

**AUTOMATED SCHEDULE GENERATION AND  
ANALYSIS FROM A CONSTRUCTION REQUIREMENT  
PERSPECTIVE**

**NGUYEN THI QUI**

*(B.Eng. (Hons.), M.Eng,*

*Hochiminh City University of Technology)*

**A THESIS SUBMITTED**

**FOR THE DEGREE OF DOCTOR OF PHILOSOPHY**

**DEPARTMENT OF CIVIL AND  
ENVIRONMENTAL ENGINEERING**

**NATIONAL UNIVERSITY OF SINGAPORE**

**2013**



# **DECLARATION**

**I hereby declare that the thesis is my original work**

**and it has been written by me in its entirety.**

**I have duly acknowledged all the sources of information**

**which have been used in the thesis.**

**This thesis has also not been submitted for any degree**

**in any university previously.**



---

**NGUYEN THI QUI**

**7 August 2013**

*This page is intentionally left blank.*

## ACKNOWLEDGEMENTS

It would not have been possible to write this thesis without the help of numerous people, and only some of whom it is possible to give particular mention here.

First and foremost, I would like to express my deepest thanks to my parents, my parents-in-law and especially my husband, who are always by my side to rigidly support me over the past five years. Without their understanding and encouragement, I would not have sustained through the stress and frustration of the research work.

I also would like to express my utmost gratitude to my supervisor, *Prof David Chua*, for his wisdom, knowledge and enthusiasm in guiding me so that I can complete my thesis. His instruction and constant motivating power have helped me to overcome the difficulties and challenges throughout the research. To me, he is not only a great supervisor but also a great teacher who has helped me in various ways, both academically and professionally.

I wish to thank many industry people, especially *Ms Zhao Yu* from House Development Board (HDB), who has helped me to gather practical knowledge on construction planning and scheduling, and provided me with valuable information for the case studies.

I would like to extend my warmest thanks to all my lab mates in NUS, namely *Wang Yueying, Shen Lijun, Liu Zhuo, Md. Aslam Hossain, Zhu Lei, Alireza Khalili*, and *Meghdad Attarzadeh* for their helps and companionship while performing this research. Also, a special note of appreciation is directed to *Mr Ernest L. S. Abbott* for teaching me my first programming lessons and proof-reading some parts of the thesis. Let me also accord my wholehearted thanks to my senior fellow, *Dr Yeoh Ker-Wei*, for his generous helps, valuable discussions, and constructive comments for my research. Lastly, I wish to thanks my best friends, *Hung* and *Duong*, for their friendship, encouragement and sharing throughout my PhD life.

My sincere appreciation also goes to members of my PhD committee, *Prof Chan Weng Tat* and *Prof Meng Qiang*, for their valuable comments during the qualification examination.

I would like to acknowledge National University of Singapore (NUS) for the award of this research scholarship throughout the four years period.

I dedicate this thesis to my beloved husband, *Duong Khanh*, for his unconditional love and unwavering support.

*This page is intentionally left blank.*

# TABLE OF CONTENTS

<b>SUMMARY .....</b>	<b>x</b>
<b>LIST OF TABLES .....</b>	<b>xiii</b>
<b>LIST OF FIGURES .....</b>	<b>xiv</b>
<b>NOMENCLATURE.....</b>	<b>xvi</b>
<b>CHAPTER 1. INTRODUCTION .....</b>	<b>1</b>
1.1. Research Motivations and Background.....	1
1.2. Research Opportunities .....	4
1.2.1. Incorporation of Construction Knowledge in Scheduling Systems .....	4
1.2.2. Automated Sequence Reasoning from Functional Requirements .....	6
1.2.3. Constraint Analysis to Improve Feasibility and Efficiency of Alternative Scheduling Approaches .....	7
1.2.4. Criticality analysis for schedule management.....	9
1.3. Research Objectives .....	10
1.4. Research Scopes .....	12
1.5. Research Methodology.....	13
1.6. Organization of Thesis .....	14
<b>CHAPTER 2. LITERATURE REVIEW .....</b>	<b>18</b>
2.1. Introduction .....	18
2.2. Construction Requirements in Schedules.....	18
2.2.1. Classification of Construction Requirements .....	19
2.2.2. Formalization of Construction Requirements .....	21
2.2.3. Modeling functional requirements for automated sequencing .....	23
2.2.4. Integrating Construction Requirements for Scheduling .....	24
2.3. Advancements of Planning and Scheduling Approaches.....	25
2.3.1. CPM/PDM: Overview and Limitations for Construction Scheduling .....	26
2.3.2. Model-based Planning and Scheduling .....	27
2.3.3. Knowledge-Based Planning Systems .....	29
2.3.4. Construction Planning using Case-based Reasoning .....	32
2.3.5. Advanced Scheduling Techniques .....	34
2.3.6. PDM++ Modeling Framework .....	35
2.4. Criticality Analysis in Construction Schedules.....	38
2.4.1. Criticality Analysis from Activity Perspective.....	38

2.4.2. Criticality Analysis from Constraint Perspective .....	40
2.5. Identified Research Gaps .....	42
2.6. Summary .....	43
<b>CHAPTER 3. GENERALIZED FRAMEWORK FOR AUTOMATED SCHEDULING FROM CONSTRUCTION METHODS AND REQUIREMENTS .....</b>	<b>45</b>
3.1. Introduction .....	45
3.2. Core Knowledge Models for Automated Scheduling.....	45
3.2.1. Extended Product Model .....	46
3.2.2. Construction Method Model .....	49
3.2.3. Construction Requirement Model.....	51
3.2.4. Construction Schedule Model.....	56
3.2.5. Schedule Data Integration Framework .....	58
3.3. Generalized Framework for Automated Scheduling from Construction Requirement (ASCoRe).....	62
3.3.1. Process P: Generating Extended Product Hierarchy.....	63
3.3.2. Process R: Identifying Construction Requirements for Scheduling .....	68
3.3.3. Process S: Generating Schedule Model .....	70
3.3.4. Process A: Computing for Alternative Schedules .....	78
3.4. The scheduling problem is formulated as a constraint satisfaction problem (CSP) and constraint logic programming (CLP) is used for schedule computation. This method is selected so that a complete solution (all alternative schedules) of scheduling problems can be obtained. The outcome generated can be either types: First, if the constraint set are still inconsistent, no result is obtained. Second, when there is no conflict in the constraint set, a collection of all alternative schedules with minimal makespan are returned as output. These schedules represent alternative construction sequences leading to similar project completion time. Concluding Remarks .....	79
<b>CHAPTER 4. AUTOMATED CONSTRUCTION SEQUENCING FROM FUNCTIONAL REQUIREMENTS .....</b>	<b>81</b>
4.1. Introduction .....	81
4.2. Modeling Perspectives of a Functional Requirement.....	82
4.3. Representing Complex Functional Requirements .....	84
4.3.1. Function User.....	85
4.3.2. Function Provider .....	86
4.3.3. Function Type .....	87
4.3.4. Provider Co-Functionality .....	87
4.3.5. Illustrative Example .....	88



4.4. Modeling Temporal Attributes of a Functional Requirement.....	89
4.4.1. Temporal Attributes of User and Provider .....	90
4.4.2. Temporal attributes of Function User and Function Provider.....	92
4.5. Sequence Reasoning Framework from Functional Requirement.....	96
4.5.1. Necessary Condition at Requirement Level .....	97
4.5.2. Necessary Conditions at Function Level.....	98
4.5.3. Necessary Conditions at User/Provider Level.....	98
4.6. Implementation of the FReMAS model .....	100
4.7. Case Study.....	102
4.7.1. Product Components and State Chains.....	102
4.7.2. Formalizing Construction Requirements.....	105
4.7.3. Construction Sequence Reasoning and Schedule Computation.....	107
4.8. Concluding Remarks .....	114
<b>CHAPTER 5. ASCoRe SCHEDULER: SYSTEM ARCHITECTURE AND</b>	
<b>SEQUENCE REASONING ALGORITHMS .....</b>	<b>116</b>
5.1. Introduction .....	116
5.2. Relevant Background .....	116
5.2.1. Overview of Constraint Satisfaction Problem.....	117
5.2.2. Overview of Constraint Logic Programming .....	118
5.2.3. Constraint Analysis in CP/CLP-based Schedulers .....	119
5.3. Overview of System Architectural Framework for Implementing ASCoRe	
Framework .....	121
5.4. Construction Knowledge Modeling Module.....	123
5.4.1. Product Component Hierarchy Template .....	124
5.4.2. Construction Method Templates .....	125
5.4.3. Construction Requirement Templates .....	126
5.4.4. Work Packaging Template .....	129
5.5. Inference and Sequence Reasoning Kernel .....	130
5.5.1. Activity Generation Mechanism.....	132
5.5.2. Functional Requirement Sequence Reasoning Mechanism .....	134
5.5.3. Constraint Transformation Mechanism.....	134
5.5.4. Key Resource Requirement Sequence Reasoning Mechanism.....	138
5.5.5. Work Space Requirement Sequence Reasoning Mechanism .....	141
5.6. Preemptive Constraint Analyzer .....	142
5.6.1. Definition and Classification of Constraint Redundancies and Conflicts. 143	

5.6.2. Pre-emptive Constraint Analysis Framework.....	147
5.6.3. Identifying Feasible Duration Range.....	156
5.6.4. Preemptive Constraint Analyzer.....	159
5.7. Schedule Generator.....	161
5.8. Concluding Remarks.....	164
<b>CHAPTER 6. CRITICALITY ANALYSIS OF CONSTRUCTION REQUIREMENTS FOR SCHEDULE CHANGE MANAGEMENT.....</b>	<b>166</b>
6.1. Introduction.....	166
6.2. Constraint Criticality.....	167
6.2.1. Definition and Classification.....	167
6.2.2. Order of Constraint Criticality.....	171
6.3. Identifying Constraint Criticality.....	172
6.3.1. Determining Constraint Relaxation Times.....	173
6.3.2. Interpreting Constraint Relaxation Times.....	177
6.4. Criticality of Construction Requirements.....	179
6.5. Schedule Change Analysis from the Perspective of Construction Requirements.....	184
6.5.1. Schedule Makespan Change by Variations of Relaxation Times.....	184
6.5.2. Change in Schedule Makespan through Adding/Removing a Constraint.....	191
6.6. Illustrative Example.....	192
6.7. Concluding Remarks.....	196
<b>CHAPTER 7. CASE STUDIES.....</b>	<b>198</b>
7.1. Introduction.....	198
7.2. Case Study 1: Schedule Generation and Analysis of a Covered Walkway Project.....	198
7.2.1. Product Hierarchy and Component State Chain.....	199
7.2.2. Construction Requirements and Constraint Network.....	200
7.2.3. Schedule Generation.....	202
7.2.4. Criticality Analysis.....	203
7.2.5. Analyzing Schedule Changes.....	209
7.3. Case Study 2: Application of the Preemptive Constraint Analysis Framework to a Pipeline Installation Project.....	211
7.3.1. Construction Requirements.....	211
7.3.2. Preemptive Constraint Analysis and Schedule Generation.....	214
7.4. Concluding Remarks.....	218

<b>CHAPTER 8. CONCLUSION AND RECOMMENDATIONS .....</b>	<b>220</b>
8.1. Introduction .....	220
8.2. Conclusion and Research Contributions .....	221
8.2.1. Generalized Framework for Automated Scheduling from Construction Requirement (ASCoRe).....	221
8.2.2. Functional Requirement Model for Automated Sequencing (FReMAS)..	222
8.2.3. Preemptive Constraint Analysis Framework.....	224
8.2.4. Criticality Concept and Schedule Change Analysis Methodology from the Perspective of Constraints and Construction Requirements .....	226
8.3. Limitations .....	228
8.3.1. Incorporating Practice Considerations into Automated Scheduling .....	228
8.3.2. Modeling and Reasoning Nonstandard Complex Functional Requirements .....	229
8.3.3. Analyzing Non-Temporal Constraints in the Pre-Scheduling Stage.....	229
8.4. Recommendations for Future Work.....	230
8.4.1. Time-cost Tradeoff Using ASCoRe .....	230
8.4.2. Using Constraint Criticality and Alternative Schedules for Dynamic Schedule Management.....	230
8.4.3. Prototyping a BIM-based System for Automated Project Planning and Dynamic Control .....	231
<b>REFERENCES.....</b>	<b>233</b>
<b>APPENDIX.....</b>	<b>240</b>
<b>CURRICULUM VITAE.....</b>	<b>243</b>

## SUMMARY

In the construction industry, feasible schedules are crucial for good project performance since they provide an appropriate basis for project execution and cooperation among different project parties. Construction knowledge which is often abstracted in the form of construction methods and requirements is the key element for generating and controlling schedules. Therefore, sufficient incorporation of construction methods and requirements into schedule generation and management is decisive to improve the feasibility of construction schedules. Moreover, scheduling is generally an intricate process and demands highly experienced personnel. Especially, in today's construction industry where construction projects are more complex, manual scheduling is found to be inefficient and inadequate. Accordingly, automated scheduling has become a dominant approach to improve the efficiency and adequacy of this process.

The main purpose of this research is to develop necessary methodologies and concepts for automated schedule generation and analysis from the perspective of construction requirements to improve the efficiency and feasibility of construction schedules. For this purpose, this dissertation proposes an overarching framework to integrate, interpret, and analyze construction requirements for schedule auto-generation and change management.

The outline of the overarching framework follows the structure of this dissertation. It includes a generalized framework for automated generation of alternative schedules from construction methods and requirements. This scheduling

framework is built upon four core knowledge models, which allow the explicit representation and integration of construction requirements and multiple methods for automated construction sequence reasoning and scheduling. Moreover, it involves four scheduling procedures, which generalize the process of automated BIM-based scheduling from construction requirements. With the incorporation of sequencing knowledge for different types of construction requirements, namely functional requirements, key resource requirements, spatial constraints and temporal constraints, the proposed scheduling framework enhances the capability and efficiency of current BIM-based schedulers, and can be applied to different project types.

A generalized functional requirement model for automated construction sequencing (FReMAS) is then developed to provide the necessary modeling tools and sequence reasoning knowledge to formalize and convert complex functional requirements into temporal constraints. This is achieved through a representation format to capture and a reasoning procedure to transform complex functional requirements into temporal constraints. By this, this model can support the integration of product and process perspectives of scheduling and facilitate the adequate identification of multiple construction sequences implicitly defined by functional requirements, so that all possible alternative schedules can be determined.

To further improve the efficiency and feasibility of scheduling from complex requirements, a preemptive constraint analysis framework which aims to identify basic constraints redundancies/inconsistencies prior to performing scheduling is developed. This framework provides planners with a deeper insight into the role of constraints for the feasibility of the schedule, and thus appropriate resolution strategies can be applied earlier in the pre-scheduling stage.

Finally, for better schedule management, an innovative concept for criticality analysis in construction schedule, which is based on the criticality of constraints and construction requirements with the consideration of multiple alternative schedules, is developed. In particular, this research presents an extended classification and a systematic approach for identifying the criticality of schedule constraints and construction requirements. This approach advocates a new concept for schedule management which is set from the perspective of constraint variation and criticality.

Keywords: Construction Scheduling; Construction Requirements; Schedule Change Analysis; Constraint Criticality; Alternative Schedules

## LIST OF TABLES

Table 4.1. Schedule solutions under Scenario 1 .....	109
Table 4.2. Schedule solutions under Scenario 2 .....	111
Table 5.1. Rules for converting constraints from quiescent states to active states .....	135
Table 5.2. Constraint relationships in according with $\Psi$ and $f$ .....	152
Table 6.1. Relationship of criticality and relaxation times .....	173
Table 6.2. Criticality of complex and simple construction requirements .....	182
Table 6.3. Changes of lag and activities' time leading to constraint tightening .....	185
Table 6.4. Impact of variation of project-critical constraint on schedule makespan ..	186
Table 6.5. Result from criticality analysis .....	192
Table 7.1. Alternative schedules .....	203
Table 7.2. Criticality of simple constraints in four alternative schedules.....	204
Table 7.3. Temporal constraints constituting the imposed requirements .....	213
Table 7.4. Conflicting constraints .....	215
Table 7.5. Redundant constraints.....	216
Table A.1. Pairwise integration of unary constraints.....	240
Table A.2. Pairwise integration of non-lag type binary constraints .....	241
Table A.3. Pairwise Integration of Lag Type and Non-lag Type Binary Constraints	241
Table A.4. Pairwise Integration of Lag Type Binary Constraints .....	242

# LIST OF FIGURES

Figure 1.1. Flow chart of research methodology .....	13
Figure 1.2. Organization of the thesis .....	15
Figure 2.1. PDM++ Temporal Relationships from Chua and Yeoh (2011).....	36
Figure 3.1. Extended product model .....	47
Figure 3.2. Examples of construction method.....	51
Figure 3.3. Classification of construction requirements .....	52
Figure 3.4. Temporal relationships based on PDM++ model .....	54
Figure 3.5. Topological relationships.....	55
Figure 3.6. Comparative relationships .....	55
Figure 3.7. Examples of construction requirement .....	56
Figure 3.8. Example of schedule model .....	57
Figure 3.9. Integrated construction information framework .....	59
Figure 3.10. IDEF $\emptyset$ representation of the ASCoRe framework.....	62
Figure 3.11. Procedure for creating an extended product hierarchy .....	64
Figure 3.12. Example rule for generating final functional requirements .....	68
Figure 3.13. Generic non-functional requirement.....	70
Figure 3.14. Conversion from component state chain to elementary activities .....	71
Figure 3.15. Three detail levels of a typical activity hierarchy.....	73
Figure 3.16. Approach for generating temporal constraints.....	74
Figure 3.17. Convert requirements from component state to elementary activities .....	75
Figure 3.18. Convert requirements from elementary activity to activity levels.....	76
Figure 3.19. Convert temporal constraints from meta-activity to activity level .....	77
Figure 4.1. Core entities representing a functional requirement .....	85
Figure 4.2. Example component state chains and functional requirements.....	88
Figure 4.3. Time windows of individual User and Provider.....	90
Figure 4.4. Time windows of function user and function provider.....	92
Figure 4.5. Flowchart for implementing FReMAS .....	100
Figure 4.6. <i>ECLiPS<sup>e</sup></i> code for implementing FReMAS for automated scheduling .....	101
Figure 4.7. 3D model of nursing house showing main entrance.....	102
Figure 4.8. State Chains with Durations of Components Involving in the Analysis ..	103
Figure 4.9. Alternative 1.1 - Scenario 1 with <i>RTWs</i> and <i>ATWs</i> .....	108
Figure 4.10. Early Schedule of Alternative 2.1 - Scenario 2 with <i>RTWs</i> and <i>ATWs</i> ..	112



Figure 5.1. ASCoRe system architectural framework .....	122
Figure 5.2. Template of product component hierarchy.....	125
Figure 5.3. Construction method template.....	126
Figure 5.4. Template for defining functional requirement .....	127
Figure 5.5. Template for defining non-functional requirement .....	128
Figure 5.6. Template for defining work package.....	130
Figure 5.7. Workflow of the inference and sequence reasoning kernel.....	131
Figure 5.8. Pseudo code for the activity generation mechanism .....	132
Figure 5.9. Illustrative example for activity generation mechanism .....	133
Figure 5.10. Converting quiescent state constraint to active state constraint .....	136
Figure 5.11. Converting state-based constraints to activity-based constraints .....	137
Figure 5.12. Pseudo code spatial requirement sequence reasoning mechanism.....	142
Figure 5.13. Example redundant and inconsistent constraints.....	144
Figure 5.14. Examples redundancy and inconsistency rules for simple constraints...	150
Figure 5.15. Examples of redundancy rules of conjunction constraints .....	154
Figure 5.16. Flowchart for implementing preemptive constraint analysis .....	160
Figure 5.17. Flowchart of scheduling process .....	162
Figure 6.1. Example schedule network.....	168
Figure 6.2. Example of sequence-critical constraint.....	171
Figure 6.3. Example constraint network showing relaxation times.....	178
Figure 6.4. Example schedule for analyzing schedule change .....	187
Figure 6.5. Schedule change analysis under constraint relaxation .....	189
Figure 6.6. Illustrative example for criticality analysis .....	192
Figure 6.7. Alternative schedules demonstrating constraint criticality.....	194
Figure 6.8. Best alternative schedule when constraint $c_2$ is removed .....	195
Figure 7.1. 3D perspectives of the covered walkway structure .....	199
Figure 7.2. Product hierarchy and component state chain .....	199
Figure 7.3. Schedule network of covered walkway project.....	202
Figure 7.4. Alternative schedules indicating critical constraints .....	205
Figure 7.5. Pipeline installation layout .....	211
Figure 7.6. Constraint network of pipeline installation project .....	213
Figure 7.7. Alternative schedules.....	217

## NOMENCLATURE

<i>Symbol</i>	<i>Description</i>
$F$	An arbitrary functional requirement $F$
$T_F$	Function type of functional requirement $F$
$C_F$	Provider co-functionality type of functional requirement $F$
$U_F$	Function user of functional requirement $F$
$u_{F,i}$	A user $i$ of functional requirement $F$
$P_F$	Function provider of functional requirement $F$
$p_{F,j}$	A provider $j$ of functional requirement $F$
$MP_{F,m}$	Meta-provider $m$ of functional requirement $F$
$RTW_{F,i}^U$	Requirement Time Window of user $i$ of functional requirement $F$
$ATW_{F,j}^P$	Availability Time Window of provider $j$ of functional requirement $F$
$FTW_{F,j,k}$	Function Time Window of component $k$ constituting provider $j$ of functional requirement $F$
$RTW_F$	Requirement Time Window of functional requirement $F$
$ATW_F$	Availability Time Window of functional requirement $F$
$ATW_F^E$	Availability Time Window of functional requirement $F$ of type $E$
$ATW_F^C$	Availability Time Window of functional requirement $F$ of type $C$
$ATW_{F,m}^M$	Availability Time Window of meta-provider $m$ of functional requirement $F$

$FD_X$	Feasible Duration Range of Activity $X$
$FD_X^L$	Lower Bound of Feasible Duration Range of Activity $X$
$FD_X^U$	Upper Bound of Feasible Duration Range of Activity $X$
$ART_C$	Aggregate Relaxation Time of constraint $c$
$IRT_C$	Intrinsic Relaxation Time of constraint $c$
$AFT_{k,c}^{FW}$	Forward Aggregation Flexibility Time of activity $k$ w.s.t constraint $c$
$AFT_{k,c}^{BW}$	Backward Aggregation Flexibility Time of activity $k$ w.s.t constraint $c$
$IFT_{k,c}^{FW}$	Forward Intrinsic Flexibility Time of activity $k$ w.s.t constraint $c$
$IFT_{k,c}^{BW}$	Backward Intrinsic Flexibility Time of activity $k$ w.s.t constraint $c$
$C_i$	Criticality of a constraint/construction requirement $i$
$\Delta T_i$	Tightening/Relaxation Amount in Time constraint $i$
$\Delta T_P$	Schedule Makespan Increase/Decrease Amount in Time
$\Omega_i$	Controlling Constraint Set when Constraint $i$ is relaxed/removed
$\mathcal{P}(A)$	Power set of the set $A$

*This page is intentionally left blank.*

# CHAPTER 1. INTRODUCTION

## 1.1. Research Motivations and Background

A construction schedule is an important tool for project planning and control. It provides a basis for project execution and cooperation among the project parties. Improper or infeasible schedules have been found to be a crucial factor in causing project delays, over-budgeted cost and unsatisfied quality (Chan and Kumaraswamy, 1997; Chua et al., 1999). Therefore, proper generation and management feasible pre-construction schedules are key prerequisites for the success of construction projects.

Scheduling involves the integration and interpretation of construction knowledge and building data to determine construction activities required to create the facility and the sequence among them. It is thus an intricate process and demands highly experienced personnel. In today's construction industry where construction projects are more complex, manual scheduling is found to be inefficient and inadequate to incorporate multiple construction methods and to produce alternative schedules (Mikulakova et al., 2010). Improving the efficiency and adequacy of schedules is thereby a current need in the construction industry.

Recently, Building Information Modeling (BIM) has become a centerpiece for Architecture, Engineering and Construction (AEC) technology. BIM concept has been applied to different areas of project management such as constructability analysis, collaboration or visualization. For schedule generation, BIM enhances model-based scheduling techniques by enabling rapid integration of product and process information. Beyond that, in many scheduling systems, preliminary schedules can be

automatically generated through the incorporation of BIM models and construction knowledge to reason about the construction sequence of building components from their topological relationships. However, since construction schedules are governed by various factors such as construction methods, contractors' experiences, site conditions, as well as project's specific characteristics, schedules generated from only topological relationships are possibly inadequate and infeasible for implementation. Accordingly, there is a need for improved scheduling approaches which make good use of major construction knowledge so that more reliable schedules can be obtained.

Construction knowledge can be abstracted as construction requirements, which are the key prerequisites for construction processes (Yeoh, 2012). Similar to function analysis in software engineering, in the Architecture, Engineering, and Construction (AEC) community, construction requirements can be modeled as functional and non-functional (Song and Chua, 2006). Functional requirements represent functional dependencies among components in both construction and completion stages. In other words, they impose constraints on the functionality behavior of product components. On the other hand, non-functional requirements are related to other construction aspects, such as temporal constraints between construction processes, availability of key resource/work space, or constraints on measurable features of product components.

Functional requirements often arise from alternative choices of construction technology, collaboration scenarios or engineering solutions for the project. Hence, they could imply multiple alternative construction sequences which are represented by complex disjunctive combinations of temporal constraints between construction processes. However, such alternative construction sequences defined by functional

requirements are often inadequately determined. Reasons for this inadequacy include the lack of available tools to represent complex functional requirements, as well as the lack of reasoning mechanism to identify and capture all alternative construction sequences resulting from functional requirements.

For good project management, appropriate schedule analysis should be carried out early in the pre-construction stage. A schedule is controlled by its constraints which are derived from construction requirements. Therefore, constraints and construction requirements should be analyzed directly to identify the critical ones driving the entire schedule (Chua and Shen, 2005). Specifically, the emphasis of identifying criticality from the perspective of activities for better project management should be changed to studying and classifying the criticality from the perspective of constraints and requirements. Furthermore, criticality analysis of construction requirements should take into account the existence of multiple alternative schedules. Despite this, analyzing the criticality of constraints and construction requirements for schedule management has not been well addressed by the research community due to the lack of a systematic approach.

In summary, construction knowledge involves primary factors determining the feasibility of construction schedules through both planning and management stages. Therefore, crucial construction knowledge should be sufficiently considered in schedule generation and management processes. Moreover, the efficiency of scheduling depends greatly on scheduling techniques, and automation in construction scheduling is thereby a necessity in current practice. In this regard, this dissertation proposes an overarching framework to integrate, interpret, and analyze construction knowledge for schedule auto-generation and change management.

## **1.2. Research Opportunities**

This section describes the gaps and research opportunities with regard to approaches for schedule generation and analysis in the AEC research community.

### **1.2.1. Incorporation of Construction Knowledge in Scheduling Systems**

Construction knowledge can be described as construction requirements, which are the capabilities and conditions to which the construction processes and the in-progress facility product must conform. If not, the construction processes may be delayed or temporary stability of the in-progress structure may not be sustained during construction (Song and Chua, 2006). Construction requirements generally have different natures. They represent a wide range of project constraints including technical constraints, design requirements, resource/space requirements, budget limits, safety regulations, precedence constraints among project parts, contractual milestones, and so on. However, in most scheduling systems they are often represented in a derived form as precedence relationships among activities. Such an unambiguous representation may not enable good traceability of changes for better project management (Yeoh, 2012).

In order to achieve feasible schedules, major construction requirements should be adequately and explicitly incorporated into scheduling. Despite this, most existing automated scheduling systems focus only on individual elements such as topological relationships, resource constraints, or space constraints (Chua et al., 2013). In particular, a large number of model-based and knowledge-based systems only center on physical relationships among components (Chevallier and Russell, 1998). Similarly, recent BIM-based scheduling systems can enable rapid integration of 3D design



models with commercial scheduler applications (like Microsoft Project or Primavera) for schedule computation and visualization, yet the knowledge embedded in these systems is still restrained to topological relationships and technical constraints (Hartmann et al., 2012).

Construction methods are also the key planning knowledge that need to be considered for scheduling. Since hundreds of methods available as options for a project and new methods are being developed all the time, consideration of multiple methods in the planning stage has become an essential need for planners to attain the best schedules (Kataoka, 2008). Nevertheless, existing scheduling systems do not provide planners with such a function. In current practice, planners with their own knowledge and experience have to manually decide which construction method can be used for the project prior to scheduling, and only one method can be implicitly incorporated into the scheduling process. Consequently, the feasibility of schedules is greatly dependent on planners' knowledge and experience which may not be sufficient and available for new methods. Moreover, while best schedules can probably be obtained by combining different methods for different parts of a project, manually analyzing all method combinations may be impossible for large projects.

In summary, construction methods and requirements are two primary elements of construction knowledge which determine the feasibility of construction schedules. The aforementioned drawbacks raise the need for improved scheduling frameworks which provide modeling mechanisms to adequately integrate and analyze different types of construction requirements as well as multiple construction methods, so that more appropriate and reliable schedules can be achieved.

### **1.2.2. Automated Sequence Reasoning from Functional Requirements**

Functional requirements are among the key sequencing logic of a construction schedule. They refer to the functional dependencies among components in both construction and completion phases, which are respectively referred to as intermediate and final functional requirements in the context of this research. In order to incorporate such requirements into scheduling, they need to be formalized, and converted into temporal constraints between the associated construction processes (Chua et al., 2013).

Researchers and practitioners have developed different methods to automate the sequence reasoning process from functional requirements. However, the major focus of the proposed models is restricted to reasoning from the final functional requirements perspective, and thus is limited to the physical relationships among permanent components. Intermediate functional requirements often involve both permanent and temporary components, and probably represent different engineering solutions for the project. They possibly lead to complex combinations of temporal constraints such as work/resource continuity or process concurrency/overlap/disjunction which may induce multiple construction sequences. In current practice such complex requirements are often treated as technical constraints, and are manually interpreted into precedence constraints (equivalently addressed as simple temporal constraints in this thesis). Consequently, planners may not adequately determine all possible construction sequences that satisfy the requirements and thus could not guarantee to obtain the best sequencing solutions.

Most existing scheduling systems are built on Critical Path Method (CPM) or Precedence Diagram Method (PDM) models, which do not capture complex temporal constraints containing conjunction and disjunction conditions and dictate only one

predefined sequence (El-Bibany, 1997). They therefore cannot represent complex temporal constraints and all possible sequences resulting from complex functional requirements. Especially, they do not provide a mechanism to reason and generate schedules from functional requirements. Recently, a number of advanced scheduling approaches based on Artificial Intelligence (AI) techniques, such as PDM++ (Chua and Yeoh, 2011) or constraint-based scheduling (Lorterapong and Ussavadiokrit, 2013), have been developed to overcome the limitations of CPM/PDM in handling complex temporal constraints. Yet, they still lack the reasoning knowledge for deriving temporal constraints from functional requirements.

In summary, construction sequence reasoning from functional requirements is generally intricate due to their complexity nature and need to be automated for more efficient and sufficient scheduling. This requires a generalized modeling and sequence reasoning framework with knowledge embedded to represent and automatically transform complex functional requirements into temporal constraints for scheduling.

