antecedent set of x. The parent that yields this minimum would mark the time that x can be possibly reached. When a backtrack is imposed by another parent of x due to the nonsatisfiability of traversal constraints, q implies the lower bound of the waiting time that these constraints will be satisfied. Even if $C((q,x)) = \epsilon$, if the traversals are disallowed, this waiting time is still imposed. Therefore, this lower bound accounts parent-child relationship accounting both Σ and ϵ values of C. Roughly speaking, this subtrahend gives the *longest path of the nearest parent of x with respect to w.*

(3) The delay δ_x of a POD x is therefore computable from the parent/s that yield/s the values in (1) and (2). A parent can yield both the minimum and maximum values in (1) and (2). Furthermore, note that there is a requirement of R to be NSC for Definition 39. Therefore, the reachability of the concerned parent/s and x are assured.

(4) $\max\limits_{l\in\{1,2\}}\{.\}$ imposes that two parents q and q' of x which are being considered in the computation of the maximum and minimum values in (1) and (2) have arcs (q,x) and (q',x) are type-alike. By Definition 21, traversals(and therefore reachability of x) is also imposed by this pair-wise characteristic of incoming arcs of x. Therefore, values for maximum and minimum values are separately computed for type-alike arcs. When $l = 1$, the arcs that are considered do not belong to a reset-bound subsystem. Otherwise, q, q' and x belong to a vertex set of a reset-bound subsystem in R.

(5) $L_{reqMinimum}((a,b))$ accounts the number of POS w along $a,b,x \in \overline{\Gamma}_x$. Note that there is a consideration whether a,b,x belong to the same reset-bound subsystem or not. If they do, their reachability solely depends on the number of POS inside the RBS itself. This number sets the value of $L_{reqMinimum}((a,b))$. That is, the minimum number of traversals of (a,b) is $L_{reqMinimum}((a,b))$ so x can still be reached each time traversals go through every POS w in the RBS. This implies that $L((a,b))$ for every $a,b \in \overline{\Gamma}_x$ is imposed by its inclusion to a "local" neighborhood. That is, whether a,b,x belong to the same RBS or not. If they do not, it is also equivalently implied that $L((a,b))$ is lower bounded by the number of POS (of $\{g,x\}$) that are in $\overline{\Gamma}_x$ to guarantee the reachability of x with respect to to these POS. For this, the inclusion/exclusion of such POS in an RBS is irrelevant in adding it to the count in $L_{reqMinimum}((a,b))$.

Note that this $L_{reqMinimum}((a,b))$ only considers the existence of at least one path from every POS w to every parent of x and measures delays of reachability δ_x of x from them. Therefore, concerns about repeatability of executions tasks corresponding to the vertices $\overline{\Gamma}_x$ along paths from w to x is irrelevant for this measure. (Nonetheless, this research also proposes a property known as L-verifiability(see Section 8) which tackles such concerns in checking these aspects of R.)

In Figure 3.6, ***x8*** is 4-delayed, i.e. $\Delta_{\boldsymbol{x8}} = diam_{\overline{\Gamma}}(\boldsymbol{w}, \{\boldsymbol{x6}{:}\boldsymbol{x8}\})$ - $diam_{\overline{\Gamma}}(\boldsymbol{w}, \{\boldsymbol{x1}{:}\boldsymbol{x8}\}) = (4-1)+1 = 4$. The value of the subtrahend in computing $\Delta_{\boldsymbol{x8}}$ is the time ***x1*** is reached where ***x1*** is a parent of ***x8*** which is first parent to be reached from ***w***. Meanwhile, the latest to be reached is ***x6***. The value $\Delta_{\boldsymbol{x8}} = 4$ means that ***x8*** is reached 4 time steps after ***x1*** was reached. The last constraint $C(\boldsymbol{x6}, \boldsymbol{x8}) =$***pc*** was satisfied by then. The satisfaction of this constraint implies that the pressure difference between the reactor-desorber and the condenser is sufficient so the succeeding tasks done by the system is to open relevant valves to connect them and pass the vaporized refrigerant to the latter. By that time, the other constraint $C(\boldsymbol{x4}, \boldsymbol{x8}) = C(\boldsymbol{x5}, \boldsymbol{x8}) =$ ***br*** had already been satisfied 2 steps before. That constraint relates to the temperature value at the reactor-desorber bed that vaporizes the refrigerant during desorption stage. The temperature causes the change of the pressure at the bed, hence, opening the required valves in the succeeding step.

For ***x8*** in Figure 3.6, all arcs (a, b) that connect two vertices in $\overline{\Gamma}_{\boldsymbol{x8}} = \{$***w***, ***x1***, ***x2***, ***x3***, ***x4***, ***x5***, ***x6***, ***x8***$\}$ have $L((a, b)) \leq 2$. This is because there are two POS in this set, i.e. ***w*** and ***x1***, and a, b, and ***x8*** do not belong to the same RBS.

Definition 40 (Bounded RDLT) *An RDLT R is* ***bounded*** *if R is connected and every POD x of R, if it exists, is Δ_x-delayed where $0 \leq \Delta_x \leq diam(R)$.*

Theorem 2 *Every $x \in V$ of in an RDLT R is reachable iff R is bounded.*

Proof 3 ($\Longrightarrow$) *We first prove that every $x \in V$ in R is reachable if R is bounded. If R is bounded, then R is NSC(by Definition 39). By Definition 40, every arc $(a, b) \in E$ that is along a path p from any POS $w \in V$ of some $g \in \overline{\Gamma}_x$ where $w, a, b, x \in Lit(p)$ is always traversable by $\mathcal{A}$ since R is NSC and $L((a, b))$ allows it. That is, every time a POS w is reached by $\mathcal{A}$ and $\mathcal{A}$ carries on traversals through (a, b), the lower bound $L_{minRequired}((a, b))$ of $L((a, b))$ guarantees a correspondence of w being used/reached and (a, b) being traversed (again) by $\mathcal{A}$ for an activity profile in R. Since R is bounded, $diam_{POS;\overline{\Gamma}}(\overline{w}, f)$ for a sink $f \in V$ is computable where $diam_{POS;\overline{\Gamma}}(\overline{w}, f) \leq diam(R)$, f is a descendant of x, $\overline{w} \in V$ is an ancestor of x and $\overline{w}$ is a source in R. Therefore, there exists an activity profile $S = \{S(1), S(2), \ldots, S(diam_{POS;\overline{\Gamma}}(\overline{w}, f))\}$ where the following holds,*

1. *$\exists(\overline{w}, v) \in S(1)$, where v is an ancestor of x,*
2. *$(u, x) \in S(t + t_x)$ for some $t \in \mathbb{N}$, $t_x = diam_{POS;\overline{\Gamma}}(w, x)$, w is a POS for some $g \in \overline{\Gamma}_x$, and $1 \leq t + t_x \leq diam_{POS;\overline{\Gamma}}(\overline{w}, f)$,*
3. *$\exists V_0 \subseteq \overline{\Gamma}_x$ where for some $a, b \in V_0$, $(a, b) \in S(k)$, $k = 1, 2, \ldots, (t + t_x)$,*
4. *$\exists V_1 \subseteq \overline{\Gamma}_x \cup \overline{\Omega}_x$ where for some $c, d \in V_1$, $(c, d) \in S(k')$, $k' = (t + t_x + 1), (t + t_x + 2), \ldots, diam_{POS;\overline{\Gamma}}(\overline{w}, f)$,*
5. *$\exists(q, f) \in S(diam_{POS;\overline{\Gamma}}(\overline{w}, f))$ for some $q \in V_1$,*
6. *$\exists a, b, c \in V_1$ where $(a, b) \in S(i)$, $(b, c) \in S(i{+}1)$, $i = 1, 2, \ldots, (diam_{POS;\overline{\Gamma}}(\overline{w}, f) - 1)$ (by Definitions 38 and 37).*

This proves that every vertex $x \in V$ in R is reachable if R is bounded.

($\Longleftarrow$) *Next, we prove that R is bounded if every $x \in V$ is reachable. From Lemma 2, we know that R is NSC when every vertex x is reachable from a source $\overline{w} \in V$. Hence, there exists a path from every POS $w \in V$ for some $g \in \overline{\Gamma}_x$ to every parent of $q \in \overline{\Gamma}_x$. From this, $diam_{POS;\overline{\Gamma}(w,q)}$ is computable where $diam_{POS;\overline{\Gamma}(w,q)} < diam(R)$, and therefore, $0 \leq \Delta_x \leq diam(R)$. (Note that every parent $q' \in \overline{\Omega}_x$ of x does not prevent traversal for (q, x) because R is NSC.) Since any sink $f \in V$ is reachable, where f is an ancestor*

of x and w – i.e. $\exists t \leq diam(R)$ where $(q, f) \in S(t)$, therefore, every $a, b \in V \in \overline{\Gamma}_x$ which are descendants of w and ancestors of x and f are reachable. Therefore, (a, b) can be traversed when $\mathcal{A}$ checks it after using w at a previous time step. This shows that $L((a, b)) \geq L_{reqMinimum}((a, b))$. This proves that R is bounded if every vertex $x \in V$ in R is reachable.
■

Remark 6 *Boundedness in RDLTs have direct relations with the conflict-freeness and deadlock-freeness in Petri nets(see Section 2). Like conflict-freeness in Petri nets in Section 2, the NSC property in (bounded) RDLTs makes sure that every vertex $x \in V$ have their corresponding constraints, i.e. $C(u, x)$ where $u \in V$, are set in such a way that at least one path can be used to reach a parent of x that makes (u, x) unconstrained, i.e. x would eventually be used. However, in contrast with conflict-freeness, the NSC property in RDLTs considers the presence of reset-bound subsystems where x might be a vertex therein. Whenever this happens, the type-alikeness of the arcs from every parent of x is considered for the reachability of x. Moreover, because every vertex is reachable in a bounded RDLT, this means that every reachability configuration in an activity profile has at least one element where its non-sink, target vertex is usable to reach another vertex in its succeeding reachability configuration. Therefore, this is operationally equivalent to having Petri nets that are deadlock-free, i.e. typing of arcs by virtue of $M(.)$ in RDLTs is not considered.*

3.3 Free-choice Structures in RDLTs

Definition 41 (Free-choice) *A bounded RDLT R is **free-choice** if for any siblings $x, y \in V$ where x is a POD, any of the following holds,*

1. *Σ**-distinct Parent-Child Structure(PCS):***
   - *$(q, x) \in E$ iff $(q, y) \in E$, $\forall q \in V$, and*
   - *$C((q, v)), C((q', v)) \in \Sigma$ and $C((q, v)) \neq C((q', v))$ where $v \in \{x, y\}$ for any $q, q' \in V$ such that $(q, v), (q', v) \in E$, (q, v) and (q', v) are type-alike, and $x \neq q \neq q' \neq y$. Furthermore, if $(y, x) \in E$, then $C((y, x)) = \epsilon = C((y, y))$. Similarly, if $(x, y) \in E$, then $C((x, y)) = \epsilon = C((x, x))$.*
   - *for every POS $w \in V$ where $w \in \overline{\alpha}_x \cup \overline{\alpha}_y$, there exist a $POS_{\overline{\Gamma}_x}$-path from w to x and a $POS_{\overline{\Gamma}_y}$-path from w to y in R.*
2. *t_0**-step Parent-Child Neighborhood(PCN):***
   $(u, x) \in S(t)$ then $(v, y) \in S(t)$ for some $u, v \in V$, $2 \leq t \leq diam(R)$, and for some t_0 with $1 \leq t - t_0 < diam(R)$ and every $q \in V$ where $\{(q, x), (q, y)\} \in E$, with q, x, and y are distinct from each other, and

$$\exists r \in V \text{ such that } (r, q) \in \bigcup_{j=1}^{t_0} S(t-j) \text{ and } \{(q, x), (q, y)\} \notin \bigcup_{j=1}^{t_0} S(t-j),$$

where t_0 is minimum, $a, q \in V$.

Lemma 3 establishes the relation of Σ-distinct PCS and PCN free-choice RDLTs by identifying the value of t_0 for a Σ-distinct PCS R.

Lemma 3 *A free-choice PCS R is a t_0-step PCN RDLT.*

Proof 4 *We prove that a free-choice PCS R is a t_0-step PCN RDLT by a proof of contradiction, i.e. for some POD $x \in V$ and its sibling $y \in V$ in a free-choice PCS RDLT, $\exists t'$ where $t - t_0 \leq t' < t \leq diam(R)$ such that, without loss of generality, $(q, x) \in S(t')$, $(q, y) \in S(t)$, $q \in V$, $(q, x) \in E$.*

By Definitions 40 and 41, every such pair x and y in a free-choice PCS R has $0 \leq (diam_{POS;\overline{\Gamma}}(w, x) = diam_{POS;\overline{\Gamma}}(w, y)) \leq diam(R)$ where $w \in V$ is a POS vertex in R. Let $w_0, w_1 \in V$ be parents of x and y such that

$$diam_{POS;\overline{\Gamma}}(w, w_0) = \min_{u \in V, (u,v) \in E, v \in \{x,y\}, \{(w_0,v),(u,v)\} \text{ are type-alike}} \{diam_{POS;\overline{\Gamma}}(w, u)\}$$

and

$$diam_{POS;\overline{\Gamma}}(w, w_1) = \max_{u \in V, (u,v) \in E, v \in \{x,y\}, \{(w_0,v),(u,v)\} \text{ are type-alike}} \{diam_{POS;\overline{\Gamma}}(w, u)\}.$$

By Definition 41.(1), every pair of parents $u, u' \in V$ have the values of their arc attributes $C((u, x)) \neq C((u', x))$ $(C((u, y)) \neq C((u', y)))$ and $C((u, x)), C((u', x)) \in \Sigma$ $(C((u, y)), C((u', y)) \in \Sigma)$. By Definition 21, neither $(u, x), (u, y) \in E$ nor $(u', x), (u', y) \in E$ become unconstrained at time $t'' = [diam_{POS;\overline{\Gamma}}(w, w_0), diam_{POS;\overline{\Gamma}}(w, w_1)]$. However, they become unconstrained-traversed by the algorithm $\mathcal{A}$ at $diam_{POS;\overline{\Gamma}}(w, w_1) + 1$, such that $\{(u, x), (u', x), (u, y), (u', y)\} \in S(t)$ for $t = diam_{POS;\overline{\Gamma}}(w, w_1) + 1$ with $t \leq diam(R)$. Hence, x and y are not reached at $t' = t'' \in [(t - t_0 = diam_{POS;\overline{\Gamma}}(w, w_0)), (t - 1 = diam_{POS;\overline{\Gamma}}(w, w_1)]$. We arrive at a contradiction. This proves that every free-choice PCS R is a t_0-step PCN RDLT.

■

For the Σ-distinct PCS case of the free-choice property in RDLTs, the difference in the values of the arc attributes $C(.) \in \Sigma$ of the arcs (u, x) and (u', x) for every pair of parents $u, u' \in V$ of x and y imposes that x and y are only reachable if all of their parents have been reached beforehand, i.e. all constraints $C((q, \{x, y\}))(\in \Sigma)$, $\forall q \in V$, are already satisfiable. With this, since y has exactly the same set of parents as x with the same scheme of the assigned values for $C(.)$, therefore, x and y are reachable at the same time. Furthermore, having $C((y, x)) = C((y, y)) = \epsilon$(or $C((x, y)) = C((x, x)) = \epsilon$) ensures that $x(y)$ is reachable.

Meanwhile, the t_0-step PCN guarantees the simultaneous reachability of the POD x and its sibling y either (1)by the connectivities of the arcs from a POS w to the parents

u and u' of x and y, i.e. equality of the lengths of paths from w to u and u' considering x and y regardless of the values of their corresponding arc attributes, or (2) differing values of $C(u, v)$ and $C(u', v)$ for $v \in \{x, y\}$, or (3)a combination of (1) and (2). Note that the t_0-step PCN requires only one common parent for x and y. With this, the reachability of y does not require the (simultaneous) reachability of x when y is not a POD itself.

In Figure 3.6, the subgraph induced by the vertex set $\{\boldsymbol{w}, \boldsymbol{x1}, \boldsymbol{x2}, \boldsymbol{x3}, \boldsymbol{x4}, \boldsymbol{x5}\}$ is a free-choice Σ-PCS RDLT. Meanwhile, the subgraph which is induced by the vertex set $\{\boldsymbol{w}, \boldsymbol{x1}, \boldsymbol{x2}, \boldsymbol{x3}, \boldsymbol{x4}, \boldsymbol{x5}, \boldsymbol{x6}, \boldsymbol{x7}, \boldsymbol{x8}, \boldsymbol{y1}\}$ and whose value $C((\boldsymbol{x8}, \boldsymbol{y1}))$ is changed to $C((\boldsymbol{x8}, \boldsymbol{y1})) = \boldsymbol{r}$, $\Sigma := \Sigma \cup \{r\}$ is a free-choice t_0-step RDLT for the siblings $\boldsymbol{x7}$ and $\boldsymbol{x8}$. These siblings are both PODs in R. Note that retaining the original value $C((\boldsymbol{x8}, \boldsymbol{y1})) = \boldsymbol{s}$ and implementing R in a non-parallelized manner will extract two activities from this subgraph where one activity S_1 contains $(\boldsymbol{x7}, \boldsymbol{y1}) \in S(6)$ and another activity S_2 contains $(\boldsymbol{x8}, \boldsymbol{y1}) \in S(6)$. This is because the arcs contained in both terminal reachability configurations of S_1 and S_2 are unconstrained when the corresponding parent vertices of $\boldsymbol{y1}$ are checked by $\mathcal{A}$. Therefore, $\mathcal{A}$ does not backtrack to involve both siblings in one activity profile for $\boldsymbol{w}$ and $\boldsymbol{y1}$. However, when this change is done on $C((\boldsymbol{x8}, \boldsymbol{y1}))$, there is only one activity profile S where S contains the reachability configuration $(\boldsymbol{x6}, \boldsymbol{x7})$, $(\boldsymbol{x6}, \boldsymbol{x8}) \in S(5)$. With this, the value $t_0 = 4$ corresponds to the earliest time of reachability of one of the siblings' parents, i.e. $\boldsymbol{x1}$ at $t - t_0 =$ 5 - 4 = 1. Furthermore, $t - 1 = 4$ is the latest time of reachability of one of the siblings' parents, i.e. $\boldsymbol{x6}$. Here, $t = 5$, i.e. when the siblings were reached by $\mathcal{A}$. Note that both siblings are not reachable when their parents $\boldsymbol{x4}$ and $\boldsymbol{x5}$ were reached at $t - 2$ because of the differing values of the attribute $C(.)$ of $(\boldsymbol{x6}, \boldsymbol{x7})$ and $(\boldsymbol{x6}, \boldsymbol{x8})$ with respect to $\{(\boldsymbol{x4}, \boldsymbol{x7}), (\boldsymbol{x5}, \boldsymbol{x7})\}$ and $\{(\boldsymbol{x4}, \boldsymbol{x8}), (\boldsymbol{x5}, \boldsymbol{x8})\}$, respectively. Therefore, none of the latter arcs are unconstrained (Definition 21). Therefore from $t - t_0$ to $t - 1$, neither of the siblings are reached by $\mathcal{A}$.

Remark 7 *The free-choice property of Petri nets(see Section 2) is equivalent to free-choice Σ-distinct PCS RDLTs without reset-bound subsystems having their vertex set containing two siblings where free choice is required with respect to their parents. More specifically, the requirement of having a uniform set of input places for two transitions t_1 and t_2 that share at least one parent, i.e. input place, in Petri nets(by Definition 16) is conceptually equivalent to requiring two siblings $x, y \in V$ in such RDLTs to have the same set of parents and having differing values of $C(.)$ for all arcs that connect a parent u to $x(y)$ where $u \neq y(u \neq x)$. Furthermore, any POS $w \in V$ in the third condition of a free-choice Σ-distinct PCS RDLT(see Definition 41.(1)) is equivalent to having any free-choice Petri net with t_1 and t_2 always enabled when either of these transitions have a parent(input place) containing a token regardless of loops, i.e. task (re)executions, in the Petri net. With all these, it can be seen that involving any of the transitions' input places inside a cancellation region(see Section 1) in existing workflows to simulate a reset-bound subsystem in RDLT can lead to invalidating the free-choiceness in the former models. This happens when a cancel task is executed in them and these input places are affected in the cancellation and lose their tokens.*

Meanwhile, the concept of free-choice property in Petri nets is further extended in RDLTs with respect to having t_0-step Parent-Child Neighborhood(PCN) structures in

RDLTs in Definition 41.(2). For this definition, connectivity with consideration of reset-bound subsystems mainly drives the satisfiability of free-choiceness with or without imposing the conditions in Definition 41.(1).

Definition 42 (Synched siblings) *Given two siblings $x, y \in V$ of a bounded RDLT R, where x is a POD, x and y are* ***synched*** *if*

$$\delta_\Sigma(P_x) = \delta_\Sigma(P_y) = \delta_\epsilon(P'_x) = \delta_\epsilon(P'_y)$$

where

$$\delta_\Sigma(P_v) = \begin{cases} \max\limits_{q \in P_v}\{diam_{POS;\overline{\Gamma}}(w,q)\}, & \textit{if } |P_v| \geq 1, \\ \delta_\epsilon(P_v) & \textit{otherwise,} \end{cases}$$

and

$$\delta_\epsilon(P'_v) = \begin{cases} \bigcap\limits_{q \in P'_v}\{diam_{POS;\overline{\Gamma}}(w,q)\} & \textit{if } |P'_v| \geq 1 \\ \delta_\Sigma(P_v) & \textit{otherwise,} \end{cases}$$

where $P_v = \{q \in V | (q,v) \in E$ and $C((q,v)) \in \Sigma\}$, $P'_v = \{q \in V | (q,v) \in E$ and $C((q,v)) = \epsilon\}$, and $w \in V$ is a POS of some $g \in \overline{\Gamma}_x \cup \overline{\Gamma}_y$ in R where there are paths from w to x and from w to y, and every pair of arcs $(a,b), (a',b') \in P_v \cup P'_v$ are type-alike.

Remark 8 *Whenever $|P'_v| \geq 1$, note that $\delta_\epsilon(P'_v)$ is taken as the intersection of all $diam_{POS;\overline{\Gamma}}(w,q)$ from every parent $q \in P'_v$ for $v \in V$ so that $\delta_\epsilon(P'_v)$ is positive if and only if every path which gives $diam_{POS;\overline{\Gamma}}(w,q)$ from any POS w (in Definition 42) leading to every such q ensures that each of these parents are reached simultaneously with respect to each other.*

Definition 43 *A bounded RDLT R is* ***diametrically-synched*** *if $\forall x, y \in V$, where x and y are siblings and at least one is a POD, x and y are synched.*

Theorem 3 *An RDLT R is diametrically-synched if R is a free-choice Σ-distinct PCS where for every POD x and its sibling y and every $w \in V$ that is a POS of some $g \in \overline{\Gamma}_x \cup \overline{\Gamma}_y$ in R, there are paths from w to x and from w to y and every $(a,b) \in E$ along these paths have $L((a,b)) \geq L_{reqMinimum}((a,b))$.*

Proof 5 *We first prove that every POD x and its sibling y are synched from any such POS $w \in V$ for $g \in \overline{\Gamma}_x \cup \overline{\Gamma}_y$. By Definition 41, any pair $q, q' \in V$ where $(q,x), (q',y) \in E$, $diam_{POS;\overline{\Gamma}}(w,q), diam_{POS;\overline{\Gamma}}(w,q') \leq diam(R)$, $v \in \{x,y\}$, $w \in V$ is a POS in R such that there is a $POS_{\overline{\Gamma}_x}$-path from w to x and a $POS_{\overline{\Gamma}_y}$-path w to y. Since R is free-choice Σ-distinct PCS, $P_x = P_y$, which results to $\delta_\Sigma(P_x) = \delta_\Sigma(P_y) = \delta_\epsilon(P'_x) = \delta_\epsilon(P'_y)$. Note that $|P'_x| = |P'_y| = 0$. Therefore, any such $\{x,y\}$ of R are synched from any any POS $w \in V$ proving that R diametrically-synched.*

Theorem 4 *An RDLT R is diametrically-synched iff R is a free-choice t_0-step PCN, $t_0 < diam(R)$, where for every POD x and its sibling y and every $w \in V$ that is a POS of some $g \in \overline{\Gamma}_x \cup \overline{\Gamma}_y$ in R, there are paths from w to x and from w to y and every $(a, b) \in E$ along these paths have $L((a, b)) \geq L_{reqMinimum}((a, b))$.*

Proof 6 *(Proof: $\Rightarrow$) We first prove that a free-choice t_0-step PCN R is diametrically-synched where $1 \leq t_0 < diam(R)$ from any such POS $w \in V$.*

For a free-choice t_0-step PCN R where every POD $x \in V$ and its sibling $y \in V$ have $diam_{POS;\overline{\Gamma}}(w, x) = diam_{POS;\overline{\Gamma}}(w, y) = t$ for some $t \leq diam(R)$, w is a POS with a $POS_{\overline{\Gamma}_x}$-path from w to x and a $POS_{\overline{\Gamma}_y}$-path w to y, $1 \leq t_0 < diam(R)$, and every $(a, b) \in E$ along these paths have $L((a, b)) \geq L_{reqMinimum}((a, b))$, the following structural relations hold,

(a) for $B = \{x, y\}$ for every $q, q' \in V$ such that $(q, c) \in E$ and $(q', c) \in E$ are type-alike and $C((q, c)), C((q', c)) \in \Sigma$ where $c \in B$, there exists paths $p = u_1u_2 \dots u_n$ and $p' = v_1v_2 \dots v_m$ where $u_i, u_j \in V$, $q = u_{n-1}$, $q' = v_{m-1}$, $u_1 = v_1 = w$, and

$$[|u_1u_2 \dots u_{n-1}| = t - 1 \ = diam_{POS;\overline{\Gamma}}(u_1, c) - 1] < diam(R), \text{ and}$$
$$[|v_1v_2 \dots v_{m-1}| = t - t_0 \ = diam_{POS;\overline{\Gamma}}(v_1, c) - t_0] < diam(R), \text{ and}$$

for every $q'' \in V$ where $(q'', c) \in E$ and $C((q'', c)) \in \Sigma$, there is a path $p'' = q_1q_2 \dots q_k$ where $q_1 = w$, $q'' = q_{k-1}$ and $q_k = c$ and $t - t_0 \leq diam(q_1, q_{k-1}) \leq t - 1$.

The first two relations above make sure the existence of parent $q' = v_{m-1}(q = u_{n-1})$ of x and/or y which determines the earliest(last) time step $t - t_0(t - 1)$ when it is reached as compared to every other parent q'' of x and/or y. The last condition makes sure that for all other remaining parents $q'' = q_{k-1}$ of x and/or y are reached between $t - t_0$ and $t - 1$.

(b) there is no path $\overline{p} = \overline{u}_1\overline{u}_2 \dots \overline{u}_{n'}$ where $\overline{u}_i \in V$, $\overline{u}_1 = w$, $\overline{u}_{n'} \in B$ such that $(q, \overline{u}_{n'})$ and/or $(q', \overline{u}_{n'})$(see (a) above) and $(\overline{u}_{n'-1}, \overline{u}_{n'})$ are type-alike such that $diam_{POS;\overline{\Gamma}}(\overline{u}_1, \overline{u}_{n'}) \notin [t - t_0, t - 1]$. Therefore, every such parent q'' of $\overline{u}_{n'}$ where $C((q'', \overline{u}_{n'}) = \epsilon$ has $diam_{\overline{\Gamma}}((w, q'')) = t - 1$.

This relation assures that both x and y are not reached between $t - t_0$ and $t - 1$. They are simultaneously reached at t when all of their parents had been reached during the said time interval. This is guaranteed because any such parents q and q' with the same values for the attribute C have $diam_{POS;\overline{\Gamma}}(w, q) = diam_{POS;\overline{\Gamma}}(w, q') = t - 1$.

With the relations in (a) and (b) of every POD x and its sibling y, it is implied that

$$diam_{POS;\overline{\Gamma}}(w, p_c) = diam_{POS;\overline{\Gamma}}(w, c) - 1 = \delta_\Sigma(P_x) = \delta_\Sigma(P_y) =$$
$$\delta_\epsilon(P'_x) = \delta_\epsilon(P'_y) = diam_{POS;\overline{\Gamma}}(w, q'''),$$

where $q''' \in P'_x \cup P'_y$ and $C((q''', c)) = \epsilon$, with $p_c \in \{q, q'\}$ and $c \in B$. This shows that x and y are synched. This proves that R is diametrically-synched if R is a free-choice t_0-step PCN, $1 \leq t_0 < diam(R)$, where every POD x and its sibling y are reachable from any POS $w \in V$ for $g \in \overline{\Gamma}_x \cup \overline{\Gamma}_y$.

(Proof: $\Leftarrow$) Next, we prove that if R is diametrically-synched then it is a free-choice t_0-step PCN, $t_0 < diam(R)$, where every POD x and its sibling y are reachable from any such POS $w \in V$ for $g \in \overline{\Gamma}_x \cup \overline{\Gamma}_y$. To do this, we first determine the value of t_0 that satisfies the second property of Definition 41. Note that for any $q \in V$ where $(q, c) \in E$ and $C((q, c)) = \epsilon$ with $c \in \{x, y\}$, $diam_{POS;\overline{\Gamma}}(w, q) = t - 1$ where w is a is a POS with paths from w to x and w to y because R is diametrically-synched, i.e. $\delta_\epsilon(x) = \delta_\epsilon(P_y) = t - 1$. Since x is a POD and R is bounded, $1 \leq \delta_\Sigma(P_x) < diam(R)$ and $1 \leq t_0 < (\delta_\Sigma(P_x) = \delta_\epsilon(P'_x) = \delta_\Sigma(P_y) = \delta_\epsilon(P'_y)) < diam(R)$. More specifically, $t_0 = \max\{\Delta_x, \Delta_y\}$ when y also a POD, $t_0 = \Delta_x$ otherwise. Since R is diametrically-synched, every $v \in V$ where $(v, c) \in E$ and $C((v, c)) \in \Sigma$, $diam_{POS;\overline{\Gamma}}(w, v) \leq \delta_\Sigma(P_x)$. Therefore, for some $v' \in V$ where $(v', c) \in E$ and $C((v', c)) \in \Sigma$, $t - t_0 \leq diam_{POS;\overline{\Gamma}}(w, v)) \leq (diam_{POS;\overline{\Gamma}}(w, v') = \delta_\Sigma(P_x) = t - 1)$. Since $C((v', x)) \in \Sigma$ and R is diametrically-synched, then $(q, x), (r, y) \in S(t)$ for every $q, r \in V$ where $(q, x), (r, y) \in E$, $C((q, x)), C((r, y)) \in \Sigma$ where $C((q, x)) \neq C((v', x)) \neq C((r, y))$(by Definition 21) and $\nexists s \in V$ where $\{(s, x), (s, y)\} \subset \bigcup_{j=1}^{t_0} S(t - j)$.
■

3.4 The Soundness Property of RDLTs

Definition 44 (Soundness of RDLTs) *An RDLT R is **sound** if for every sink $f \in V$ and a source $w \in R$ where w is an ancestor of f, there exists an activity profile $S = \{S(1), S(2), \ldots, S(k)\}$ where $1 \leq k \leq diam_{POS;\overline{\Gamma}}(w, f) \leq diam(R)$, $(w, v) \in S(1)$, $(q, f) \in S(k)$, $v, q \in V$, and $S(k + 1) = \emptyset$, the following conditions hold,*

(i) $\forall S(t) \neq \emptyset$, $1 \leq t \leq k - 2$, there exists a sequence $vS = N_t N_{t+1} \ldots N_{k-1} N_k$ of vertex sets $N_i \subseteq V$ such that $\exists\{a, b, c\} \in V$ where $a \in N_j$, $b \in N_{j+1}$, and $(a, b) \in S(j)$ and $(b, c) \in S(j + j')$, $j = t, t + 1, \ldots, k - 1, k - 2$, and $1 \leq j' \leq k - j$,

(ii) $\forall (x, y) \in S(k)$, $y = f$, $x \in V$,

(iii) $\biguplus_{i=1}^{t} S(i) \subset \biguplus_{i=1}^{t+1} S(i)$ where $\biguplus$ is the multi-set union operator used on S, $1 \leq t < k$, and

(iv) $\forall (x, y) \in E$, there exists an activity profile with a reachability configuration $S'(t)$ where $(x, y) \in S'(t)$, $1 \leq t \leq diam(R)$.

The first condition (i) in Definition 44 means that there is at least one vertex $v \in V$ where $(u, v) \in S(j)$ that connects to the vertex $q \in V$ where $(q, r) \in S(j + 1)$ which $\mathcal{A}$ traversed the latter arc at time $j + 1$(by Definition 21), $t \leq j \leq k - 2$. Furthermore, if $j + 1 \leq k - 2$, r is not a sink vertex, and therefore r connects to some other vertex $c \in V$

where (r,c) is traversed by $\mathcal{A}$ at a later time $j+j' \leq k$ where $j' \geq 1$, i.e. $(r,c) \in S(j+j')$. The second condition (ii) means that no other arc that does not end at the final vertex f is included in the last reachability configuration $S(k)$ of S. Altogether, (i) and (ii) corresponds to the unhindered continuity of traversals of $\mathcal{A}$ from w to f, i.e. proper termination of an activity.