### **1.2.3. Constraint Analysis to Improve Feasibility and Efficiency of Alternative Scheduling Approaches**

A number of advanced scheduling approaches using Constraint Programming (CP) or Constraint Logic Programming (CLP) techniques have been developed to overcome the limitations of CPM/PDM in processing complex temporal constraints. With the ability to handle precedence and disjunctive constraints, these approaches are capable of generating all feasible schedule solutions.

There are two major problems with CSP/CLP schedulers: solution feasibility and computational efficiency, which are greatly influenced by the relationships among the

imposed constraints. Solution feasibility, which refers to the capability of producing a feasible solution, is defined by the consistency of the constraint set. Computational efficiency is governed by the total number of constraints, especially the number of backtrackings which increases exponentially with the number of disjunctive constraints. In other words, conflicting constraints obstruct the scheduling solver to generate a feasible solution, while redundant constraints produce unnecessary search spaces, and decrease the efficiency of the scheduling process. Therefore, redundant and inconsistent constraints should be identified and removed in the pre-scheduling stage to improve scheduling feasibility and efficiency.

Preemptive constraint analysis in scheduling problems has, however, received little research attention. Redundant constraints are often manually identified and completely eliminated from the schedule problem. However, completely deleting redundant constraints could distort the structure of the scheduling problem. Moreover, in many CP/CLP-based schedule solvers, constraint inconsistencies are identified and probably resolved along the scheduling process using constraint propagation, and thus dependent on constraint ordering. From a management perspective, this approach does not guarantee the best (or optimal) set of constraints. In addition, since activity durations and lag times often play a significant role for the relationships among constraints, constraints should be analyzed in accordance with activity durations to provide planners with better management strategies.

In brief, CP/CLP techniques are promising approaches to alternative scheduling in construction. To improve the solution feasibility and computational efficiency of CP/CLP-based schedulers, the constraint set should be preemptively analyzed in the pre-scheduling stage so that basic redundant and conflicting constraints can be

identified and removed. In addition, constraint analysis should be conducted in relation with activity durations and lag requirements to provide planners with more useful information for appropriate resolution strategies.

#### **1.2.4. Criticality analysis for schedule management**

If generating constructible schedules is the necessary condition of good schedule performance, schedule management could be considered as the sufficient condition. Essentially, schedule management has been found to be the most crucial process for schedule performance (Chua et al., 1999; Iyer and Jha, 2006). For good schedule management, it is necessary to identify the crucial parts of the schedule which need to receive more management attention than others, and criticality analysis is thus a crucial schedule management task.

Traditionally, criticality analysis is carried out from the activity perspective. In CPM networks, the criticality of an activity is identified using its float times. However, researchers have indicated that using floats to study criticality of an activity is inadequate in PDM networks due to the existence of non-finish-to-start relationships, and that understanding the nature of constraints associated with it is a necessity (Moder et al., 1983; Valls and Lino, 2001). In addition, since activity criticality is set within the scope of a specific schedule, it could not be applicable to circumstances where multiple alternative schedules exist (Bowers, 2000; Rivera and Duran, 2004).

Theory of constraint (TOC) also advocates the need for identifying the key constraints driving the entire schedule, and the key constraint analysis approach developed by Shen and Chua (2005) is one of the primary research in this area. However, as it is from the production viewpoint, this approach addressed only simple

precedence relationships and enabling constraints (resource and information), and it analyzed only a single schedule.

The feasibility of a schedule is governed by its requirements which possibly result in multiple schedules. It is therefore necessary to identify key requirements, and criticality analysis should be carried out in the relation with multiple alternatives. Furthermore, since requirements represent construction knowledge and practice from which schedule constraints are derived, this research proposes that schedule management should be carried out from the perspective of construction requirements. To achieve this requires novel approaches for analysing the criticality of constraints and construction requirements with regards to multiple alternative schedules, as well as the impact of constraint variations on schedule performance.

### **1.3. Research Objectives**

The primary purpose of this research is to improve the efficiency and feasibility of construction schedules via the adequate incorporation of primary construction knowledge into schedule auto-generation and analysis processes. For this goal, this dissertation will provide the necessary frameworks, concepts and methodologies for formalizing integrating, reasoning and analyzing construction requirements for automated generation and analysis of alternative construction schedules.

In particular, the specific research objectives include:

1. To develop a generalized framework for automated scheduling from alternative construction methods and requirements. This framework will provide modeling tools to represent and integrate primary construction

knowledge for scheduling, and develop generalized procedures for BIM-based scheduling. With these features, the framework will allow the explicit representation and integration of construction requirements and multiple methods into automated construction sequencing and scheduling processes.

2. To develop a generalized model for automated construction sequencing from functional requirements by providing a representation format to capture and a reasoning procedure to transform complex functional requirements into temporal constraints. This model aims to support the integration of product and process perspectives of scheduling and to facilitate the adequate identification of multiple construction sequences which may lead to alternative schedules.
3. To propose a reasoning framework to preemptively identify basic constraint redundancies and inconsistencies in the pre-scheduling stage from a project management perspective. The framework will form the foundation for the development of a preemptive constraint analyzer which aims to improve the solution feasibility and computational efficiency of advanced scheduling approaches. It will also provide planners with a deeper insight into the impact of lag and activity durations upon the relationship between constraints, so that appropriate strategies can be implemented to resolve constraint conflicts.
4. To develop a systematic methodology for analyzing the criticality of constraints and construction requirements in construction schedules in regards to multiple alternative schedules. In particular, this research presents an extended classification and a systematic approach for identifying the criticality of schedule constraints and construction requirements using

constraint criticality indicators. This approach will advocate a new set of indicators for constraint-based schedule management.

#### **1.4. Research Scopes**

The feasibility of construction schedules is probably affected by various requirements and project constraints. This research however focuses only on primary requirements captured in the form of functional requirements, temporal constraints, key resource and work space requirements, which directly govern construction sequence and/or the start/finish of construction operations. Dissatisfaction of such requirements, construction processes cannot be started or the stability of structural systems is not maintained, causing infeasible schedules. Productivity-related requirements such as pool resource availability, inventory, crew's productivity, or site congestion are not explicitly addressed in this research. The proposed frameworks can be extended to incorporate these requirements in future development.

Although topological dependencies among components can often be derived from a 3D model using existing approaches such as Nguyen et al (2005), Khalili and Chua (2012), the deriving process is not presented in this dissertation. Instead, a generalized functional requirement model is developed to represent both topological dependencies and intermediate function requirements, and to derive the temporal constraints among the associated construction processes. Incorporating these approaches into the frameworks proposed in this research will be considered in the future prototype extension.

As a consequence of dynamic construction environment, construction projects often are subjected to a variety of changes originated from different sources. For



schedule management, this research focuses on variations directly affecting the existence and/or temporal attributes of the constraints (lag and process times) which have direct impacts on schedules and project completion time. Changes related to the removal or introduction of activities can be elaborated or transformed into variations of associated constraints and thus are not directly addressed in this research.

### 1.5. Research Methodology

The research methodology is illustrated in the flow chart shown in Figure 1.1. The research methodology consists of three main steps: (a) Developing Research Objectives and Scopes, (b) Generating Research Outputs, and (c) Analyzing and Validating Research Outputs through Illustrative Case Studies.

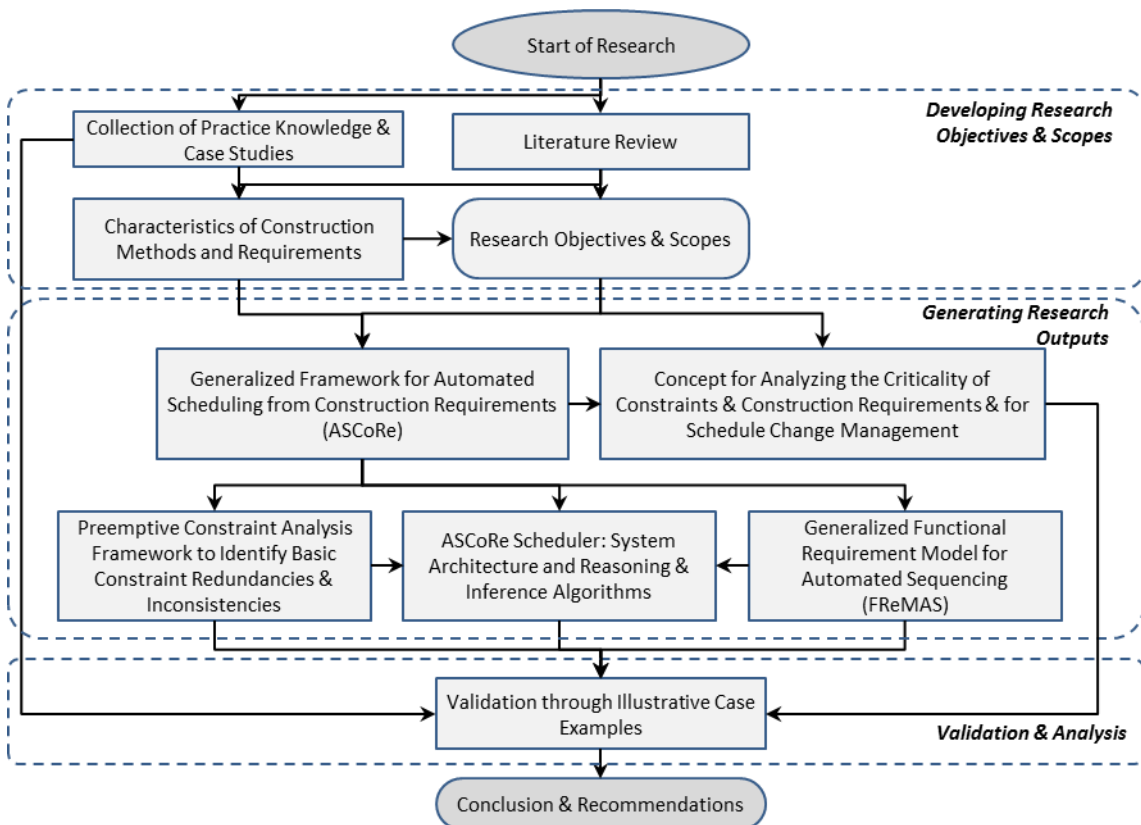


Figure 1.1. Flow chart of research methodology

The Research Objectives and Scopes were iteratively developed through various sources of data. First of all, an extensive study of academic literature has been conducted throughout the project for a deep insight into concepts and techniques of construction schedule generation and analysis. Concurrently, practical construction knowledge and experiences were collected from project reports, construction drawings and schedules, and expert interviews.

Research Outputs were subsequently generated to achieve the defined objectives. In particular, a generalized framework automated scheduling from construction requirements (ASCoRE) was first developed. A functional requirement model (FReMAS) was then developed for automatically reasoning construction sequences from functional requirements. Also, a constraint preemptive analysis framework was developed to identify basic constraint redundancies and inconsistencies in the pre-scheduling stage, so that the solution feasibility and computational efficiency of scheduling can be further enhanced. These models form the input for the system architecture design of the ASCoRE scheduler. Finally, an approach for analyzing the criticality of construction requirements to schedules is developed in order to provide better understanding of the schedule so that good schedule management could be achieved. The methodologies developed were validated with illustrative examples, and finally with industrial case studies.

## **1.6. Organization of Thesis**

This thesis is organized into eight chapters including this introduction as shown in Figure 1.2. Each chapter explicitly illustrates the steps taken to achieve the research objectives. Chapter Two presents a detailed review on research related to concepts and

techniques for schedule generation and management. It contains reviews and existing approaches for identifying and capturing requirements in construction, existing automated scheduling systems and the current research on schedule analysis for change management. From this, major research gaps have been identified, shaping the direction of this research.

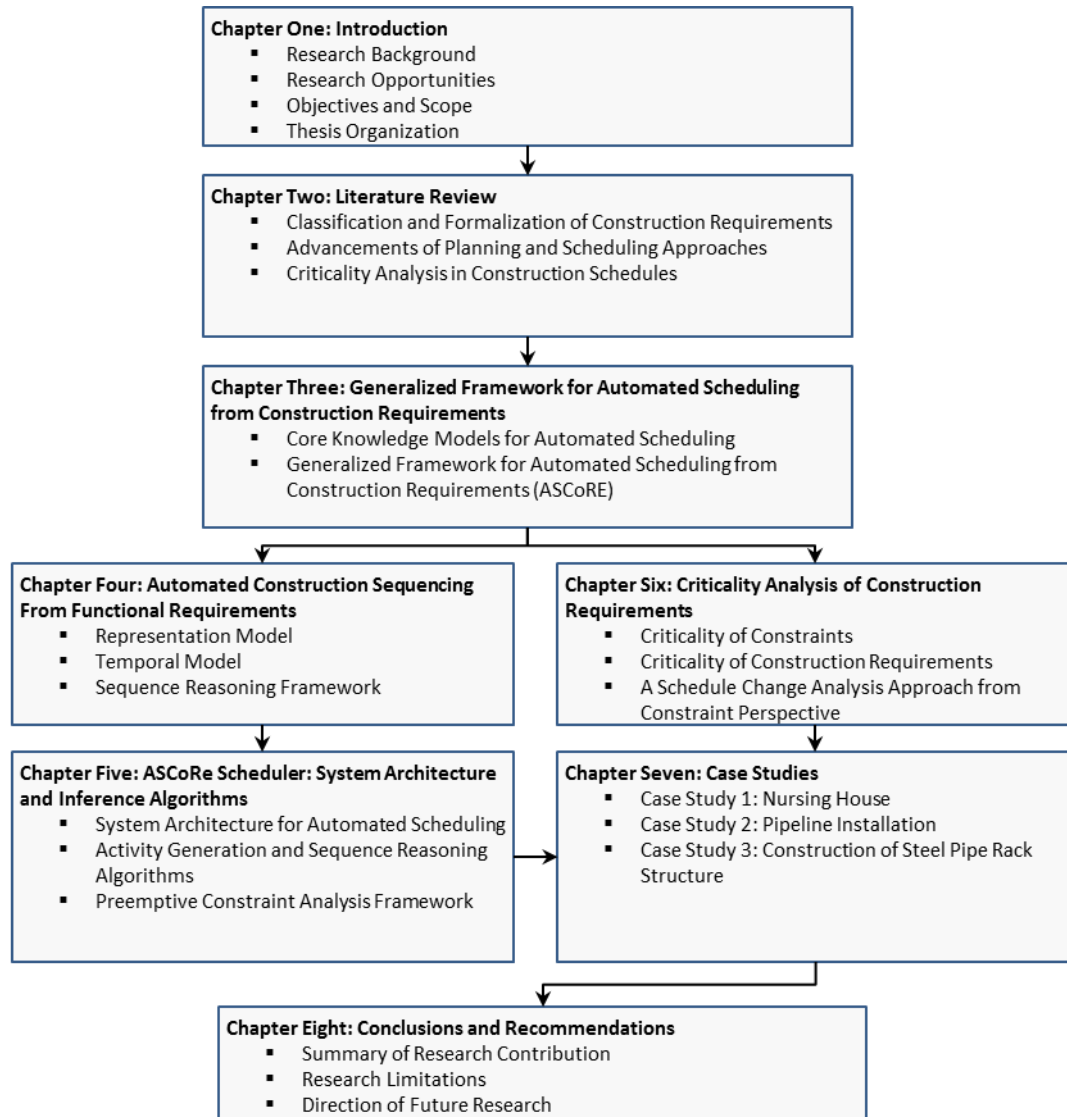


Figure 1.2. Organization of the thesis

Chapter Three presents a generalized framework for automated scheduling from construction requirements (ASCoRe). It is facilitated by four core knowledge models representing construction knowledge and building data necessary for the identification of construction processes and the sequences among them. The ASCoRe framework consists of four main processes to capture, represent and convert major construction requirements into temporal constraints from which alternative schedules are generated. Chapter Four describes a functional requirement model called FReMAS for representing and converting functional requirements into temporal constraints. This modeling framework contains three main subcomponents: (a) a Representation to provide a generalized representation format of functional requirements, (b) a Temporal Model to explicitly define temporal attributes of functional requirements and (c) a Sequence Reasoning Framework to automatically convert functional requirements into temporal constraints from which construction sequences and schedules can be derived.

Chapter Five documents the system architecture and the necessary reasoning and solving algorithms of the ASCoRE scheduler. Especially, a preemptive constraint analyzer is developed to identify redundant and conflicting constraints in the pre-scheduling stage to improve scheduling feasibility and efficiency.

Chapter Six introduces a new concept for analyzing the criticality of construction requirements in the context of multiple alternative schedules. It includes a classification of criticality, a systematic methodology to identify criticality and the application of the proposed concept for schedule management.

Chapter Seven presents three industrial case studies to demonstrate the application of the concepts and methods proposed in Chapters 3, 4, 5, and 6. Each case study is analyzed with management implications presented herein.

Chapter Eight concludes the thesis, summarizing the research contributions derived from this dissertation. Further suggestions for future research and development directions are also presented in this chapter.

## **CHAPTER 2. LITERATURE REVIEW**

### **2.1. Introduction**

This literature review presents the current state of the art with regard to improving the feasibility and efficiency of construction scheduling from two aspects: the sufficient incorporation of major construction requirements into automated schedule generation and the systematic identification of crucial construction requirements for schedule management. For the former aspect, to provide the readers with a fundamental understanding of the origins and natures of construction requirements, the literature presents an overview of classification schemas and methods to formalize construction requirements for scheduling. It then describes the advanced scheduling techniques for sequence reasoning and schedule generation from construction requirements. From this, the major limitations of automated sequence reasoning and scheduling from complex construction requirements of existing auto-planning systems and scheduling techniques are identified. For the latter aspect, the literature describes relevant studies on criticality analysis for schedule evaluation and change management from two main perspectives: activity and constraint, and pinpoint key their key drawbacks for understanding the criticality of construction requirements with the existence of multiple alternative schedules.

### **2.2. Construction Requirements in Schedules**

Construction requirements are the abstract form of construction knowledge and play a key role in determining the feasibility of construction schedules. They arise from various aspects of construction and have different characteristics. In order to

provide readers a background understanding of construction requirements, this section will first summarize the existing classification schemes of construction requirements from different perspectives. It will then present key research on formalizing construction requirements in general and functional requirements in particular. This section will also discuss the existing methods for integrating construction requirements for scheduling. The final part of this section will address the major drawbacks of existing research on modeling and integrating construction requirements for automated sequencing and scheduling which set major directions of this research.

### **2.2.1. Classification of Construction Requirements**

Construction requirements can be classified from two main perspectives: origin and nature (Koo et al., 2007). According to their origin, construction requirements are commonly referred to as sequencing knowledge. Physical building component dependencies were identified as a common sequencing knowledge in early studies, such as Gray (1986), Navinchandra (1988), Zozaya-Gorostiza et al. (1989), and Kartam and Levitt (1990). Echeverry et al. (1991) identified sequencing knowledge with respect to physical relationships among building components, trade interaction, path interference, and code regulations. Sripraset and Dawood (2002) described construction requirements as constraints which can obstruct commencement or progress of construction processes to achieve successful project performance. Accordingly, they classified schedule constraints into three main groups, including physical constraints (technological dependencies, space, safety, and environment), contract (time, cost, quality, and special agreement), and enabler constraints (resource and information). Failure to fulfilling these constraints could lead to infeasible schedule and project delay. Yeoh (2012) introduced a broader definition of

construction requirements. According to his description of the requirement evolution process, construction requirements are the collection of project requirements at all stages. They exist as a form of derived requirements and are the abstraction of the client's intention and design specifications. Construction activities and their corresponding relationships may be derived from the requirements arising from different perspectives, including: topological precedence, intermediate function requirements, space, key resources, safety, contracts, site/environment and logistic/procurement.

For constructability analysis, Song (2006) described construction requirements as the concerns and constraints that should be fulfilled for conducting procurement, construction and logistic processes. According to their nature, construction requirements are classified into two categories: functional and nonfunctional requirements. Functional requirements refer to construction intentions for supporting a construction process. As such, functional requirements are narrowly defined as *intermediate function* requirements, which are temporary functions that are required for supporting the construction of a facility project. On the contrary, nonfunctional requirements refer to performance constraints such as capacity, productivity and inventory. To this extent, construction requirements could be understood as requisites during the construction phase but not those in other project phases. In addition, it can be inferred that functional requirements relate to the engineering behaviors of product components and thus they are product-based. In this context, non-functional requirements can be considered as process-based requirements.

The above literature provides an insight into requirements in construction. In general, construction requirements arise from different project aspects including



clients' intentions, design codes and regulations, and construction technologies and practices. They can be product- or process-related, and sequence- or performance-governing constraints. A key advantage of origin-oriented classification schemas could enable planners to determine the party in charge of a particular requirement for better project management (Yeoh, 2012). However, since the impact of a construction requirement on schedules is governed by its nature, the nature-oriented classification schema is found more suitable for sequence reasoning and scheduling purpose.

### **2.2.2. Formalization of Construction Requirements**

Construction requirements are the rationale driving construction schedules. Therefore, an unambiguous and systematic formalization of construction requirements at the planning stage is crucial for integrating them into schedule generation and management. Yet, this issue has received quite little research attention (Yeoh, 2012). Existing approaches for representing construction requirements mainly aim for constructability analysis or knowledge management. For constructability analysis, some of construction requirements were established in the form of constructability rules. Fischer (1993) developed a rule-based format for representing geometrical and topological design-relevant constructability requirements. Song (2006) proposed a modeling methodology for capturing and analyzing intermediate functional requirements for constructability assessment of construction schedules. This model captures an intermediate function from three perspectives: function user, function provider, and their temporal and spatial relationships. Based on this model, designers and constructors can explicitly describe and incorporate intermediate functional requirements into construction schedules for constructability verification.

For knowledge management, many ontological modeling frameworks have been developed for capturing knowledge in construction as requirements. Domain ontologies (El-Diraby and Kashif, 2005; El-Gohary and El-Diraby, 2010) often involve a set of high-level core ontology with logical rules to allow additional inferences in the specific domains. However, these researches focus mainly on representing knowledge yet not providing knowledge for sequence reasoning from construction requirements. They were still centered on specific requirement types such as physical relationships, precedence activity constraints, or resource requirements, and were centered at specific project types such as highway or infrastructure projects. Consequently, they were found inflexible for different types of construction requirements and projects.

In order to improve the flexibility of the aforementioned domain ontological models, Yeoh (2012) proposed a generalized ontology model for construction requirement by establishing core characteristics and flexible taxonomies. In particular, this model comprises three core characteristics: spatial, temporal, and abstract, and three flexible taxonomies: *Purposive*, *Operational*, and *Necessity Conditions* to allow the representation of a requirement of any type from any construction projects. In brief, any construction requirement can be generally represented by one or all of three constraints: spatial, temporal, and abstract. Such generality allows the model to be applied to any requirement and project type. However, the generality characteristic of this model also results in its major drawbacks for being applied to automated sequencing and scheduling. Firstly, it lacks a syntactical structure to support automated sequencing so that temporal constraints can be automatically generated. Consequently, planners have to manually specify the temporal constraint defined by a requirement if it is not explicitly defined. Secondly, the lack of syntactical structure specific for

common requirement types also obstructs the integration of construction requirements into project data model which is essential for automated sequencing and scheduling. Therefore, for scheduling purpose, this model should be modified to improve the representation and sequence reasoning of common specific requirement types.

From the above literature review, it is found that existing methods for formalizing construction requirements are still restricted for specific types of requirements. The more generalized models on the other hand lack the necessary syntactical structure for automated sequencing. These limitations raise the need for an improved formalization framework which is sufficiently general to capture requirements of different types and efficiently support automated sequence reasoning.

### **2.2.3. Modeling functional requirements for automated sequencing**

Functional requirements are primary elements of construction requirements. They represent the functional dependencies among components in both construction and completion stages, which are respectively referred to as intermediate and final functional requirements in the context of this research. In general, functional requirements may involve both permanent and temporary components. For example, a beam may require the support of a scaffolding-formwork system while it is being constructed, and later it needs to be supported by two columns at its ends after it has been constructed. By this definition, functional requirements encompass the topological precedence constraints in common context and the intermediate functions proposed by Song and Chua (2006).

As discussed in the previous section, functional requirements are product-oriented. For scheduling purposes, they need to be converted into temporal constraints

from which construction sequences are derived (Chua et al., 2013). Researchers and practitioners have developed different methods to automate the sequence reasoning process from functional requirements. Some of these can be found in Shaked and Warszawski (1995), Vries and Harink (2007), and Kataoka (2008). However, the major focus of the proposed models is restricted to reasoning from the final functional requirements perspective, and thus is limited to the physical relationships among permanent components. Intermediate functional requirements are usually treated as technological constraints, and are still manually interpreted into precedence constraints. These requirements often involve both permanent and temporary components, and may result in complex temporal constraints such as work/resource continuity or process concurrency/overlap/disjunction. They could also result in multiple feasible construction sequences which may not be adequately identified using manual techniques. Consequently, a modeling approach allowing for representing and reasoning complex functional requirements is necessary to improve the adequacy and efficiency of automated sequencing and scheduling.

#### **2.2.4. Integrating Construction Requirements for Scheduling**

Data integration is fundamental for automated schedule generation. Several pieces of research have been carried out for effective information integration in construction. Especially, a variety of core models for modeling process information in construction have been developed (Froese, 1996). Yamazaki (1995), Stumpf, et al. (1996), and Bouchlaghem, et al. (2004) developed object-oriented modeling approaches for product-process information integration. Staub-French, et al (2003) focused on feature-based process and product modeling for cost estimation. Halfawy and Froese (2007) developed a multitier component-based framework to facilitate the

implementation of modular and distributed integrated project systems for multidisciplinary project processes throughout the project life cycle. Since the major emphasis of these approaches is the integration of product and process models, they may not be applicable to incorporate construction requirements existing in different formats like functional requirements, temporal constraints or resource/spatial requirements. Consequently, these models could not support automated sequence reasoning and scheduling.

Recently, BIM (Building Information Modeling) technology has become a new approach to design, construction and facilitate management (Vozzola et al., 2009). Researchers have developed different BIM-based frameworks to integrate a wide range of information such as product, process, resource, or safety (Goedert and Meadati, 2008; Babic et al., 2010; Jung and Joo, 2011; Singh et al., 2011; Zhang et al., 2012). The major emphasis of these researches is to create a digital representation of the building information for better documentation, collaboration and project management. Consequently, construction requirements are not explicitly represented as a core knowledge component in these models. This ambiguity does not allow planners to efficiently manage and exploit construction knowledge for automated scheduling.

In summary, the identified limitations raise the need for an improve data integration framework which allows construction knowledge to be explicitly represented and integrated with other project data for automated sequencing.

### **2.3. Advancements of Planning and Scheduling Approaches**

Planning and scheduling in construction demands considerable time, knowledge and experience. The AEC research community has thereby put much effort into

improving the feasibility and efficiency of this complex process. This section first provides an overview of CPM/PDM and their major limitations for construction scheduling which motivate various research in this area. The next three subsections describe existing approaches in automated scheduling. The last two subsections review the development of advanced scheduling techniques which provides fundamental knowledge and methodology for this research.

### **2.3.1. CPM/PDM: Overview and Limitations for Construction Scheduling**

The critical path method (CPM) is a widely used and important tool for planning and control of construction projects. It is facilitated by an activity on arrow (AOA) diagram which represents project with activity nodes linked by precedence relationships (Lu and Lam, 2009). As such, CPM can only handle strict precedence constraints, i.e. Finish-to-Start (FS), and requires the use of artificial dummy activities. In addition to this strict precedence constraint, Precedence diagram method (PDM) involves three other relationships, Start-to-Start (SS), Finish-to-Finish (FF), and Start-to-Finish (SF) and positive/negative lags to depict partially concurring or overlapped working progress between activities (Moder et al., 1983). Accordingly, compared with CPM featuring FS logic only, PDM networks with “smart” relationships are more compact, flexible, and realistic to represent construction projects (Harris, 1978; Valls and Lino, 2001; Lock, 2003). In addition, CPM/PDM calculations are generally simple and straightforward. Therefore, they have been used intensively in the AEC industry. Popular commercial scheduling software systems (such as Primavera Project Planner, and Microsoft Project) also incorporate these methods.

The suitability and effectiveness of these models have been widely criticized. Firstly, CPM/PDM only provide mathematical models to simulate the construction process and manipulate the data provided by the planners but not the knowledge used to generate the plan (Morad and Beliveau, 1994). In other words, they lack a mechanism to capture and reason construction requirements for automated schedule generation. Secondly, CPM/PDM perform scheduling only from the process perspective. Component-based schedule constraints are therefore cannot be captured and processed using the models. Thirdly, since they handle only precedence constraints among activities, CPM/PDM have been found to be inadequate to cope with complex temporal constrains such as process concurrency/overlap/disjunction and work/resource continuity (Jaafari, 1984; El-Bibany, 1997). Lastly, CPM/PDM dictate only one predefined sequence (Chua and Yeoh, 2011) and lack the flexibility and expressiveness to cope with multiple alternative sequences and the varied patterns of construction methods (Jaafari, 1996; Choo et al., 1999).

The above limitations of CPM/PDM for construction scheduling have directed various research in this area, the key of which will be reviewed in the following sections. In particular, model-based, knowledge-based and case-based reasoning scheduling (described in sections 2.3.2 to 2.3.4) are three major paradigms to overcome the first two limitations of CPM/PDM, while other advanced techniques addressing the other two limitations are summarized in sections 2.3.4 and 2.3.5.

### **2.3.2. Model-based Planning and Scheduling**

Model-based scheduling is about linking the information from three domains: architectural design, construction scheduling and quantity take-off. Due to its popular use in the construction industry, CAD (Computer-Aided Design) models have been

widely employed to assist the planning and scheduling process. In particular, Cherneff et al. (1991) developed a system that interpret a CAD model into declarative presentation from which a list of activities and an associated activity network are generated. Relationships between components in CAD models have also been a primary source for generating and sequencing construction activities in other early research, such as Winstanley et al. (1993), McKinney and Fischer (1998), and de Vries and Harink (2007). In these models, predefined rules are used to generate activities and their precedence relationships. Recently, CAD models have been further exploited for both schedule generation and quantity take-off, or time-cost trade off planning. Kataoka (2008) proposed a method to automatically generate construction plans and quantity take-off from primitive architectural information and predefined construction method templates. On the other hand, Feng et al. (2010) used CAD models to develop a time-cost integrated scheduling approach, in which activities were sequenced based on physical constraints and genetic algorithms were used for time-cost tradeoff.

With the development of BIM commercial applications like Bentley, Tekla or Revit, generating construction schedules from BIM and/or IFC (Industry Foundation Classes) models has recently become a new trench of model-based scheduling. Tauscher et al. (2009) proposed schedule generation approach using case-based reasoning technique based on historical data extracted from IFC models. Weise and Liebich (2009) developed a 4D Simulation package which allowed to import IFC models and link with Microsoft Project. Kim et al. (2013) proposed a framework for automating the generation of construction schedules by using data (e.g. spatial, geometric, quantity, relationship and material layer set information) stored in BIM. Since IFC can represent design data created in most of existing modeling tools,



BIM/IFC-based scheduling approaches have no restriction on input design models. By this, generating construction schedules from design/IFC models could improve the scheduling efficiency and data accuracy (Porkka and Kähkönen, 2007).

Despite their benefits for visualization and simulation, the aforementioned model-based scheduling approaches retain two major drawbacks. Firstly, they lack a mechanism for capturing complex construction requirements and thus consider only atomic topological relationships and simple temporal constraints. Secondly, they incorporate CPM/PDM methods thus the construction sequences have to be predefined by planners based on only one construction method. Consequently, they still do not support the consideration and incorporation of multiple construction methods for alternative scheduling.

### **2.3.3. Knowledge-Based Planning Systems**

While model-based scheduling systems normally comprise a predefined set of rules for generating and sequencing activities, knowledge-based planning systems (KBPS) often consist of a knowledge representation/acquisition facility, inference engine and a knowledge base of domain rules and facts to capture some of key requirements and determine the construction sequence.

There is a large library of KBSPs in the AEC industry. Most early systems such as PLATFORM (Levitt and Kunz, 1985), IKBS9 (Gray, 1986), GHOST (Navinchandra et al., 1988), ACP (Waugh, 1989), SIPE (Kartam and Levitt, 1990), ESCHEULER (Moselhi and Nicholas, 1990), or MIRCI (Alshawi and Jagger, 1991) were only developed at the level of proof of concept or prototype. On the other hand, Construction Planex (Hendrickson et al., 1987) was one of the first working-model

systems. In Planex, design components are represented in a hierarchical structure and elementary components are a representation of the project at the lowest level. Predefined element activities are assigned to components and then aggregated into project activities. The element activities form the basis for rules which are based on physical and resource relationships to determine the construction sequence. In addition, Planex could demonstrate the feasibility of KBS for construction planning in specific domains; yet, the construction knowledge is implicitly stored in the knowledge database. This made it applicable only to specific project types.

In OARPLAN (Winstanley et al., 1993) developed by the Centre for Integrated Facility Engineering (CIFE) from Stanford University, the component hierarchy is one element of the generic model, which comprises component/object, action, and resource hierarchies. Activities are defined as an action applied to an object and requires resources and are represented as a hierarchical structure. This allows greater granularity for plan control. The dependencies and precedencies among activities are inferred from the relationship between sub-activities, other activities, and the physical constraints among components. Although OARPLAN did not explicitly consider construction methods, the OAR structure developed in this system forms the foundation for the development of CMD Scheduler (Fischer and Aalami, 1996) in which construction methods are treated as the basic knowledge concept for automated model-based scheduling. However, although they can capture resource requirement, the sequencing knowledge in these two models is based only on topological constraints among permanent building components yet does not address complex functional requirements occurring in the construction process.

KNOW-PLAN (Morad and Beliveau, 1994) utilizes an object oriented representation to capture building components with their geometric attributes and the type of their inter-connections. Different from previous system, activity sequences are determined based on predefined sequence templates which are based on continuity of employment, repetition location and the relationships between tasks rather than components. The knowledge base of KNOW-PLAN contains a database storing data extracted from CAD model and the dynamic sequencer with rules and explanations. Especially, the system allows the sequence to be interactively modified based on user-defined rules which can subsequently be added to the knowledge base.

CONSCHEDED (Shaked and Warszawski, 1992) and HISCHED (Shaked and Warszawski, 1995) were designed for automated planning of high-rise buildings. These systems also utilize object-oriented representation to capture building components under zones, systems and their attributes. Activities are generated from predefined list of tasks associated with categories of components, and the sequencing of activities is based on the start/end attributes of activities rather than physical relationships. As such, they could allow flexibility in construction technology.

Taking into account both resource and spatial requirements, ScaRC (Thabet and Beliveau, 1997) is more developed than the aforementioned systems in terms of complexity of sequencing knowledge. The scheduling knowledge of ScaRC contains data of four constraint types: horizontal construction logic, vertical construction logic, resource and space. Incorporating different types of major constraints would result in more feasible schedules. However, ScaRC is developed at prototype level only. In addition, it neither automates the activity generation nor reasons the logics. Thus, it requires a large amount of manual work by the planners.

The above review shows that the use of KBS technique to automated construction planning has bloomed in the recent decades. One of the major advantages of this approach is that generic construction knowledge is systematically defined, represented and applied to produce schedules. However, similar to model-based scheduling systems, the existing KBSPs have two common shortcomings. Firstly, most of them only consider atomic topological constraints, and lack a means to capture and reason complex functional requirements. Secondly, they are built on CPM/PDM and thus, cannot generate multiple alternative schedules which possibly happen from different choices of construction methods or technologies.

#### **2.3.4. Construction Planning using Case-based Reasoning**

In recent years, it has been common to use case-based reasoning (CBR) for automated planning. The main emphasis of this approach is to reuse construction knowledge and historical project data for schedule generation in order to save time and effort and reduce errors (Faris, 1991). Various techniques may be applied to assigning importance weights, measuring similarity, and adapting cases so that past experience can be exploited. Most CBR planning systems use similar concepts and approaches, but differ in their combination and modification of techniques thereof to suit their domain of application.

In the approach proposed by Chevallier and Russell (1998), historical projects with similar basic features are grouped together so that recurring project information and sequencing logic can be adapted to future projects through the use of standard structures and rules contained in project templates. The combination of rules and project templates could reduce some reasoning work which has already been

performed by the user in defining the template. Tah et al. (1999) developed CBRidge Planner for highway bridge projects. In this system, projects are defined as hierarchical structures with single components. Afterwards, these components are coupled with defined activities, grouped into cases and stored in a database for reuse. Similar construction processes are identified in terms of project specifications and using similarities of predefined sections of the structure. Dzung and Tommelein (2004) presented CasePlan, a case-based system that automates the generation of construction schedules for power plant boiler erection. CasePlan uses a generic product model to establish the basis for project comparison so that existing schedules can be retrieved and reused for similar projects. A further research approach, CONPLA-CBR (Ryu et al., 2007), uses crude descriptions of project properties, such as subsoil specification, costs of construction, or floor-count to determine similar schedules.

Some researchers also used IFC models as the basis for project description and comparison in CBR-planning. Mikulakova et al. (2010) use IFC data and IFC-based constraints to compute structural and content similarity measures. These similarity indicators are incorporated with weightings to retrieve the schedule of similar projects. Tauscher et al. (2009) and Hartmann et al. (2012) also used IFC models to find design similarity among projects.

In general, CBR-based planning could help reduce some laborious and repetitive scheduling steps and exploit existing knowledge and experience so that the scheduling efficiency can be improved. However, while construction projects are commonly unique with different contract agreements, site conditions, or applied code and regulations, the similarity among projects which is the core concept of this approach may not sufficient to obtain a proper or feasible schedule.

### **2.3.5. Advanced Scheduling Techniques**

Previous research has attempted to improve the practical application of CPM/PDM to construction. Various analytical and heuristic methods were developed to resolve the resource allocation problems in construction planning, including Chan et al. (1996), Hegazy (1999), Leu and Yang (1999), Abeyasinghe et al. (2001) or Liu and Wang (2008). The proposed approaches aimed to incorporate resource capacity constraint into CPM and to identify a schedule solution with good project makespan while fulfilling this constraint. However, these approaches still lack a mechanism to explicitly capture and represent complex temporal constraints.

Plotnick (2006) developed a Relationship Diagramming Method (RDM) as an extension of the traditional PDM with programmatically added “reason” codes so that planners can have a better understanding of the reasons for a relationship or an activity. By this, important data could be included in the representation of activities, and the semantic description of activities and relationships could thereby be enhanced. However, this extended feature of RDM is still insufficient to represent construction requirements systematically for automated sequencing and scheduling. On the other hand, Tamimi and Diekmann (1988) and Fan and Tserng (2006) aimed to generate alternative schedules using soft logic. Accordingly, a heuristic algorithm called SOFTCPM was developed to sequence the activities under the impacts of soft and fixed logics. Nevertheless, the scope of soft logic is still limited and unsuitable for construction requirements.

Recently, Constraint Logic Programming (CLP) has emerged as a common planning and scheduling tool to overcome the limitations of CPM/PDM in processing complex temporal constraints and generating multiple alternatives (Van Hentenryck,

1989; Zweben and Fox, 1994). This is due to its ability to use constraints actively to reduce the computational effort to solve the combinatorial nature of scheduling problems (Caseau and Laburthe, 1994; Goltz, 1995). Baptiste and Le Pape (2000) also noted that the performance of CLP schedulers is comparable to traditional operational research approaches, if not better for most problem instances, while offering greater model flexibility. Readers may wish to refer to Jaffar and Maher (1994), (Wallace, 2002), or Apt (2007) for a more in-depth discussion on the basic concepts of CLP in Prolog and its extension to CLP. With the ability to process both precedence and disjunctive constraints, CLP has been widely used for resource-constrained scheduling problems. Some existing methodologies may be found in (Beck and Fox, 2000; Dorndorf et al., 2000; Fromherz, 2001; Cesta et al., 2002; Laborie, 2003; Lorterapong and Ussavadilokrit, 2012). For construction scheduling, these methods still cannot capture complex combinations of temporal constraints which possibly result from conditional constraints or inter-dependencies between construction requirements. In addition, they are not able to reason construction requirements into feasible schedules due to the lack of a modeling syntax and embedded reasoning knowledge.

### **2.3.6. PDM++ Modeling Framework**

The PDM++ model (Chua and Yeoh, 2011) which has been adopted for this research, is one attempt to improve the application of CLP to construction scheduling. It extends the traditional PDM model by incorporating two basic logical operators “AND” and “OR” with the enriched syntax inspired by the Artificial Intelligence developed by Allen (1984). Accordingly, PDM++ not only maintains the capability of Allen’s representation but also subsumes the PDM model by allowing both minimum and maximum lag time requirements to be explicitly described.

Type	Constraint	Short Form	Mathematical Definition	Pictorial Representation
Simple Unary	$X$ Start-Before( $m$ )	SB	$X^- \leq m$	
	$X$ Start-After( $m$ )	SA	$X^- \geq m$	
	$X$ Due-Before( $m$ )	DB	$X^- + d_X \leq m$	
	$X$ Due-After( $m$ )	DA	$X^- + d_X \geq m$	
Simple Binary	$X$ Before( $m$ ) $Y$	B	$X^- + d_X + m \leq Y^-$	
	$X$ Before( $\sim m$ ) $Y$		$X^- + d_X + m \geq Y^-$	
	$X$ Starts( $m$ ) $Y$	SS	$X^- + m \leq Y^-$	
	$X$ Starts( $\sim m$ ) $Y$		$X^- + m \geq Y^-$	
	$X$ Finishes( $m$ ) $Y$	FF	$X^- + d_X + m \leq Y^- + d_Y$	
	$X$ Finishes( $\sim m$ ) $Y$		$X^- + d_X + m \geq Y^- + d_Y$	
	$X$ Start-Finish( $m$ ) $Y$	SF	$X^- + m \leq Y^- + d_Y$	
	$X$ Start-Finishes( $\sim m$ ) $Y$		$X^- + m \geq Y^- + d_Y$	
Complex Unary	$X$ Cannot-Occur [ $L, U$ ]	CO	$(X^- + d_X \leq L) \vee (X^- \geq U)$	
Complex Binary	$X$ Overlaps( $m$ ) $Y$	O	$(X^- + m \leq Y^- + d_Y) \wedge (Y^- + m \leq X^- + d_X)$	
	$X$ Overlaps( $\sim m$ ) $Y$		$(X^- + d_X \leq Y^- + m) \vee (Y^- + d_Y \leq X^- + m)$	
	$X$ Start-Overlap( $m$ ) $Y$	SO	$(X^- + m \leq Y^- + d_Y) \wedge (X^- \geq Y^-)$	
	$X$ End-Overlap( $m$ ) $Y$	EO	$(X^- + d_X \leq Y^- + d_Y) \wedge (X^- + d_X \geq Y^- + m)$	
	$X$ Meets $Y$	M	$(X^- + d_X + m \leq Y^-) \wedge (X^- + d_X + m \geq Y^-)$	
	$X$ Contains $Y$	C	$(X^- \geq Y^-) \wedge (X^- + d_X \leq Y^- + d_Y)$	
	$X$ Disjoint $Y$	D	$(X^- + d_X \leq Y^-) \vee (Y^- + d_Y \leq X^-)$	

Figure 2.1. PDM++ Temporal Relationships from Chua and Yeoh (2011)

PDM++ generally consists of two different types of relationships: Unary and Binary. Unary relationships are defined as constraints affecting the start/finish time of a single activity while binary relationships specify the temporal constraints between two activities. For easy reference, a brief summary of PDM++ model is presented in



Figure 2.1 where  $m$  and  $\sim m$  respectively denote minimal and maximal lag types. The third column shows the respective short form formats of PDM++ constraints which will be used throughout the next chapters.

Based on mathematic definitions, PDM++ temporal constraints can be classified into two groups: simple and complex. Simple temporal constraints are those represented by only one mathematical inequality constraint. This group includes 4 unary constraints and 8 binary constraints (with minimal and maximal lag requirements), forming the basic constructs used to represent complex constraints. In contrast, complex constraints are mathematically represented by either a conjunctive or disjunctive combination of inequality constraints. For example, constraint  $X$  Contains  $Y$  is represented by two inequality constraints  $X^- \leq Y^-$  and  $(\wedge)Y^- + d_Y \leq X^- + d_X$ , which respectively refer to constraints  $(X \text{ SS}(0) Y)$  and  $(Y \text{ FF}(0) X)$ . In other words, a complex temporal constraint is a conjunctive/disjunctive combination of at least two simple temporal constraints.

Of various advanced scheduling techniques that have been developed to overcome the limitations of CPM/PDM in capturing and processing construction requirements, PDM++ could be the most effective in representing complex temporal constraints for generating alternative schedules. This attribute makes PDM++ employed as the background modeling tool to represent complex temporal constraints derived from construction requirements. However, similar to the traditional CPM/PDM, PDM++ does not include the necessary knowledge and reasoning tools to represent construction requirements and convert them into temporal constraints, which are vital for automated sequencing and scheduling. Hence, this research will overcome

this limitation by proposing construction requirement formalization tools and sequence reasoning frameworks which will be presented in the following chapters.

## **2.4. Criticality Analysis in Construction Schedules**

If generating constructible schedules is the necessary condition of good schedule performance, schedule management could be considered as the sufficiency condition. Essentially, schedule management has been found to be the most crucial for schedule performance (Chua et al., 1999; Iyer and Jha, 2006). For good schedule management, it is necessary to identify the crucial parts of the schedule which need to receive more management attention than others, and criticality analysis is thus a crucial schedule management task. Currently, the concept of criticality could be examined from both activity and constraint viewpoints. This section therefore reviews the key research on criticality analysis in construction schedules from two perspectives of criticality: activity and constraint.

### **2.4.1. Criticality Analysis from Activity Perspective**

The concept of criticality already has been introduced since the formation of the CPM. CPM (Kelley, 1961) allows planners to identify critical paths as a series of critical activities from beginning to the end of the project network (Wiest and Levy, 1977). Accordingly, total and free floats are used to demonstrate the impact of delaying an activity on project makespan and on the early starts of subsequent activities, respectively. In particular, a critical activity has zero floats and any delay of its start time or any increase in its duration will delay its successor activities and prolong project makespan.

Criticality concept with float times has been used effectively in anticipating schedule changes arising from activity changes in CPM networks. However, it has been indicated in previous research that using floats to study the impact of changing the duration of a critical activity on schedule makespan is inadequate in PDM models due to the existence of non-finish-to-start relationships. Further information – critical arcs or constraints incident with a critical activity – need to be taken into account for the analysis. Various methodologies have been proposed to address this issue. Wiest (1981) suggested that there should be four types of critical activities, namely *normal*, *reserve*, *neutral* and *perverse*. A critical activity is classified as normal if a critical path passes through it from its start to finish, while it is called reserve if a critical path passes through it from finish to start. When a critical path enters and exists from the starting or finish point of an activity, changes in duration of this activity does not affect schedule duration and thus this activity is defined as neutral. Finally, if an activity has both normal and reserve impacts on project duration, it is called a perverse activity.

The classification proposed by Wiest (1981) forms the background for various later research. A similar classification was also proposed by Moder et al. (1983). However, when incorporating activity splitting the authors further divided neutral class activities into two groups: start-critical and finish-critical. Elmaghraby and Kamburowski (1992) examined the criticality in activity networks with generalized precedence relations, and proposed that an activity is critical if at least its start- or finish- node belongs to a critical path. Accordingly, they classified critical activities into five groups: forward-critical, backward-critical, bi-critical, start-critical, and finish-critical. Valls and Lino (2001) developed a more detailed classification of

critical activities including six classes: normal, reserve, neutral, perverse, increasing normal, and decreasing reserve.

From the above review, the proposed criticality classifications provide a deeper insight into how changes of critical activities influence project duration. These classifications were developed commonly by analyzing the nature of the critical arcs or constraints linking the critical activities. This implies that constraints play a key role in defining the way a critical activity affects project makespan, and analyzing the characteristics of constraints is thereby essential for criticality and change analysis in construction schedules.

#### **2.4.2. Criticality Analysis from Constraint Perspective**

Theory of constraint (TOC) advocates that most crucial constraints should be identified and resolved with the highest priority to enhance the overall system performance (Rahman, 1998). In traditional CPM networks, critical constraints are the critical precedence relationships linking critical activities and are often inferred from critical paths. In resource-constrained CPM models, the concept of critical path is extended to critical sequence to comprise critical activities linked by both technological and resource precedence relationships (Wiest, 1964). Various analytical approaches for identifying critical sequences have been developed, including those proposed by Wiest (1964); Woodworth and Shanahan (1988); Bowers (1995); Lu and Li (2003); Kim and de la Garza (2005); (Lu and Lam, 2008) and Liu and Shih (2009). However, as highlighted in Bowers (2000) and Rivera and Duran (2004), critical sequences might be different for different schedules, and might depend on the specific method being applied. Hence, critical constraints inferred from critical sequences might not be identical in different schedules or with different methods.

Chua and Shen (2005) developed an analytical methodology to identify key resource and information (RI) constraints causing project delays. The impacts of these enabling constraints on the overall project performance are represented by their floats. More specifically, a RI constraint is critical to project delays when it has zero floats since any delay in its Estimated Availability Time (EAT) will push back the latest time of the associated activity and consequently delay the overall project. It was found from their analysis that constraints related to non-critical activities could be critical and cause project delays when they are delayed. Therefore, in addition to normal critical constraints associated with critical activities, these constraints should also be well-managed to reduce and prevent project delays. As such, this theoretical methodology has opened a novel perspective for schedule management from the viewpoint of constraint criticality. However, since this approach was built upon the traditional CPM, its application is still restricted to single constraints and from the perspective of a single schedule (Nguyen and Chua, 2012). Consequently, it is found inadequate to handle construction requirements which may result in complex temporal constraints and multiple alternative schedules.

It is found from the above review that a constraint is identified critical if it is involved in a critical path or sequence. In other words, only constraints linking critical activities and governing the project duration are critical. Hence, those governing non-critical activities' times are sometimes intuitively considered non-critical or unimportant. However, in construction stage, besides project duration, the start/ finish times of non-critical activities and activity sequence are also important to contractors as they may affect the overall work plan among related parties or different projects. In addition, existing approaches still focused on a single constraint and from a single

schedule perspective. Consequently, they are found insufficient for analyzing the criticality of construction requirements which usually comprise multiple temporal constraints and possibly lead to multiple alternative schedules.

## **2.5. Identified Research Gaps**

The above literature review shows that numerous researches have attempted to improve the adequacy and efficiency of schedule generation and analysis using different approaches and perspectives. Schedules are controlled by construction requirements which arise from different construction aspects with various natures and often lead multiple temporal constraints. Thus, systematically formalizing construction requirements, automatically converting them into temporal constraints, and analytically analyzing them are necessary for automated scheduling. Nevertheless, these issues are not fully addressed in the current research, and the literature has prompted the following research gaps.

Firstly, there is a lack of scheduling framework that allows the incorporation of multiple construction methods and complex construction requirements per se. Generally, construction requirements can exist in such different forms as functional dependencies between product components, temporal constraints on a single or between construction processes, and requirements of key resources and work space availability. Integrating these complex requirements for automated sequencing and scheduling requires an extended information modeling framework. Unfortunately, the present information modeling approaches focused mostly on elements of requirements such as atomic physical relationships, resource or spatial constraints. Hence, an extended framework is a necessity for automated scheduling from key construction requirements.

Secondly, there is a need for a generalized model to capture functional requirements and convert them into temporal constraints. Functional requirements, which encompass topological dependencies and intermediate function requirements, are among the most complex construction requirements. Generally, functional requirements are product-oriented and involved both permanent and temporary components. They could also result in complex combinations of temporal constraints which could lead to multiple alternative schedules. Despite this, existing sequencing approaches are found insufficient in automatically derive temporal constraints from such requirements. Hence, a more adequate sequencing framework from functional requirements is necessary for automated sequencing and alternative scheduling.

Finally, there is a lack of a methodology for identifying critical construction requirements. Construction requirements are the governing factors of the feasibility of schedules; hence, identifying crucial requirements is necessary for good schedule management. Despite this, present criticality analysis approaches emphasis mainly on activities or single constraints. Besides, they are still carried out from the perspective of a single schedule. Since complex construction requirements normally involve many temporal constraints and possibly lead to multiple alternative schedules, these approaches are found inadequate to handle complex requirements. Therefore, a systematic approach for analyzing the criticality of constraints and construction requirements from the perspective of multiple alternative schedules is necessary so that crucial requirements can be identified and managed.

## **2.6. Summary**

This chapter has presented a literature review on approaches for generating and analyzing construction schedules. The detail review shows that key construction

requirements have been inadequately captured and analyzed for schedule generation and management. This is due to the lack of representation and integration frameworks which allow for identifying, interpreting and integrating key requirements for automated sequencing and scheduling. The literature also depicts that a systematic approach for analyzing the criticality of construction requirements is needed for schedule analysis to achieve better schedule management.

Subsequent chapters of this dissertation presents the research attempts to overcome the identified research gaps. In particular, a generalized framework for automated scheduling from construction requirements will be presented in the next chapter. This framework aims to improve the formalization and integration of complex construction requirements into the scheduling process. A generalized functional requirement model for representing and reasoning complex functional requirements into temporal constraints will be described in chapter four. A system architectural framework and necessary reasoning algorithms for implementing the proposed models will be presented in chapter five. Especially, to further enhance the efficiency of scheduling with complex constraints, a pre-emptive constraint analysis framework will also be developed in this chapter. Finally, chapter six will present an innovative approach for analyzing and managing construction requirements based on the impact of construction requirements on schedules. Altogether, these research results will form a generalized framework for automated schedule generation and analysis from the perspective of construction requirements, which allows the consideration, incorporation, pre- and post-analysis of construction requirements to achieve better schedule feasibility and efficiency.



# **CHAPTER 3. GENERALIZED FRAMEWORK FOR AUTOMATED SCHEDULING FROM CONSTRUCTION METHODS AND REQUIREMENTS**

## **3.1. Introduction**

Construction methods and requirements are key construction knowledge that should be explicitly represented and sufficiently incorporated into scheduling for good schedule generation and analysis. This chapter attempts to address these issues by developing a generalized framework for automated generation of schedules with consideration of alternative methods and major requirements. The development of this framework starts by defining core knowledge models for representing key project information originated from product, process, and construction knowledge perspectives. Four fundamental scheduling processes are then developed to automatically generate activities and temporal constraints from building models and the imposed construction requirements, and compute for alternative schedules. In combination, the proposed approach will help improve the current practice of schedule generation by allowing for the explicit representation and incorporation of construction knowledge in the form of construction methods and requirements, by automating the sequence reasoning and schedule computation processes, and by producing all feasible alternative schedules.

## **3.2. Core Knowledge Models for Automated Scheduling**

This research proposes four core knowledge models: Extended Product, Construction Method, Construction Requirement, and Schedule to formalize the

project information as well as construction knowledge necessary for automated scheduling from construction requirements.

### 3.2.1. Extended Product Model

Product models are conceptual structures that represent the project-specific components. In many planning systems, product decomposition models are normally directly derived from design models, and describe only permanent components. This research employs an extended product model which includes both temporary structures and site works required for the construction of the permanent facility into the product hierarchy similar to Song (2006) as shown in Figure 3.1. Since temporary structures and site works are the main elements describing construction methods, such an augmentation enables an explicit description of the lifecycle, and functional behaviors of all components, as well as construction requirements associated with them.

- ***Permanent component:*** Permanent components represent the permanent structures that will be delivered to the project owner after the construction. Once a permanent component enters the product system, it will remain there until the facility is demolished.
- ***Temporary component:*** Temporary components refer to temporary facilities whose existence is governed by the applied construction method to maintain the stability of other components or support construction processes. As such, temporary component will be removed when the need for its functionality no longer exists.
- ***Site work component:*** Site work components represent the site components that depict the site environment of the permanent facility, but do not belong to the

permanent facility. Example of site work components can be earth works and temporary accesses, which provide construction spaces or accesses to support construction processes.

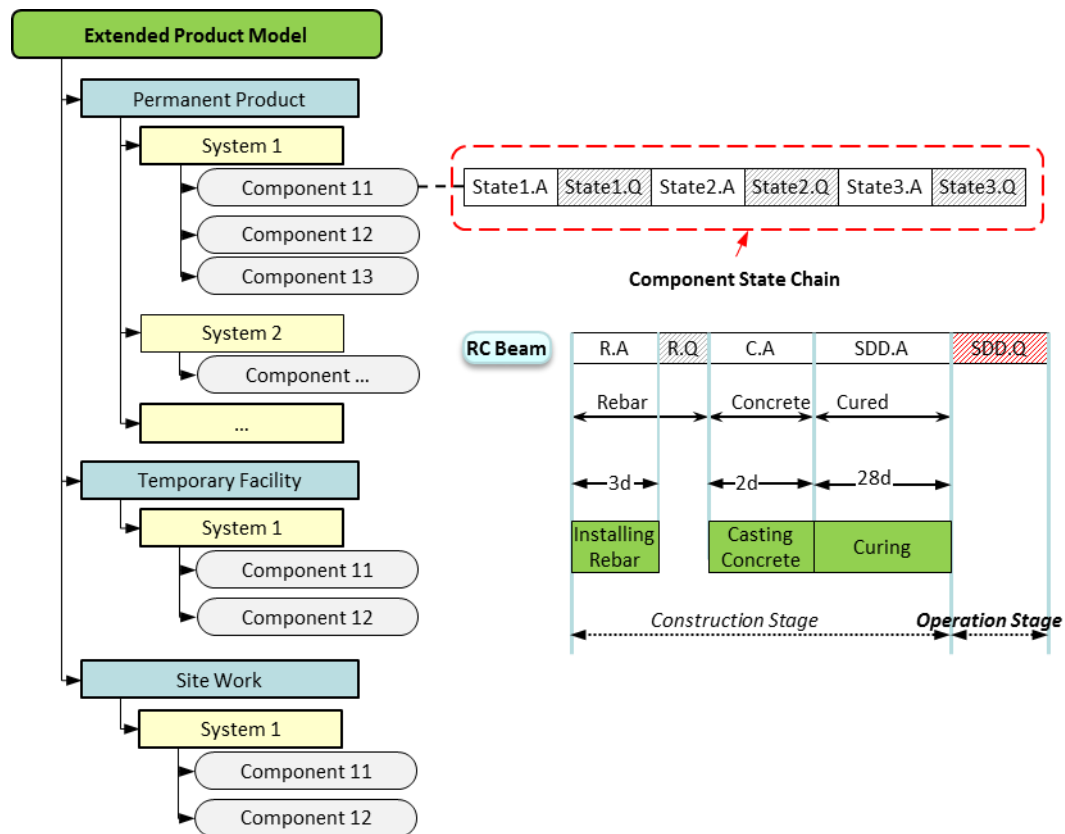


Figure 3.1. Extended product model

Similar to traditional product models, components in each category are arranged in decomposition (i.e. *part-of*) hierarchies. In particular, the entire facility model is gradually decomposed into systems and subsystems (or zones and subzones if the decomposition is area-oriented), and components with no subcomponents are the lowest level of detail. By this, most of the facility components in the engineering design like beams, columns, or piles can be represented as product components in the extended product model. When large-size building elements like long shear walls or

wide slabs are divided into multiple segments, each segment is defined as a component. The decomposition levels of each component system can be determined by planners to a granulation degree suitable for describing construction requirements and planning intention. A product component comprises a set of main attributes as follows:

- **Geometry:** Components have geometries, i.e. height, length, width or diameter.
- **Location:** Components have locations in a 3D space.
- **Decomposition:** Components can have constituting components.
- **Functionality:** Components can have functional behaviors according to design intentions (for permanent components) or technological purposes (for temporary and site work components). Typical functions that a component can provide to another component are: *support, suspend, contain, protect, and balance*.
- **State chain:** Each component has a state chain defined by the applied construction methods and depicting its transitive engineering behavior along its construction lifecycle. A state chain consists of a sequence of states, each of which describes an intermediate status of a component in the construction process. This research adopts and extends the component state concept proposed by Song (2006) in which a component state is divided into an *active phase* and a *quiescent phase* to distinguish the transitive engineering characteristics that determine the behavior of the in-progress component. Active phases are associated with construction processes while quiescent phases are the duration when no process happens to the component. Especially, in order to further distinguish the construction and completion stages, this research has augmented Song's component state chain with a final quiescent state (Complete.Q) which represent the duration in which the component has been already constructed and

can perform its designed engineering behavior. This extension is necessary for representing construction requirements occurring in the service stage of a component like final functional dependencies.

As shown in Figure 3.1, the construction lifecycle of the “RC Beam” has three consecutive processes: “Installing Rebar”, “Casting Concrete”, and “Natural Hydration”. The entire state chain representing this construction lifecycle is divided into four sequential states named: Rebar, Concrete, Strength Development, and Completion. Among these, state “Concrete” only contains an active phase as the beam the Natural Hydration starts right after Casting Concrete is finished, leading to an immediate change from states “Concrete” to “Strength Development”. The construction phase ends at the end of state “Strength Development” when the RC Beam meets its designed status, starting the completion phase, and thus the completion state has only quiescent phase. The final state of this component is “Completion” indicating the time period in which the beam has been fully constructed and can provide all functionalities or performances as design intention.

Syntactically, a product component can be represented as follows:

```
product_component(name, category, type, (geometries), (location), [decomposition],  
[functionalities], [state chain]).
```

### **3.2.2. Construction Method Model**

Construction method model abstracts the knowledge of construction technology in terms of generic construction processes, resources and temporary structures required to facilitate the processes. To this extent, the following attributes are defined to represent a construction method.

- **Type:** A construction method can have either type: *elementary* indicating that the method refers to only one construction process, or *aggregation* defining a method as a combination of multiple elementary methods and thus involving multiple construction processes.
- **Construction Process:** Construction process represents generic construction work that needs to take place with respect to the method. One process can be involved in different construction methods. In this case, the methods differ in other attributes.
- **Component Type:** This attribute specifies the type or class of product component to which the method can be applied.
- **Temporary Structures:** A method may require one or a set of temporary structures to support the process.
- **Key Resources:** A method may also need one or more types of key resource to facilitate the process.
- **Quiescent Phase Allowed:** This Boolean attribute describes whether the process allows for any gap between it and the subsequent process when two methods are sequentially applied to one component. In other words, it specifies if a component state associated with the process has a quiescent phase. A “Yes” value for this attribute indicates that the construction method does not require the method subsequently applied to the component to be carried out right after it. Consequently, the component state associated with this method has two phases: active and quiescent. In contrast, a “No” attribute requires the subsequent method to be carried out immediately after the method, and thus the associated component state has only active phase.