Meanwhile, (iii) means that an arc (x,y) is always traversed at time $t < k$ and $y \in V$ is connected to at least one vertex that is previously reached by $\mathcal{A}$. Note that (x,y) can be a self-loop, i.e. $(x=y)$. The multi-set union operation is used in (iii) to account these traversals which either reaches a vertex not in previously reached by $\mathcal{A}$ and those which had been. The fourth condition (iv) means that all arcs in R is used at least once in S. This means that the RDLT does not include connections and/or vertices that are unnecessary in executing activities in R. We shall discuss in Section 8 how condition (iv) is verified in R that contains multiple activities.

Remark 9 *Soundness in RDLTs are operationally equivalent with classical soundness in Petri nets(see Section 2), i.e. simply viewing reachability with respect to the reachability configurations of activity profiles and disregarding the typing of arcs due to the values of $M(.)$. Additionally, every activity profile S that is derivable from an RDLT R for its sink/output vertex $f \in V$ can be a basis for constructing a sequence that is equivalent to sound firing sequences of Petri nets(see Section 2). More specifically, the former sequence is constructed by the string "$S(1)S(2)\ldots S(k)$" where $2 \leq k \leq diam_{POS;\overline{\Gamma}}(w,f) \leq diam(R)$, $S(i) \in S$, $i = 1,2,\ldots,k$, and $w \in V$ is a source vertex of R.*

We illustrate soundness property in an RDLT model of a real-world complex system, i.e. an adsorption chiller [68]. Shown in Figure 3.7 is an instance of an RDLT model of the reactor and valve systems of this said chiller. (The model of all the chambers and valve system of the chiller is shown in Section 4.1.) We abstracted the cooling and chilling temperature sources into one source vertex, i.e. controller w, in the model. Furthermore, this model shows a detailed specification of the reactors, their modes of operation, i.e. desorption/adsorption and idle(refrigerant and heat recovery), and the valve subsystem of the chiller. This subsystem is an RBS whose center is $\boldsymbol{y1}$, i.e. $M(\boldsymbol{y1}) = 1$. Furthermore, it shows one instance of labelling and settings of L and C. These labels and settings can illustrate one cycle of operation of the chiller, i.e. the refrigerant is coursed through the reactor chamber where it is adsorbed/desorbed and then passed to the condenser. Note that the labelling $l:c$ on an every arc (x,y) sets $L((x,y)) = l$ and $C((x,y)) = c$ where $c \in \Sigma \cup \{\epsilon\}$, respectively. (The label $l:$ signifies that $c = \epsilon$.)

From Figure 3.7, the continuity of the traversal of arcs from $\boldsymbol{w}$ to $\boldsymbol{z}$ as required by (i) and (ii) of the soundness property is as follows: $S = \{S(1), S(2), \ldots, S(19)\}$, where
$S(1) = \{(\boldsymbol{w},\boldsymbol{x1})\}$,
$S(2) = \{(\boldsymbol{x1},\boldsymbol{x2}), (\boldsymbol{x1},\boldsymbol{x3})\}$,
$S(3) = \{(\boldsymbol{x2},\boldsymbol{x5}), (\boldsymbol{x3},\boldsymbol{x5})\}$,
$S(4) = \{(\boldsymbol{x5}, \boldsymbol{x6}), (\boldsymbol{x1},\boldsymbol{x6})\}$,
$S(5) = \{(\boldsymbol{x6},\boldsymbol{x8}),(\boldsymbol{x5},\boldsymbol{x8}), (\boldsymbol{x1},\boldsymbol{x8})\}$,
$S(6) = \{(\boldsymbol{x8},\boldsymbol{y1})\}$,

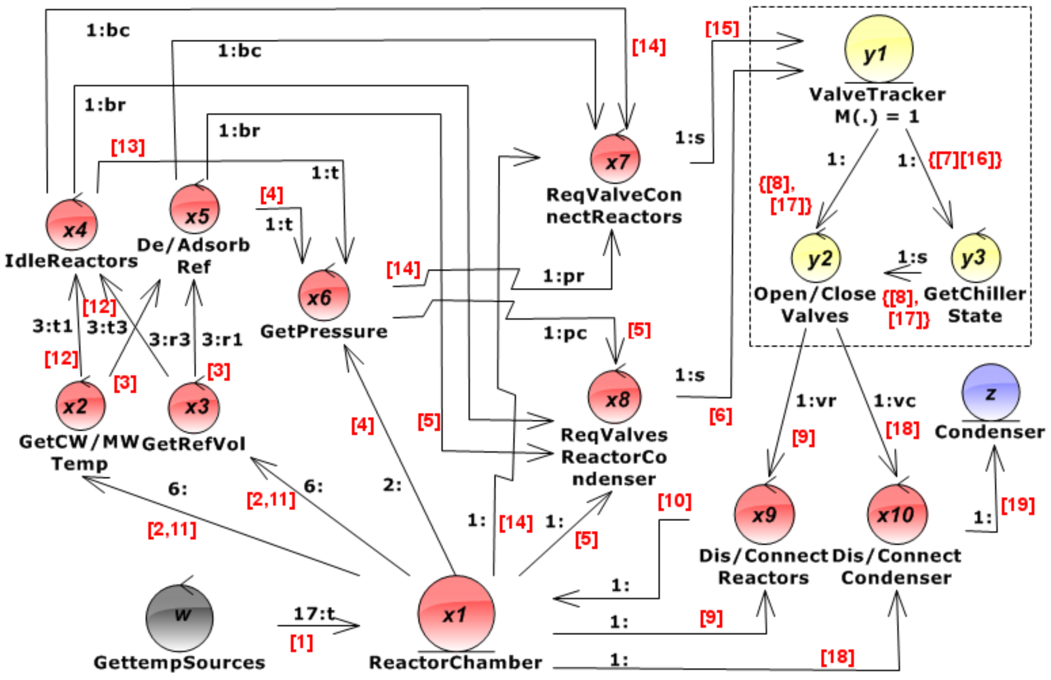


Figure 3.7: RDLT model of the reactors and valve subsystem of an Adsorption Chiller

$S(7) = \{(\boldsymbol{y1},\ \boldsymbol{y3})\}$,
$S(8) = \{(\boldsymbol{y3},\boldsymbol{y2}),\ (\boldsymbol{y1},\boldsymbol{y2})\}$,
$S(9) = \{(\boldsymbol{y2},\boldsymbol{x9}),\ (\boldsymbol{x1},\boldsymbol{x9})\}$,
$S(10) = \{(\boldsymbol{x9},\boldsymbol{x1})\}$,
$S(11) = \{(\boldsymbol{x1},\boldsymbol{x2}),\ (\boldsymbol{x1},\boldsymbol{x3})\}$,
$S(12) = \{(\boldsymbol{x2},\boldsymbol{x4}),\ (\boldsymbol{x3},\boldsymbol{x4})\}$,
$S(13) = \{(\boldsymbol{x4},\boldsymbol{x6})\}$,
$S(14) = \{(\boldsymbol{x6},\ \boldsymbol{x7}),(\boldsymbol{x4},\ \boldsymbol{x7}),\ (\boldsymbol{x1},\boldsymbol{x7})\}$,
$S(15) = \{(\boldsymbol{x7},\boldsymbol{y1})\}$,
$S(16) = \{(\boldsymbol{y1},\boldsymbol{y3})\}$,
$S(17) = \{(\boldsymbol{y3},\boldsymbol{y2}),\ (\boldsymbol{y1},\boldsymbol{y2})\}$,
$S(18) = \{(\boldsymbol{y2},\boldsymbol{x10}),\ (\boldsymbol{x1},\boldsymbol{x10})\}$,
$S(19) = \{(\boldsymbol{x10},\boldsymbol{z})\}$.

Note that the values of the time of traversals shown in $S(t)$ are the vectors in the red font. The vectors on the arcs in $E_{\boldsymbol{y1}}$ of the RBS $G_{\boldsymbol{y1}}$ in Figure 3.7 are annotated as a set to indicate that their arc attribute T are reset to $\mathbf{0}$, hence, the second vector in the set corresponds to the second time the algorithm $\mathcal{A}$ uses $G_{\boldsymbol{y1}}$'s in- and out-bridges.

Meanwhile, the conditions (iii) and (iv) of the soundness property can be easily shown as $\mathcal{A}$ always explores new arcs from time step 1 until the last step to reach z. There is also one activity for $\boldsymbol{w}$ and $\boldsymbol{z}$ in this example.

3.5 Structural Properties of Reachability in RDLTs

Reachability by C-based constraints

Definition 45 (Vertex-simplified R) *A **vertex-simplified RDLT** $G = (V', E', C')$ of $R = (V, E, T, M)$(with arc attributes C and L) is a multidigraph whose vertices $v \in V$ have $V_{type}(v) =$ 'c' where G is derived from R such that the following holds,*

1. *$x \in V'$ if any of the following holds,*
   - *$x \in V$ and $x \notin V_{G_u}$ of an RBS G_u in R, or*
   - *there exists an in-bridge $(q, x) \in E$ of $x \in V \cap V_{G_u}$, $q \in V$ of R, or*
   - *there exists an out-bridge $(x, q) \in E$ of $x \in V \cap V_{G_u}$, $q \in V$ of R*
2. *$(x, y) \in E'$ with $C'((x, y)) = C((x, y))$ for $x, y \in V'$ if $(x, y) \in E$*
3. *$C((x, y)) = \epsilon$ if $x, y \in V' \cap V_{G_u}$ and x is an ancestor of y in R and $(x, y) \notin E_{G_u}$.*

*We refer to this simplification of R as **level-1 vertex-simplification** of R with respect to every RBS G_u in R. A **level-2 vertex-simplification** of R with respect to its RBS G_u is the level-1 vertex-simplification of G_u where G_u is treated as an RDLT where the value of the vertex attribute M of u is redefined to 0, i.e. $M(u) = 0$. With this, the verification of model properties(i.e. maximally composed, sound RDLTs in Section 3.5 and onwards) are separately done for the level-1 and level-2 vertex simplifications of R. However, this separation does not affect the validity of proving for any of these properties on the entire RDLT itself. Therefore, any discussion on vertex simplification of RDLTs in the succeeding sections is assumed to be level-1.*

In Definition 45.(1), the vertices of R that are present in its vertex-simplified RDLT G are those that do not belong to any RBS or those that are inside an RBS but has an in-bridge and/or an out-bridge. For the first set of vertices, any $v \in V' \cap V$ of this set has the values of the arc attributes $C'(q, v)$ and $C'(v, r)$ retained in G where $q, r \in V' \cap V$ with respect to $C(q, v)$ and $C(v, r)$ in R, respectively, as stated in Definition 45.(2). Furthermore, Definition 45.(2) specifies that the values of $C(.)$ of all in-bridges of any vertex $v \in V'$ are retained from R to G. Finally, Definition 45.(3) establishes an arc for every $x, y \in V' \cap V_{G_u}$ in G if there exists at least one path p from x to y in G_u where $|p| \geq 2$. This arc is labelled with $C((x, y)) = \epsilon$ to represent this path and the (yet-to-be-verified) reachability of y from x (using the level-2 vertex simplification of R with respect to to G_u).

By the use of the vertex-simplification G of R, we can verify many properties for R and generalize the verification to R itself. Shown below are the operations that can be performed in G to aid in these verifications.

Definition 46 (Contraction of arcs in RDLTs) *Given an RDLT $R = (V, E, T, M)$ and its vertex-simplified RDLT $G = (V', E', C')$, a **contraction** of $(x, y) \in E'$ of G, $x, y \in V'$, $x \neq y$, results to a multidigraph $G^* = (V^*, E^*, C^*)$ such that*

1. $V^* = V' \backslash \{x,y\} \cup \{x'\}$. Here, x' is a dummy vertex representing x and y.

2. $$E^* = \left\{ E' \bigcup_{\forall z \in V', \exists (z,v)_i \in E', v \in \{x,y\}} \{\bigcup_{\forall i} \{(z,x')_i\}\} \bigcup_{\forall z \in V', \exists (v,z)_j \in E', v \in \{x,y\}} \{\bigcup_{\forall j} \{(x',z)_j\}\} \right\} \backslash$$
$$\left\{ \bigcup_{\forall q \in V', \exists (q,v)_i \in E', v \in \{x,y\}} \{\bigcup_{\forall i} \{(q,v)_i\}\} \bigcup_{\forall q \in V', \exists (v,q)_j \in E', v \in \{x,y\}} \{\bigcup_{\forall j} \{(v,q)_j\}\} \right\},$$

where $z \notin \{x,y\}$, $(a,b)_i \in E'$ is the i^{th} arc from $a \in V'$ to $b \in V'$, $i = 1, 2, \ldots, n$, where n is the outdegree of a.

3. $C^*((z,x')_i) = \epsilon$, $\forall z \in V'$ where $\exists (z,y)_i \in E', i \geq 1$,

4. $C^*((x',z)_i) = C'((v,z)_i)$, $v \in \{x,y\}$, $\forall z \in V'$ where $(v,z)_i \in E', i \geq 1$.

Definition 47 (Feasible contraction) *A contraction of (x,y) is **feasible** if $\nexists (u,x) \in E'$ where $C((u,x)) \in \Sigma$, $u \in V'$ and*

$$\bigcup_{\forall i} \{C'((x,y)_i)\} \cup \{\epsilon\} \supseteq \bigcup_{\forall z \in V',\ where\ \exists (z,y)_j \in E', j \geq 1} \{\bigcup_{\forall j} \{C'((z,y)_j)\}\},$$

where $(x,y)_i \in E'$ is the i^{th} arc from x to y, $i = 1, 2, \ldots, n$, and n is the outdegree of x.

In essence, contraction of $(x,y) \in E$ would represent a merge of x and y in V^* of G^*. The merging of the nodes results to $x' \in V^*$ which represents both x and y of R as described in Definition 45.(1). This representation should therefore also account the connectivities of x and y in R as shown in Definition 45.(2). With this, any arc with the vertex z that is connected to either x or y in R, if it exist, is represented by the connectivity of x' and z in G^*. Since y is considered reachable at this contraction of (x,y), the value of $C^*((z, y = u')_i)$ is set to ϵ in G^* to account that all constraints $C'((z,y)_i) \in \Sigma$ for all $z \in V$ in R had been already satisfied, i.e. a feasible contraction of $(x,y) \in E'$ exists as defined in Definition 47. Furthermore, every arc from x to y in E' are not included in E^* to account the information that all ancestors of y, inclusive of x, which are required to make (x,y) unconstrained(by Definition 21) and y reachable in R had already been previously reached by the algorithm $\mathcal{A}$. Note that every $(q,x)_i \in E'$ has $C'((q,x)_i) = \epsilon$ by Definition 46 since x had been reached at a previous time/contraction step. Finally, Definition 46.(4) accounts $C'((v,z))$, $v \in \{x,y\}$, in G' by representing it as $C^*((u',z))$ with the same values in G^*.

Definition 48 (Contraction Path in RDLTs) *Given an RDLT $R = (V,E,T,M)$ and its vertex-simplified RDLT $G_1 = (V_1,E_1,C_1)$, a **contraction path** from x_1^1 to x_n in G_1 is a sequence $p = x_1^1 x_2 \ldots x_n$, $n \leq |V_1|$, where a contraction is feasible on $(x_1^{i-1}, x_i) \in E_{i-1}$ in G_{i-1} resulting to $G_i = (V_i,E_i,C_i)$ for $i = 2,3,\ldots,n$, and $x_1^i \in V_i$ represents $x_1^{i-1} \in V_{i-1}$ and $x_i \in V_{i-1}$ whose arc (x_1^{i-1}, x_i) is contracted.*

For the succeeding discussions, we shall simply use the name of the leftmost vertex in the contraction path p without its superscript to refer to the dummy vertex that represents the two vertices involved in the contraction of their corresponding arc.

Shown in Figure 3.8 is the RBS of the RDLT in Figure 3.1 with some arcs removed to provide a clearer illustration of the contraction process.

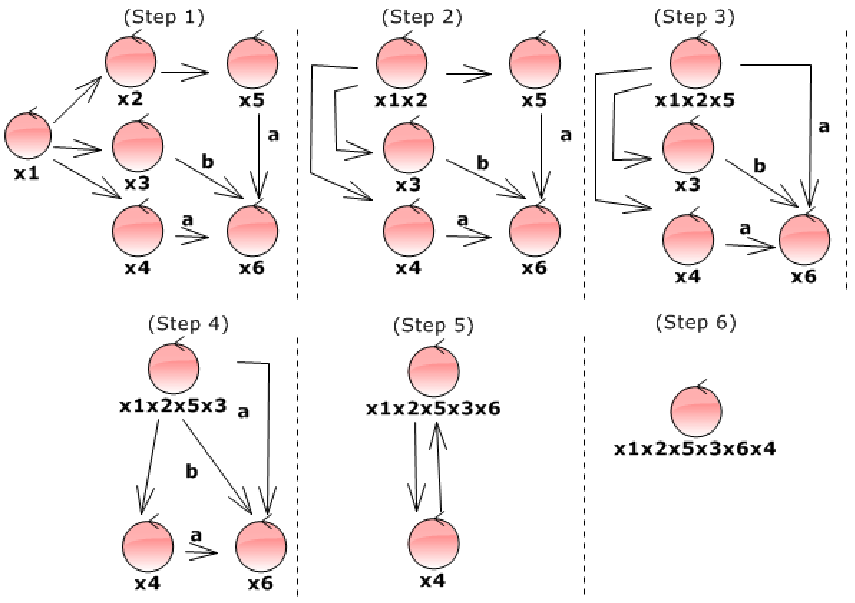


Figure 3.8: Contraction of the RBS of the sample RDLT in Figure 3.1.