Figure 3.2 presents two examples of construction method: (a) elementary, and (b) aggregated methods. In case (a), the elementary construction method involves only one process, and allows a quiescent phase after the process has been finished. This means that when this method is applied to a beam component, the state “Rebar” can have both active and quiescent phases. On the other hand, in case (b), the aggregated construction method “Cast-in-situ” involves three construction processes.

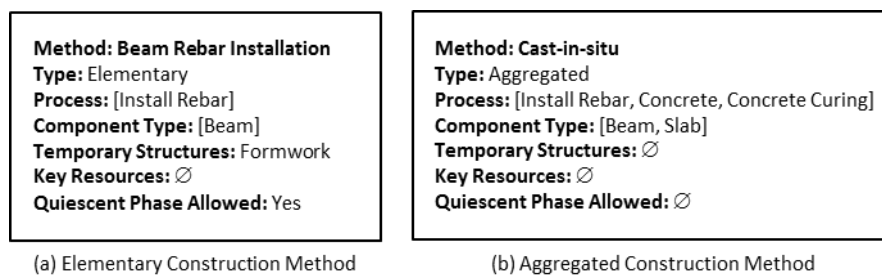


Figure 3.2. Examples of construction method

Syntactically, a construction method can be represented as follows:

**construction\_method(name, type, construction process, [component types], [temporary structures], [key resource types], quiescent phase allowed).**

### 3.2.3. Construction Requirement Model

A construction schedule is controlled by its constraints derived from construction requirements. The satisfaction of these requirements determines the appropriateness of the schedule. Therefore, to complete the representation of planning knowledge, construction requirements should be modeled as a fundamental data class in addition to products and construction methods.

Construction requirements are grouped into two main categories (as shown in Figure 3.3): functional and non-functional. Functional requirements are classified

respectively as intermediate and final functional requirements. Non-functional requirements can be categorized into four sub-classes: temporal, key resource, work space, and value requirements. Temporal requirements refer to constraints imposed on the start/finish or the sequence among construction processes. Resource and space requirements respectively represent the needs of resource and space availability for constructing a product component and/or carrying out a construction process. Value requirements refer to constraints on measurable features of product components such as weight or geometries.

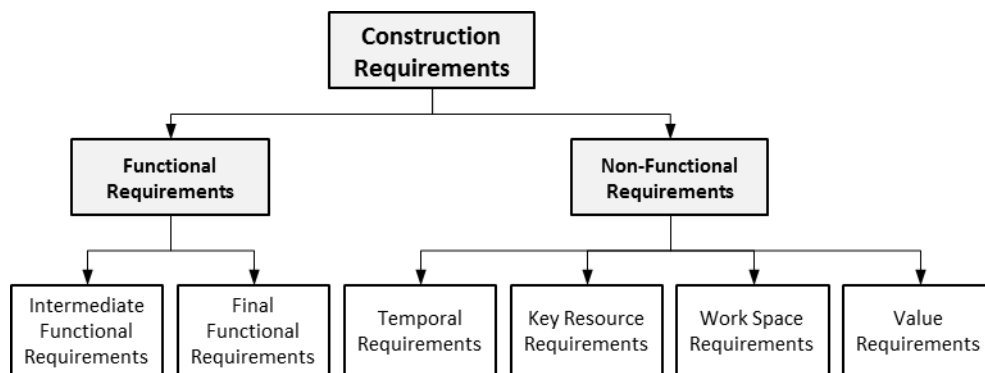


Figure 3.3. Classification of construction requirements

In general, a construction requirement implicitly imposes one or many functional, temporal, topological or measurable constraints on a single or some components, construction processes, resources and space entities. This research extends the generalized ontological model developed by (Yeoh, 2012) to describe a requirement using three attributes: Purpose, Operator and the Necessary Condition that need to be satisfied for the fulfillment of the requirement.

- **Purpose:** The purpose refers the agent that drives the requirement. The purpose attribute of a functional requirement is called “Function User” which is the



requester for the functionality behaviors of other product components to sustain its stability or construction. For a non-functional requirement, the purpose attribute refers to a construction process or a key resource or work space entity whose execution/performance is enabled by the fulfillment of the requirement.

- **Operator:** The operator of a requirement depicts the product components, construction process, resource, or space entity whose behaviors, inter-relationships or attributes need to meet some constraints for the fulfillment of the requirement. In particular, the operator of a functional requirement is called “Function Provider” which involves a set of product components whose functionality behaviors provide the required functionality from the function user. The operator of a non-functional requirement is the resolution of the requirement and can involve a single or a combination of component states, construction processes, space or key resource entities, and measurable attributes of product components or construction processes.
- **Necessary Condition:** The necessary condition involves the constraint(s) which must be fulfilled before the requirement is available for proceeding. It may be represented as functional dependencies, topological relationships between product components, temporal relationships among construction processes, or constraints of measurable features like the clearance between objects, weight of loads, or number of key resources. Measurable constraints can be defined in the form of arithmetic comparative relationships. Essentially, the necessary condition of a functional requirement normally comprises a functional relationship between function user and function provider. The temporal relationship between these two parties needs to be derived from the relationship

between the components involved in the function provider. For this purpose, a generalized framework for modeling and reasoning from functional requirements built upon this basic requirement model will be presented in the next chapter. On the other hand, the necessary condition of a non-functional requirement can comprises one or many constraints of other types (temporal, topological or quantitative). Typical taxonomies for functional dependencies are: *support*, *suspend*, *contain*, *protect*, and *balance*. Taxonomies for topological, temporal relationships and measurable constraints follow those defined in previous studies such as Nguyen and Oloufa (2002) and Chua et al. (2010) and are summarized in Figures 3.6 – 3.8. Necessary conditions having impact upon construction sequences need to be converted into temporal constraints for scheduling.

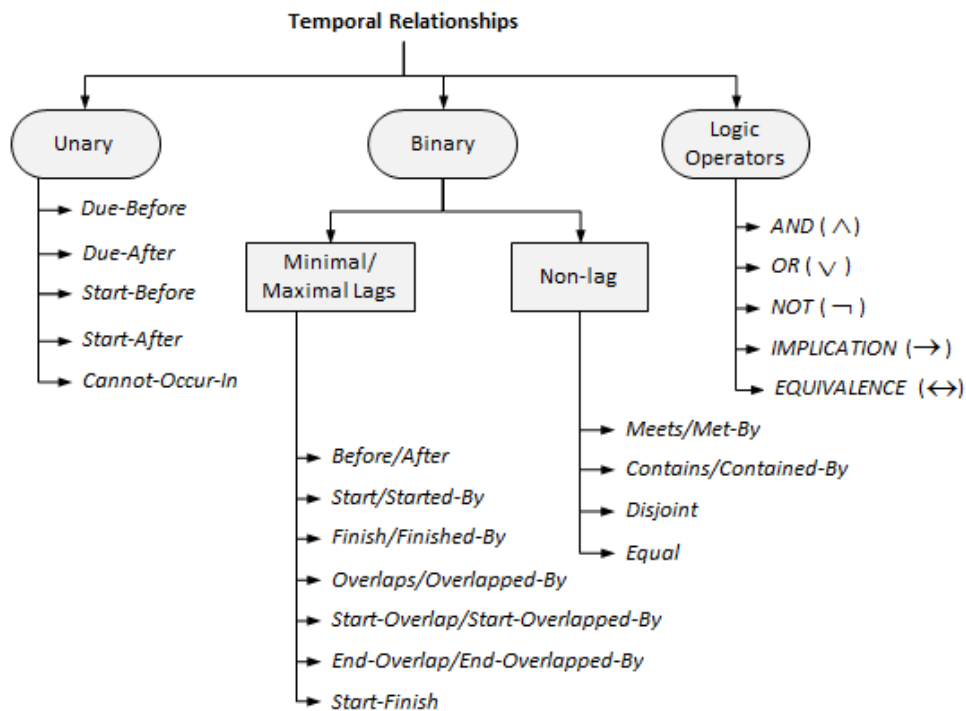


Figure 3.4. Temporal relationships based on PDM++ model

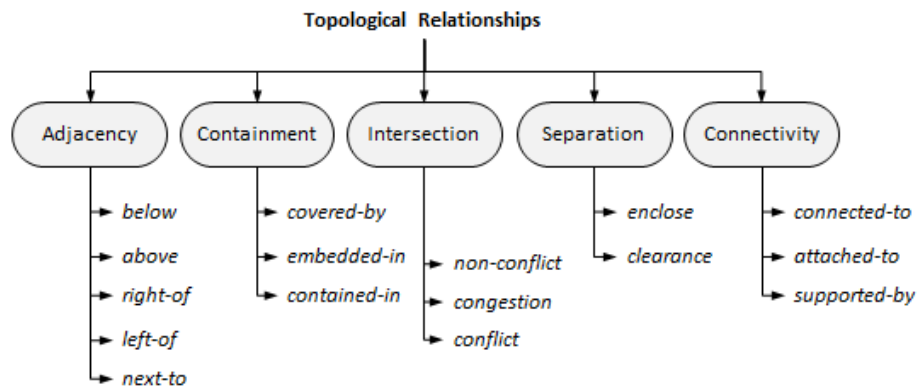


Figure 3.5. Topological relationships

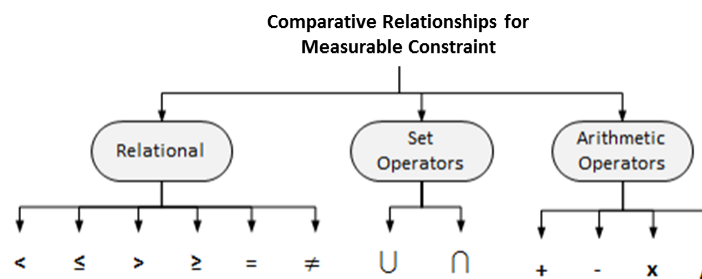


Figure 3.6. Comparative relationships

Figure 3.7a describes a functional requirement “R1: Beam B2 needs to be supported by two columns C1 and C2”. The purpose of R1 is represented by a function user which comprises of an individual or a set of components requesting for the functionality (B2). The operator of R1 is represented a set of component states of product components performing the required function (C1 and C2). The necessary conditions of R1 are the functional dependency between function user and provider. In Figure 3.7b, requirement R2 defines a dependency between the erection of beam B4 and the construction sequence of beams B2 and B3. The purpose attribute of R2 is the erection process of beam B4 (B4-Erection). The operator comprises the two erection processes of beam B2 and B3 (B2-Erection and B3-Erection). The necessary condition involves a conditional temporal constraint among three processes.

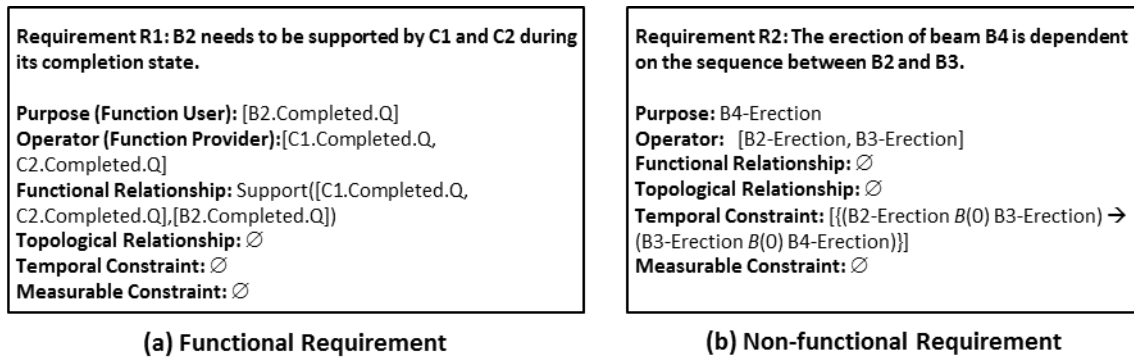


Figure 3.7. Examples of construction requirement

Syntactically, a construction requirement can be represented as follows:

**construction\_requirement(description, [purpose], [operator], [functional relationships], [topological relationships], [temporal constraints],[measurable constraints]).**

### 3.2.4. Construction Schedule Model

Construction schedule model formalizes the construction processes, and their temporal dependencies involved in the project. The schedule model is described in a hierarchical structure. Each hierarchy represents the construction of a system or subsystem of the product model. The decomposition attribute of a schedule can assist planners with rapid and concise representation of temporal constraints among groups of activities. It also allows for elaborating the schedule to a desired level of detail. Elementary activity is the lowest detail level of the schedule model. Each elementary activity has a one-to-one relationship with an active component state phase and involves the following attributes:

- **Decomposition:** Schedules can contain other constituting sub-schedules.
- **Temporal Constraint Set:** Temporal constraint set contains all the temporal relationships between their constituting schedules (activities). Temporal

constraints or relationships are represented using relationships defined in PDM++ Model.

- **Start Time:** Each schedule has a time range indicating its earliest and latest time of its commencement.
- **Finish Time:** Each schedule has a time range indicating its earliest and latest time of its completion.
- **Duration:** Each schedule has duration, which is the difference between earliest/latest finish and earliest/latest start times.
- **Resource Use:** Resource use attribute is a list of key resources and amount required to support any activities in the schedule. At the activity level, this attribute is useful for estimating activity durations, while at a higher level, it provides information for resource management.
- **Space Use:** Similar to resource use, this attribute is a list of work space entities used by all activities in the schedule.

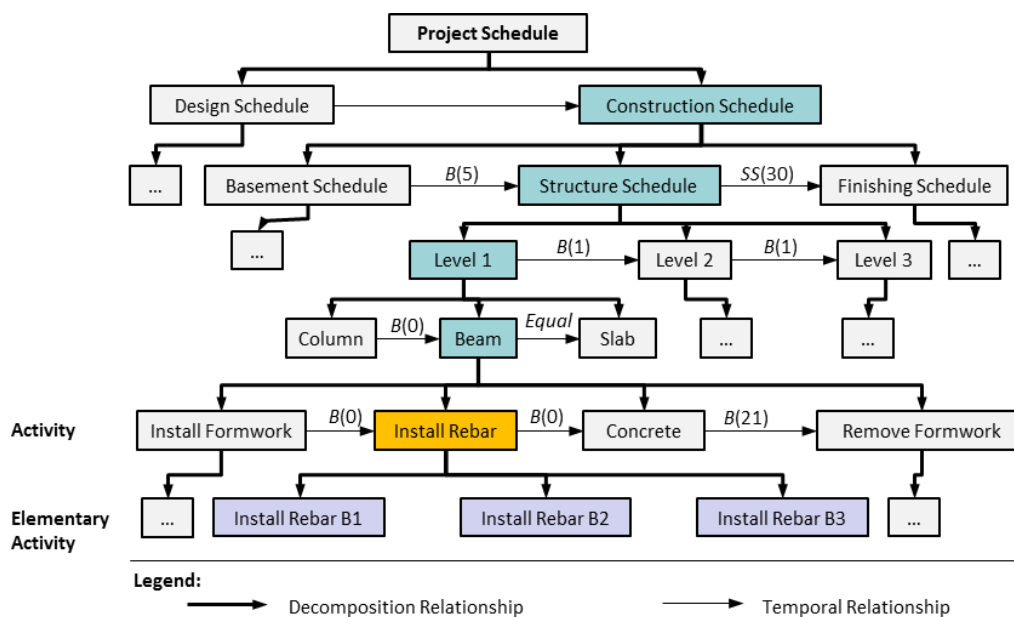


Figure 3.8. Example of schedule model

Figure 3.8 shows an example schedule model of a construction project with six levels of detail. The schedule model differs from a work breakdown structure (WBS) in that it contains temporal constraints between high-level schedules. For example, Super Structure Schedule must be at least 5 days after Basement Schedule (described as constraint  $B(5)$ ). These constraints can be further elaborated into a set of constraints between their constituting schedules when schedules at a lower detail level are needed.

### 3.2.5. Schedule Data Integration Framework

Although a construction project may involve a variety of information and data, this research specifies seven types of data which are indispensable for a construction requirement oriented automated scheduling system. The conceptual integration framework of these core data based on the proposed knowledge models is depicted in Figure 3.9. **Error! Reference source not found.** It is based on an object-oriented paradigm that defines the relationships between product components, construction methods, construction requirements, activities and temporal constraints.

The *Product Component* class is devised to implement the proposed Extended Product Model. A product component object has a *Name* for identification. Its *Geometries* define its physical dimension such as height, length, or width. The *Location* attribute is defined in the form of (x, y, z) coordination in 3D space. The *Category* attribute takes one of three default values: Permanent, Temporary, and Site Work. The *Type* attribute defines the structural function, such as column, beam, and so on. The *Decomposition* attribute specifies the direct subcomponents constituting the component. The *Functionality* attribute is used to capture the designed final functionality behaviors of the component. Finally and most especially, the *State Chain*

attribute describes the transitive engineering behavior along the component's construction lifecycle. It contains a series of *component state*, each of which is defined by a *construction method*.

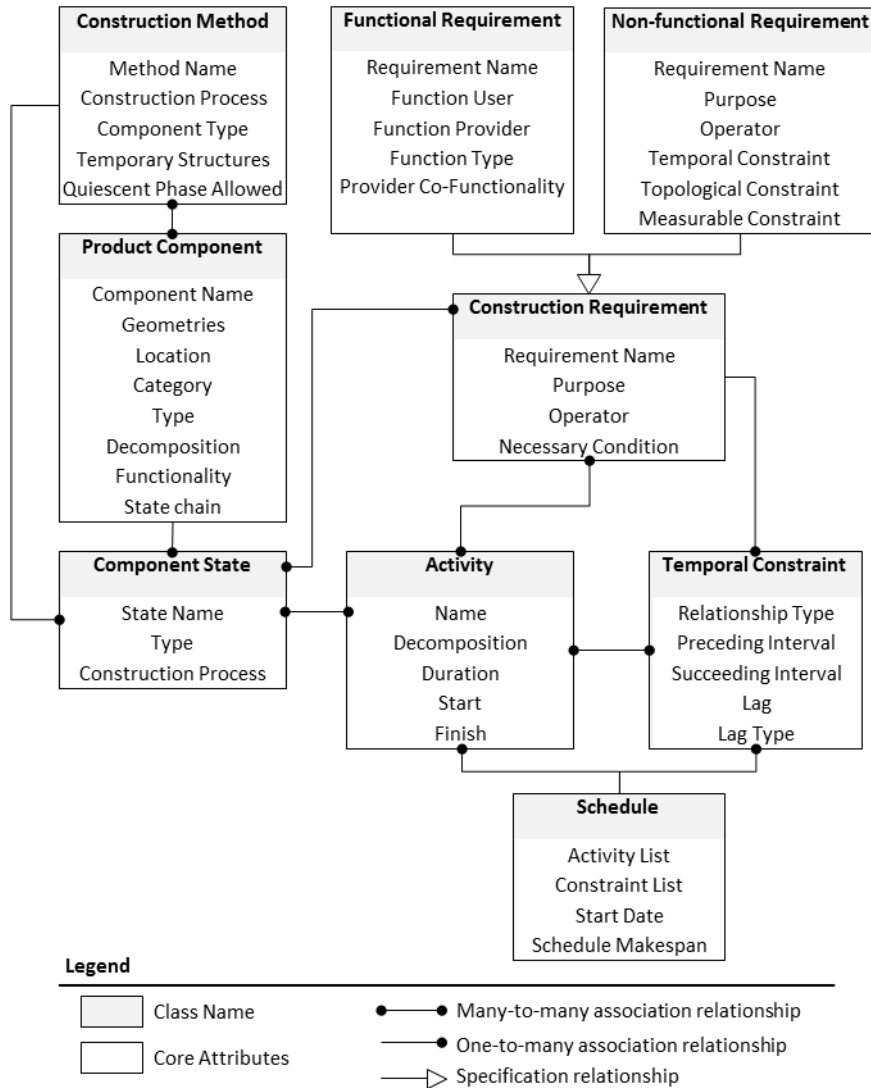


Figure 3.9. Integrated construction information framework

The *Component State* object describes an intermediate status of a component in the construction process. It is defined by three core attributes: a *Name* or a unique ID, a *Type* with two default values: active and quiescent, and a *Construction Process* associated with the state. The value of construction process attribute is extracted from

the construction method defining the component state. Component state objects are also the key elements for representing construction requirements.

The *Construction Method* object represents the core construction process and its corresponding requirements. It is defined by a unique *Name*, and a *Construction Process*. The *Component Type* attribute indicates the type of product component to which the method can be applied. The value of this attribute is presented in a set format. The *Temporary Structures* attribute define the set of temporary component types required for executing the method. Finally, the boolean *Quiescent Phase Allowed* attribute describes whether the process allows for any time gap between it and the subsequent process when two methods are sequentially applied to one component. The relationship between *Construction Method* and *Product Component* objects is many-to-many, meaning that a construction method can be applied to many product components, and at the same time, a product component can be constructed using many construction methods. On the other hand, the association relationship between *Construction Method* and *Component State* objects is one-to-many, since a *Component State* is defined by only one *Construction Method*.

The *Construction Requirement* class abstracts construction knowledge and project constraints imposed on the project. Construction requirement objects have three main attributes: *Purpose*, *Operator*, and *Necessary Condition* as defined in the construction requirement model. *Construction requirement* class has two sub-classes: *Functional Requirement* class for representing functional requirements and *Non-functional Requirement* class for non-functional requirements. Accordingly, the *Purpose* and *Operator* attributes are inherited as *Function User* and *Function Provider* in the functional requirement sub-class. The *Necessary Condition* attribute is



elaborated into *Function Type* and *Provider Co-functionality* in the former, while as *Temporal*, *Topological* and *Measurable* constraints in the later. In essence, the *Functional Requirement* objects have a *Provider Co-functionality* attributes to capture the relationship among the providers involved in the requirement. More detailed description on formalizing functional requirements will be provided in chapter four. In general, a *Construction Requirement* object can be associated with one or many *Component States* and/or *Activities*, forming a many-to-many association relationship between *Construction Requirement* and *Component State* and *Activity* classes. Moreover, since a construction requirement can be converted into a set of temporal constraints, the relationship between *Construction Requirement* and *Temporal Constraint* objects is one-to-many.

The *Activity* class represents the construction processes required for the project. An *Activity* object is distinguished by its *Name*, and has three core temporal attributes: *Duration*, *Start*, and *Finish*. Especially, an *Activity* object can be constituted by other *Activity* objects, which are captured using the *Decomposition* attribute. The temporal relationships among activities are captured by the *Temporal Constraint* class. Generally, a *Temporal Constraint* object defines an interval-to-interval relationship between two time intervals. Accordingly, it is abstracted with five key attributes: *Relationship Type* specifying the constraint type (such as *Before*, *Start*, *Finish*, etc.), *Preceding Interval*, *Succeeding Interval*, *Lag*, and *Lag Type* (minimal or maximal). Finally, the *Schedule* class incorporates activities and temporal constraints for schedule computation. A *Schedule* object consists of an *Activity List*, a *Constraint List*, a *Start Date* and a *Makespan*.

Overall, the integrated information framework proposed in this section allows for the unambiguous formalization and incorporation of construction knowledge in the form of construction methods and requirements. The association relationship among the core seven data classes also forms the foundation for linking three main perspectives: product, construction knowledge, and process, so that inference and reasoning mechanisms for automated sequencing and scheduling based on construction knowledge can be performed.

### 3.3. Generalized Framework for Automated Scheduling from Construction Requirement (ASCoRe)

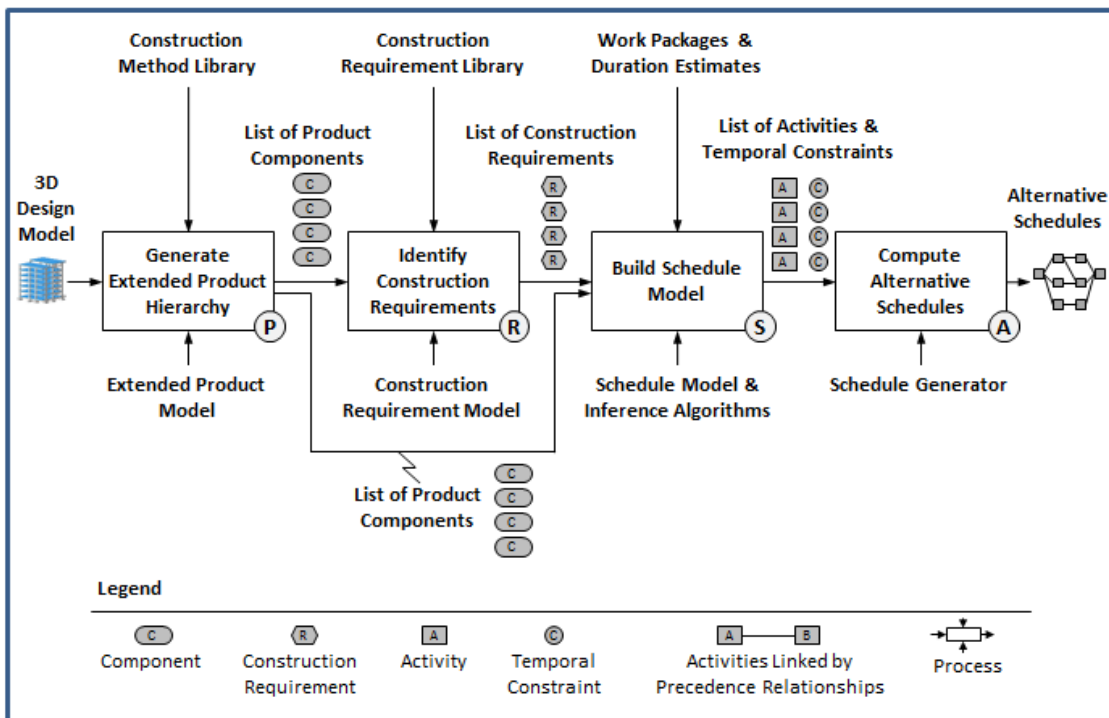


Figure 3.10. IDEF0 representation of the ASCoRe framework

Figure 3.10 depicts a generalized framework for Automated Scheduling from Construction Requirements (ASCoRe). This framework comprises four kernel inference and reasoning processes necessary for an automated scheduling system. The

scheduling process begins with process “P” (for product) which aims to create an extended hierarchy of product components using 3D design models and construction methods defined in a method library. Output of this process is a product component list organized in a hierarchical structure using the extended product model.

The product component collection obtained from process “P” is used by process “R” (for requirement) to identify construction requirements imposed on the project. Common construction requirements can be inferred from basic requirements stored in a library, and represented using the construction requirement model. Subsequently, the list of requirements obtained from process “R”, the product component list, defined work packages and production estimates are input to process “S” (for schedule) to create the schedule model. This network contains a list of activities with associated durations and a list of temporal constraints defining the precedence relationships among activities. Finally, activity and constraint lists are input to process “A” (for alternative scheduling) to compute for alternative schedules. This process is facilitated by a set of inference and computation algorithms embedded in a schedule generator. The output of the entire scheduling is a set of alternative schedules fulfilling the imposed construction requirements while also optimizing the project makespan if such a schedule exists.

### **3.3.1. Process P: Generating Extended Product Hierarchy**

This process transforms graphical project description to data representation. Figure 3.11 depicts its three main procedures which sequentially refine an arbitrary 3D design model into a standard structure, incorporate the refined design with construction method, and generate an extended collection of product components.

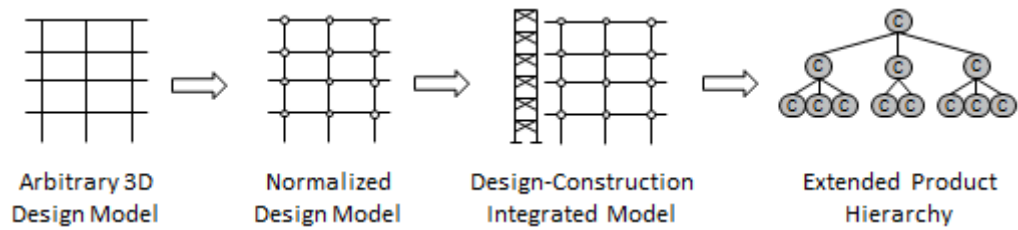


Figure 3.11. Procedure for creating an extended product hierarchy

### 3.3.1.1. Normalizing Design Model

A standard 3D design model is an important input for model-based scheduling. However, 3D design models do not have a clear definition of components due to different modeling practices. A building element may be modeled as a combination of multiple standard components. For instance, a designer may draw a column from ground slab to roof, while another may draw columns for each floor only. Similarly, in some designs, a multi-span beam is modeled as one beam from the first supporting column to the last one, or it can be divided into multiple beams, each of which corresponds to one span. Such differences will lead to ambiguous recognition of components and their functionality behaviors. Therefore, design elements need to be decomposed into a standard granularity level. For a clear and accurate functionality representation and analysis, this research adopts the component definition from the structural analysis perspective, in which components are defined based on their structural joints. For example, beams must be decomposed into single-span beams, slabs are defined by its supporting beams, and so on. Besides, design mistakes such as wrong connections, and design elements that are not necessary for planning such as annotations or comments can also be removed in this process.

The normalization procedure also facilitates the reasoning about functional relationships among components. Especially, the most common functional dependency

“support” can be inferred based on topological relationships and component type. For example, a column supports a beam if it is connected to the beam at its upper end. Other types of functional relationships like “*suspend*”, “*protect*” or “*balance*” need to be specified directly by engineers or planners.

### ***3.3.1.2. Creating Design-Construction Integrated Model***

Construction knowledge is incorporated into normalized design models by assigning construction methods to product components. This assignment enables the automated generation of component state chains. For instance, if column is linked with an aggregate construction method “Cast-in-situ” = [Rebar, Concrete, Curing], its state chain is defined by the elementary methods constituted in the aggregate method and can be derived as [Rebar.A, Rebar.Q, Concrete.A, Curing.A, Completed.Q]. When multiple methods are assigned to a component for considering choice of methods, their corresponding state chains will be generated accordingly. Moreover, multiple methods can be assigned to one component. In this case, a component may have multiple state chains, each of which is corresponding to one method.

The requirement of temporary structure defined in construction methods is used as a guideline for planners to identify temporary structures for the project, and they can decide if it is necessary to add these structures into the 3D model. Especially, temporary components will be automatically added to the product collection after a construction method is assigned to a permanent product, and a functional relationship between it and the permanent product is also set up. When different methods applied to a permanent component require a same type of temporary structure, only one temporary component of the common type will be added to the product collection to avoid generating unnecessary components. The state chain of temporary components

can also be defined through the methods applied to the components. However, a default state chain of [Erect.A, Erect.Q, Dismantle.A] can be assigned to a temporary component if no construction method is explicitly assigned to it. Pre-emptively generating temporary components from method assignment helps to ensure temporary components are adequately defined. If visualizing temporary is required for spatial or structural analysis, planners have to manually insert them into the design model, and then link them to pre-generated temporary components so that functional relationships between the temporary and permanent components are retained.

In addition, key resource requirements defined in construction methods can be linked to the associated component states through the assignment of construction methods to components. In particular, key resource requirement is defined as an attribute of component states. The value of this attribute can be automatically derived from the associated construction methods.

### ***3.3.1.3. Generating Extended Product Hierarchy***

In this step, product data are extracted from normalized construction-design integrated models and structured into a hierarchical format. Data extracted should be sufficient to set up major attributes of product components as defined in the previous section, including: component category, component class, decomposition, geometry, location, functionality, and state chain.

The detailed structure of the product hierarchy can vary for different projects and should be specified by planners. By manually defining the structure of the product hierarchy, planners can control the level of detail for any part of the project based on their management strategies. For instance, they may want to elaborate the beams into

individual components at every floor level, and at the same time represent all columns in one story as one component only. However, standard structures can be predefined for basic project types. For building projects, a standard product hierarchy can be predefined according to functional component systems like piles, footings, beams, columns, etc. and floor levels. Similarly, a generic product hierarchy for bridge projects may include basic component systems such as piles, piers, beams, decks, tendons and so on.

#### ***3.3.1.4. Generating Space Entities and Spatial Interference Matrix***

When spatial requirement is considered for planning, key space elements can be included into 3D models, and a collection of space entities also can be generated within this process. Space requirements may also be automatically defined using existing approaches such as Akinci et al. (2002), Gominuka and Sadeghpour (2008), or Shih-Chung and Miranda (2008). Space entities have types which are categorized following the space utilization hierarchy model developed by Chua et al. (2010) (see Figure 2-7) which defined four major space types: Interdiction Space Element (type *I*), Dead Space Element (type *D*), Work Space Element (type *W*), and Path Space Element (type *P*). Interdiction Spaces are spaces where no product, process or resource is allowed to occupy, and typically specified for reasons of hazards or protection. Dead Spaces are generally occupied by a “permanent” physical product component such as slabs and walls. Work spaces are defined as space entities where processes are carried out, and are typically adjacent to work faces, while Path spaces are defined as entities where movement of workers, equipment and/or physical materials from an initial designated origin to the final destination takes place.

A spatial interference matrix contains information of pair-wise topological relationships is also created in this process. Topological relationships follow the classification described in **Error! Reference source not found.** They play a key role for planners to reason for a proper sequence when a spatial conflict occurs between construction processes.

### 3.3.2. Process R: Identifying Construction Requirements for Scheduling

Basic requirements can be automatically derived from product model or modified from generic requirements stored in libraries, while complex or project-specific ones need to be determined by planners.

#### 3.3.2.1. Representing Functional Requirements

Final functional requirements describe the functional relationships between permanent components in their completion stage according to the design intentions, and are equivalent to physical relationships in other planning systems. These requirements normally include completed states of permanent components. Such simplicity enables them to be automatically derived from functionality attributes of permanent components. Figure 3.12 presents simple reasoning rule for automatically generating final functional requirements from the product model.

```
If
  (the functionality attribute of component X includes:
   function: f
   component: Y
  )
Then
  (generate a functional requirement FR with:
   Function User: [Y.Complete.Q]
   Function Provider: [X.Complete.Q]
   Necessary Condition: f([X.Complete.Q],Y.Complete.Q))
  )
```

Figure 3.12. Example rule for generating final functional requirements



Similarly, simple intermediate functional requirements like those defined in construction methods can be obtained from the functionality attributes of temporary components. However, complex intermediate functional requirements may involve all component categories. The function provider of these requirements normally represents the engineering solution for the requirement which could be derived from different construction methods, and thus they are often project-specific and generally are specified directly by planners. For instance, in a basement construction, the retaining wall requires a *support* function to maintain its stability. There are two possible solutions for this requirement resulting from two construction methods: a steel shoring system and a ground anchor system, and the function provider of this requirement refers to multiple component systems.

#### ***3.3.2.2. Representing Non-functional Requirements***

For easy and rapid generation, generic non-functional requirements can also be predefined in libraries. For example, a generic safety requirement can be predefined as shown in Figure 3.13a, in which purpose refers to a generic safety requirement, and performance attributes are defined by generic construction process, and the necessary condition is represented as a temporal constraint. When this requirement is added to the project, it will be applied to all welding and painting activities, and a set of requirements can be automatically generated from this pattern. Similarly, Figure 3.13b presents an example of resource requirement. In this case, the performance attribute refers to a key resource requirement of a construction process while the performance to a generic construction process and resource type. The necessary condition is defined as a key resource requirement with an abstract constraint defining the number of resources required by the excavation process.

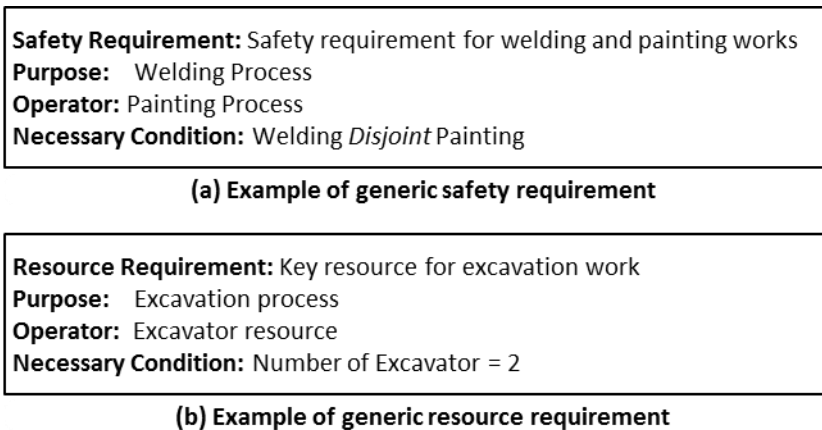


Figure 3.13. Generic non-functional requirement

### 3.3.3. Process S: Generating Schedule Model

Activities and temporal constraints are key elements of a schedule. This section describes the generalized procedure for generating them from a product hierarchy and construction requirements.

#### 3.3.3.1. Generating Activity Hierarchy

An activity refers to a construction or management process that facilitates the production of product components. Accordingly at the lowest level of detail, an elementary activity is equivalent to an active component state phase. In this framework, the term “elementary activity” is used to indicate a construction process happening on one product component. They are the core entities from which activities (in normal context) are created. Based on this equivalence, the collection of elementary activities can be directly derived from component state chains in the product model.

As illustrated in Figure 3.14, component B1 has a state chain of three active state phases Rebar.A, Concrete.A, and Curing.A linked with three construction processes Rebar, Concrete and Curing extracted from the applied methods. These processes are associated with three elementary activities, and thus the associated active state phase

can be intuitively converted into elementary activities. Especially, a quiescent state phase are converted into a  $B(0)$  relationship between its immediate precedent and succeeding active state phases. The relationship between two consecutive active state phases in a component state chain is converted into a *Meets* relationship to maintain the continuity nature of the state chain.

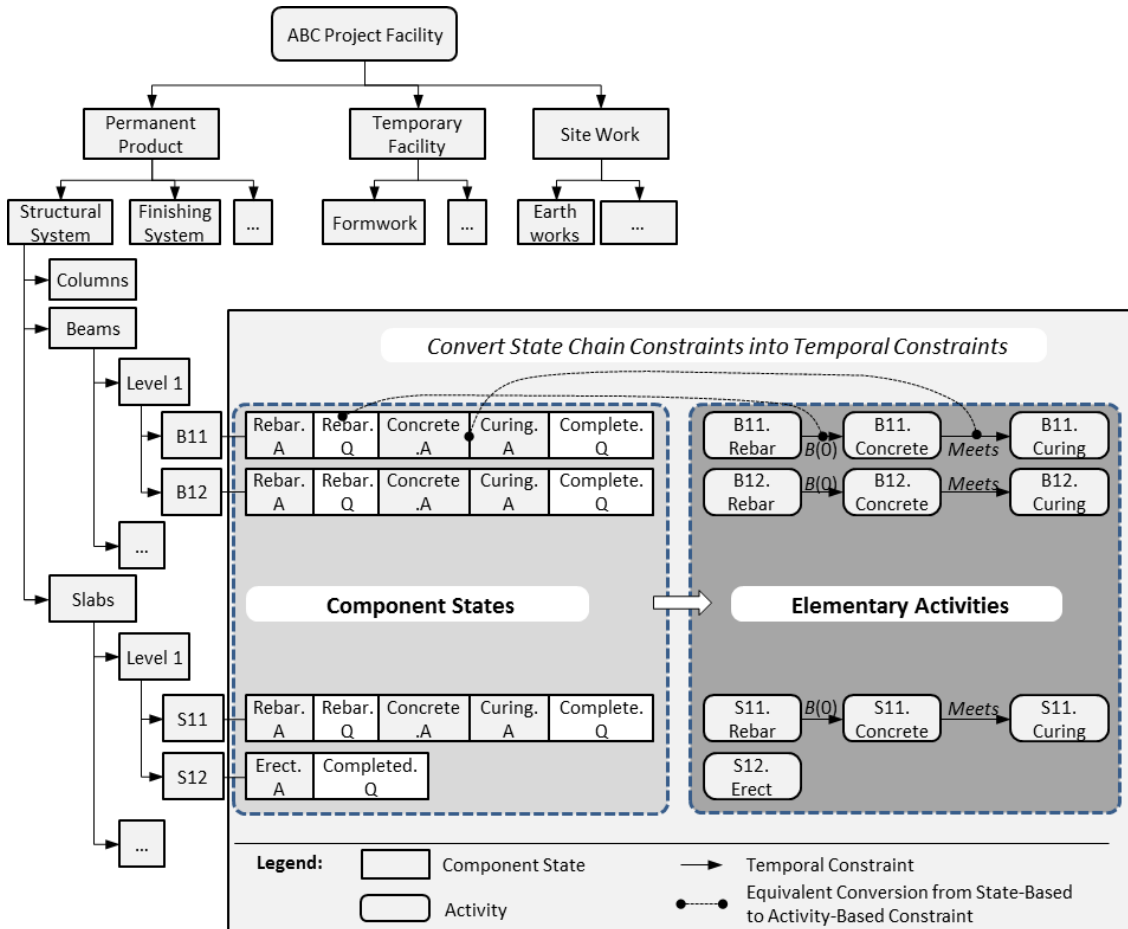


Figure 3.14. Conversion from component state chain to elementary activities

The one-to-one transformation from component states to elementary activities provides a clear link between product and process models. It also allows flexibility for updating the process model when any change occurs in the product model. For example, if a planner wants component B1 to be precast instead of cast-in-situ, the

state chain of this component will be replaced with a new state chain [Erect.A, Completed.A] describing the precast method. The elementary activities associated with the old state chain will correspondingly be replaced by new ones.

Activities at higher levels in the hierarchy are defined as combinations of those in the lower levels. Since an activity represents a construction process happening on a group of components, it is formed by the aggregation of elementary activities associated with the same construction process. By this, an activity can be considered as a work package – the amount of work produced by a construction process. Moreover, this definition of activities does not require their constituting elementary activities to be associated with components at the same level of detail. Hence, planners can have more flexibility in defining scope of work for construction processes as well as choosing different level of details for different parts of project when necessary.

As shown in Figure 3.15, the constituting elementary activities of activity Level1-Rebar refer to component states of components at different levels of detail. In particular, component B1 is a component system comprising all beams in level 1, while S11 is an individual slab belonging to a slab system Level 1. Moreover, when unnecessary for schedule computation, elementary activities can be replaced by their activities to simplify the schedule model and reduce computational effort.

A meta-activity, which is equivalent to a “meta-interval” used by Yeoh (2012) or the “summary activity” in Microsoft Project, is a contains a collection of activities of similar or different construction processes. Meta-activities are necessary for hierarchical planning through higher level abstractions of a group of activities. With this construct planner can also divide the project into sub-projects according to any

intention for better management. In addition, temporal, key resource and work space requirements can be represented at this level.

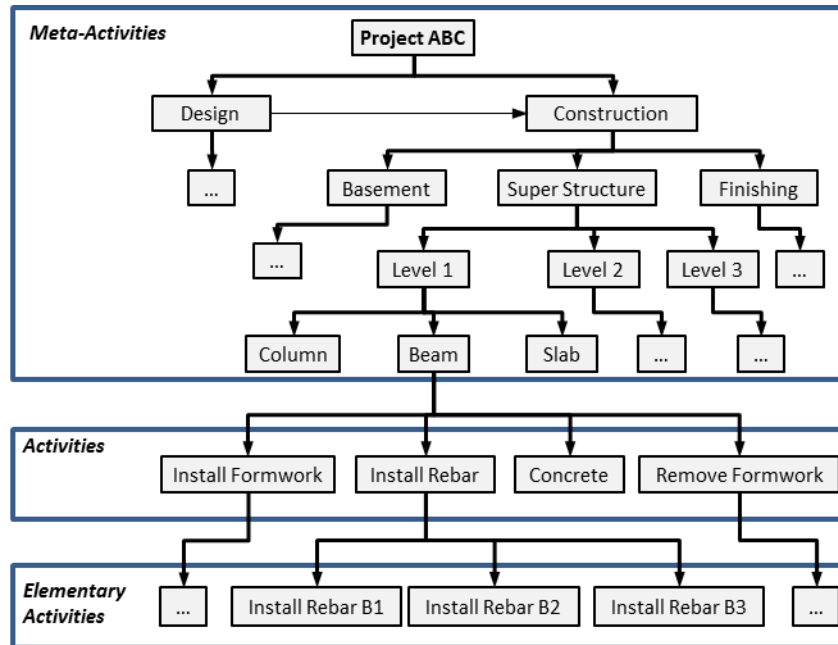


Figure 3.15. Three detail levels of a typical activity hierarchy

### 3.3.3.2. Deriving Temporal Constraints between Activities

Generating sequencing constraints could be the most difficult scheduling task, especially in this framework as they are derived from various types of construction requirement. In general, construction requirements can be defined at three levels: component states, activities, and meta-activities, and refer to four main requirements: functional dependencies, temporal relationships, space, and key resource constraints. They are converted into temporal constraints at the activity level.

Figure 3.16 depicts the approach for generating temporal constraints between activities used in this framework. Each requirement type is converted throughout three levels: component state, elementary and activity, and finally reasoned into temporal constraints. Firstly, requirements defined at component state level are converted into

those at elementary activity level. In particular, functional necessary conditions of functional requirements are transformed into temporal constraints among component states. This transformation process is facilitated by a reasoning framework called FReMAS described in chapter 4.

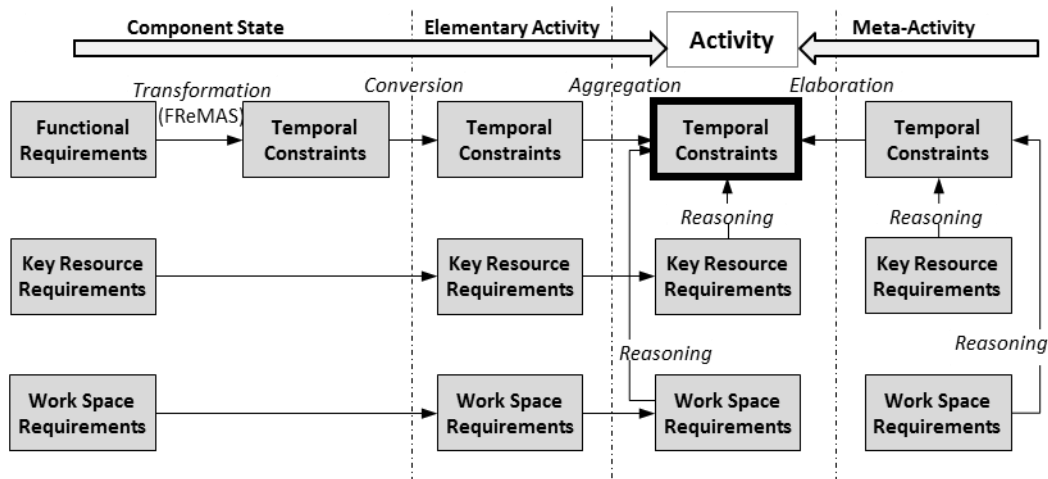


Figure 3.16. Approach for generating temporal constraints

Component state-based temporal constraints are converted into those between elementary activities based on the one-to-one relationship between an elementary activity and the active phase of a component state (as illustrated in Figure 3.17). In other words, this step will remove all quiescent state phases from the scheduling model, and transfer key resource and workspace requirements related to a component state to those of associated elementary activities. The quiescent phase between two active phases in a state chain is represented by a *Before* relationship between the elementary activities corresponding to the active phases, and the continuity constraint between two active phases is captured by relationship *Meets* between their associated elementary activities.

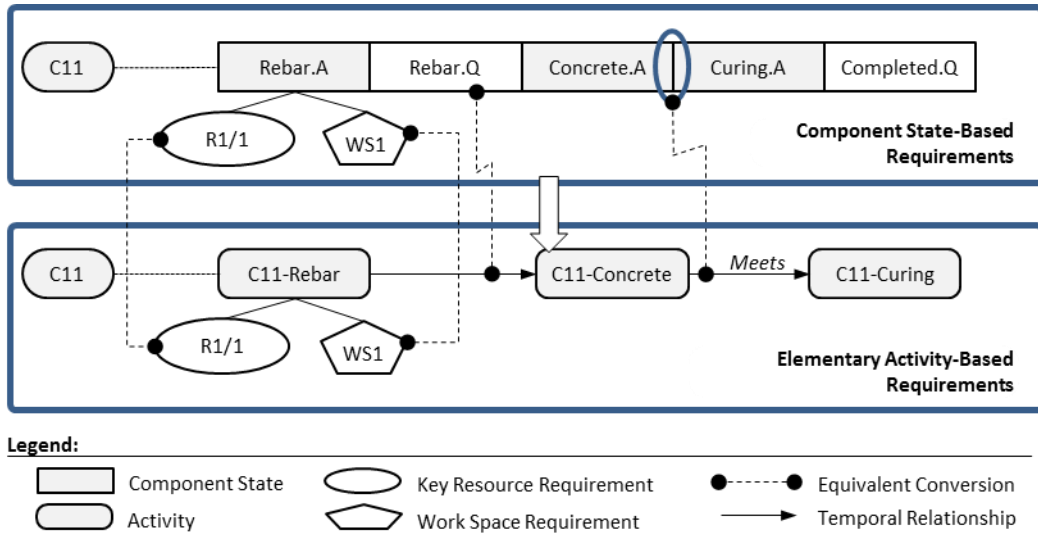


Figure 3.17. Convert requirements from component state to elementary activities

Requirements at activity level are generated by aggregating those in their constituting elementary activities. Since an activity has identical temporal attributes (duration, start and finish times) to those of its constituting elementary activities, it will “inherit” all temporal constraints in which they are involved. In other words, any temporal constraints between two elementary activities involved in two different activities will be maintained as a temporal constraint between the two activities. This conversion is supported by the assumption in which the start and finish times of all elementary activities are the same as those of their activity. Since this one-to-one conversion could result in some duplicate constraints, a constraint refining process will be applied to remove such duplications. Similarly, resource and workspace requirements are also transferred from its constituting elementary activities, and duplicate requirements will then be removed using a refining process.

As illustrated in Figure 3.18, the temporal constraints between activities A1 and A2 are derived from those among their constituting elementary activities [a11, a12, a13] and [a21, a22]. In particular, four  $B(0)$  constraints a11-a21, a12-a21, a12-a22, and

a13-a22 are combined into one, and two constraints  $SS(2)$  and  $FF(1)$  are also maintained between A1 and A2. In a similar way, the resource and space requirements of A1 and A2 are the combination of all required in their elementary activities. In addition, all resource requirements of the same resource type will be aggregated into one with maximal required value as this will subsume all requirements with smaller required numbers.

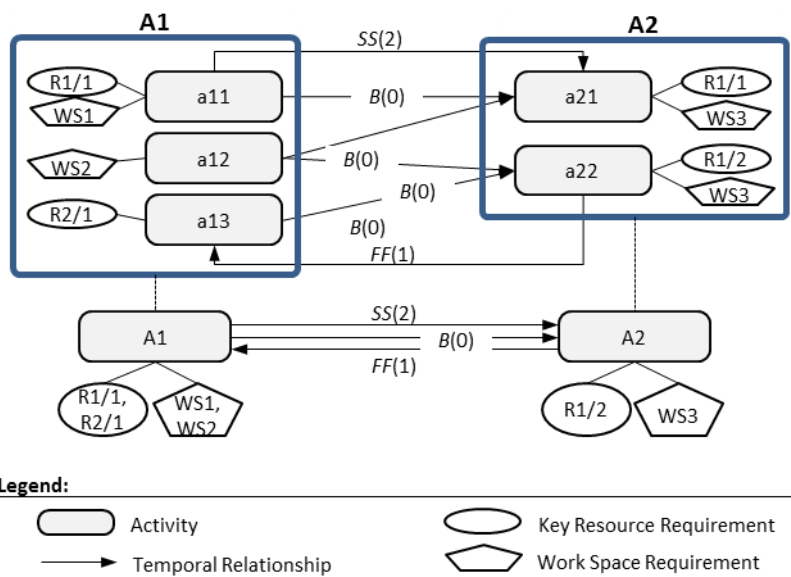


Figure 3.18. Convert requirements from elementary activity to activity levels

The obtained key resources and space requirements will then be reasoned into temporal constraints. The reasoning rules are respectively based on the number of available resources and topological relationships between space entities. For example, a typical resource reasoning rule can be defined as: “If two activities require the same resource type, and the total required amount exceeds the available amount then they must be taken place disjunctively.” If, for instance in Figure 3.18, only 2 items of resource R1 are available, then an additional *Disjoint* constraint between activities A1 and A2 will be added to the constraint collection. Similarly, a *Disjoint* constraint will



occur between two activities if their required space entities have a *conflict* relationship. The syntax and procedure of the inference mechanisms for reasoning key resource and work space requirements will be described in more detail in Chapter 5.

Requirements can also be assigned to a project at a meta-activity level, or in other words, between meta-activities. They are also need to be elaborated into requirements at activity level. Resource and spatial requirements among meta-activities are first reasoned into temporal constraints among them using the similar reasoning rules for activity level. Subsequently, temporal constraints between meta-activities are elaborated into those between their constituting activities. In brief, if a meta-activity MA1 has a simple temporal constraint  $C$  (such as  $B$ ,  $SS$ ,  $FF$ , and  $SF$ ) with another meta-activity MA2, then there will be a constraint  $C$  between each activity constituting MA1 and every activity in MA2. This inference rule is supported by the implicit temporal constraints between a meta-activity and its activities as well as the transitive attribute of temporal relationships. In fact, meta-activities are equivalent to the meta-interval concept in the PDM++ model developed by Yeoh (2012). Readers may need to refer to this reference for a more discussion on elaborating constraints at meta-activity level to those at activity level.

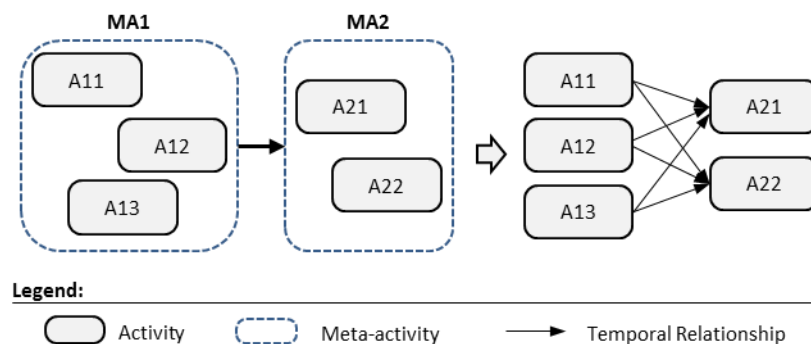


Figure 3.19. Convert temporal constraints from meta-activity to activity level

As illustrated in Figure 3.19, the precedence relationship ( $B(0)$ ) between two meta-activities representing two groups of activities can be equivalently represented by six  $B(0)$  constraints, each of which is between one activity in one meta-activity and another from the other meta-activity.

### **3.3.4. Process A: Computing for Alternative Schedules**

This final process in the ASCoRe framework is to generate all alternative schedules for the project. Inputs for this process include a list of activities and a list of temporal constraints between them. Due to the reasoning process, there may be multiple constraints between a pair of activities, some of which may be redundant while some conflicts each other. In addition, there may be many disjunctive constraints resulting from key resource requirements or alternative methods, and this could increase the problem size. Therefore, a constraint pre-analyzing process is developed to determine redundant and conflicting constraints between any pair of activities. This process will help to resolve some constraint inconsistencies and remove unnecessary disjunctive constraints, reducing computational effort. A detail description of this preemptive constraint analysis approach is presented in chapter 5.

**3.4. The scheduling problem is formulated as a constraint satisfaction problem (CSP) and constraint logic programming (CLP) is used for schedule computation. This method is selected so that a complete solution (all alternative schedules) of scheduling problems can be obtained. The outcome generated can be either types: First, if the constraint set are still inconsistent, no result is obtained. Second, when there is no conflict in the constraint set, a collection of all alternative schedules with minimal makespan are returned as output. These schedules represent alternative construction sequences leading to similar project completion time. Concluding Remarks**

This chapter has proposed a generalized framework for automated scheduling from construction methods and requirements (ASCoRe). This scheduling approach is built upon four core knowledge models: product, construction method, construction requirement and schedule. The significant advantage of these core knowledge models is that they allow construction requirements to be flexibly and explicitly captured and incorporated into scheduling. Such a clear and unambiguous elicitation of construction requirements is also essential for schedule analysis and management. Especially, it allows for the identification of critical requirements and their impact upon schedules when changes happen.

The ASCoRe approach, which generalizes model-based scheduling techniques, consists of four fundamental processes: (P) to generate an extended product hierarchy, (R) to identify main construction requirements, (S) to create a schedule model by generating activities and temporal constraints, and finally (A) to compute for alternative schedules. With these processes, the ASCoRe framework determines all necessary procedures for automated scheduling. Therefore, it can be applied any project types such as building, bridge or highway projects. Moreover, by using component state as the elementary construct linking the product and process

perspectives, ASCoRe facilitates both product- and process-based planning, and at different levels of detail.

Background concepts for generating activities and deriving temporal constraints from project descriptions – product model and construction requirements are also presenting in this chapter, providing the foundation for the development of reasoning and inference methodologies in the succeeding chapters. In particular, a generalized model for representing complex functional requirements and transforming them into temporal constraints is described in chapter four. This framework plays a key role for the ASCoRe framework by facilitating the adequate identification of possible construction sequences. Chapter five describes a system architectural framework and reasoning algorithms for implementing the ASCoRe approach.

# **CHAPTER 4. AUTOMATED CONSTRUCTION SEQUENCING FROM FUNCTIONAL REQUIREMENTS**

## **4.1. Introduction**

Functional requirements are a special class of construction requirements. They relate to the engineering behavior of product components. In order to incorporate functional requirements into schedule, they have to be captured and converted into temporal constraints for schedule computation.

To address the above issue, this chapter proposes a generalized Functional Requirement Model for Automated Sequencing (FReMAS). FReMAS extends the requirement model introduced in Chapter 3 and the Intermediate Function concept proposed by Song and Chua (2006) for modeling and interpreting complex functional requirements. In essence, it contains three main components: a Representation Model to formalize a functional requirement, a Temporal Model to systematically define temporal attributes of a functional requirement, and a Construction Sequence Reasoning Framework to convert its temporal attributes into temporal constraints. One primary advantage of this modeling framework is its ability to derive all construction sequences from complex functional requirements, thus efficiently facilitating the ASCoRe framework.

This chapter starts with a brief overview of the Intermediate Function concept to provide readers with necessary understanding of the background of FReMAS. It then proceeds with the descriptions on three main components of FReMAS and ends with case study demonstrating its application into automated scheduling.

## 4.2. Modeling Perspectives of a Functional Requirement

The present research employs and modify the concept of Intermediate Function proposed by Song (2006) to produce a more generic and flexible representation schema, facilitating the generating, updating, and reasoning both intermediate and final functional requirements for scheduling purpose.

The Intermediate Function concept captures an intermediate functional requirement from three perspectives: user (purpose), provider (operation), and the interaction relationship between them. From a purposive aspect, a functional requirement refers to a “functionality demand” of a product component or a structure system to sustain its existing status. As such, the term “purpose” is described from a viewpoint of the user, who can select different engineering solutions to achieve his demands. Generally, functionality demands can occur during any period of time along the lifecycle of a product component, in both construction and service stages. In construction stage, a product component may demands various intermediate functionalities to sustain its status changing along with the construction progress. These intermediate functionality demands also vary accordingly. For example, in addition to demanding of a supporting functionality through its construction period, a cast-in-situ concrete beam also requires a containing functionality when concrete is cast to retain its shape. In the service stage, a product component can also require certain functionalities to maintain the design intention. These requirements are defined as final functionality demand.

On the other hand, described from a provider viewpoint, the “operation” or “behavior” is inherent to the product, and is independent of the purposes of the potential user. In other words, as an “operation”, a functional requirement refers to the

functionality performance or behavior of a product component. Similar to functionality demand, functionality performance also varies along the lifecycle of a product component. It is also distinguished since intermediate and final functionality operations respectively refer to those performed in construction and service periods. A precast column, for instance, generally could not perform any functionality during erection; yet when erected (in the service stage), it can provide a support functionality to the connected beams.

Interaction relationship between user and provider is represented by temporal and spatial interactions. The temporal interaction is described by the *requirement time window* and *availability time window* from the user and provider perspectives respectively, while the spatial interaction is evaluated based on the spatial-temporal relationship between *user space* and *provider space*. These interactions allow constructability conflicts in a schedule solution to be identified.

The concept of Intermediate Function provides a systematic approach to examine the fulfillment of intermediate functional requirements for constructability analysis. However, there remain three major drawbacks making it inadequate for schedule generation. Firstly, the Intermediate Function concept defines a one to one relationship between the user and provider. In other words, a functional requirement comprises one user and one provider. This is inadequate to capture multiple complex requirements which involve multiple engineering solutions. Secondly, the temporal attributes of function user and provider are defined at an aggregated level and cannot be applied for sequence reasoning at individual user and provider level. Finally, this concept does not provide any reasoning knowledge for translating complex functional requirements into temporal constraints at component state level, the key constraints for schedule

generation. The generalized functional requirement model presented in this chapter aims to overcome these limitations by providing a generalized representation model, a detailed temporal model and a systematic sequence reasoning framework for formalizing and converting complex functional requirements into temporal constraints for schedule generation.

### **4.3. Representing Complex Functional Requirements**

Generally, the user and provider of a functional requirement may involve multiple product components. Function provider represents the engineering solution for a functional requirement. The engineering solution can possibly be derived from the applied construction method or resource usage. Practically, when multiple construction methods or resources are utilized, there are probably more than one engineering solution for a functional requirement. For example, in a basement construction, the retaining wall requires a *support* function to maintain its stability. There are two possible solutions for this requirement resulting from two construction methods: a steel shoring system and a ground anchor system. Thus, to capture these situations, the definition of function provider is extended to contain multiple providers, each of which refers to an engineering solution for the requirement. In other words, each provider represents one producer of the required functionality. A provider may involve a set of components sharing their performance to jointly produce the functionality. Each component is also specified by a set of component states during which the functionality exists.

In order to capture these special characteristics, this chapter extends the construction model presented in chapter three with two more attributes to better



describe a complex functional requirement. By this, a functional requirement is captured with four basic modeling entities, termed as: *function user*, *function provider*, *function type*, and *provider co-functionality*, as shown in Figure 4.1. The “dot” notation is used to define component state as : “Component.State.StatePhase”.