In Figure 3.8, **x1** is always used as the parent vertex where contraction is performed on G_i. The contraction path for this model is shown as the name of the vertex at step i=6, i.e. p = **x1 x2 x5 x3 x6 x4**.

Meanwhile, a contraction path for the adsorption chiller model in Figure 3.7 is p = ***w x1 x2 x3 x4 x5 x6 x7 x8 y1 y3 y2 x9 x10 z***. Note that ***x7*** can only be contracted after ***x4*** and ***x6***. The same holds for ***y2*** with respect to ***y1*** and ***y3***.

A View of Compositionality of RDLTs

Structural Composition of Multi-activity RDLTs

Definition 49 (Maximal substructures, deadlock-freeness of R) *Given an RDLT $R = (V, E, T, M)$ and its vertex-simplified RDLT $G = (V', E', C')$, a* ***support*** *$V'' \subseteq V'$ of $x, y \in V'$, where x is an ancestor of y, is an intersection of the sets corresponding to the descendants of $x \in V'$ and the ancestors of $y \in V'$ and includes $\{x, y\}$ in G. The union of the support of x and y, $\forall x \in V'$, is called the* ***maximal support*** *$V_{max} \subseteq V'$ of y. If y is a sink vertex of G, i.e. y has an outdegree 0, the subgraph $R_{V_{max}}$ of R that is induced by the V_{max} is referred to as a* ***maximal substructure*** *of R.*

*For $x, y \in V'$, we say y is C-**reachable** from x in G (and in R) if there exists a support V'' of x and y where there is a contraction path $p = x_1x_2 \ldots x_n$ from $x_1 = x$ to $x_n = y$ in G, $x_i \in V''$, $i = 1, 2, \ldots, n$, $n = |V''|$, where $x_i \neq x_j$, $j = 1, 2, \ldots, n$, and $i \neq j$. In this case, the subgraph $R_{V''}$ of R that is induced by the V'' is called a C-**deadlock-free** substructure of R.*

An RDLT R can contain multiple activities such that it has a set of multiple output/sink vertices $O \subset V$, $|O| \geq 1$, and possibly a set of multiple input/source vertices $I \subset V$, $|I| \geq 1$. With this, there will be multiple maximal substructures in R with vertices that can be shared or exclusively used in some activities corresponding to these substructures. The algorithm $\mathcal{A}$ relies on the input specification of one start vertex $s \in V$ and one output vertex $f \in V$. If $|I| \geq 1$, we can supplement a dummy vertex s_0 to V of R and connect every $v \in I$ with $C((s_0, v)) = \epsilon$ and $L((s_0, v)) = 1$. From this, we would be able to extract the maximal substructure that supports the execution of an activity which uses f and all of $v \in I$. We can therefore use this maximal substructure to produce the values of T to be able to realize the dependency of the vertices in this structure with one another based on connectivities and arc attributes. Using $\mathcal{A}$, the profile $S(t)$ for an activity Act is computed, where $1 \leq t \leq diam(R)$. This profile describes the arcs used for accomplishing Act and their time of traversals. Therefore, from $S(t)$, we can identify processes, control and information flow, and constraints involved in Act as represented by this profile. Since output vertices in R are distinct, there exists at least one arc different in their respective activity profiles. With the nature of $\mathcal{A}$ arbitrarily choosing the next arc to consider, it can select an arc that can be unnecessary to reach the desired output vertex f or can lead to including another output vertex using an arc in $S(t)$. The backtrack process of $\mathcal{A}$ ensures that f is reached, though the time of traversal t is not optimal, i.e. a smaller value of t exists which satisfy all requirements of f, and reaching it. For real world systems, this time difference could be crucial and therefore should be avoided. Furthermore, we can verify model properties in every maximal substructure in R. Hence, we provide a preprocessing algorithm P to isolate these maximal substructures before using $\mathcal{A}$ on these substructures as discussed below.

To ensure that Act is **maximally composed**(see Definition 53) with respect to its output f, i.e. only those necessary inputs, tasks and requirements of f are executed and satisfied at their optimal values of t (including the ϵ-labelled arcs), we supplement pre-processing algorithm P to help in accomplishing this as shown in Algorithm 3.4. Note that ϵ-labelled arcs are included in a profile since they can ensure reachability of f from its corresponding input vertices.

Note that at Line 1 of P, we identify the set of all output vertices O in R. Output vertices have zero outdegrees in R, i.e. sinks. A user can alternatively preset $O \subset V$ in Line 1. At Line 17, P stores in $VofOutput(f)$ for every $f \in O$ such that $VofOutput(f)$ contains $v \in V$ when v is necessary to execute the activity that supports f.

Remark 10 *The time and space complexity of P is $O(|V|^4)$ and $O(|V|^2)$, respectively.*

Algorithm 3.4: P: Generate Maximal Composition of Activities in R

```
Input: RDLT R
Output: Maximal substructures in $R$ for every $v \in O$ where $O$ is the set of sinks
        in $V$ of $R$
1  $V \leftarrow \{s_0\}$, where $s_0$ is a dummy input created in $R$;
2  $V_{type}(s_0) =$ ‘$c$‘;
3  for each $v \in I$ where $I$ is the set of source vertices in $V$ do
4      $E \leftarrow E \cup \{(s_0, v)\}$;
5      $C((s_0, v)) \leftarrow \epsilon$;
6      $L((s_0, v)) \leftarrow 1$;
7      $T((s_0, v)) \leftarrow \mathbf{0}$;
8  end
9  $O \leftarrow \{v \in V | \nexists(v, w) \in E\}$;
10 $VofOutput(v) \leftarrow \emptyset, \forall v \in V$;
11 for each $f \in O$ do
12     $Descendants \leftarrow \{f\}$;
13     while $Descendants \neq \emptyset$ do
14         $Ancestors \leftarrow \emptyset$;
15         for each $v \in Descendants$ do
16             $Ancestors \leftarrow Ancestors \cup \{u \in V | (u, v) \in E\}$;
17             $VofOutput(f) \leftarrow VofOutput(f) \cup \{v\}$;
18         end
19         $Descendants \leftarrow Ancestors$;
20     end
21 end
22 return $VofOutput$;
```

To use P in conjunction with $\mathcal{A}$, we first use P to generate the activity profiles found in $VofOutput(f)$ for each $f \in O$. For each $f \in O$, a subgraph $R_{sub}(f)$ of R which represents an activity resulting to f is generated, where $R_{sub}(f)$ is the graph induced by the vertices in $VofOutput(f)$. R_{sub} is then used as input to $\mathcal{A}$ to produce a set of optimal values of T for each vertex in $R_{sub(f)}$. Due to the addition of the dummy input vertex s_0, $\mathcal{A}$ adds 1 to each value of T. We can simply adjust T to get a set of optimal values for time of traversals by subtracting 1 from it and disregard s_0 from the output of $\mathcal{A}$. Lastly, using $\mathcal{A}$ also shows if the values of L of $R_{sub}(f)$ are correctly set to ensure reachability of the activity's input vertices to f. Shown in Algorithm 3.5 is the pseudocode $ExtractActivityProfiles$ that performs P and sequentially provides $R_{sub}(f)$, and $s_0, f \in V$ found in $R_{sub}(f)$, $\forall f \in O$ as input to $\mathcal{A}$.

Remark 11 *The time and space complexity of $ExtractActivityProfiles$ is $O(|V|^4)$ and $O(|V|^2)$, respectively.*

Algorithm 3.5: *$ExtractActivityProfiles$*: Generate ActivityProfiles in R

```
1 $ActivityProfiles \leftarrow \emptyset$;
2 $VofOutput \leftarrow P$ //preprocessing algorithm;
3 $O \leftarrow \{v \in V | \nexists (v,w) \in E\}$;
4 for each $f \in O$ do
5     $R_{sub}(f) \leftarrow InducedGraph(VofOutput(f)), \forall f \in O$;
6     $ActivityProfiles(f) \leftarrow \mathcal{A}(R_{sub}(f),)$;
7 end
8 return $ActivityProfiles$;
```

Satisfiability of Model Properties in Multi-activity RDLTs

Apart from the presence of multiple maximal substructures in R, we also need to deal with the entire design of R itself. Note that reachability also depends on the values of the arc attribute L. For example, when a conditional structure is encountered that leads to paths that are involved in multiple (and separate) maximal substructures, the values of L must be set appropriately. Loosely speaking, the number of times a vertex $v \in V$ is reached because the condition of the maximum allowable traversals on $(u, v) \in E$ is still satisfied(by Definition 21, must be distributed properly to all the values of $L(.)$ of all outgoing arcs $(v, q) \in E$ of v. This should be done so as to allow reachability of every sink that uses the path that uses every such v, q.

To enable a holistic analysis of the properties of such multi-activity R, we define the concept of an *extended RDLT* as shown below. From the extended RDLT of R, we produce structural relationships that can be established between the substructures and R itself with respect to the defined model properties in this research.

Definition 50 (Extended RDLT) *An **extended RDLT** $R' = (V', E', T, M)$, with arc attributes C' and L', is derived from R such that*

1. *$V' = V \cup \{i\} \cup \{o\}$, where i and o are dummy source and sink where $V_{type}(i) = V_{type}(o) =$ 'c'.*

2. *$E' = E \cup \{(i, u)\} \cup \{(x, o)\}$, $\forall u \in I \backslash \{i\}$, $I \subset V$, $I = \{u \in V | u$ is a source in $V\}$, $C'((i, x)) =$ 'ϵ', $L'((i, x)) = 1$, and $\forall x \in O \backslash \{o\}$, $O \subset V$, $O = \{x \in V | x$ is a sink in $V\}$, $C'((x, o)) =$ 'x_o', $\Sigma := \Sigma \cup \{$'x_o'$\}$, and $L'((x, o)) = 1$. Note that the new labels are distinguishable from each other.*

Definition 51 (C-verifiability of RDLTs) *We say that R is **C-verifiable** if its extended RDLT R' is C-deadlock-free.*

Lemma 4 *Every maximal substructure of R is C-verifiable iff R is C-verifiable.*

Proof 7 *We first prove that every maximal substructure R_k of R is C-verifiable if R is C-verifiable, where $k = 1, 2, \ldots, |O|$, $O = \{x \in V | x$ has an outdegree $0\}$. By Definition*

51, the extended RDLT R' of R is C-deadlock-free. By Definitions 49 and 51, the vertex-simplified RDLT $G = (V', E', C')$ of R' contains $i, o \in V'$ where a support V_{sup} of i and o exists in G where there is a contraction path $p = ix_1x_2 \ldots x_no$ from i to o, with $i, o, x_j \in V_{sup}$, $n = |V_{sup}| - 2$, and $V_{sup} = V'$.

Let R_k be any maximal substructure of R. Let $R'_k = (V_k, E_k, T_k, M_k)$ be the extended RDLT of R_k. Let $G_k = (V'_k, E'_k, C'_k)$ be the vertex-simplified RDLT of R'_k where i_k and o_k are the dummy source and sink vertices in V'_k, respectively. Let $p'_k = x_{(k,1)}x_{(k,2)} \cdots x_{(k,m)}$ be a subsequence of the contraction path p from i to o in G, where $x_{(k,j)} \neq i \neq o$, for $i, o \in V'$, $m = |V'_k| - 2$, $x_{(k,j)} \in V'_k \cap V_{sup}$, $\forall j$. Let $p_k = i_kp'_ko_k$, where $i_k, o_k \in V'_k$.

By Definitions 47 and 50, a contraction from $i_k \in V'_k$ using any $x_{(k,j)} \in V'_k$ using p_k where $x_{(k,j)}$ has indegree 0 is feasible.

Let G^d_k be the graph derived from G_k after performing d feasible contractions on G_k using p_k, $d \in \{1, 2, \ldots, m\}$. For any substring s of the contraction path p in G, where $s = x_{(k,j)}s'x_{(k,j')}$ that contains the substring s' where $Lit(s') \subset V_{sup}\backslash V'_k$, and $x_{(k,j)}, x_{(k,j')} \in V'_k$ of the path p_k, we know that there is no arc $(a, x_{(k,j')})$ in G^j_k, where $a \in Lit(s')$, $1 \leq l \leq t$, which makes the contraction of i_k using $x_{(k,j')}$ in p_k on G^{j-1}_k not feasible because G_k itself is a maximal substructure.

Since o_k has an indegree 1 in G_k with respect to $x_{(k,m)} \in V'_k$, therefore contraction on i_k using o_k in G^m_k by p_k can be performed. Therefore, p_k is a contraction path from i_k to o_k in G_k derived from the support V'_k of i_k and o_k. This proves that the extended RDLT R'_k C-deadlock-free and R_k is C-verifiable. Furthermore, this also proves that every maximal substructure R_k of R is C-verifiable if R is C-verifiable.

(Proof: $\Longleftarrow$) Secondly, we prove that R is C-verifiable if every maximal substructure R_k is C-verifiable. We still use the definitions for R and its extended RDLT R', the vertex-simplified RDLT G of R', R_k and its extended RDLT R'_k, and the vertex-simplified RDLT G_k of R'_k as shown above, for $k = 1, 2, \ldots, |O|$.

Let G^e (G^d_k) be the graph derived from G (G_k) after performing e (d) feasible contractions on G (G_k). Let $p_k = i_kx_{(k,1)}x_{(k,2)} \cdots x_{(k,m)}o_k$ be a contraction path derived from the support $V_{k,sup} = V'_k$ of i_k and o_k, where $m = |V_{k,sup}| - 2$ and $i_k, x_{(k,j)}, o_k \in V'_k$ of G_k, $j = 1, 2, \ldots, m$. Let p'_k be equal to p_k with i_k and o_k removed. Let $q = p'_1p'_2 \cdots p'_{|O|}$. Let $q' = unique(q)$, where $unique(q)$ removes x_i from the sequence q if $\exists x_j$ in q, for $i > j$. Let $p = iq'o$. We now prove that p is a contraction path from $i \in V'$ to $o \in V'$ in G where the support V_{sup} of i and o is $V_{sup} = V'$.

By Definition 47 and 50, by using p, we can contract $i \in V'$ using any $x \in V \cap V'$ in G where x has indegree 0 in R.

For any substring $p' = x_{(k,i)}x_{(t,i+1)}$ of p, $1 \leq i, j \leq |V'| - 2$, two cases arise:

- *if p' is equal to the substring $x_{(k,i')}x_{(t,i'+1)}$ of q, therefore, i can be contracted using $x_{(k,i)}$ on G^{i-1} and thereafter using $x_{(t,i+1)}$ on G^i since either (1) p' in itself is a part of a contraction path p_k of G_k, where $k = t$ and G_k is the vertex-simplified RDLT of the maximal substructure R_k of R, or (2) $x_{(k,i+1)}$ has an indegree 0 in G_k and therefore a contraction on i using $x_{(k,i+1)}$ is feasible in G^i by Definition 47. In the latter case, $k \neq t$.*

- *if p' is a subsequence of q, i.e. there exists the substring $x_{(k,i')}yx_{(t,j')}$ of q where $y = x_{(k_1,i'+1)}x_{(k_2,i'+2)}\dots x_{(k_n,i'+n)}$, $1 \leq n \leq j'-1$, then there exists a contraction path p_r of G_r where $x_{(k_e,i'+d)} = x_{(r,l)}$ in p of G, $l < i$, and therefore, a contraction of i using $x_{(r,l)}$ (or $x_{(k_e,i'+d)}$) had already been performed on G^{l-1}, for $d = 1,\dots,n$. (Note that y was removed from p by unique(q).) After contracting i using $x_{(k,i)}$ on G^{i-1}, a contraction on i using $x_{(t,i+1)}$ in G^i is possible since p_t is a contraction path in G_t, i.e. a contraction on i_t using $x_{(t,l)}$ of p_t is feasible in G_t^{l-1}, where $x_{(t,l)} = x_{(t,i+1)}$ of p.*

With the contraction of i using $x_{(k,m)}$ of p on G^{m-1}, i can be contracted using o on G^m since it is a feasible contraction. This finally proves that p is a contraction path from i to o in G, where the support of i and o is V'. This proves that R is C-verifiable if every maximal substructure R_k is C-verifiable. ■

Remark 12 *The time and space complexity to check if R is C-verifiable is $O(|V|^3)$ and $O(|V|^2)$, respectively.*

Remark 13 *The RDLT of the adsorption chiller in Figure 3.7 is C-verifiable.*

Reachability by L-based constraints

Definition 52 (L-verifiability of graphs) *R is **L-verifiable**(where L sets the maximum number of allowable traversals for each arc in R) if the following structural relations are satisfied,*

1. *For any type-alike arc pair $(x_1, y), (x_2, y) \in E$, if $C((x_1, y)) \neq C((x_2, y))$, where $C((x_1, y))$, $C((x_2, y)) \in \Sigma$, then $L((x_1, y)) = L((x_2, y))$.*

2. *For each $y \in V$ where $\exists(y, z) \in E$ and (y, z) is an out-bridge of y, and $InPlaces(y) = \{x \in V | x \text{ is an ancestor of } y \text{ and } \exists(w, x) \in E \text{ where } (w, x) \text{ is an in-bridge of } x\}$,*

   $$\sum_{x\in InPlaces(y)} \sum_{(w,x)\in E, C((w,x))=\epsilon} L((w, x)) + \sum_{x\in InPlaces(y)} W(x) \geq sumOut(y),$$

   where $W(x) = L((w', x))$, for $(w', x) \in E$ that is an in-bridge of x where $C((w', x)) \in \Sigma$, $sumOut(y) = \sum_{\forall(y,z)\in E} L((y, z))$, where (y, z) is an out-bridge of y, and $L((y, z)) \geq 1$.