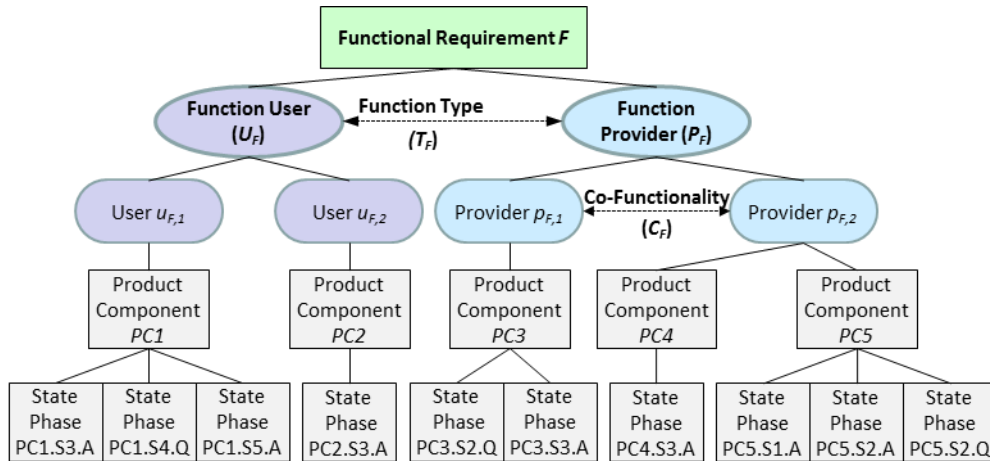


Figure 4.1. Core entities representing a functional requirement

Function user ( $U$ ) and function provider ( $P$ ) entities respectively refer the requester (or the p) and the supplier of the required functionality. The function type ( $T$ ) entity is employed to define the nature of the required functionality, such as “support”, “protect” or “balance”. The final entity – provider co-functionality ( $C$ ) refer to the interactions among the providers in the function provider. Syntactically, a functional requirement  $F$  is defined as relation of the corresponding entities as follows:

$$F = T_F(U_F, P_F, C_F) \tag{4.1}$$

### 4.3.1. Function User

*Function user* refers to all requesters which demand a similar functionality performance from the providers. As such function user may involve one or more

components, each of which is called a *user* and specified by a set of component states. Accordingly, a functional requirement with multiple users is the aggregate of the similar functional requirements of all individual users.

In Figure 4.1, the function user of the requirement  $F$  consists of two users,  $u_{F,1}$  and  $u_{F,2}$ , representing two product components  $PC1$  and  $PC2$  respectively.  $PC1$  requires for the functionality during three state phases from  $PC1.SP3$  to  $PC1.SP5$  while  $PC2$  requires that functionality during its  $PC2.SP3$  phase. The representation format of individual users and the function user in this example is described as:

$$u_{F,1} = [PC1.S3.A, PC1.S4.Q, PC1.S5.A]$$

$$u_{F,2} = [PC2.S3.A]$$

$$U_F = [[PC1.S3.A, PC1.S4.Q, PC1.S5.A], [PC2.S3.A]]$$

#### 4.3.2. Function Provider

Function provider represents the engineering solution for a functional requirement. The engineering solution could be derived from construction method or resource usage. Practically, when multiple construction methods or temporal structures are utilized, there are probably more than one engineering solutions for a functional requirement. A function provider consists of one or many providers, each of which represents one engineering solution that could resolve the functionality required by function user. A provider may involve a set of components sharing their performance to jointly produce the functionality. Each component is also specified by a set of component states during which the functionality exists.

For the example shown in Figure 4.1, there are two providers,  $p_{F,1}$  and  $p_{F,2}$  available for the functional requirement  $F$ .  $p_{F,1}$  refers to the functionality performed by

component PC3 from states  $PC3.SP2$  to  $PC3.SP3$ , while  $p_{F,2}$  refers to the engineering solution resulting from the simultaneous functionality behaviors of component PC4 during  $PC4.SP3$  and component PC5 during phases  $PC5.SP1$  to  $PC5.SP3$ . These providers are represented as follows:

$$p_{F,1} = [[PC3.S2.Q, PC3.S3.A]]$$

$$p_{F,2} = [[PC4.S3.A], [PC5.S1.A, PC5.S2.A, PC5.S2.Q]]$$

$$P_F = [[[PC3.S2.Q, PC3.S3.A]], [[PC4.S3.A], [PC5.S1.A, PC5.S2.A, PC5.S2.Q]]]$$

### 4.3.3. Function Type

The Function Type entity is used to capture descriptive information about the nature of the required functionality. As such, its major use is for distinguishing the nature of the required function. Some examples of function type taken from literature are: *support*, *suspend*, *hold*, *contain*, *protect*, *balance* and *generate*.

### 4.3.4. Provider Co-Functionality

The interaction among different engineering solutions presented by providers in a functional requirement is termed *provider co-functionality* in the context of this research. When only one construction method or resource can be used for the requirement, only one engineering solution can be applicable at any time. This leads to a mutually exclusive relationship among them. Consequently, the associated providers are also mutually exclusive. In this case, the functioning interaction is classified as *type E*. For the above basement construction example, the two methods are mutually exclusive, and thus the co-functionality between the related providers is defined as *type E*. On the other hand, if the construction methods can be used regardless of the existence of the others, all engineering solutions can be jointly used for the required

functionality. As such, all providers can share their performance for joint functionality. In this case, the functioning interaction is classified as *type C* where all the providers are compatible and can be jointly applied for the requirement.

Examining the co-functionality of providers is necessary for scheduling since it can impact the schedule results. When mutually exclusive, only one engineering solution or provider can be used at one time to satisfy the requirement. In contrast, when mutually compatible, all engineering solution could be combined to jointly perform the required functionality so that the project completion time can be enhanced.

#### 4.3.5. Illustrative Example

Figure 4.2 presents the state chains of six components: Cast-in-situ walls W1 and W2, precast beams B1 and B2, scaffolding system SC1 used for beam installation and the earthwork component TR1.

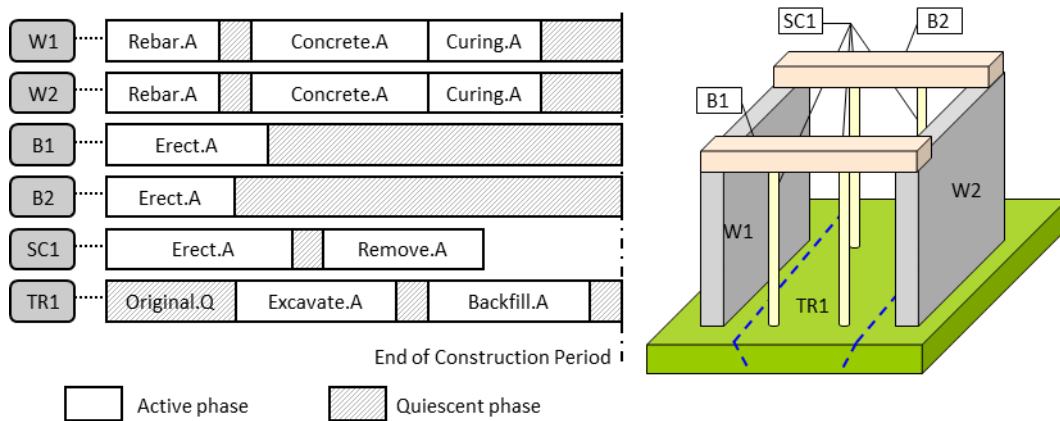


Figure 4.2. Example component state chains and functional requirements

Some functional requirements among these components are captured using the representation model and shown as follows:

- Requirement F1: Beams B1 and B2 need a function support from walls W1 and W2 after they are erected. This final functional requirement consists of two users – B1 and B2, both during their Erect.Q phases. The support function is provided by one provider comprising two components W1 and W2 during their Curing.Q phases. The [-] is used as a list notation. In addition, the provider co-functionality of a single-provider requirement is defined as type E. Accordingly, this requirement is shown as:

$$F1 = \text{support}([B1.Erect.Q, B2.Erect.Q], [[W1.Curing.Q, W2.Curing.Q]], E)$$

- Requirement F2: Scaffold SC1 needs a support function during all its construction lifecycle. This supporting function is provided by the trench TR1 in its either Original.Q or Backfill.Q states. Consequently, this requirement is formalized as:

$$F2 = \text{support}([SC1.Erect.A, SC1.Erect.Q, SC1.Remove.A], [[TR1.Original.Q], [TR1.Backfill.Q]], E)$$

- Requirement F3: B1 and B2 need to be supported during their Erect.A states by scaffold SC1 within its Erect.Q state, and/or by walls W1 and W2 during their Curing.Q states. As such, this requirement involves two providers of type C, expressed as:

$$F3 = \text{support}([B1.Erect.A, B2.Erect.A], [[SC1.Erect.Q], [W1.Curing.Q, W2.Curing.Q]], C)$$

#### **4.4. Modeling Temporal Attributes of a Functional Requirement**

Temporal attributes of a functional requirement are described by the temporal attributes of the function user and provider. They are formed from the temporal interval of the component states phases involved. Determining these attributes is necessary for sequence reasoning as they are the link between product and process

perspectives. Subsequently, the present study develops a framework for a systematic presentation of these attributes. The framework is built on two levels: (1) User/Provider Level to capture the temporal attributes of individual user/provider, and (2) Function Level to derive the aggregate temporal attributes of multiple users/providers.

#### 4.4.1. Temporal Attributes of User and Provider

At the User/Provider level, the temporal attributes of a functional requirement are represented by those of individual users and providers. These attributes refer to the duration during which a user requires the functionality, or a provider can provide the required functionality.

##### 4.4.1.1. Temporal Attribute of a User

The temporal attribute of a user is defined by a time window called *User Requirement Time Window* ( $RTW^U$ ). It is the time window during which the functionality is required by the user.

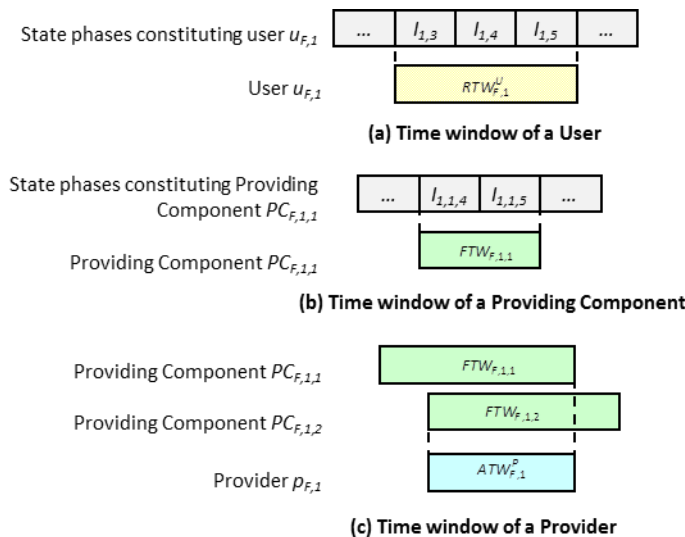


Figure 4.3. Time windows of individual User and Provider

As shown in Figure 4.3a, user  $u_{F,1}$  contains three component states. As the functionality is needed throughout three states, the  $RTW^U$  of  $u_1$  contains the combination of these three state intervals. Mathematically, the  $RTW^U$  of a user  $i$  of a functional requirement  $R$  denoted as  $RTW_{F,i}^U$  is the union of all component state intervals ( $I_{i,j}$ ) shown as:

$$RTW_{F,i}^U = \bigcup_i (I_{i,j}) \quad \forall I_{i,j} \in u_{F,i}, \forall u_{F,i} \in U_F \quad (4.2)$$

#### 4.4.1.2. Temporal Attribute of a Provider

The temporal attribute of a provider is also represented by a time window during which the provider can produce the required functionality. It is called *Provider Availability Time Window* ( $ATW^P$ ). As a provider may contain multiple components, its  $ATW^P$  is defined by the time windows during which the constituting components perform the required functionality. These time windows are called *Function Time Window* ( $FTW$ ) and are specified by the involved component state intervals. For the example in Figure 4.3b, component  $PC_{F,1,1}$  of provider  $p_{F,1}$  can perform the functionality during 2 states  $I_1$ , and  $I_2$ . Hence, its  $FTW$  is the combination of these state intervals. In terms of set operation, with regards to a requirement  $F$ , the  $FTW$  of a component  $PC_{F,j,k}$  constituting a provider  $p_{F,j}$  is the union of all the component state intervals ( $I_l$ ), expressed as:

$$FTW_{F,j,k} = \bigcup_l (I_l) \quad \forall I_l \in PC_{F,j,k} \quad (4.3)$$

To produce the required functionality, all components in a provider have to share their functionality performances. Thus, their  $FTWs$  must simultaneously coexist so that the  $ATW^P$  of a provider results from the joint existence of all  $FTWs$ . As shown in

Figure 4.3c, provider  $p_{F,1}$  contains two components  $PC_{F,1,1}$  and  $PC_{F,1,2}$  with two  $FTWs$ :  $FTW_{F,1,1}$  and  $FTW_{F,1,2}$  respectively, and the time window during which the required functionality is available is the intersection of  $FTW_{R,1,1}$  and  $FTW_{R,1,2}$ . Therefore, the  $ATW^P$  of a provider  $j$  of a functional requirement  $F$  ( $ATW_{F,j}^P$ ) must be the intersection of all  $FTWs$  of the constituting components, shown as:

$$ATW_{F,j}^P = \bigcap_k (PC_{F,j,k}) \quad \forall PC_{F,j,k} \in p_{F,j}, \forall p_{F,j} \in P_F \quad (4.4)$$

#### 4.4.2. Temporal attributes of Function User and Function Provider

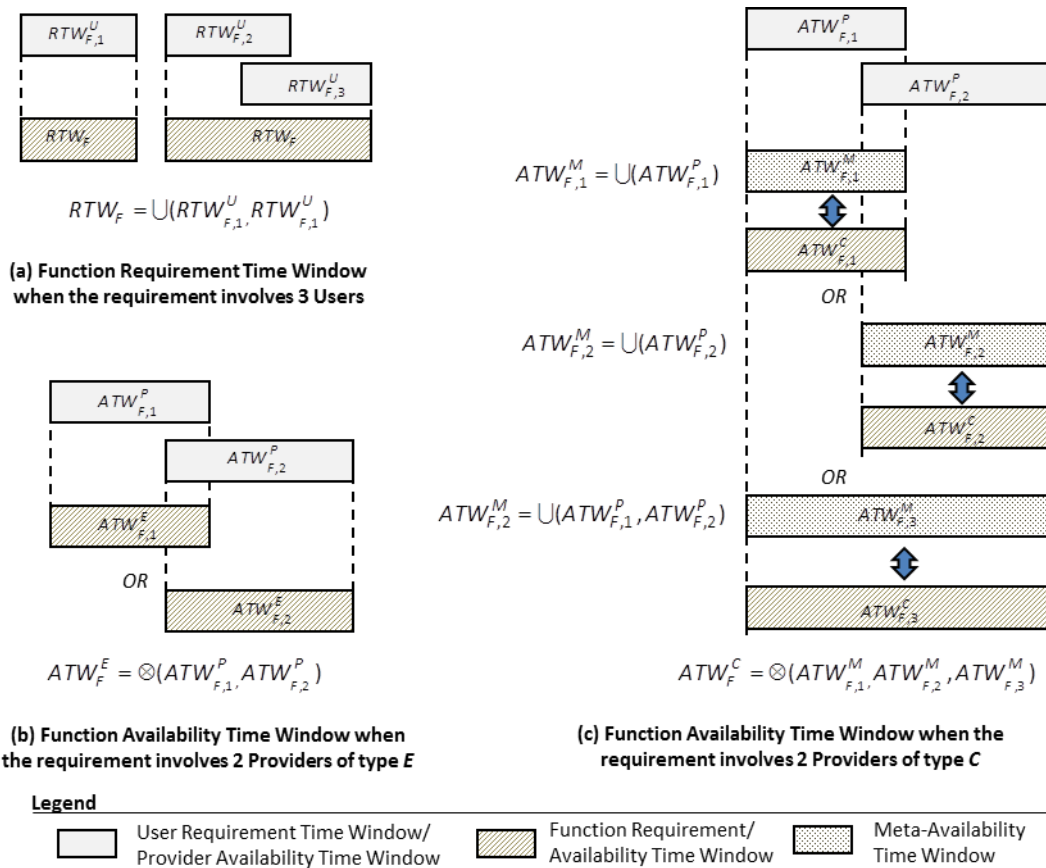


Figure 4.4. Time windows of function user and function provider



At the function level, the temporal attributes of a functional requirement are described by those of its two parties: function user and function provider. When there are multiple users/providers, the temporal attributes of the function user/provider are represented by the aggregate time windows of all users/providers. They are called *Function Requirement Time Window (RTW)* and *Function Availability Time Window (ATW)* respectively.

#### **4.4.2.1. Temporal Attribute of the Function User**

$RTW_F$  is the time window during which the function  $F$  is required by any one of its users. For example, the function user shown in Figure 4.4a contains three users with three  $RTW^U$  intervals.  $RTW_F$  is the combination of all  $RTW_F^U$  intervals, given by the union of all  $RTW_F^U$  as:

$$RTW_F = \bigcup_i (RTW_{F,i}^U) \quad \forall u_{F,i} \in U_F \quad (4.5)$$

#### **4.4.2.2. Temporal Attribute of the Function Provider**

$ATW_F$  (denoted as  $ATW_F^E$  and  $ATW_F^C$  for functioning interaction types  $E$  and  $C$  correspondingly) is the time window during which the required function  $F$  can be provided by the providers. This attribute is determined by the co-functionality nature among providers.

##### **a) Provider Co-functionality Type E**

When all providers are mutually exclusive, only one provider can be used provide the required functionality at any time. As such, although multiple providers can perform the functionality, only one of them is the engineering solution for the

requirement. Consequently,  $ATW_F^E$  is equal to any time window of an individual provider  $ATW_F^P$ . Figure 4.4b shows a functional requirement  $R$  with two providers  $p_{F,1}$  with  $ATW_{F,1}^P$  and  $p_{F,2}$  with  $ATW_{F,2}^P$ . When these two providers are mutually exclusive, they cannot share their time windows to jointly produce the functionality. Thus, the time window of the function user can be formed from either  $p_{F,1}$  or  $p_{F,2}$ , showing as:  $ATW_F^E = ATW_{F,1}^P$  or  $ATW_F^E = ATW_{F,2}^P$ . This aggregation rule is generally expressed as follows:

$$ATW_F^E = \bigotimes^j (ATW_{F,j}^P) \quad \forall p_{F,j} \in P_F \quad (4.6)$$

The relation  $= \bigotimes^j$  in Equation (4.6), in the context of this study, represents the mutually exclusive equality, defined as:

$$a = \bigotimes^j (b_j) \quad \forall b_j \in [b_1, b_2, \dots, b_n] \Leftrightarrow (a = b_1) \vee (a = b_2) \vee \dots \vee (a = b_n) \quad (4.7)$$

#### ***b) Provider Co-functionality Type C***

In the case of co-functionality type  $C$ , providers can share their functionalities or time windows to jointly provide the required functionality. They can be combined in various ways to form new providers which can possibly fulfill the requirement. The combination of providers is called a *meta-provider*. From construction perspective, meta-providers represent different patterns of combining engineering solutions being considered. Since only one combination can be applied in a planning scheme, all meta-providers are mutually exclusive.

The temporal attribute of a meta-provider is represented by its time window called *meta-Availability Time Window* ( $ATW^M$ ). As each meta-provider is a combination some providers, its time window is also the combination of the time windows of all constituting providers. For the example in Figure 4.4c, meta-provider  $MP_{F,1}$  includes only provider  $p_{F,1}$ ; thus its  $ATW_{F,1}^M$  is equal to  $ATW_{F,1}^P$ .  $MP_{F,3}$  however involves both  $p_{F,1}$  and  $p_{F,2}$ ; therefore  $ATW_{F,3}^M$  is the joint of two time windows  $ATW_{F,1}^P$  and  $ATW_{F,2}^P$ . Generally,  $ATW_{F,m}^P$  is the union of all  $ATW^P$  of providers given by

$$ATW_{F,m}^M = \bigcup_n (ATW_{F,n}^P) \quad \forall p_{F,n} \in MP_{F,m}, \quad \forall MP_{F,m} \in MPS_F \quad (4.8)$$

where  $MPS_F$  refers to the meta-provider collection of the functional requirement  $F$ .

In addition, since each meta-provider represents a combination of providers, the collection of all meta-providers refers to all possible provider combinations that can be generated from the function provider. In other words,  $MPS_F$  is the power set of  $P_F$  excluding the empty set, given by:

$$MPS_F = \mathcal{P}(P_F) \quad (4.9)$$

where  $\mathcal{P}(S)$  represents the power set of set  $S$  excluding the empty set (denoted as []).

Moreover, as all meta-providers are mutually exclusive, similar to the case of mutually exclusive providers, the aggregate time window of the function provider,  $ATW^C$ , equals to only one  $ATW^M$  at any time. The mathematical definition of  $ATW^C$  is shown in Equation (4.10), where  $MSP_F$  refers to the collection of all possible meta-providers of the functional requirement  $F$ .

$$ATW_F^C = \bigotimes^m(ATW_{F,m}^M) \quad \forall MP_{F,m} \in MPS_F \quad (4.10)$$

It can be further noted that  $ATW_F^E$  is a special case of  $ATW_F^C$  in which all meta-providers only involve one provider. The difference between two cases is the collection of meta-providers. Under the scenario of provider co-functionality type  $E$ , each meta-provider has only one provider. As such, the meta-provider collection in this case is defined as:

$$MPS_F = \mathcal{P}_{[1]}(P_F) \quad (4.11)$$

with  $\mathcal{P}_{[n]}(S)$  denoting the set of all subsets of  $S$  consisting of  $n$  elements.

In fact, the meta-provider collection of a type  $E$  requirement is exactly similar to the provider collection. Consequently, the definition of  $ATW$  can be generalized for both provider co-functionality type  $E$  and  $C$  as:

$$ATW_F = \bigotimes^m(ATW_{F,m}^M) \quad \forall MP_{F,m} \in MPS_F \quad (4.12)$$

## 4.5. Sequence Reasoning Framework from Functional Requirement

For scheduling, functional requirements must be converted into temporal constraints. The sequence reasoning framework presents a method to automate this conversion. It incorporates reasoning knowledge to translate the necessary condition from functional to temporal constraints using the proposed  $RTW/ATW$ . These constraints which are often represented as disjunctive constraints among mutually exclusive providers/meta-providers cannot be modeled using traditional CPM/PDM models. Thus, the PDM++ Framework developed by Chua and Yeoh (2011) is

employed to represent the complex temporal constraints. The reasoning framework comprises three levels: (1) Requirement Level, (2) Function Level, and (3) User/Provider Level, demonstrating the necessity conditions between user and provider so that the requirement is fulfilled.

#### 4.5.1. Necessary Condition at Requirement Level

Generally, a functional requirement is satisfied if and only if the required functionality is available at any time during the requirement period of all users. It can be inferred that, to ensure a functional requirement fulfilled, its *ATW* must subsume its *RTW*. In terms of temporal interval relationship, this satisfaction condition can be modeled using the *Contains*, expressed as:

$$ATW_F \text{ Contains } RTW_F \quad (4.13)$$

In a general case, the  $ATW_F$  is exclusively equal to only one of  $ATW_{F,m}^M$  at any time. By applying Equation (4.12) to (4.13), the following constraints are obtained:

$$\bigvee_m (ATW_{F,m}^M \text{ Contains } RTW_F) \quad \bigvee_m MP_{F,m} \in MPS_F \quad (4.14)$$

Consequently, the original necessary satisfaction condition is elaborated into a set of temporal constraints, each of which represents a constraint between each meta-provider (or each provider in the case of functioning interaction type *E*) and the function user. As such, the mutually exclusive interactions among meta-providers have been translated into disjunctive relationships among these sets of constraints using the logic operator *OR* ( $\bigvee$ ). Since each meta-provider represents an engineering solution option for the requirement, this reasoning process allows all alternative schedules resulting from these engineering solutions to be examined.

### 4.5.2. Necessary Conditions at Function Level

The necessary conditions at the Function Level refer to the temporal constraints between the function user and its constituting user, as well as those between function provider and individual providers. These constraints are essential for the reasoning process as they link high-level constraints defined at the requirement level to the basic ones determined in the User/Provider Level.

By definition, the *RTW* of a functional requirement must cover the time window of all constituting users. Following this, as shown in equation (4.5), *RTW* is defined as the union of all  $RTW^U$ . From the scheduling perspective, *RTW* must contain all constituting  $RTW^U$ , shown as:

$$\bigwedge_i (RTW_F \text{ Contains } RTW_{F,i}^U) \quad \forall u_{F,i} \in U_F \quad (4.15)$$

The *ATW* is exclusively presented by each  $ATW^M$ ; thus the relationship between *ATW* and  $ATW^P$  is equivalent to that of  $ATW^M$  and  $ATW^P$ . As each meta-provider represents a combination of providers, its availability time window must incorporate that of all providers involved in it. Therefore, similar to function user, the time window of a meta-provider,  $ATW^M$  is defined as the union of all  $ATW^P$  as shown in equation (4.8). This relationship is converted to a set of *Contains* constraints between  $ATW^M$  and all  $ATW^P$  as follows:

$$\bigwedge_i (ATW_{F,m}^M \text{ Contains } ATW_{F,i}^P) \quad \forall p_{F,i} \in MP_{F,m} \quad (4.16)$$

### 4.5.3. Necessary Conditions at User/Provider Level

The necessary conditions specified in this lowest level define the constraints between each user/provider and its constituting component state intervals. By this, the

overall necessary condition can eventually be calibrated into constraints among basic schedule elements – component state intervals. The time window of a user –  $RTW^U$  – must subsume all constituting component state intervals. This relationship is represented by a *Contains* constraint, expressed as:

$$\wedge_i (RTW_{F,i}^U \text{ Contains } I_{i,j}) \quad \forall I_{i,j} \in u_{F,i} \quad (4.17)$$

The time window of a provider is also constrained to that of its constituting product components. However, since the functionality can only be generated when all the product components involved simultaneously perform it,  $ATW^P$  of a provider is the joint of all  $FTW$ , and represented by a constraint *Contained-By* as shown in (4.18). Subsequently, the relationship between each  $FTW$  and its constituting component state intervals are captured by a constraint *Contains* as defined in (4.19).

$$\wedge_k (ATW_{F,j}^P \text{ Contained-By } FTW_{F,j,k}) \quad \forall PC_{F,j,k} \in p_{F,j} \quad (4.18)$$

$$\wedge_l (FTW_{R,j,k} \text{ Contains } I_{F,k,l}) \quad \forall I_{F,k,l} \in PC_{F,j,k}, \forall PC_{F,j,k} \in p_{F,j} \quad (4.19)$$

In summary, the reasoning process proposed in this framework allows the original functional dependency to be converted into temporal constraint between two time windows:  $RTW$  and  $ATW$  which represents the necessary condition of fulfilling a functional requirement. In addition, the mutually exclusive relationship between providers and meta-providers are captured using the logic operator “OR” ( $\vee$ ). By this, all possible sequencing options can be examined in the scheduling process, facilitating the generation of all feasible schedule alternatives.

### 4.6. Implementation of the FReMAS model

The implementation of FReMAS in *ECLiPS<sup>e</sup>* as an automatic sequence reasoning mechanism consists of two main procedures (as described in Figure 4.5): a Pre-processing Procedure (1) to normalize complex requirements into a list of simple constraint, and a Sequence Reasoning Procedure (2) to convert each normalized requirement into a set of temporal constraints.

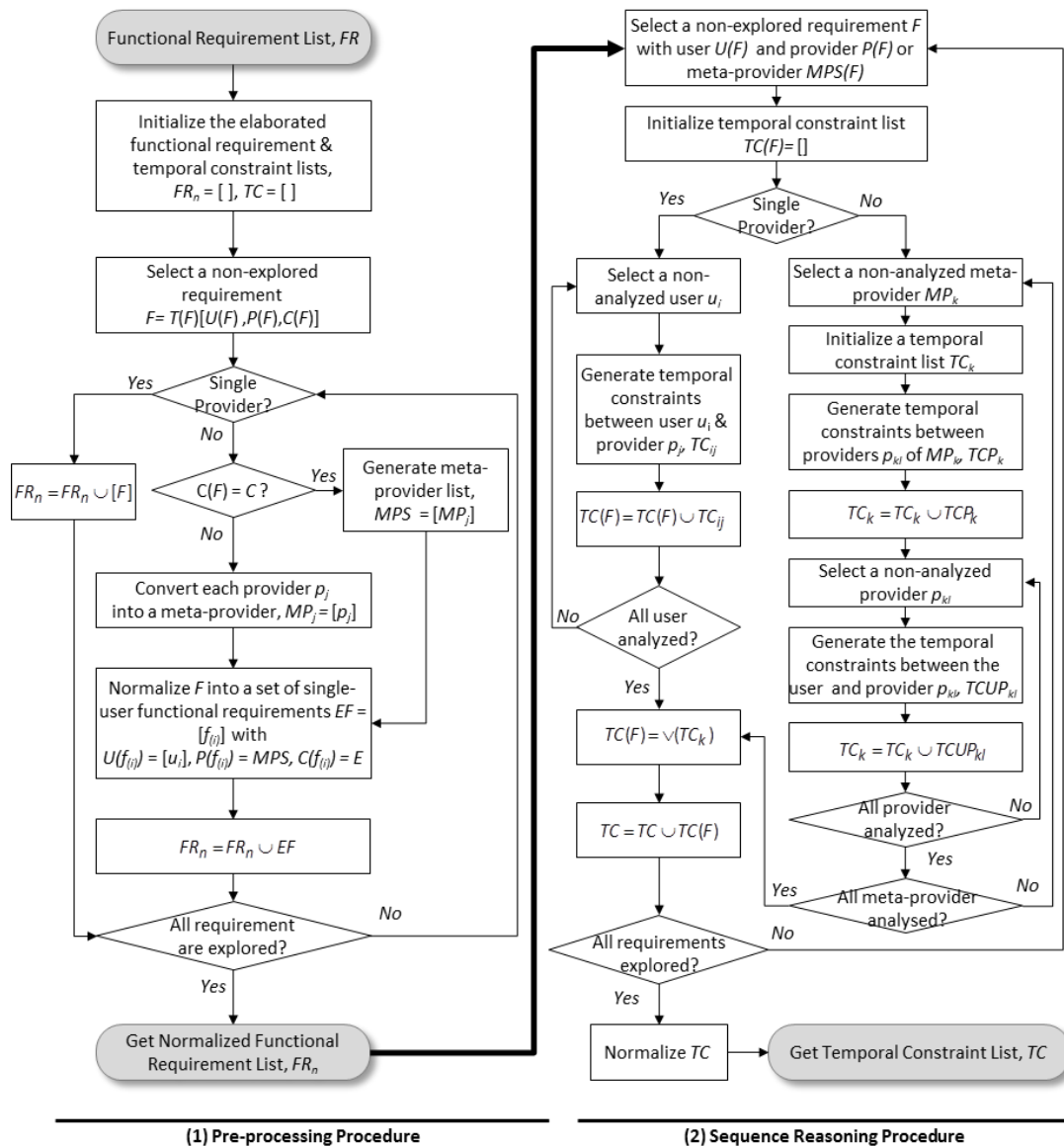


Figure 4.5. Flowchart for implementing FReMAS



In brief, the pre-processing procedure will first generate the meta-provider list of a complex requirement based on the co-functionality type. Then, if the requirement consists of multiple users, it will be replaced by a list of simple requirements, each of which comprises one user of the requirement. The final requirement list is then input to the main sequence reasoning procedure which subsequently generates a list of temporal constraint between the user(s) and provider/meta-provider(s). Finally, the temporal constraint list is normalized to become a list of disjunctive combination of conjunction constraints. The final output  $TC$  is a disjunctive combination of multiple groups of constraints ( $TC = \bigvee_i (C_i) \forall C_i \in TC$ ), represented as a nested constraint set:

$TC = [C_1, C_2, \dots, C_i, \dots, C_n]$ , where each constraint subset  $C_i = [c_{i,1}, c_{i,2}, \dots, c_{i,j}, \dots, c_{i,m}]$  denotes a conjunction combination of temporal constraint  $c_j$  ( $C_i = \bigwedge_j (c_{i,j}) \forall c_{i,j} \in C_i$ ).

The example code for implementing FReMAS in  $ECL^iPS^e$ , a Constraint Logic Programming language is described in Figure 4.6.

```
:-local struct(state(name, type, dur, start, fin)).
:-local struct(t_cons(type, aname, astate, bname, bstate, lag)).
:-local struct(f_cons(name, type, pname, ps, mps, uname, us, nop)).
:-local struct(provider(pname, pstate, start, fin, atw)).
```

```
schedule(SList, PList, CList, FList, Makespan):- states(S),
  t_constraints(TC),
  f_constraints(FC),
  get_boundary(S, TC, U),
  build_StateList(S, SList, U),
  build_TempCons(TC, CList),
  build_FuncList(FC, FList),
  build_ProList(FC, PList),
  constraints_map(SList, CList),
  provider_map(SList, PList, U),
  functions_map(SList, PList, FList),
  msolve(SList, CList, FList, Makespan),
  minimize(Makespan).
```

(a) Main predicates for schedule generation

```
msolve(SList, CList, FList, Makespan):-
  tcons_solve(SList, CList, Makespan),
  fcons_solve(FList).
fcons_solve([F|Fs]):-
  arg(us of f_cons, F, User),
  arg(mps of f_cons, F, MPS),
  arg(type of f_cons, F, Type),
  (Type = e -> pu_typeE(MPS, User);
  Type = c -> pu_typeC(MPS, User); true),
  fcons_solve(Fs).
pu_typeE([P|MPS], US):-
  pr_st_constraint(P),
  (foreach(U, US), param(P)
  do
    arg(start of provider, P, Ps),
    arg(fin of provider, P, Pf),
    arg(start of state, U, Us),
    arg(fin of state, U, Uf),
    without(Ps, Pf, 0, Us, Uf); pu_typeE(MPS, US).
  pu_typeC(MPS, [U|US]):- match_pu(MPS, U),
  pu_typeC(MPS, US).
```

(b) Main predicates for implementing FReMAS reasoning framework

Figure 4.6.  $ECL^iPS^e$  code for implementing FReMAS for automated scheduling

## 4.7. Case Study

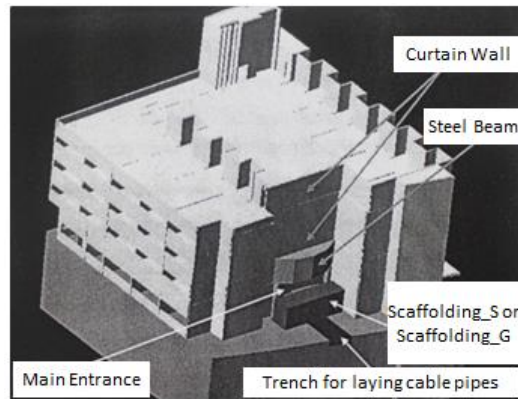


Figure 4.7. 3D model of nursing house showing main entrance

A case project is presented in this section to demonstrate how the proposed model (FReMAS) can be implemented for automated sequencing and scheduling. It involves the construction of the main entrance of a nursing house (shown in Figure 4.7) which consists of three major tasks performed by three different contractors: (1) design and construction of glass work of the curtain wall by subcontractor “SubCon\_1”, (2) design and construction of pre-fabricated steel beam by subcontractor “SubCon\_2”, and (3) laying of cable pipe by the main contractor (MainCon).

### 4.7.1. Product Components and State Chains

The components associated with the work are arranged into three groups as shown in Figure 4.8. The “PC” group contains all permanent product components that are involved in the project. The “TC” group includes temporary components, the “SC” contains the site work components, and the “IC” group refers to special components which are not product-related but information-related. Information-related components provide the necessary information for construction processes. The active phase of each state is denoted by “.A”, and any hatched phase refers to the quiescent phase of the

previous active phase. Note that the length of the states is only for presentation purpose and not related to state durations. The component state chains are generated from the construction methods applied to the components.

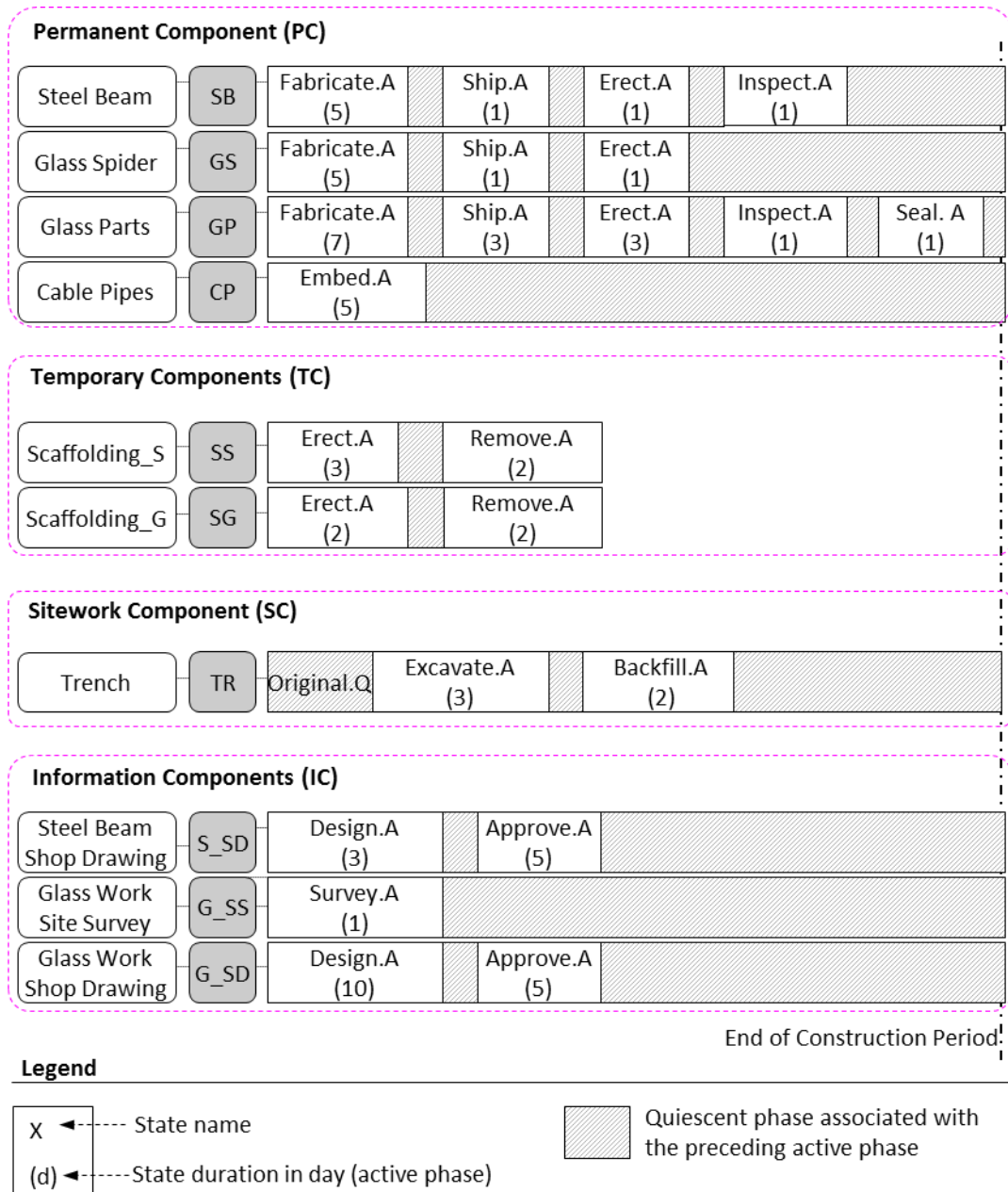


Figure 4.8. State Chains with Durations of Components Involving in the Analysis

In the “PC” group, the Steel Beam component (*SB*) is first fabricated offsite, transported to site, then erected and finally inspected before completion. The Glass Spider component (*GS*) is a part of glass work. It is also prefabricated offsite, shipped to site and erected on site. Similarly, the Glass Parts component (*GP*) is also fabricated, transported to site and erected. Then they are inspected before sealant is applied. The Cable Pipes component (*CP*) is embedded in the excavated trench.

In the “TC” group, Scaffolding\_S (*SS*) and Scaffolding\_G (*SG*) are used for the steel and glass works, respectively. They have the same state chain type corresponding to two processes: erection and removal. As these two components will leave the component system after dismantled, their final states – *Remove* – only contain the active phase.

The earthwork component named as Trench (*TR*) belongs to the “SC” group. It is firstly in its original status, then excavated for pipe installation, and finally backfilled. The *Original* state has only quiescent phase since it is not associated with any construction process.

There are three components belonging to the “IC” group. The Steel Beam Shop Drawing (*S\_SD*) and Glass Work Shop Drawing (*G\_SD*) are designed and approved before being used on site. These two processes are reflected by states *Designed.A* and *Approved.A* in the state chain. The Glass Work Site Survey (*G\_SS*) has only one state – *Survey* during which SubCon\_2 measures the as-built information of the completed steel beam for the completion of glass work shop drawing.

## 4.7.2. Formalizing Construction Requirements

In this project, there are various construction requirements governing the schedule. For illustration purpose, only the major construction requirements are described in the following subsections.

### 4.7.2.1. Functional Requirements

The following major functional requirements were identified. The  $(a, \dots, b)$  notation represents a group of consecutive states from phase  $a$  to phase  $b$ . These requirements are captured using the construction knowledge templates built upon on FReMAS described in chapter five, and summarized as follows:

**FR1.** Glass Parts need to be supported by the Glass Spider during its construction and service phases, specified as

$$\text{FR1} = \text{support}([(GP.Erect.A, \dots, GP.Seal.A)], [(GS.Erect.Q)], E)$$

**FR2.** Scaffolding\_S and Scaffolding\_G need a supporting base provided by Trench component during its original status or after the Trench is backfilled, specified as

$$\text{FR2} = \text{support}([(SS.Erect.A, \dots, SS.Remove.A), (SG.Erect.A, \dots, SG.Remove.A)], \\ [[TR.Original.Q], [TR.Backfill.Q]], C)$$

**FR3.** The design work of Steel Beam Shop Drawing needs a support function from Scaffolding\_S to collect site information for design work, and construction of Steel Beam also needs to be supported by Scaffolding\_S, specified as

$$\text{FR3} = \text{support}([(SB.Erect.A, \dots, SB.Inspect.A), (SSD.Design.A)], [[SS.Erect.Q]], E)$$

**FR4.** Erection of Glass Parts and Glass Spider need to be supported by Scaffolding\_G.

$$\text{FR4} = \text{support}([GS.Erected.A, (GP.Erect.A, \dots, GP.Seal.A)], [[SG.Erect.Q]], E)$$

**FR5.** Survey work of Glass Work Site Survey requires a `support` function from `Scaffolding_G` to collect site information, specified as

**FR5 = support([SS\_G.Conduct.A],[[SG.Erect.Q]],E)**

**FR6.** Cable Pipe needs to be enclosed by the excavated Trench, specified as

**FR6 = enclose([CP.Embed.A],[[TR.Excavat.Q]],E)**

#### **4.7.2.2. Non-functional Requirements**

Besides the above functional requirements, major non-functional requirements for the construction of these works are also identified and captured in the form of temporal constraints. These requirements refer to managerial constraints such as procurement, material inventory or information availability.

**NR1.** Fabrication of Steel Beam must start at least 3 days after steel beam shop drawing is done due to material procurement process, specified as

**NR1: S\_SD.Approve.A B(3) SB.Fabricate.A**

**NR2.** Fabrication of Glass Parts and Glass Spider must start at least 4 days after Glass Work shop drawing due to material procurement process, specified as

**NR2: G\_SD.Approve.A B(4) GS.Fabricate.A**

**NR3.** Glass Work Site Survey must finish at least 5 days before design of Glass Work Shop Drawing finishes to ensure sufficient as-built information acquired, specified as

**NR3: G\_SS.Conduct.A FF(5) G\_SS.Design.A**

**NR4.** Site survey must be done after the steel beam is fully constructed, specified as

**NR4: SB.Inspect.A B(0) GSS.Conduct.A**

**NR5.** Scaffolding\_G must be erected after Scaffolding\_S is removed due to space constraint, specified as

**NR5: SB.Inspect.A B(0) GSS.Conduct.A**

**NR6.** Steel Beam Shop Drawing must be designed after erection of Scaffolding\_S so that site information can be collected, specified as

**NR6: SS.Erect.A B(0) SSD.Design.A**

### **4.7.3. Construction Sequence Reasoning and Schedule Computation**

The functional requirements are modeled and reasoned into temporal constraints using FReMAS implemented in the Functional Requirement Sequence Reasoning Mechanism. Component state chains are transformed into precedence constraints between component states using the Constraint Transformation Mechanism. All generated and imposed temporal constraints are finally input to the schedule generator for schedule computation at component state level. This level of detail is chosen since each construction activity is associated with one component. The scheduling problem of the project portion is solved under two scenarios. Scenario 1 is the original case situation where the subcontractors do not have any collaboration. Scenario 2 examines the schedule results where collaboration in terms of resource sharing between two subcontractors is allowed.

#### ***4.7.3.1. Scenario 1 – No Collaboration between Two Subcontractors***

By minimizing the project makespan with no collaboration applied, 3 schedule solutions with similar makespans of 68 days are generated. The result of the first solution named Alternative 1.1 is shown in Figure 4.9 in the form of a Gantt chart. In this solution, the site is first used by SubCon\_1 to do the Steel Beam work (Day 1 to

Day 24), then it is occupied by SubCon\_2 for the Glass work (Day 24 to Day 58), and finally it is used by MainCon for the Pipe laying work (Day 58 to Day 68). The work sequence determined here ensures no site clashing among contractors.

The *RTW* and *ATW* of 6 functional requirements are also presented in Figure 4.9, showing that this schedule solution satisfies all requirements imposed. For the example of functional requirement FR4, its *RTW* and *ATW* are computed as follows:

$$RTW_4 = \cup(RTW_{4,1}^U, RTW_{4,2}^U) = \cup([47..48], [51..56]) = [47..48], [51..56]$$

$$ATW_4 = [26..56]$$

Since *ATW*<sub>4</sub> Contains *RTW*<sub>4</sub>, FR4 is fulfilled. The other functional requirements can be similarly verified to be satisfied.

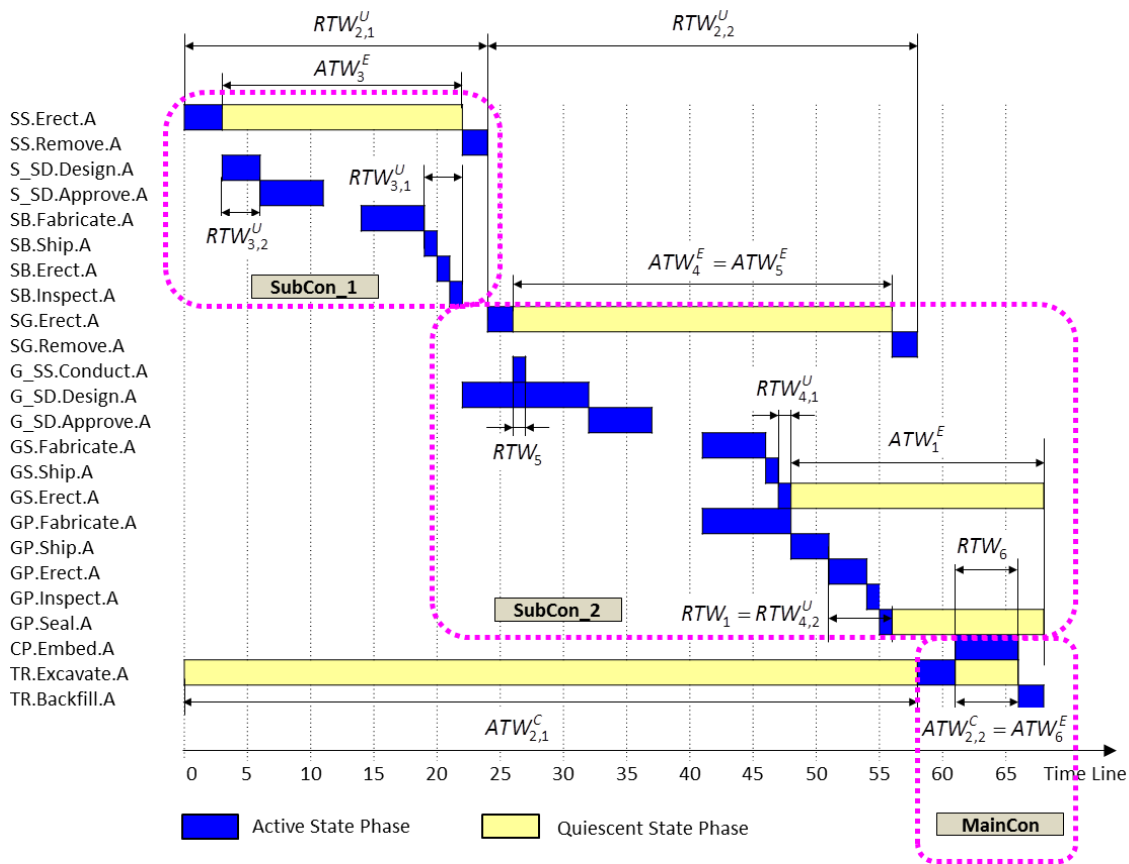


Figure 4.9. Alternative 1.1 - Scenario 1 with *RTWs* and *ATWs*



Table 4.1. Schedule solutions under Scenario 1

Component	Component State (Active Phase)	Duration (days)	Alternative 1.1		Alternative 1.2		Alternative 1.3	
			Start	Finish	Start	Finish	Start	Finish
Scaffolding_S (SS)	SS.Erected.A	3	0	3	0	3	10	13
	SS.Removed.A	2	22	24	22	24	32	34
Steel Beam Shop Drawing (SSD)	S_SD.Designed.A	3	3	6	3	6	13	16
	S_SD.Approved.A	5	6	11	6	11	16	21
Steel Beam (SB)	SB.Fabricated.A	5	14	19	14	19	24	29
	SB.Shipped.A	1	19	20	19	20	29	30
	SB.Erected.A	1	20	21	20	21	30	31
	SB.Inspected.A	1	21	22	21	22	31	32
Scaffolding_G (SG)	SG.Erected.A	2	24	26	34	36	34	36
	SG.Removed.A	2	56	58	66	68	66	68
Site Survey (G_SD)	G_SS.Conducted.A	1	26	27	36	37	36	37
Glass Work Shop Drawing (SSD)	G_SD.Designed.A	10	22	32	32	42	32	42
	G_SD.Approved.A	5	32	37	42	47	42	47
Glass Spider (GS)	GS.Fabricated.A	5	41	46	51	56	51	56
	GS.Shipped.A	1	46	47	56	57	56	57
	GS.Erected.A	1	47	48	57	58	57	58
Glass Parts (GP)	GP.Fabricated.A	7	41	48	51	58	51	58
	GP.Shipped.A	3	48	51	58	61	58	61
	GP.Erected.A	3	51	54	61	64	61	64
	GP.Inspected.A	1	54	55	65	65	64	65
	GP.Sealed.A	1	55	56	65	66	65	66
Cable Pipe (CP)	CP.Embedded.A	5	61	66	27	32	3	8
Trench (TR)	TR.Excavated.A	3	58	61	24	27	0	3
	TR.Backfilled.A	2	66	68	32	34	8	10

The summary of all 3 schedule solutions with active phase times of each component state is presented in Table 4.1. Similar to Alternative 1, the other two solutions (Alternative 1.2 and Alternative 1.3) can be verified to fulfill all identified requirements. In addition, the schedule solutions obtained represent 3 different work sequences among the contractors. The sequence in Alternative 1.1 is SubCon\_1 (Day 1 – Day 24) → SubCon\_2 (Day 24 – Day 58) → MainCon (Day 58 – Day 68). In Alternative 1.2, work sequence also starts with SubCon\_1 (Day 1 – Day 24) first, followed by MainCon (Day 24 – Day 34), and finally ends with SubCon\_2 (Day 34 – Day 68). Alternative 1.3 defines another work sequence which is started by MainCon

(Day 1 – Day 10), followed by SubCon\_1 (Day 10 – Day 34) and ended by SubCon\_2 (Day 34 – Day 68). These results also show that there is no site clashing among the contractors following these work sequences.

Furthermore, from management perspective, these schedule alternatives provide planners with different choices for a planning scheme that most suits their conditions. For example, the main contractor has 3 options to conduct his work which is at the beginning (Day 0 to Day 10), in the middle (Day 24 to Day 34) or at the end (Day 58 to Day 68) of the project portion with Alternative 1.1, 1.2, and 1.3, respectively.

#### ***4.7.3.2. Scenario 2 – Collaboration between Two Subcontractors***

In this scenario, the original project data is modified to capture the collaborative situation between the subcontractors. It is assumed that SubCon\_1 and SubCon\_2 can share their scaffolding resources with each other. This means that any functional requirements provided by either Scaffolding\_S or Scaffolding\_G can now be combined. In other words, they become compatible providers in functional requirements FR3, FR4 and FR5 so that they are modified as follows:

**FR3 = support([(SB.Erect.A, ..., SB.Inspect.A), SSD.Design.A], [[SS.Erect.Q], [SG.Erected.Q]], C)**

**FR4 = support([GS.Erect.A, (GP.Erect.A, ..., GP.Seal.A)], [[SS.Erect.Q], [SG.Erect.Q]], C)**

**FR5 = support([ms(SS\_G.Conduct.A)], [[ms(SS.Erect.Q)], [ms(SG.Erect.Q)]], C)**

With these changes, two schedule solutions with duration of 54 days are obtained as summarized in 2 showing active phase times of each component state. The domain values in the columns refer to the feasible start/finish time for each state. For example, in Alternative 2.1, the removal of Scaffolding\_S has a start interval of [25..31], meaning that this state has an early start on Day 25 and late start on Day 31. In terms

of float time, the process associated with this state has a total float of 6 days. The states with one value for start/finish time do not have float and are critical.

Table 4.2. Schedule solutions under Scenario 2

Component	Component State (Active Phase)	Duration (days)	Alternative 2.1		Alternative 2.2	
Scaffolding_S (SS)	SS.Erect.A	3	0	3	0	3
	SS.Remove.A	2	23 .. 29	25 .. 31	6	8
Steel Beam Shop Drawing (SSD)	S_SD.Design.A	3	3	6	3	6
	S_SD.Approve.A	5	6	11	6	11
Steel Beam (SB)	SB.Fabricate.A	5	14	19	14	19
	SB.Ship.A	1	19	20	19	20
	SB.Erect.A	1	20	21	20	21
	SB.Inspectd.A	1	21	22	21	22
Scaffolding_G (SG)	SG.Erect.A	2	35 .. 41	37 .. 43	18	20
	SG.Remove.A	2	52	54	52	54
Site Survey (G_SD)	G_SS.Conduct.A	1	22	23	22	23
Glass Work Shop Drawing (SSD)	G_SD.Design.A	10	18	28	18	28
	G_SD.Approve.A	5	28	33	28	33
Glass Spider (GS)	GS.Fabricate.A	5	37	42	37	42
	GS.Ship.A	1	42	43	42	43
	GS.Erect.A	1	43	44	43	44
Glass Parts (GP)	GP.Fabricate.A	7	37	44	37	44
	GP.Ship.A	3	44	47	44	47
	GP.Erect.A	3	47	50	47	50
	GP.Inspect.A	1	50	51	50	51
	GP.Seal.A	1	51	52	51	52
Cable Pipe (CP)	CP.Embed.A	5	28 .. 34	33 .. 39	11	16
Trench (TR)	TR.Excavate.A	3	25 .. 31	28 .. 34	8	11
	TR.Backfill.A	2	33 .. 39	35 .. 41	16	18

The early schedule of Alternative 2.1 is presented Figure 4.10 in the form of a Gantt chart. The time windows of functional requirements are also presented, demonstrating that condition  $ATW_F \text{ Contains } RTW_F$  is fulfilled for all functional requirements. It is also interesting to note that the collaboration helps reduce the project duration by 14 days (from 68 days to 54 days) compared with Scenario 1. By enlarging the  $ATW$  of related functional requirements, it allows some construction

processes of SubCon\_1 and SubCon\_2 to be re-sequenced for better resource usage; thereby shortening project duration.

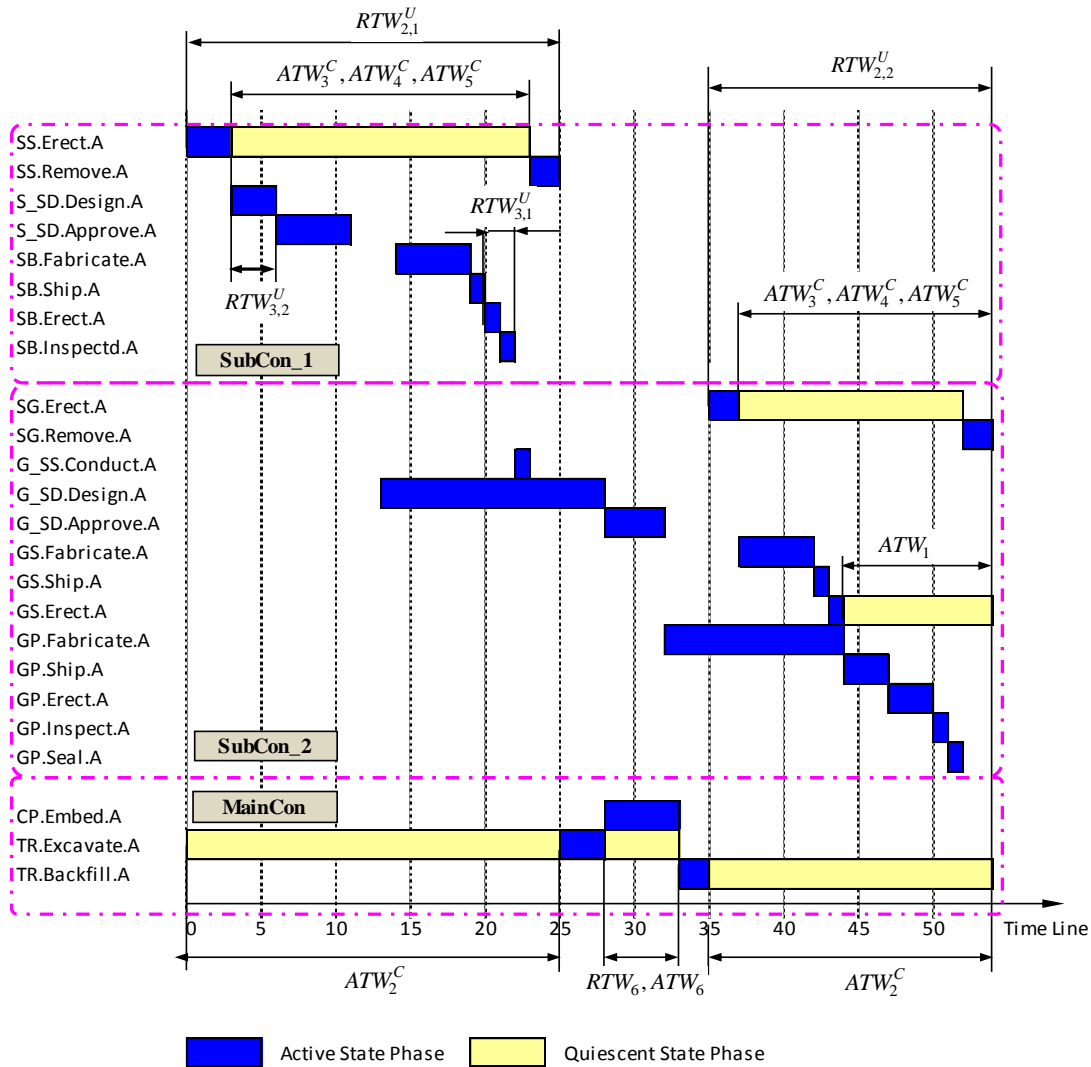


Figure 4.10. Early Schedule of Alternative 2.1 - Scenario 2 with  $RTWs$  and  $ATWs$

For an example, consider FR5.  $ATW_5^C$  contains two intervals [3..23] and [35..52] with a total duration of 37 days, longer than  $ATW^{FC}(FR5)$  in Scenario 1 with 32 days. With this extension, Glass Work Site Survey can be done (from Day 22 to Day 23) before the erection of Scaffolding\_G (Day 35) as it is supported by

Scaffolding\_S of SubCon\_1 which is available from Day 3 to Day 23. The time windows of this requirement are calculated as follows:

$$RTW_5 = [22..23]$$

$$ATW_5^C = \cup(ATW_{5,1}^P, ATW_{5,2}^P) = [3..23], [35..52]$$

Alternative 2 can be easily verified to fulfill all imposed construction requirements using a similar method. The effect of collaboration in re-sequencing the works is also found in this solution. For an example, consider FR3. Although Scaffolding\_S is dismantled from Day 6, the construction of Steel Beam of SubCon\_1 (from Day 20 to Day 22) can still be done with the support from Scaffolding\_G of SubCon\_2 which is available from Day 20 to Day 47.

Similar to Scenario 1, two schedule solutions obtained in this scenario also present different work sequences among the contractors. In the early schedule of Alternative 2.1, the site is first used by SubCon\_1 and SubCon\_2 for Steel Beam work and Glass Work Site Survey (from Day 0 to Day 25). It is then transferred to MainCon for the Cable Pipe work (from Day 25 to Day 35) and finally returned to SubCon\_2 for the Glass Work (from Day 35 to Day 54). In Alternative 2.2, the site is first used by SubCon\_1 to get information for the design of steel beam shop drawing (from Day 1 to Day 8). It is subsequently occupied by the MainCon (from Day 8 to Day 18) and finally by SubCon\_1 and SubCon\_2 for the remaining work (from Day 18 to Day 54). Comparing with scenario 1 where the two subcontractors have separate work sequences, their work sequences are now integrated in this scenario due to resource sharing. However, despite the work sequence integration, both alternative schedules do not impose any site clashes between the contractors.

From management perspective, the schedules obtained also provide alternative planning schemes for the planners. Alternative 2.1 is more flexible than Alternative 2.2 as it contains more states having float time. In addition, Scaffolding\_S is more effectively used in this case as it supports more processes than in Alternative 2. The results of this scenario show that collaboration does help reduce project duration as it allows resource to be allocated to the works of all involved parties, leading to shorter project duration.