3. *For $y \in V$, $L\_Sum_{in}(y) + \delta(InBridge(y)) \geq L\_Sum_{out}(y) + |OutBridge(y)|$ holds, where $L\_Sum_{in}(y) = \sum_{\forall(x,y)\in E, C((x,y))=\epsilon} L((x, y)) + L((w, y))$, $(w, y) \in E$ where $C((w, y)) \in \Sigma$, $w \in V$, $InBridge(y) = \{(x, y) \in E | (x, y) \text{ is an in-bridge of } y\}$, $\delta(S) = 1$ if $|S| \geq 1$, else 0, and $L\_Sum_{out}(y) = \sum_{\forall(y,z)\in E} L((y, z))$, $z \in V$, $OutBridge(y) = \{(y, z) \in E | (y, z) \text{ is an out-bridge of } y\}$.*

Remark 14 *The time and space complexity to check whether R is L-verifiable is $O(|V|^3)$ and $O(|V|^2)$, respectively.*

Remark 15 *The RDLT model of the adsorption chiller in Figure 3.7 is L-verifiable.*

Definition 53 (Maximal Compositionality) *An RDLT R is **maximally composed** if R is C-verifiable and L-verifiable.*

Remark 16 *The time and space complexity to check whether R is maximally composed is $O(|V|^3)$ and $O(|V|^2)$, respectively.*

3.6 Proving Soundness by Structural Properties in RDLTs

Theorem 5 *An RDLT R is sound if R is maximally composed.*

Proof 8 *For any activity profile $S = \{S(1), S(2), \ldots, S(k)\}$ of a maximally composed $R = (V, E, T, M)$ where $S(t) \subseteq E$, $1 \leq t \leq k \leq diam_{POS;\overline{\Gamma}}(w, f) \leq diam(R)$, for a source $w \in V$, and its corresponding vertex sequence $vS = N_t N_{t+1} \ldots N_k N_{k+1}$, $N_j \subseteq V$, $j = t, \ldots, k+1$, which satisfies the first condition of soundness is determined as follows:*

Since R is maximally-composed, we know R and its every maximal substructure R_q is C-verifiable. Therefore, we can pick some R_q, and its extended RDLT R'_q with its vertex-simplified RDLT $G_q = (V'_q, E'_q, C'_q)$, where all the vertices in V'_q corresponds to all the vertices $v \in V$ of R where $\exists(a, b) \in S(t)$, $v \in \{a, b\}$. Since R_q is C-verifiable, therefore there exists a contraction path $p_q = i_q x_1 x_2 \ldots x_n o_q$ from i_q to o_q in G_q, $i_q, x_j, o_q \in V'_q$, $n = |V'_q|$, where the support of i_q and o_q is also V'_q. Therefore, such $S(t)$ and $S(t+1)$ of S is determinable where for $1 \leq a \leq b \leq c \leq n$, $(x_a, x_b) \in S(t)$, $(x_b, x_c) \in S(t+1)$ for the subsequence $x_a x_b x_c$ of p_q, with $x_a, x_b, x_c \in V'_q \cup \{\emptyset\}$, and $(x_a, x_b), (x_b, x_c) \in E'_q$. Therefore, $N_t = \{x_a \in V'_q | (x_a, x_b) \in S(t)\}$ and $N_{t+1} = \{x_b \in V'_q | (x_b, x_c) \in S(t+1)\}$.

Since R'_q is C-verifiable, we know i_q can be contracted using x_n by using the contraction path p_q on G_q. With reference to R, x_n is a sink vertex. Therefore, $\nexists(x_n, x_s) \in S(k+1) \cap E$. With this, the second condition for soundness is satisfied.

With the successive contractions performed on G_q using p_q resulting to $G_q^{(g)} = (V_q^{(g)}, E_q^{(g)}, C_q^{(g)})$, $g = 1, \ldots, |V'_q|$, i.e. $G_k^{(g)}$ is the gth step of contraction from using G_q, we know that either of the following cases is true,

- *if $V_q^{(i+1)} \subset V_q^i$, then a node $v \in V_q^{(i+1)} \subset V_q^i$ is usable for traversal at some time l of the activity extraction algorithm $\mathcal{A}$, wherein $\nexists(u, v) \in E$ which is unconstrained for any time step $l' < l$. (Note that $G_q^{(0)} = G_q$.) For this case, $\bigcup_{l'=1}^{l-1} N_{l'} \subset \bigcup_{l'=1}^{l} N_{l'}$ and for some $(x_a, x_b) \in S(l-1)$, $(x_b, x_c) \in S(l)$, $x_a \neq x_b$.*
- *if $V_q^{(i+1)} \subseteq V_q^i$, then this implies that a vertex $v \in V_q^{(i+1)} \subset V_q^i$ was reused by $\mathcal{A}$ at some time l and a $(v, v) \in E_q^{(g)}$ was deleted in the contraction using the subsequence $x_a x_b$ of p_q where $x_a = x_b = v$. For this case, $\bigcup_{l'=1}^{l-1} N_{l'} \subseteq \bigcup_{l'=1}^{l} N_{l'}$ and for some*

$(x_a, x_b) \in S(l-1)$, $(x_b, x_c) \in S(l)$, $x_a = x_b$. However, we can hide this case from the entire contraction process $G_q^{(0)} G_q^{(1)} \ldots G_q^{(|V_q'|)}$ since there is no change in $G_q^{(i+1)}$ from $G_q^{(i)}$ other than a deletion of $(v, v) \in E_q^{(i)}$.

Note that for either of the cases, the arcs (u, v) and (v, v) in the first and second cases are unconstrained since R_q' is both C-verifiable and L-verifiable. Thus, we have satisfied the third condition for soundness.

By using Lemma 4, we know that all vertices for each maximal substructure R_q of R are contracted to i_q at G_q^n, $n = |V_q'|$. Additionally, with both cases in a step of contraction, we know that all incoming and outgoing vertex of $v \in V$ of the vertex-simplified G derived from R are used in the contraction process for all maximal substructures and G itself. This proves that the fourth condition for soundness is satisfied. This proves that R is sound if R is maximally composed.
■

Remark 17 *The time and space complexity to check whether R is sound is $O(|V|^3)$ and $O(|V|^2)$, respectively.*

Remark 18 *The RDLT of the adsorption chiller in Figure 3.7 is sound and maximally-composed.*

Corollary 2 *An RDLT R is sound iff its extended RDLT R' is sound.*

3.7 Derivable Relations for Model Properties in RDLTs

On C-verifiability and NSC

Lemma 5 *Every RDLT R is C-verifiable iff R is NSC.*

Proof 9 ($\Longrightarrow$) *First, we prove that if R is NSC then R is C-verifiable. To do this, we use the extended RDLT R' of R. Furthermore, we obtain the vertex simplified graph $G = (V', E', C')$ of the extended RDLT R'. From V', we construct a contraction path from the dummy source $i \in V'$ to the dummy sink $o \in V'$ of G. Using this pair, a support V'' of i and o includes all the vertices of V of the RDLT R. V'' is also the maximal support of i and o. From V'', a contraction path $p = u_1 u_2 \ldots u_n$ such that $u_1 = i$, $u_n = o$, $u_i \in V''$, $n = |V''|$, and for every $u_j, u_{j'} \in Lit(p)$ where $1 < j < j' \leq n$, $u_j \in \overline{\Gamma}_{u_{j'}}$. That is, u_j precedes u_j' in p if $u_j \in \overline{\Gamma}_{u_{j'}}$. Here, p is a contraction path because R is NSC therefore every arc (q, x) in the derivations of G where $q \in \overline{\Gamma}_x \cap Lit(p)$ is a parent of (a POD) $x \in Lit(p)$ with $C((q, x)) \in \Sigma$ had already been contracted when x is used for the next contraction. Note that a contraction on x is possible since every other parent $q' \in \overline{\Omega}_x$ has $C((q', x)) = \epsilon$ since R is NSC. With respect to the contraction path p, x also precedes q'. Since the dummy sink $u_n = o \in V''$ is a POD that is connected to every sink o_k of R where for every $\{o_k, o_{k'}\}$, $k \neq k'$, $C((o_k, o)) \neq C((o_{k'}, o))$ and $C((o_k, o)), C((o_k', o)) \in \Sigma$, o is not used for contraction until every o_k has been used for contraction in the last $m + 1$*

derivations of G, where m is the number of sinks in R. Note that $\overline{\Gamma}_o \backslash \{o\} = V'' \backslash \{o\} = V$ of R. This proves that R is C-verifiable if R is NSC.

($\Longleftarrow$) *Next, we prove that if R is C-verifiable then R is NSC. By Definitions 49 and 51, if R is C-verifiable, then there exists a contraction path $p = x_1x_2 \dots x_n$ where $x_i \in V'$ of the vertex-simplified RDLT G for the extended RDLT R' of R, and x_1 is a source, x_n is a sink. Suppose $x_j \in Lit(p)$ is a POD and a POS of some $g \in V'$ in R where $g = x_{j'} \in Lit(p)$. That is, for some $y \in \overline{\Omega}_g$ where $y = x_{j''} \in Lit(p)$, (y, x_j) is a looping arc of g in R. With these relations of x_j, $g = x_{j'}$, and $y = x_{j''}$, the order of these vertices in p would be $p = \dots x_j \dots x_{j'} \dots x_{j''} \dots$, i.e. $1 < j < j' < n-1$ and $j' < j'' < n$. Since p is a contraction path from x_1 to x_n, then $C'((y, x_j)) \in \bigcup_{q \in \overline{\Gamma}_{x_j} \cap Lit(p)} \{C'((q, x_j))\} \cup \{\epsilon\}$ in G by Definition 47. This proves that R is NSC if R is C-verifiable.*

■

Corollary 3 *From Lemma 5, the following are provable,*

1. *Every maximal substructure in R is NSC iff R is NSC.*
   (Follows from Lemma 4.)
2. *Every maximally-composed RDLT R is NSC.*
   (Follows from Theorem 5.)
3. *Every bounded RDLT is C-verifiable.*
   (By Definitions 38 and 40.)
4. *Every sound RDLT R is NSC.*
   (Follows from Theorem 5.)
5. *Every free-choice RDLT is C-verifiable.*
   (By Definitions 38, 40, and 41.)
6. *If every vertex in R is reachable then R is C-verifiable.*
   (Follows from Theorem 2)

On Soundness and Reachability

Theorem 6 *Every RDLT R is sound if every vertex $v \in V$ of R is reachable.*

Proof 10 ($\Longrightarrow$) *First, we prove that if every vertex $v \in V$ of R is reachable, then R is sound. By Lemma 2, if every $v \in V$ is reachable, then R is NSC. From Theorem 5 and Corollary 2, R is sound if its extended RDLT R' is sound. Furthermore, it is also trivially provable that R' is NSC iff R is NSC. That is, $o_j \in V$ of R is reachable, then the dummy sink $o \in V'$ of R' is reachable. Furthermore, a source $w \in V$ of R is reachable too since the dummy source $i \in V'$ of R' has $C'((i, w)) = \epsilon$ and $\nexists v' \in V$ such that $C'((v', w)) \in \Sigma$. Hence $(i, w) \in E'$ of R' is always unconstrained(Definition 21). With these, use R' that contains reachable vertices to prove R' is sound.*

By Definition 23, every POD $v \in V'$ of R', there is an activity profile $S = S(1), S(2), \ldots,$ $S(k)$, $k \leq diam(R')$, where $S(1) = \{(i, w)\}$, and $(q, f) \in S(k)$ for some $q \in V'$ and $f \in V'$ is a sink and a descendant of i. Since R' is NSC, then there exists a contraction path $p = x_1x_2 \ldots x_n$ for the vertex-simplified RDLT G where $v = x_j \in V'$. For $x_j \in Lit(p)$, $\exists(x_{j'}, x_j) \in S(t)$, $1 \leq t \leq diam(R)$. Since x_j is reachable, there is a path $p' = x'_1x'_2 \ldots x'_m$ from $x'_1 = i$ to $x'_m = x_j$ where $x'_{j'} \in Lit(p') \cap \overline{\Gamma}_{x_j}$, $Lit(p') \subseteq Lit(p)$. Therefore, $(x'_{j'}, x'_{j'+1}) \in S(j')$ and $(x'_{j'+1}, x'_{j'+2}) \in S(j'+1)$, $j' = 1, 2, \ldots, m$. Similarly, there is a path $p'' = x''_1x''_2 \ldots x''_{m'}$ from $x''_1 = x_j$ to $x''_{m'} = o$ where $x''_{j'} \in Lit(p'') \cap (\overline{\Gamma}_{x_j} \cup \overline{\Omega}_x)$, and $Lit(p'') \subseteq Lit(p)$. Since o is a sink, therefore and every pair of sink vertices o_1, o_2 in V of R have $C'((o_1, o)) \neq C'((o_2, o))$ in R' where $C'((o_1, o)), C'((o_2, o)) \in \Sigma$, then $S(diam(i, o) = diam(R')) = \{(o_j, o) | (o_j, o) \in E'$ of $R'\}$. These satisfy the first and second conditions for soundness. Since $diam(R') \in \mathbb{N}$ and o is reachable using S, therefore the third condition for soundness is satisfied. Lastly, because of the values $C'((o_1, o)), C'((o_2, o)) \in \Sigma$ of every pair $(o_1, o), (o_2, o) \in E'$ and there exists a contraction path from i to o, therefore $E' = \bigcup_{j=1}^{diam(i,o)} S(j)$. This satisfies the last condition for soundness. These prove that R' is sound. Finally, note that $S(diam(i, o))$ only contains arcs from the sink vertices $o_j \in V$ of R to $o \in V'$ of R'. This also shows that there exists at least one activity profile for o_j in R. By Corollary 2, R' is sound iff R is sound. This proves that R is sound if every vertex $v \in V$ of R is reachable.

($\Longleftarrow$) Secondly, we prove that if R is sound, then every vertex $v \in V$ is reachable. To establish this, we also use the extended RLDT R' of R'. By Definition 44, there exists an activity profile $S = S(1), S(2), \ldots, S(diam(R'))$ for the source i and sink o of R'. We establish that every $v \in V'$ of R' is reachable by obtaining an activity profile $S_v = \{S'(1), S'(2), \ldots, S'(k)\}$, $k \leq diam(R)$ for the RDLT R that uses i, v, and some sink $o_j \in V$ of R, such that $\exists(q, v) \in S'(t')$ for some $q \in V'$, $1 \leq t' \leq k$. S_v is determinable from S using the following construction,

Note that we bypass $t = 1$ and $t = diam(R')$ because this corresponds to the arcs used by the dummy source i and o in R'. Since the algorithm $\mathcal{A}$ reaches $v \in V'$ by traversing the arcs found in the activity profile S, extracting the same arcs that are used by paths containing v gives a minimal subgraph of R' that uses v and some output/s $o_j \in V'$. For a sound RDLT R, every arc in R is used for some activity profile(see the fourth condition of Definition 44), therefore there exists an activity profile S_v for every $v \in V$ of R. Lastly, by using only those paths that contain i, v, o_j guarantees that only one activity profile is generated and $S(k)$ contains (q, o_j) for any $q \in V$, and $\nexists(q', o_{j'}) \in S(k)$ where $o_j = o_{j'}$, $q' \in V$.

■

Corollary 4 *Every RDLT R is bounded iff R is sound.*
(Follows from Theorem 2.)

Corollary 5 *From Theorem 6 and Corollary 4, the following are provable,*

Algorithm 3.6: Construction of activity profile S_v that uses $v \in V'$ using the activity profile S of R'

```
   Input: Activity Profile $S$ of $R'$
   Output: Activity profile $S_v$ that uses $v \in V$
1  $validIndices \leftarrow \emptyset$;
2  for $2, 3, \ldots, diam(R') - 1$ do
3      $S_{temp}(t) \leftarrow \{(a, b) \in S(t) |$there exists a path $p$ from $i$ to $o_j$ where
        $a, b, v \in Lit(p)\}$;
4      if $S_{temp}(t) \neq \emptyset$ then
5          $validIndices \leftarrow validIndices \cup \{t\}$;
6      end
7  end
8  $t' = 1$;
9  for $t = 2, 3, \ldots, diam(R') - 1$ do
10     if $t \in validIndices$ then
11         $S_v : S'(t') \leftarrow S_{temp}(t)$;
12         $t' \leftarrow t' + 1$;
13     end
14 end
15 return $S_v$;
```

1. *Every maximally-composed RDLT R is bounded.*
   (Follows from Theorem 5.)

2. *Every free-choice RDLT R is sound.*
   (By Definitions 40 and 41)

3. *Every diametrically-synched RDLT is sound.*
   (By Definitions 40 and 43.)

4. *Every vertex in R is reachable if R is maximally-composed.*
   (Follows from Theorems 5 and 2.)

5. *Every vertex in R is reachable if R is free-choice.*
   (Follows from Theorem 2 and by Definitions and 40 and 41.)

6. *Every vertex in R is reachable if R is diametrically-synched.*
   (Follows from Theorem 2 and by Definitions 40 and 43.)

A Summarization of the Relations of Model Properties for RDLTs

Shown below in Figure 3.9 is the Venn Diagram that pictorially presents the relationships of these properties for RDLTs.

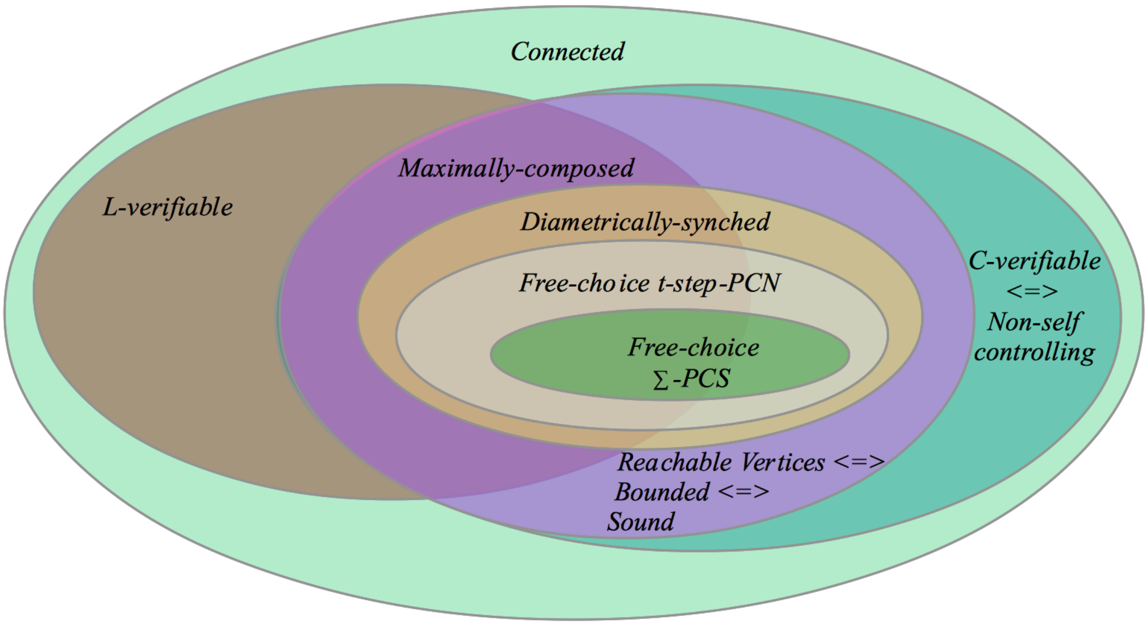


Figure 3.9: Venn Diagram of the model properties that can be used to verify RDLTs.

4

RDLT Modelling for Real-world Complex Systems

In this chapter, we shall use our proposed framework of modelling and activity extraction to profile and describe complex systems. In particular, we focus modelling adsorption chillers and extract an activity from their modes of operations.

4.1 On Applying the Proposed Framework On Energy Systems

Adsorption chillers are designed as closed-cycle (vacuum) machines with an evaporator, adsorber/desorber reactor beds, and a condenser wherein the refrigerant is passed through them during cycles of operation of chillers as shown in Figure 4.1. Associated to these components are three main temperature loops, i.e. heating water circuit (HT), chilling water circuit (LT), and cooling water circuit (MT) as seen in the figure. These loops aid in circulating the refrigerant throughout all the components of the chiller by the processes of evaporation, adsorption/desorption, and condensation.

We see in Figure 4.1 the chiller's midsection that provides two reactor beds which continuously alternate as an adsorber/desorber throughout the machine operation for a stable and continuous cooling. The bed with its temperature line coloured in blue performs adsorption, the red one desorption of the refrigerant. Adsorption takes place as the refrigerant heated and vaporized in the evaporator. By pressure value differences of the evaporator and the reactor bed, one valve is opened thereby transporting the vaporized refrigerant to the adsorber bed. Simultaneously, the other reactor bed desorbs the previously-adsorbed refrigerant. In a similar manner, opening of another valve transports the desorbed refrigerant to the condenser. In the condenser, the refrigerant is liquified and is eventually coursed back towards the evaporator to close the cycle. Before the two beds switch roles as adsorber and desorber, the system of pneumatically-actuated

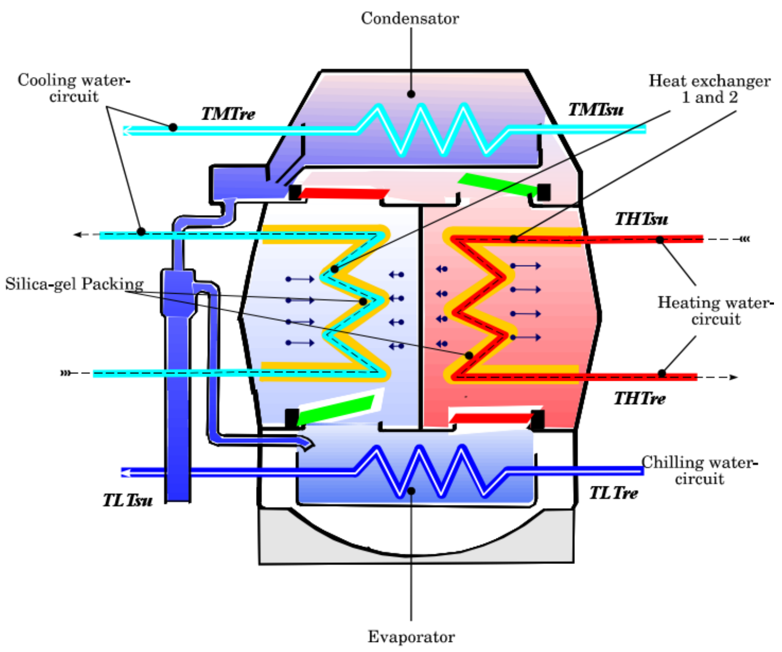


Figure 4.1: The adsorption chiller cooling specifications [68]

valves open and close to connect or disconnect appropriate chambers of the chiller to enable mass and heat recovery periods(see details in [68]).

Shown in Figure 4.2 is the RDLT representing the 2-bed adsorption chiller in Figure 4.1. In this model, we created two (2) entity objects e1 and h1 to represent the evaporator and the condenser, respectively. The two bed reactors are abstracted as g1 with a controller g4 and g5 to accommodate the requirement of alternating bed states (i.e. either act as an adsorber or desorber) and the change of states of operation (i.e. adsorption, desorption, idle bed states (which covers the mass and heat recovery states)). Notice that the set of valves of the chiller is modelled as a reset-bound system with center f1. We use the boundary object a1 as a start node of our proposed algorithm's walk. Furthermore, the RDLT is modelled with $L((x, y)) = 1$, $C((x, y)) \in \{$'r', 't','p', 'b', 's', 'v'$\}$, where 'r' represents the refrigerant mass (either in liquid or vapor form), 't' and 'p' represent temperature and pressure values/conditions, 'b' represents the bed states, 's' represents the list of valves which needs to be change their state (either open or close), and 'v' represents the list of valves that actually changed in state based on 's', $\forall(x, y) \in E$. These labels are shown in black font texts alongside the arcs. The subsystems of this RDLT have $M(\text{f1}) = 1$ and 0 for other objects in the model.

Figure 4.2 shows markings of T representing one of the many activities which we can extract from RDLT model. These activities describe the various processes taking place during duty cycles in adsorption chillers. In this model, the extracted activity starts when the chiller is activated by starting at the boundary object a1 (at time step 1) and ends when the liquid refrigerant exits the condenser and flowing back to the evaporator. Essentially, the extracted activity models the refrigerant flow from its stages of evaporation, desorption/adsorption, and condensation. The final arc which was traversed by the algorithm that represents this terminal process of this activity is (h7, e1) at time step 31. All other intermediate nodes which the algorithm walked through are also annotated with their corresponding time of traversals shown in red font text. Note on the values of T for the arcs of the reset-bound subsystem with center f1. For example, we know that (f1, f2) was traversed at the 11th time step of the walk. When (f3, e7) was traversed by the algorithm at the 12th time step, a reset was consequently performed on the reset-bound system with center f1. With the valve state change request from g7 to f1 at time step 18, we know that the algorithm is capable of using/traversing f1 to f2, and f2 to f3 to comply with the request since $T((\text{f1, f2}))$ and $T((\text{f2, f3}))$ have been previously reset to 0 and their maximum number of traversals $L((\text{f1, f2})) = L((\text{f2, f3})) = 1$. Traversals on these arcs were done again at time steps 19-20 and 28-29.

The resets of T (or the absence thereof) on arcs which are not included in reset-bound systems but when traversed triggers the reset, e.g. (f3, e7) at time step 12, is an important aspect in effective activity extraction and system modelling. Notice that the algorithm does not perform traversal (f3, e7) at any moment during the time steps $\{20, 21\}$ and $\{29, 30\}$ since we obtain $T((\text{f3, e7})) = L((\text{f3, e7})) = 1$ already. This correctly models the real-world operation and thermodynamically-bound design of the chiller for which only the valves involved during the adsorption/desorption and condensation are changed. Had we set $L((\text{f3, e7})) > 1$ with the assertion that the subsystem representing the valve components are continually reused anyway, a wrong control scheme would have proceeded after time step 12. This is one of the main reasons why a scheme for loops and resets are disjointly formulated for modelling various systems.

With the application of our proposed algorithm on the RDLT in Figure 4.2, we obtain the sequence of the extracted activity profile as:

a1 a2 [c1, b1, d1] b2 e1 [e2 e3] e4 e5 e6 (f1 f2 f3) e7 g1 [g2, g3, g5] g4 g6 g7 (f1 f2 f3) g8 h1 [h2, h3] h4 h5 h6 (f1 f2 f3) h7.

This sequence above shows the order of reachability and execution of the nodes of the RDLT, i.e. from a1, the algorithm reaches and executes a2 at time step 1. The nodes enclosed in square brackets [ ] represent nodes which are reachable at the same time step by parallel flows of execution from its parent node (e.g. parallel flows from from a2 to either c1 or b1 or d1). The nodes in parentheses represent those which belong to reset-bound systems. Note that the algorithm can yield multiple (distinct) sequences representing different process flows given the same starting input node to the same final arc. These sequences can be further used to provide bases for isomorphisms, model simplifications, and model validations across various designs of energy systems such as chillers. On the application side, the results of this research shall support improved energy system modelling, optimization, redesign schemes for system improvements, and fault-detection and diagnosis on such systems.

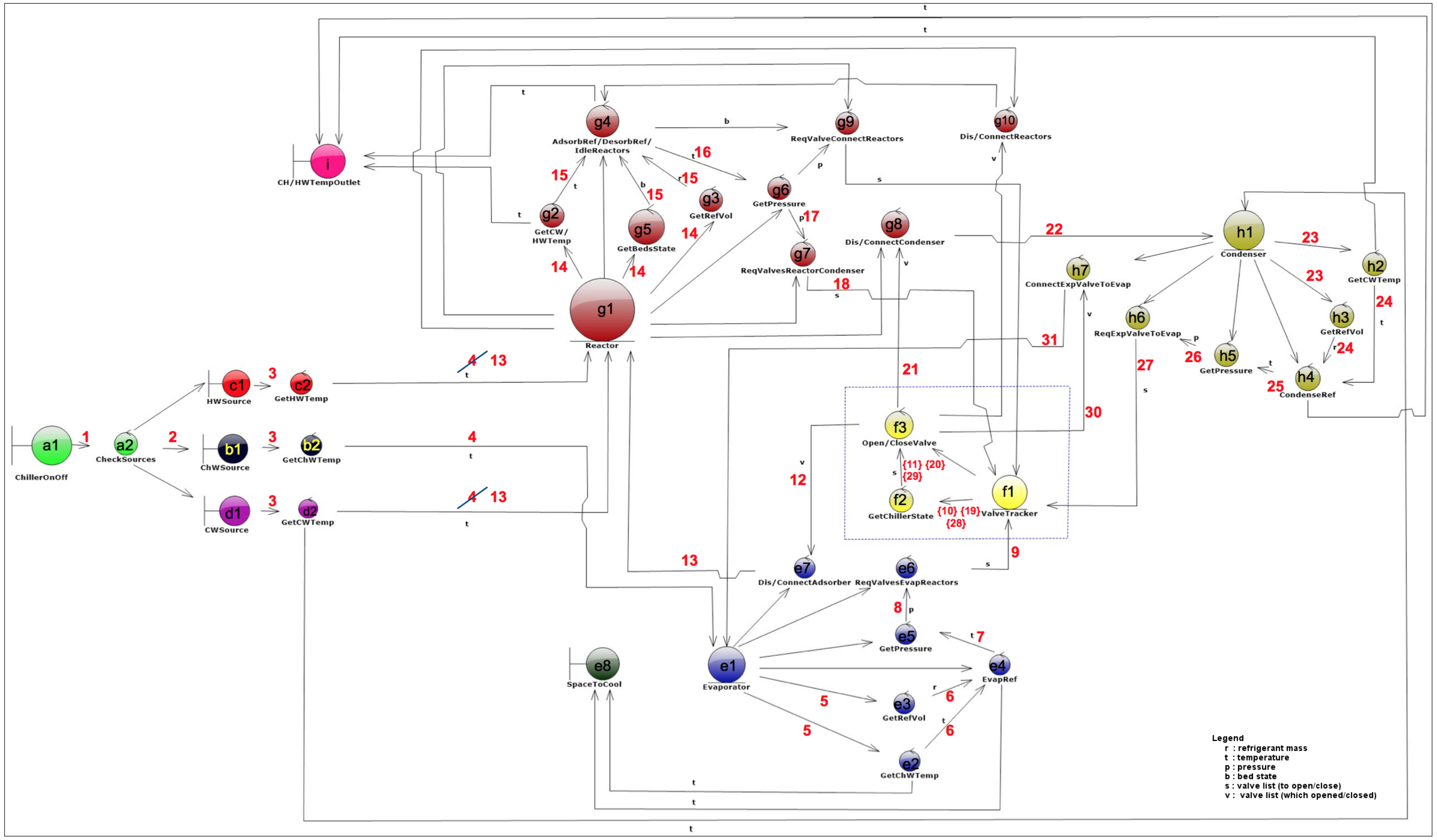


Figure 4.2: Robustness Diagram with Loops and Reset Controls Representation of a 2-bed adsorption system. The subsystems of this RDLT have $M(\text{f1}) = 1$ and 0 for other objects in the model.

Object: Label	**Object Specification**	**Controller: Label**	**Controller Specification**
a1:ChillerOnOff	System Interface to activate chiller system as on/off	a2:CheckSources	Surveys activation of heat sources for chilling, cooling and heating
b1:ChWSource (w:GettempSources)	System Interface to activate Chilling Water(ChW) temperature source	b2: GetChWTemp	Gets ChW temperature setting
c1:HWSource (w:GettempSources)	System Interface to activate Heating Water(HW) temperature source	c2:GetHWTemp	Gets HW temperature setting
d1:CWSource (w:GettempSources)	System Interface to activate Cooling Water(CW) temperature source	d2:GetCWTemp	Gets CW temperature setting
e1:Evaporator	System component representing Evaporator	e2:GetChWTemp	Gets ChW temperature setting
		e3:GetRefVol	Gets refrigerant volume in evaporator
		e4:EvapRef	Executes evaporation of refrigerant
		e5:GetPressure	Gets pressure values inside evaporator
		e6:ReqValvesEvapReactors	Triggers valve system to change states of valves between the evaporator and reactors
		e7:Dis/ConnectAdsorber	Dis/connection evaporator chamber and the adsorber through changes of valve states
e8:SpaceToCool	System Interface to relay chilling temperature to the intended space for cool		

Table 4.1: Adsorption Chiller specifications and labelling reference(Part 1/2)

Object: Label	**Object Specification**	**Controller: Label**	**Controller Specification**
f1:ValveTracker (y1)	System Component representing the set of valves in the chiller	f2:GetChillerState (y3)	Gets the state of the chiller
		f3:Open/CloseValve (y2)	Changes a set of valves from open to close or vice versa
g1:Reactor (x1:ReactorChamber)	System component representing a reactor bed	g2:GetCW/HWTemp (x2)	Gets the temperature of the CW or HW
		g3:GetRefVol (x3)	Gets the refrigerant volume in the reactor bed
		g4:AdsorbRef/ DesorbRef/ IdleReactors (x4:IdleReactors + x5:De/AdsorbRef)	Executes a change of state of a reactor into either of the following: adsorbing refrigerant from its vapor state to solid onto the adsorbate, desorbing solid refrigerant to vapor from the adsorbate, changes the reactor bed into its idle mode
		g5:GetBedState	Gets the state of the bed, whether it is adsorbing, desorbing, or idle
		g6:GetPressure (x6)	Gets pressure values inside the reactor bed
		g7:ReqValves ReactorCondenser (x8)	Triggers valve system to change states of valves between the reactors and condenser
		g8:Dis/ConnectCondenser (x10)	Dis/connects reactor and condenser through changes of valve states

Table 4.2: Adsorption Chiller specifications and labelling reference(Part 2/3)

Object: Label	**Object Specification**	**Controller: Label**	**Controller Specification**
		g9:ReqValveConnectReactors (x7)	Triggers valve system to change states of valves between the two reactor beds
		g10:Dis/ConnectReactors (x9)	Dis/connects the two reactor beds through changes of valve states
h1:Condenser (z)	System component representing condenser	h2:GetCWTemp	Gets CW temperature setting
		h3:GetRefVol	Gets refrigerant volume in condenser
		h4:CondenseRef	Executes condensation of refrigerant
		h5:GetPressure	Gets pressure values inside condenser
		h6:ReqExpValveToEvap	Triggers valve system to change the state of the expansion valve to send refrigerant back to the evaporator
		h7:ConnectExpValveToEvap	Sends the refrigerant from condenser to evaporator through the expansion valve
i:CH/HWTempOutlet	System interface to relay CH and HW temperature to outlet		

Table 4.3: Adsorption Chiller specifications and labelling reference(Part 3/3)

5 Conclusions

5.1 Conclusions

In this chapter, the conclusions that are based from the theoretical and applied results in this research are listed. A short discussion of the main contents pertaining to each of these conclusions are provided for reference. These discussions and listing is enumerated based on the ordering of the results from the main sections in the two previous chapters. These are the following,

Section 3.1 provided the framework in constructing Robustness Diagrams with Loop and Time Controls that can support modelling and verification of workflows with all workflow dimensions, i.e. process, resource, and case, as well as persistence/volatility included in one design. This framework is proposed in cognizance of the lack of literature with such support on modelling particularly serving representations for complex systems. The current literature provides business and scientific models and verification framework that are either process-, resource-, case-centric, or a pairwise combination of them. This research accomplishes its intended framework through the inclusion of multi-type vertices that addresses the need for representations of resources, explicit associations of resources to tasks(i.e. controllers), and the implied typing of the arcs through these associations. Through these differentiations of types in vertices and arcs, both basic and advanced process flows can be supported. In addition to sequential and looping, this research enriches conditional control schemes by its use of arc attributes C, L, T. This enrichment is apparent in a way that reachability does not only rely on topology but also parameter-based constraints as well as bounded repeatability of resource/task reuse. Furthermore, the inclusion of the vertex attribute M further provides more controls for process flow execution by providing means to define substructures in RDLTs, the management thereof, and their encapsulation from the entire model. Therefore, RDLTs do not only provide means for nested activities/tasks, but also for role- or function-based encapsulation of subsystems and their interactions in designs. Through the support of these basic and advanced control schemes for splits and joins accounting all these attributes and subsystems enables effective embedding and/or extraction of multiple activities in RDLTs. Section 3.1 provides an algorithm for activity extraction from RDLTs with such required specifications.

Section 3.1 provided a view of behavioral relations among components of RDLTs with regard to the activities that these components support during case executions. From this view, the concept of reachability of vertices with regard to the satisfaction of attribute-driven traversal requirements is given. Furthermore, different schemes for the construction of controls for process flows that account these traversal requirements are also provided in this section. Samples of design specifications on the structural profile in RDLTs are given so as to effectively implement the different control flows for sequential, parallel, conditional(splits and joins), and iterative processes. Furthermore, these design specifications also highlight the importance of using the attributes C, L, T, M to enable more complex control flows such as the k-out-of-n join patterns, XOR and basic OR-splits, AND-splits, AND-joins, or a combination thereof that establish well-structured workflows in modelling. Furthermore, the concept of cancellation regions in workflows are now adopted in RDLTs through the use of reset-bound subsystems. However, through these subsystems, this research addresses the problems of encapsulation, memory utility, interactions, and poor control schemes for resets in topologically- and behaviorally-related components in workflows. Therefore, this section establishes the ability of these control flow schemes in RDLTs to cover the modelling of these well-known patterns in workflows. It is also illustrated how the latter set of patterns cannot be directly related to the former because RDLTs now incorporate the resource dimension with profiles of persistence and volatility. With this combination, it is concluded that RDLTs support the three workflow dimensions altogether in terms of modelling both the structural and behavioral aspects of complex system representations. Furthermore, this section also provided a structural view of reachability of components in RDLTs. Without using the proposed algorithm for activity extraction that is discussed in Section 3.1, the traversal conditions, and T, a set of paths and diameters in RDLTs are established to determine whether a model is connected or not. This is accomplished by considering the frequency of usage of every component in a model in creating these paths. It is shown that this frequency is determined by using the values of the attributes L, C, and M. In the succeeding discussions, it is proved that this structural view can be used to conclude behavioral aspects of the models.

Section 3.2 presented concepts of Points-of-Interests(POIs) in RDLTs. These POIs were conceptualized to enable structural characterizations in subsystems where delays of reachability, profiles causing non-reachability, points of synching, and possible regions of bottlenecks(though points of reuse) are present when handling multiple activities in RDLTs. Results in Section 3.2 proved relationships between these POIs and the resulting neighborhood types that are generated from them by providing metrics for reachability, boundedness, and synchronicity of task execution. Note that these metrics are determined with considerations of topological and attribute-driven relationships of the model specifications in all three workflow dimensions. Additionally, this research has also shown how these metrics are dealt with the addition of reset-bound subsystems in RDLT models.

Section 3.3 contain results that address the aspect of the Free-choice property in RDLTs. Although this property is mainly used in verifying process-centric workflows, this research adopts to RDLTs under the multidimensional view of workflows that contain persistent and volatile components. The adoption relies on the results of reachability and boundedness in Section 3.2 to formulate the concept of Free-choice property for such RDLTs. In this research, the aspect of this property with respect to the freedom in selecting choices whenever they are posed during task execution is maintained in RDLTs. This aspect is abstracted as the Σ-distinct PCS condition of the Free-choice property in RDLTs. This condition provides that conditional joins are maintained between parent-child arcs for every two sibling nodes. These siblings are also imposed to have the same set of parents to enforce this condition. However, the Free-choice property is further extended in this research by accounting paths and path lengths with consideration of the three workflow dimensions and reset-bound subsystems. It is proved in this research that a t_0-step PCN condition can simulate structures with this freedom of making choices for task execution. From this proof, it is concluded that a mixing of conditional joins, parallelism in process flows, and certain topological configurations involving POIs can duplicate the behavior of structures Σ-distinct PCS.

Section 3.4 proposed a definition of soundness in multi-activity RDLTs. This definition relies on the behavioral aspects that are established from the relationships of reachability during arc traversals of the algorithm for activity extraction in Section 3.1. This definition maintains the concept of proper termination of case executions for every substructure that support the execution of an activity. By its use of the concept of reachability configurations in every activity profile in RDLTs, the concept of reachable markings in the processing of tokens pertaining to a completion of case execution in nets. Furthermore, the requirement that there are no dead markings, that every transition is used in at least one case execution, and that the liveness property [25] in workflow nets are also carried on through this proposed definition of soundness for RDLTs. Therefore, the concept of classical soundness in workflows [55] is adopted in RDLTs. However, note that the latter definition is further adjusted to support the composition of RDLTs where all three dimensions in place. This adjustment is accomplished and well-supported through the use of the results in Section 3.2 on POIs and their resulting neighborhoods. These are used to identify the resources, process specifications, and the corresponding cases that they support to determine disjointness of the multiple activities in RDLTs.

Section 3.5 showed two aspects of RDLTs can be used to provide an efficient means to verify soundness(Definition 44) in these models. The aspect involving constraints that mainly drive splits and joins is separated from the view of repeatability of usage of tasks. The former mainly focuses on the use of the arc and vertex attributes C and M, respectively. The formulation and detection of deadlocks by structural analysis through these attributes are accomplished. With the presence of multiple activities in RDLTs, this research provided an extension of these models to establish a holistic analysis on them with respect to deadlock detection. In order to prove the absence of deadlocks, and therefore, proper termination of process executions with respect to C and M, this research established a multi-level yet simplified graph representations of RDLTs.

The levels are brought about by the presence of reset-bound subsystems in the models. However, the introduction of levels in these simplifications do not hinder in a proper and effective verification of deadlocks even for multi-activity RDLTs. This research provided a provable means of effective encapsulation of the components and their interactions within their subsystem and across other components in designs. Through these multi-level encapsulations and simplifications, both level- and hierachical-based characterizations of models are therefore achieved in this research. Regardless of the approach of analysis, whether activity-based or holistic-based, properties can be verified and then specialized or generalized in RDLTs using their multi-level and simplified representations.

Meanwhile, another aspect in RDLTs that is usable for an efficient verification of their soundness(Definition 44) relies on the repeatability of usage of tasks. Checking for an aspect of soundness using the latter is still driven by static information which are mainly pertaining to the values of L and M. The multi-level view of RDLT representations from Section 3.5 are still partially used in this checking. However, a holistic approach is performed using this perspective of analysis. Different relationships among the values of L were established in this research to guarantee the completeness of the execution of each case that is handled by its set of activities. These relationships were established for type-alike arcs, for in- and out-bridges of a reset-bound subsystem, and for mixed-type of incoming and outgoing arcs of nodes.

Using these aforementioned aspects of structural composition in RDLTs, this research concludes that soundness(Definition 44) in RDLTs can be verified in polynomial amount of time and space. The details of the supporting proof was discussed in Section 3.6. It covers both activity-based and holistic-based verification of soundness in RDLTs containing multiple activities.

Section 3.7 showed provable relationships among the properties for RDLTs with respect to verifying reachability, boundedness, deadlock-freeness, synchronicity, free-choiceness, and soundness. With all these results, this research concludes that the proposed formalizations of construction and verification of RDLTs address the lack of frameworks for modelling and verification of workflows that support all three workflow dimensions. A multi-level approach for modelling and analysis was also provided for support, encapsulation, and verification of subsystems. In addition to this, this research also addresses the need to have modelling and verification of such models where persistence and volatility are required. These requirements are apparent when modelling complex systems. Section 4.1 illustrated how these aspects of the proposed framework are used by providing RDLT representations for real-world systems, i.e. energy systems particularly on adsorption chillers. The algorithm for activity extraction in this proposed framework was applied on this representation to illustrate the extraction of one activity profile in the chiller representation. Finally, the core results of this research is summarized in Table 5.1.

Result	**Label**
For an RDLT R that is not NSC, there exists a POD $x \in V$ that is not reachable from a source $w \in V$ of R.	Theorem 1
If every vertex $v \in V$ of R is reachable from a source $w \in V$ then R is NSC.	Lemma 2
Every $x \in V$ of in an RDLT R is reachable iff R is bounded.	Theorem 2
A free-choice PCS R is a t_0-step PCN RDLT.	Lemma 3
An RDLT R is diametrically-synched if R is a Free-choice Σ-distinct PCS where for every POD x and its sibling y and every $w \in V$ that is a POS of some $g \in \overline{\Gamma}_x \cup \overline{\Gamma}_y$ in R, there are paths from w to x and from w to y and every $(a, b) \in E$ along these paths have $L((a, b)) \geq L_{reqMinimum}((a, b))$.	Theorem 3
An RDLT R is diametrically-synched iff R is a Free-choice t_0-step PCN, $t_0 < diam(R)$, where for every POD x and its sibling y and every $w \in V$ that is a POS of some $g \in \overline{\Gamma}_x \cup \overline{\Gamma}_y$ in R, there are paths from w to x and from w to y and every $(a, b) \in E$ along these paths have $L((a, b)) \geq L_{reqMinimum}((a, b))$.	Theorem 4
Every maximal substructure of R is C-verifiable iff R is C-verifiable.	Lemma 4
An RDLT R is sound if R is maximally composed.	Theorem 5
An RDLT R is sound iff its extended RDLT R' is sound.	Corollary 2
Every RDLT R is C-verifiable iff R is NSC.	Lemma 5
Every maximal substructure in R is NSC iff R is NSC.	In Corollary 3
Every maximally-composed RDLT R is NSC.	In Corollary 3
Every bounded RDLT is C-verifiable.	In Corollary 3
Every sound RDLT R is NSC.	In Corollary 3
Every free-choice RDLT is C-verifiable.	In Corollary 3
If every vertex in R is reachable then R is C-verifiable.	In Corollary 3
Every RDLT R is sound if every vertex $v \in V$ of R is reachable.	Theorem 6
Every RDLT R is bounded iff R is sound.	Corollary 4
Every maximally-composed RDLT R is bounded.	In Corollary 5
Every Free-choice RDLT R is sound.	In Corollary 5
Every diametrically-synched RDLT is sound.	In Corollary 5
Every vertex in R is reachable if R is maximally-composed.	In Corollary 5
Every vertex in R is reachable if R is Free-choice.	In Corollary 5
Every vertex in R is reachable if R is diametrically-synched.	In Corollary 5

Table 5.1: Summary of the main results for modelling and verification of RDLT models for complex systems modelling.

List of Figures

List of Tables

List of Algorithms

Bibliography

[1] Hollingsworth, D.: Workflow Management Coalition The Workflow Reference Model WFMC-TC-1003, v. 1.1. (1995)

[2] Weske, M.: Business Process Management: Concepts, Languages, Architectures. Springer-Verlag Berlin Heidelberg. ISBN 978-3-540-73521-2. (2007)

[3] Decker, G.: Design and Analysis of Process Choreographies. PhD thesis, Hasso Plattner Institute, University of Potsdam, Potsdam, Germany. (2009)

[4] Belhajjame, K., Vargas-Solar, G., Collet, C.: A flexible workflow model for process-oriented applications. Proceedings of the Second International Conference on Web Information Systems Engineering, DOI: 10.1109/WISE.2001.996468. (2002)

[5] Deelman, E., Gannon, D., Shields, M., and Taylor, I.: Workflows and e-science: An overview of workflow system features and capabilities. Future Generation Computer Systems, 25(5):528–540 (2009)

[6] Ludäscher, B., Altintas I., Berkley C., Higgins, D., Jaeger-Frank, E., Jones, M., Lee, E., Tao, J., Zhao, Y.: Scientific Workflow Management and the Kepler System. Concurrency and Computation: Practice & Experience, 18:1039–1065. (2006)

[7] Oinn, T., Greenwood, M., Addis, M., Alpdemir, M.N., Ferris, J., Glover, K., Goble, C., Goderis, A., Hull, D., Marvin, D., Li, P., Lord, P., Pocock, M.R., Senger, M., Stevens, R., Wipat, A., Wroe, C.: Taverna: lessons in creating a workflow environment for the life sciences. Concurrency and Computation: Practice and Experience. 2005;18:1067–1100.

[8] Taylor, I., Shields, M., Wang, I., Harrison, A.: Visual Grid Workflow in Triana. Journal of Grid Computing, 3:153–169. (2005)

[9] Medeiros, C.B., Perez-Alcazar, J., Digiampietri, L., Pastorello, G.Z., Jr, Santanche, A., Torres, R.S., Madeira, E., Bacarin, E.: WOODSS and the Web: Annotating and Reusing Scientific Workflows. SIGMOD Record, 34:18–23. (2005)

[10] Leopold, H., Mendling, J., and Gunther, O.: What we can learn from quality issues of BPMN models from industry, IEEE Software 33(4), 1-9. (2015)

[11] Hsieh, F. S.: Analysis of a class of controlled Petri net based on structural decomposition, In Proceedings of The 10th IFAC/IFORS/IMACS/IFIP Symposium on Large Scale Systems: Theory and Applications, Osaka. (2004)

[12] Ballarini, P., Djafri, H., Duflot, M., Haddad, S., and Pekergin, N.: Petri nets compositional modeling and verification of Flexible Manufacturing Systems, 2011 IEEE Conference on Automation Science and Engineering (CASE), DOI: 10.1109/CASE.2011.6042488. (2011)

[13] Lara-Rosano, F.: Petri Net Models of Purposeful Complex Dynamic Systems, ISCS 2014: Interdisciplinary Symposium on Complex Systems, Springer International Publishing, ISBN 978-3-319-10759-2, pp. 183–191 (2015)

[14] Zhu, P., Schnieder, E.: Holistic modeling of complex systems with Petri nets, 2000 IEEE International Conference on Systems, Man, and Cybernetics, DOI: 10.1109/ICSMC.2000.884470. (2000)

[15] van der Aalst, W.M.P., Stahl, C., and Westergaard, M.: Strategies for Modeling Complex Processes using Colored Petri Nets, Transactions on Petri Nets and Other Models of Concurrency VII, Springer Berlin Heidelberg, ISBN 978-3-642-38143-0, pp. 6–55. (2013)

[16] Verdi, K. K., Ellis, H. J., and Gryk, M. R.: Conceptual-level workflow modeling of scientific experiments using NMR as a case study. BMC Bioinformatics, 8, 31. (2007)

[17] Liu, F., and Heiner, M.: Colored Petri nets to model and simulate biological systems, In International Workshop on Biological Processes and Petri. (2010)

[18] Papadimitriou, C.: Computational Complexity. Addison-Wesley, Reading, Mass. (1994)

[19] Lima, I.S., Perkusich, A., and de Figueiredo, J.C.A.: An interactive Petri net tool for modelling, analysis and simulation of complex systems, IEEE International Conference on Systems, Man, and Cybernetics, DOI: 10.1109/ICSMC.1996.571156. (1996)

[20] Zhou, M., and Dicesare, F.: Petri Net Synthesis for Discrete Event Control of Manufacturing Systems. Kluwer Academic Publishers, New Jersey, USA. (1993)

[21] Mitchell, M., Newman, M.: Complex systems theory and evolution. Encyclopedia of Evolution, Oxford University Press, New York. (2002)

[22] Rocha, L.M.: BITS: Computer and Communications News. Computing, Information, and Communications Division. Los Alamos National Laboratory. (1999)

[23] Ladyman, J., Lambert, J., and Wiesner, K.: What is a complex system?. European Journal for Philosophy of Science, Vol. 3, No. 1, p. 33-67. (2013)

[24] Kaplan, H.: Persistent data structures. Handbook on Data Structures and Applications. CRC Press. (2001)

[25] van der Aalst, W.M.P., Structural Characterizations of Sound Workflow Nets. Computing Science Reports 96/23, Eindhoven University of Technology (1996)

[26] van der Aalst, W.M.P., Workflow Verification: Finding Control-Flow Errors Using Petri-Net-Based Techniques. In Business Process Management: Models, Techniques, and Empirical Studies, Springer Berlin Heidelberg, ISBN 978-3-540-45594-3, DOI 10.1007/3-540-45594-9-11, pp. 161–183. (2000)

[27] Murata, T.: Petri Nets: Properties, analysis and application. Proc. of the IEEE, 77(4), 541-580. (1989)

[28] Rosenberg, D., and Scott K.:Applying Use Case Driven Object Modeling with UML: An Annotated e-Commerce Example. Addison Wesley, First Edition, ISBN: 0-201-73039-1. (2001)

[29] Rosenberg, D., Stephens, M., Use Case Driven Object Modeling with UML: Theory and Practice. Berkley: Apress. ISBN-10: 1590597745, (2007)

[30] Rosenberg, D., Stephens, M. and Collins-Cope, M.: Agile Development with ICONIX Process. (2005)

[31] Jensen, K.: Coloured Petri Nets (2 ed.). Berlin: Heidelberg. p. 234. ISBN 3-540-60943-1. (1996)

[32] Georgakopoulos, D., Hornick, M., and Sheth, A.: An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure, Distributed Parallel Databases, Vol. 3(2), ISSN 0926-8782, Kluwer Academic Publishers, Hingham, MA, USA, DOI 10.1007/BF01277643, pp. 119–153. (1995)

[33] Oracle®: Oracle® Fusion Middleware Developer's Guide for Oracle SOA Suite 11g, Release 1(11.1.1.5.0), Part Number E10224-09. (2012)

[34] van der Aalst, W.M.P., and ter Hofstede, A.H.M.: YAWL: yet another workflow language, Information Systems, Vol. 30, No. 4, ISSN 0306-4379, DOI 10.1016/j.is.2004.02.002, pp. 245-275. (2005)

[35] Dufourd, C., Finkel, A., and Schnoebelen, Ph.: Reset Nets Between Decidability and Undecidability, In Proceedings of the 25th International Colloquium on Automata, Languages and Programming, Vol. 1443, Lecture Notes in Computer Science, Springer-Verlag, pp. 103-115. (1998)

[36] Dufourd, C., Jancar, P., and Schnoebelen, Ph.: Boundedness of Reset P/T Nets. In Lectures on Concurrency and Petri Nets, Vol. 1644, Lecture Notes in Computer Science, Springer-Verlag, pp. 301-310, Prague. (1999)

[37] Owen, M., and Raj, J.: BPMN and Business Process Management: An Introduction to the New Business Process Modelling Standard, Business Process Trends. (2004)

[38] Omelayenko, B., Ding, Y., Klein, M., Flett, A., Schulten, E., Brown, M., Botquin, G., Dabiri, G.: Intelligent Information Integration In B2B Electronic Commerce. Kluwer Academics Publishers Boston, Dordrecht, London. (2001)

[39] The Workflow Management Coalition: The Workflow Management Coalition Specification: Workflow Management Coalition Workflow Standard, Process Definition Language, Document Number WFMC-TC-1025, Version 2.2. (2012) (http://www.xpdl.org/standards/xpdl-2.2/XPDL%202.2%20(2012-08-30).pdf)

[40] van der Aalst, W.M.P., Netjes, M., and Reijers, H.A.: Supporting the Full BPM Life-Cycle Using Process Mining and Intelligent Redesign, In Contemporary Issues in Database Design and Information Systems Development, IGI Global, Hershey, USA, pp. 100-132. (2007)

[41] Kohler, H.J., Nickel, U., Niere, J., Zundorf, A.: Integrating UML diagrams for production control systems, Proceedings of the 2000 International Conference on Software Engineering, IEEE, ISSN 1-58113-206-9, DOI 10.1145/337180.337207. (2000)

[42] Russell, N., van der Aalst, W.M.P., ter Hofstede, A., and Wohed, P., On the suitability of UML 2.0 activity diagrams for business process modelling. APCCM '06: Proceedings of the 3rd Asia-Pacific conference on Conceptual modelling, pp. 95–104. (2006)

[43] Eshuis, R., and Wieringa, R., Verification support for workflow design with UML activity graphs. Proceedings of the 24rd International Conference on Software Engineering (ICSE 2002), pp. 166–176. (2002)

[44] van der Aalst, W. M. P.: Business Process Management Demystified: A Tutorial on Models, Systems and Standards for Workflow Management. Lectures on Concurrency and Petri Nets: Advances in Petri Nets, Springer Berlin Heidelberg, ISBN 978-3-540-27755-2, DOI 10.1007/978-3-540-27755-2-1. (2004)

[45] van der Aalst, W. M. P.: Business process management: a personal view. Business Process Management Journal, Volume 10, Issue No. 2, Emerald. ISSN 1463-7154. DOI 10.1108/bpmj.2004.15710baa.001. (2004)

[46] Verbeek, H.M.W., van der Aalst, W.M.P., and ter Hofstede, A.H.M.: Verifying Workflows with Cancellation Regions and OR-joins: An Approach Based on Invariants. Technical Report WP 156, Eindhoven University of Technology, The Netherlands. (2006)

[47] Wynn, M.T., van der Aalst, W.M.P., ter Hofstede, A.H.M., and Edmond. D.: Verifying Workflows with Cancellation Regions and OR-joins: An Approach Based on Reset Nets and Reachability Analysis. In International Conference on Business Process Management (BPM 2006), volume 4102 of Lecture Notes in Computer Science, pages 389-394. Springer-Verlag, Berlin. (2006)

[48] Ko, R., Lee, S., and Lee, E.W.: Business process management (BPM) standards: a survey, Business Process Management Journal, Vol. 15 Issue 5, pp.744-791. (2009)

[49] Silver, B.: BPMN Method and Style, with BPMN Implementer's Guide, Cody-Cassidy Press, 2nd Edition, ISBN 978-0982368114. (2011)

[50] Allweyer, T.: BPMN 2.0 - Business Process Model and Notation, Books on Demand GMBH, Norderstedt, 2nd edition. (2009)

[51] White, S., and Miers, D.: BPMN Modeling and Reference Guide: Understanding and Using BPMN, Future Strategies Incorporated. (2008)

[52] Stiehl, V.: Process-Driven Applications with BPMN, Springer International Publishing Switzerland, DOI 10.1007/978-3-319-07218-0. (2014)

[53] Anderson, P. W.: More Is Different: Broken Symmetry and the Nature of the Hierarchical Structure of Sciences. Science. 177: 4047. (1972)

[54] Castellani, E.: Reductionism, emergence, and effective field theories, Studies in History and Philosophy of Science Part B: Studies in History and Philosophy of Modern Physics, Vol. 33, No. 2, ISSN 1355-2198, pp. 251-267. (2002)

[55] van der Aalst, W. M. P., van Hee, K. M., ter Hofstede, A. H. M., Sidorova, N., Verbeek, H. M. W., Voorhoeve, M., and Wynn, M. T.: Soundness of workflow nets: classification, decidability, and analysis, In: Formal Aspects of Computing, Vol. 23, No. 3, pp. 333–363. (2011)

[56] van der Aalst, W.M.P., van Hee, K.M., ter Hofstede, A.H.M., Sidorova, N., Verbeek, H.M.W., Voorhoeve, M., and Wynn, M.T.: Soundness of Workflow Nets with Reset Arcs is Undecidable!, In: Proceedings of the International Workshop on Concurrency Methods Issues and Applications (CHINA'08), pp. 57-72. (2008)

[57] Lyazidi, A., and Mouline, S.: Formal Verification of BPMN Models using Petri Nets, Proc. of the The 1st International Workshop on Models and Algorithms for Reliable and Open Computing. (2013)

[58] Kog, F., Dikbas, A., and Scherer, R.: Verification and validation approach of BPMN represented construction processes, In the Proceedings of Creative Construction Conference. (2014)

[59] Puhlmann, F., and Weske, M.: Investigations on Soundness Regarding Lazy Activities, In: Proceedings of Business Process Management: 4th International Conference (BPM 2006), Springer Berlin Heidelberg, ISBN 978-3-540-38903-3, pp. 145–160. (2006)

[60] Mendoza-Morales, L.: Business Process Verification: The Application of Model Checking and Timed Automata, In: : Computing Conference (CLEI) Electronic Journal, Vol. 17, No. (2). (2014)

[61] Szpyrka, M., Nalepa, G., Ligęza, A., and Kluza, K.: Proposal of Formal Verification of Selected BPMN Models with Alvis Modeling Language, In: Intelligent Distributed Computing V: Proceedings of the 5th International Symposium on Intelligent Distributed Computing, Springer Berlin Heidelberg, ISBN 978-3-642-24013-3, pp. 249–255. (2012)

[62] Recker, J., Rosemann, M., Green, P., and Indulska, M.: Do ontological deficiencies in modelling grammars matter? MIS Quarterly, 35(1):57-79. (2011)

[63] Respicio, A., and Domingos, D.: Reliability of BPMN Business Processes, In Procedia Computer Science 64:623-650. (2015)

[64] Martens, A.: On Usability of Web Services. In: Proceedings of 1st Web Services Quality Workshop, Rome, Italy. (2003)

[65] van der Aalst, W.M.P., van Hee, K.M., ter Hofstede, A.H.M., Sidorova, N., Verbeek, H.M.W., Voorhoeve, M., and Wynn, M.T.: Soundness of Workflow Nets: Classification, Decidability, and Analysis. Formal Aspects of Computing, 23(3):333-363. (2011)

[66] Gotel, O.C.Z., and Finklestein, A.C.W.: An analysis of the requirements traceability problem (PDF). Proceedings of ICRE94, 1st International Conference on Requirements Engineering, Colorado Springs. IEEE CS Press. (1994)

[67] van der Aalst, W.M.P., and van Hee, K.: Workflow Management: Models, Methods, and Systems Information Systems, MIT Press, ISBN 026229690X. (2004)

[68] Rezk, A.: Theoretical and Experimental Investigation of Silica Gel/Water Adsorption Refrigeration Systems. PhD Dissertation, Univ. of Birmingham. (2012)

[69] van der Aalst, W. M. P., ter Hofstede, A. H. M., Kiepuszewski, B., and Barros, A. P.: Workflow Patterns. Distributed and Parallel Databases, 14(3):5-51. (2003)

[70] Wohed, P., van der Aalst, W. M. P., Dumas, M., ter Hofstede, A. H. M., and Russell, N.: On the Suitability of BPMN for Business Process Modelling, Proceedings in the 4th International Conference: Business Process Management (BPM 2006), ISBN 978-3-540-38903-3, pp. 161–176. (2006)

[71] Malinao, J., Judex, F., Selke, T., Zucker, G., Adorna, H., Caro, J., and Kropatsch, W.: Robustness Diagram with Loop and Time Controls For System Modelling and Scenario Extraction with Energy System Applications. Energy Procedia, Vol. 88, pp. 537–543, ISSN 1876-6102. (2016)

[72] Malinao, J., Judex, F., Selke, T., Zucker, G., Caro, J., and Kropatsch, W.: Pattern mining and fault detection via COPtherm-based profiling with correlation analysis of circuit variables in chiller systems, Computer Science – Research and Development, DOI 10.1007/s00450-014-0277-5, ISSN 1865-2034, Springer Berlin Heidelberg. (2015)

[73] Zucker, G., Malinao, J., Habib, U., Leber, T., Preisler, A., and Judex, F.: Improving Energy Efficiency of Buildings Using Data Mining Technologies, 2014 IEEE 23rd International Symposium on Industrial Electronics (ISIE), DOI 10.1109/ISIE.2014.6865041, pp. 2664-2669. (2014)

[74] Malinao, J., Tiu, K., Lozano, L.M., Pascua, S., Chua, R.B., Magboo, Ma. S., and Caro, J.: A Metric for User Requirements Traceability in Sequence, Class Diagrams, and Lines-of-Code via Robustness Diagrams. Proceedings on Information and Communications Technology, Vol. 7, pp. 50-63, Springer Japan 2013, ISBN 978-4-431-54435-7. (2013)

[75] Fujitsu Ten Limited, CATT Ver. 4.8.0: Reference Manual, 2000-2012 (2012).

[76] Nicolis, G., and Nicolis, C., Foundations of Complex Systems: Nonlinear Dynamics, Statistical Physics, Information and Prediction. World Scientific, Singapore (2007).

[77] Sun, J.Q. and Luo, A.C.J., Bifurcation and Chaos in Complex Systems. Edited series on advances in nonlinear science and complexity, ISBN 9780444522290, Elsevier. (2006)

[78] David, R., Alla, H., Continuous Petri Nets. Proceedings of the 8th European Workshop on Application and Theory of Petri Nets. (1987)

[79] Fraca, E., and Haddad, S., Complexity Analysis of Continuous Petri Nets. Proceedings: Application and Theory of Petri Nets and Concurrency: 34th International Conference, PETRI NETS 2013, Springer Berlin Heidelberg, ISBN 978-3-642-38697-8, DOI 10.1007/978-3-642-38697-8-10, pp. 170–189. (2013)

[80] Ambler, S.: The Object Primer 3rd Edition: Agile Model Driven Development with UML 2, Cambridge University Press, ISBN: 0-521-54018-6(2004).

[81] Zerhouni, N., Alla, H.: Dynamic Analysis of Manufacturing Systems using Continuous Petri Nets. Proceedings of the IEEE International Conference on Robotics and Automation. (1990)

[82] Holzmann, G.J., The SPIN Model Checker: Primer and Reference Manual, Addison-Wesley. (2004)

[83] Ahmadon, MAB, and Yamaguchi, S.: On State Number Calculation in Petri Nets. Proceedings: Second International Symposium on Computing and Networking, IEEE, Electronic ISBN: 978-1-4799-4152-0, DOI: 10.1109/CANDAR.2014.114, (2014)

[84] Desel, J., Esparza, J.: Free Choice Petri nets. Cambridge tracts in theoretical computer science., Vol. 40, Cambridge University Press (1995)

[85] Cheng, A., Esparza, J., and Palsberg, J.: Complexity results for 1-safe nets. In: Foundations of software technology and theoretical computer science, Lecture Notes in Computer Science, Springer-Verlag, Berlin, Vol. 761 (326-337). (1993)

[86] F.G. Cabarle, H.N. Adorna: On Structures and Behaviors of Spiking Neural P Systems and Petri Nets. Proceedings in the International Conference on Membrane Computing 2012: 145-160. (2012)

[87] van der Aalst, W.M.P.: The Application of Petri Nets to Workflow Management. Journal of Circuits, Systems and Computers. vol. 8(1), 21-66. (1998)