#### **4.8. Concluding Remarks**

This chapter presents a modeling framework called FReMAS for automated construction sequence reasoning from functional requirements. One advantage of FReMAS is that it can capture complex functional requirements with multiple users and multiple providers and different provider co-functionality types. This capability facilitates the generation of alternative schedules possibly resulting from different engineering solutions for the required function during planning phase.

In summary, FReMAS consists of three components: a Representation Model, a Temporal Model, and a Construction Sequence Reasoning Framework. The Representation Model identifies a functional requirement from four perspectives: function user, function provider, function type, and provider co-functionality. The generality of this model makes it surpass the Intermediate Function Concept. In essence, it provides a generalized format for representing both final and intermediate requirements with multiple users and providers, and capturing alternative engineering solutions which often result from alternative construction methods or collaborations.

A two-level Temporal Model is then developed to define the time window of an individual user/provider ( $RTW^U/ATW^P$ ), and of the aggregate function user/provider ( $RTW/ATW$ ). Especially, a concept of meta-provider is introduced to represent a group of providers which can share their functionalities or time windows to jointly provide the required functionality. With this vital construct, different combinations of engineering solutions can be systematically captured during the planning phase.

Finally, the Construction Sequence Reasoning Framework converts the necessary condition in the form of functional dependency between function user and function provider into temporal constraints between their time windows. The final disjunctive constraint set represents alternative construction sequences fulfilling the requirement, which could lead to multiple schedule solutions. Accordingly, this framework can help enhance the adequacy and efficiency of alternative construction scheduling techniques.

# **CHAPTER 5. ASCoRe SCHEDULER: SYSTEM ARCHITECTURE AND SEQUENCE REASONING ALGORITHMS**

## **5.1. Introduction**

This chapter describes a system architectural framework, knowledge modeling templates and reasoning and inference algorithms for implementing the ASCoRe framework and FReMAS model proposed in previous chapters. For easy reading, this chapter first presents a brief overview of relevant backgrounds about constraint satisfaction problem (CSP), constraint logic programming (CLP) and the limitations of CLP-based solvers in constraint analysis from a construction management perspective. It then provides a general description on the proposed system architectural framework, followed by more detailed discussions and explanations on the necessary knowledge modeling tools, inference and sequence reasoning algorithms and the solving engine for generating alternative schedules. In particular, a pre-emptive constraint reasoning framework is developed for identifying basic redundant and conflicting constraints in the pre-scheduling stage to enhance the feasibility and efficiency of scheduling.

## **5.2. Relevant Background**

This section presents a brief review of Constraint Satisfaction Problem (CSP) and Constraint Logic Programming (CLP). These concepts are the key background knowledge on which reasoning algorithms are developed. This section also summarizes major gaps of CLP-based solving engine in constraint analysis, which provide the impetus for the development of a new preemptive constraint analysis approach for construction scheduling.



### 5.2.1. Overview of Constraint Satisfaction Problem

A constraint satisfaction problem (CSP) can be formulated as comprising a set of  $n$  variables  $X = \{x_1, x_2, \dots, x_n\}$ , each of which has a finite set  $D_i$  of possible values (its domain), and a set of constraints  $C$  restricting the values that the variables can simultaneously take. A typical scheduling problem can be modeled as a CSP with activity start/finish times as variables, and temporal constraints. A feasible solution to a CSP is an assignment of a value from its domain to every variable in such a way that the imposed constraints are satisfied.

A variety of approaches can be used to tackle CSPs. The algorithms for solving CSPs can be grouped under two broad categories: inference and search, and various combinations of those two approaches. In inference techniques, local constraint propagation can eliminate values from the domains which do not take part in any solution. The procedure of a typical constraint propagation algorithm proceeds can be described as follows. When a given variable is assigned a value, either directly by the user or by the system, the algorithm re-computes the possible value sets and assigned values of all its dependent variables. This process continues recursively until there are no more changes in the network. Accordingly, the effectiveness of CSPs depends on how well the constraints are represented and the techniques used to propagate them. More detailed descriptions of constraint propagation algorithms are available in the literature (Dechter, 2003; Bessiere, 2006; Lecoutre, 2009).

Search algorithms explore the search space either systematically or locally, often eliminating subspaces with a single failure. Backtracking is the most common systematic search algorithm, which incrementally attempts to extend a partial

assignment, which specifies consistent values for some of the variables, towards a complete assignment (Barták, 2008). It is the fundamental ‘complete’ search method for CSPs, in the sense that one is guaranteed to find a solution if one exists. On the other hand, local search approaches, such as simulated annealing, tabu search or genetic algorithms, provide an approximation solution (Brailsford et al., 1999).

Constraint propagation and backtracking are usually combined in most applications and many constraint solvers to maximize the solving efficiency (Marriott et al., 2006). In this research, the scheduling problem is also modeled as a CSP, and both constraint propagation and backtracking algorithms are combined to generate all alternative schedules.

### **5.2.2. Overview of Constraint Logic Programming**

Constraint logic programming (CLP) is a merger of two paradigms: constraint solving and logic programming. The CLP methodology extends the initial Prolog language by incorporating several types of constraint solvers, where each constraint solver is particularly suited for a specific domain. One important characteristic of CLPs is that they allow succinct, natural conceptual modeling of CSPs. In addition, CLP languages allow the programmer to define search strategies for solving their model. Modern CLP languages, such as Prolog or *ECL<sup>i</sup>PS<sup>e</sup>*, also allow programmers to define how the constraint solver processes the constraints for solving their models.

This research employs CLP approach and *ECL<sup>i</sup>PS<sup>e</sup>* is chosen as the main platform for developing sequence reasoning and scheduling algorithms. Since PDM++ is used as the background model for representing temporal constraints, utilizing *ECL<sup>i</sup>PS<sup>e</sup>* will ease the integration of PDM++ language with the proposed reasoning

and inference mechanisms in the system. In addition, its high-level language provides support for object-oriented modeling which allows for rapid software prototyping. It is also assumed that the reader has a certain level of familiarity with some of the basic programming concepts in Prolog and/or *ECLiPS<sup>e</sup>*. Readers may wish to refer to (Wallace, 2002; Apt, 2007) for a more in-depth discussion on the basic concepts of CLP and *ECLiPS<sup>e</sup>*.

### 5.2.3. Constraint Analysis in CP/CLP-based Schedulers

There are two major problems with CLP-based schedulers: solution feasibility and computational efficiency, which are greatly influenced by the relationships among the imposed constraints. Solution feasibility refers to the capability of producing a feasible solution and is defined by the consistency of the constraint set. In order to improve solution feasibility, conflicting constraints should be identified and resolved in the pre-scheduling stage. On the other hand, computational efficiency is governed by the total number of constraints, especially the number of backtrackings which increases exponentially with the number of disjunctive constraints. In addition, among the constraints, some could be subsumed by others and be redundant (Nguyen and Chua, 2012). Ignoring such constraints thus, while not affecting the schedule result, will reduce computation time. Especially, removing redundant disjunctive constraints eliminates unnecessary backtrackings, improving overall scheduling efficiency.

In many CLP-based schedule solvers, constraints are sequentially called in the propagation to reduce the feasible domains of activities' start times. Constraint inconsistencies are reactively identified along the constraint propagation process (Lorterapong and Ussavadiokrit, 2013), and thus dependent on constraint ordering.

From a management perspective, there remain three major drawbacks. Firstly, a constraint may contradict with multiple constraints. Such constraints should be identified so that multiple conflicts can be simultaneously resolved. The propagation methodologies do not facilitate such resolution strategy. Secondly, sequentially resolving local inconsistencies could lead, after many changes, to a final schedule that may not be executable, since the modified constraints may deviate from original construction intention or represent an impractical method. Moreover, sequentially resolving conflicts does not guarantee the best (or optimal) set of constraints. Finally, activity durations often impact the relationships among constraints but propagation methods do not provide information about how durations can be modified to resolve conflicts without causing new conflicts.

In summary, to enhance the feasibility and efficiency of construction scheduling, conflicting and redundant constraints, especially disjunctive constraint combinations should be preemptively identified and resolved in the pre-scheduling stage, and should be analyzed in accordance with activity durations. Determining all redundant and conflicting constraints in an initial stage would require a complete constraint propagation procedure. Instead, this research focuses only on those existing within one activity or between two activities using a Constraint Integration Reasoning Framework without constraint propagation. Detailed description of this framework will be presented in section 5.6.

### **5.3. Overview of System Architectural Framework for Implementing ASCoRe Framework**

The system architectural framework in Figure 5.1 describes a hybrid knowledge-based system (KBS) for implementing the proposed ASCoRe framework. The hybrid KBS approach is employed to better exploit construction knowledge for efficient data generation and sequence reasoning, and at the same time provide the flexibility in defining specific data for a project schedule. Essentially, the framework is designed to combine the strengths offered by a construction knowledge modeling module, inference and a sequence reasoning kernel, and a schedule generation engine for automated scheduling.

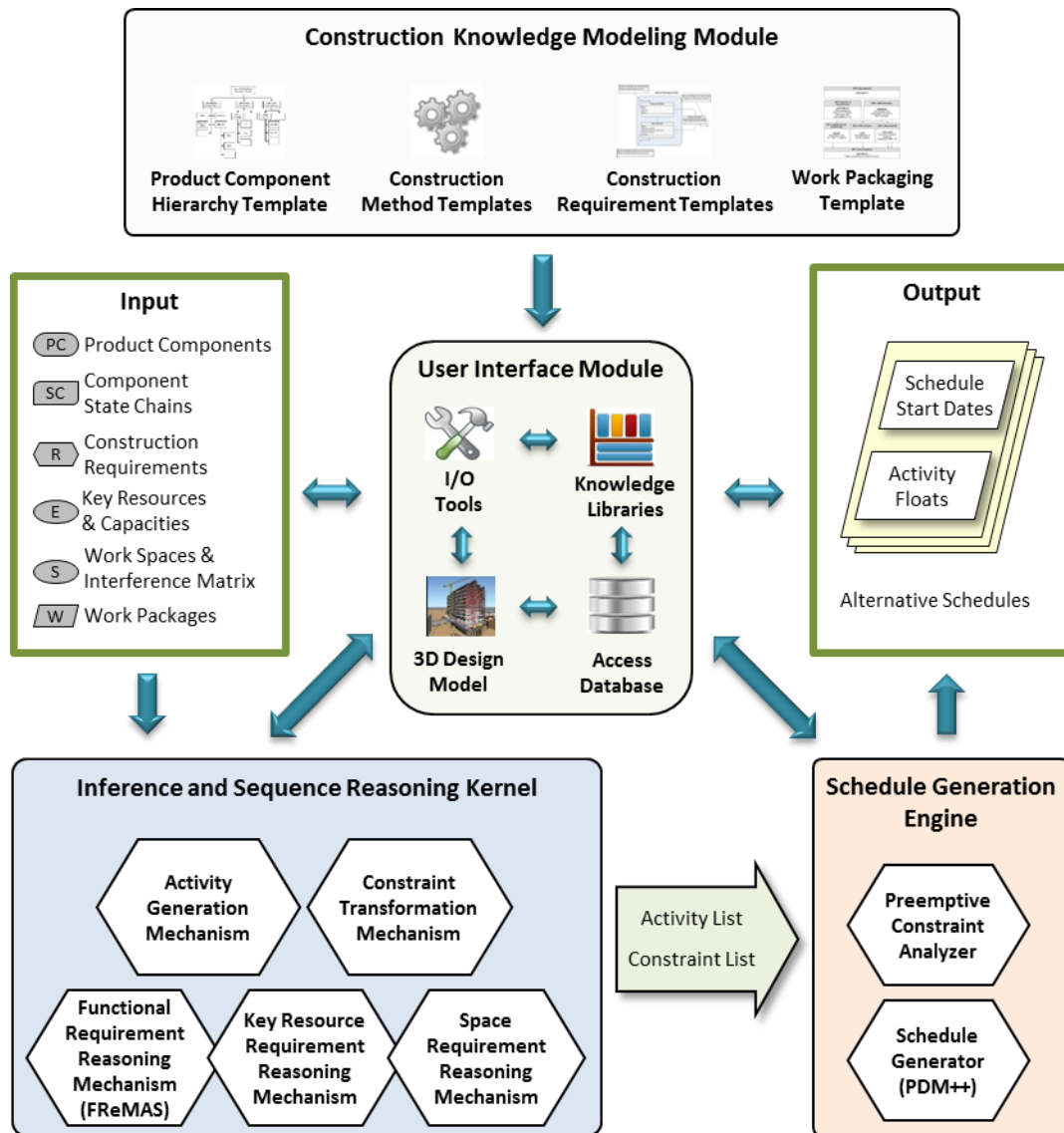


Figure 5.1. ASCoRe system architectural framework

The construction knowledge modeling module includes templates to capture standard product component hierarchies, construction methods, requirements and define work packages. The inference and sequence reasoning kernel consists of five main mechanisms to automatically generate activities from component states and work packages, and convert construction requirements into temporal constraints at both component state and activity levels. The other core solving engine of the framework is the schedule generation engine, which comprises a preemptive constraint analyzer to

identify basic redundant and conflicting constraints in the pre-scheduling stage, and a schedule generator to compute all alternative schedules for the project. These will be covered in the following sections.

A user interface module is developed on .NET framework platform providing planners with different input/output (I/O) tools. Project data are stored in an Access database. Scheduling input can be generated automatically from 3D design models and knowledge libraries or manually specified by planners using I/O tools. The main input for reasoning mechanisms includes a list of components with component state chains defined by construction methods, lists of construction requirements including functional requirements, key resource requirements, spatial requirements and other requirements defined in the form of temporal constraints, project data about key resource capacities, a spatial interference matrix, and defined work packages. The inference kernel and scheduling engine are mainly developed on *ECLiPS<sup>e</sup>* platform to generate all possible construction sequences and determine best schedules with minimal makespan. The scheduling output is a collection of all best alternative schedules with activities' times and floats, if a feasible solution exists.

#### **5.4. Construction Knowledge Modeling Module**

This section describes typical templates for capturing construction knowledge including product models, construction methods, and construction requirements during two processes P and C of the ASCoRe framework. The development of these templates is based on the core knowledge models presented in Chapter 3, and is facilitated by different project data built in the form of libraries, including but not limited to component category, component type, resource type, space type, construction process,

functional relationships, temporal relationships, topological relationships, and comparative relationships. These templates will help planners to pre-define some construction knowledge in the form of libraries as well as automate some data entry process to improve the efficiency.

#### **5.4.1. Product Component Hierarchy Template**

Standard hierarchy templates can be pre-designed to assist the generation of product model (process P of the ASCoRe framework) and thus accelerating the planning process. A typical template of product component hierarchy for a building project is depicted in Figure 5.2a. In this template, components are first arranged by their category type, i.e. permanent, temporary or site work, then by the functioning system to which they belong such as structural, architectural or MEP, next by floor level, subsequently by type, i.e. column, beam, etc. and finally by component name.

The application of this template is presented in Figure 5.2b. A building structure can be elaborated into different hierarchical format with different levels of detail dependent on the project's nature and planning objectives. With this template, planner can flexibly modify the number of levels as well as the criterion for organizing product components to attain the most suitable hierarchy for their own intentions.



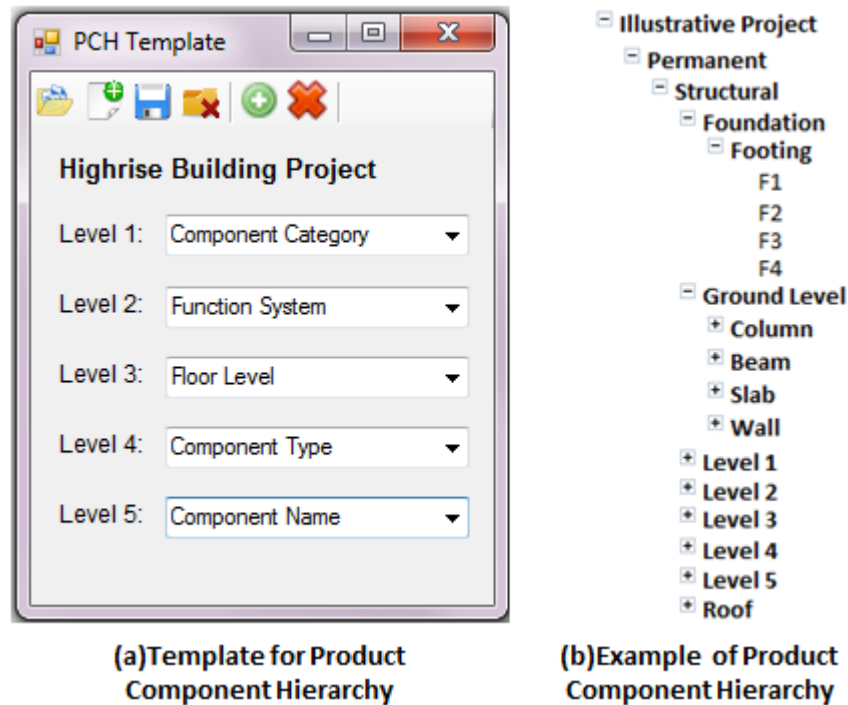


Figure 5.2. Template of product component hierarchy

#### 5.4.2. Construction Method Templates

Construction method template is established on the construction method model described in section 3.2.2 to assist users with building a construction method library. Figure 5.3a presents the template for defining an elementary method in which a method is represented by a generic construction process, the component types which it can be applied to, and temporary structure types and key resource types required for the construction process. In this example, a “Formwork Installation” method is defined with a construction process “Install”. This method can be applied to formwork components and requires a scaffolding temporary structure. It also requires a mobile crane to carry out the work and allows for quiescent phase after its completion. In addition, as show in the figure, the attributes *Component Type*, *Temporary Structure*, and *Key Resource* are defined in the list format.

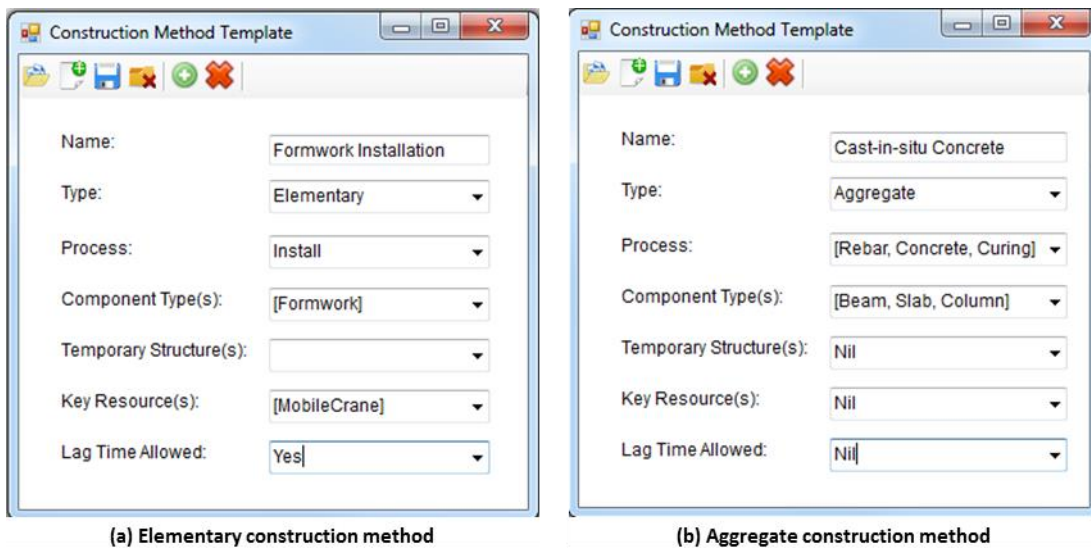


Figure 5.3. Construction method template

Figure 5.3b presents the template for defining an aggregated method which consists of a sequence of construction processes. When an aggregated method is applied to a product component, the corresponding sequence of construction processes involved in the elementary methods helps define the state chain of the component accordingly. The aggregated “Cast-in Site Concrete” defined in Figure 5.3b includes three elementary methods: Rebar, Concrete, and Curing, which altogether describe the necessary processes for constructing a concrete beam/slab/column on site.

#### 5.4.3. Construction Requirement Templates

Two templates are designed for defining functional and non-functional requirements to assist the identification of requirements for scheduling (process R in the ARSCoRe framework). The former is built on FReMAS model presented in chapter four (section 4.3), while the latter is established on the generic construction requirement described in section 3.2.3. These templates allows planner to create libraries of generic requirements or manually define project-specific requirements. The

template for defining functional requirements is depicted in Figure 5.4. The registration of functional requirement may be tedious for large projects. Final functional requirements can be automatically acquired from the functionality analysis of the 3D design model. Simple intermediate functional requirements can also be defined through the same process. By this, this template can be used for defining complex functional requirements which are probably project-specific.

Figure 5.4. Template for defining functional requirement

As described previously in chapter four, the function user of a functional requirement can comprise multiple users, each of which is represented as a set of component states (using the [·] notation). The function user consists of two users from two components [B1.Erect.A] and [B2.Erect.A]. The function provider of the example requirement in Figure 5.4 consists of two providers, the first of which involves two component states from two walls W1 and W2, and is defined as [W1.Complete.Q, W2.Complete.Q], while the second of which contains a component states from a scaffolding structure SB1 and is defined as [SC1.Erect.Q]. Since beam erection can be

supported by either individual or both providers, the provider co-functionality is defined as “C” (Mutually Compatible” in the last field.

The template for defining non-functional requirements is presented in Figure 5.5. It is designed to formalize of three major requirement types: resource, work space and temporal constraints. The necessary condition is elaborated into temporal, topological and comparative relationships as described in section 3.2.3.

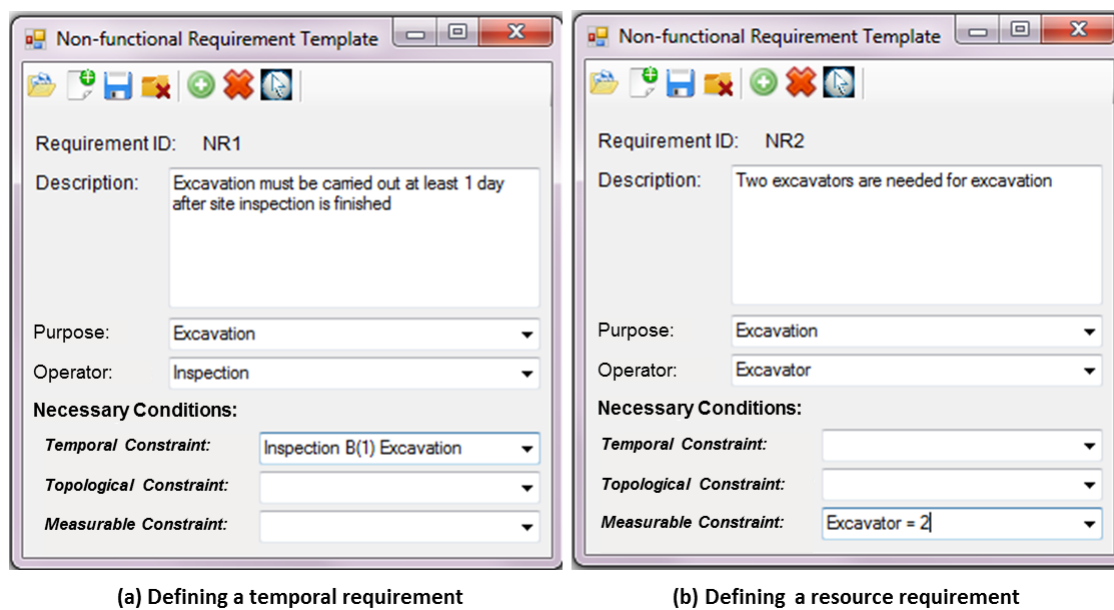


Figure 5.5. Template for defining non-functional requirement

Figure 5.5a illustrates the use of the template to define a precedence requirement between excavation and site inspection processes. The temporal relationship “B(1)” is the short form of the PDM++ constraint Before(1). Readers can refer to Figure 2.1 for the short form formats of all PDM++ constructs used in this dissertation. Figure 5.5b describes a resource requirement in which two excavators must be provided for the excavation. In this template, “=” is a comparative relationship for measurable condition. Other relationships for measurable condition are presented in Figure 3.6.

#### **5.4.4. Work Packaging Template**

Work package is the concept of breaking down a project into smaller sub-project for better planning and management. Different work package definitions have been proposed in the ACE community. According to Halpin (1985), “a work package is a sub-element of a construction project on which both cost and time data are collected for project status reporting. All work packages combined constitute a project’s work breakdown structure”. Song (2006) defined a work package to include a group of component states (active phases) of the product components that are concurrently transited by the associated process. As such, a work package serves as a link between an activity and component states. This research extends this definition by recognizing a work package as a group of components that are constructed using the same method. In other words, components involved in a work package will be created by same processes. In this way, a work package is used as a construct to link product (components) and process data (activities) with a many-to-many relationship.

A work package template is design for manual/automatic generation of work packages using some grouping rules. AND and OR logic operators can be used to combine the defined rules and provide more flexibility in defining a work package. In the example shown in Figure 5.6, with the specified rules, a work package WP1 comprising all precast beam in Level 1 will be automatically created. With this design, the template allows new rules to be easily added.

Work Package ID: WP1

Name: L1\_CastInSituBeam

**Component Grouping Rules:**

	Attribute	Constraint	Value	Rule Combination
Rule 1	Level	=	Level 1	AND
Rule 2	Type	=	Beam	AND
Rule 3	Aggregated Method	=	Cast-in Situ Concrete	

Figure 5.6. Template for defining work package

The relationship between work packages and product components is one-to-many, requiring that one work package can comprise one or many product components but one component can belong to only one work package.

## 5.5. Inference and Sequence Reasoning Kernel

The inference and reasoning kernel is to facilitate process S of the ASCoRe framework by automatically generating an activity list from the product model and work packages, and converting the imposed construction requirements into temporal constraints. As described in section 3.3.3.2, four major construction requirements are examined in the ASCoRe framework, including: functional, key resource, work space and temporal requirements. While functional requirements are generally defined at component state level, key resource, work space, and temporal constraints can be represented at both component state and activity levels. Thus, all construction requirements have to be converted into temporal constraints at activity level for schedule computation.

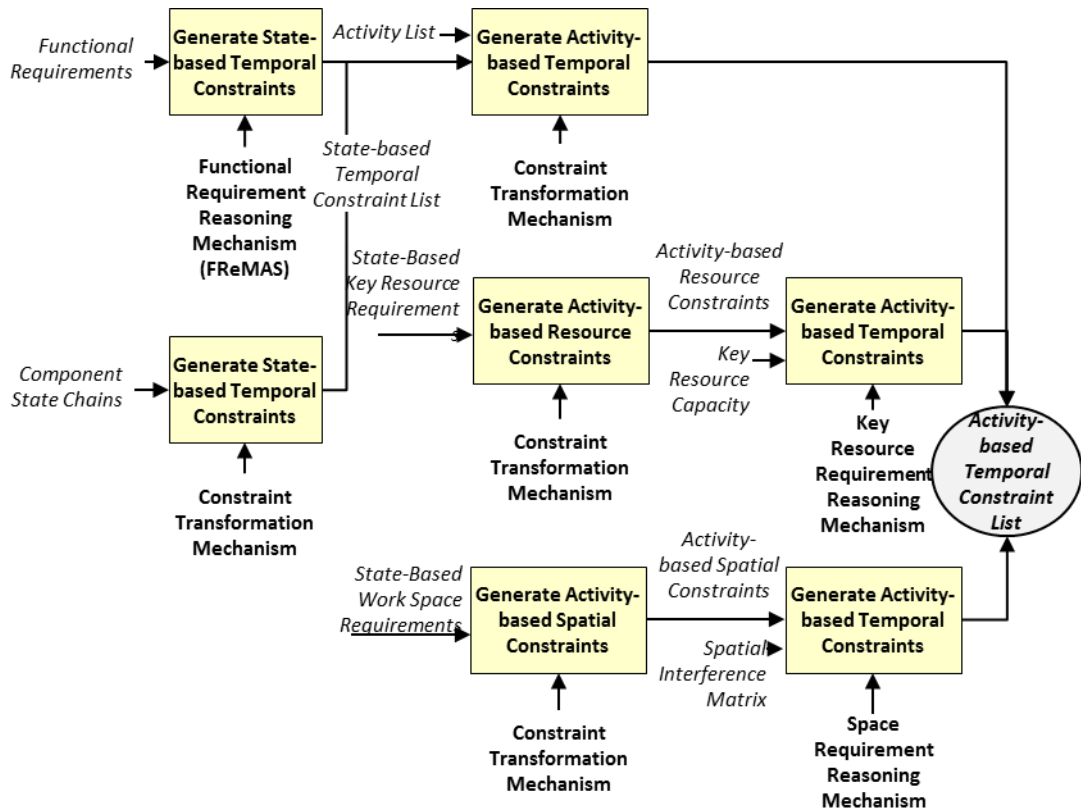


Figure 5.7. Workflow of the inference and sequence reasoning kernel

Figure 5.7 presents the general workflow of the inference and sequence reasoning kernel developed in this research. The reasoning process starts by generating an activity list from the defined work packages and component states using the Activity Generation Mechanism. Next, the Functional Requirement Reasoning Mechanism is employed to convert functional requirements into state-based temporal constraints. Subsequently, all state-based constraints, including key resource, spatial and temporal constraints are transformed into corresponding activity-based constraints using the Constraint Transformation Mechanism. Finally, activity-based resource and spatial constraints are converted into activity-based temporal constraints using the Resource and Space Requirement Reasoning Mechanisms. The final output of the sequence reasoning process is a combined list of activity-based temporal constraints

derived from functional, resource, and space requirements and an activity list, which are then transferred to the Schedule Generation Engine for schedule computation.

### 5.5.1. Activity Generation Mechanism

The purpose of this inference mechanism is to obtain the activity list from component states and work packages. The inference process of this mechanism is illustrated in Figure 5.8. To summarize, for each work package  $wp_{(i)}$ , the mechanism first generates the collection of all component states of the components belonging to  $wp_{(i)}$ . Then all component states which are defined by the same elementary construction method are grouped into one activity. The final output of this mechanism is a list of activities, each of which has two important attributes: the associated elementary construction method, and a list of constituting component states.

```

Input: List of work packages,  $WP$ 
         List of product components,  $PC$ 
Output: List of activities,  $AL$ 

For each Work package  $wp_{(i)} \in WP$ 
  Initialize List of Activities associated with  $wp_{(i)}, A_{(i)}$ 
  Generate List of Active Component States Belonging to  $wp_{(i)}, CS_{(i)}$ 
  For each  $cs_{(i,j)} \in CS_{(i)}$ 
    Get The construction method associated with  $cs_{(i,j)}, cm_{(i,j)}$ 
    If There is no exiting activity  $a_{(j)} \in A_{(i)}$  that contains  $cm_{(i,j)}$ 
      Generate a new activity  $a_{(j)}, A_{(i)} \leftarrow a_{(j)} \cup A_{(i)}$ 
    EndIf
    Add  $cs_{(i,j)}$  to the component state list of  $a_{(j)}$ 
  Next  $j$ 
   $AL \leftarrow A_{(i)} \cup AL$ 
Next  $i$ 
End

```

Figure 5.8. Pseudo code for the activity generation mechanism



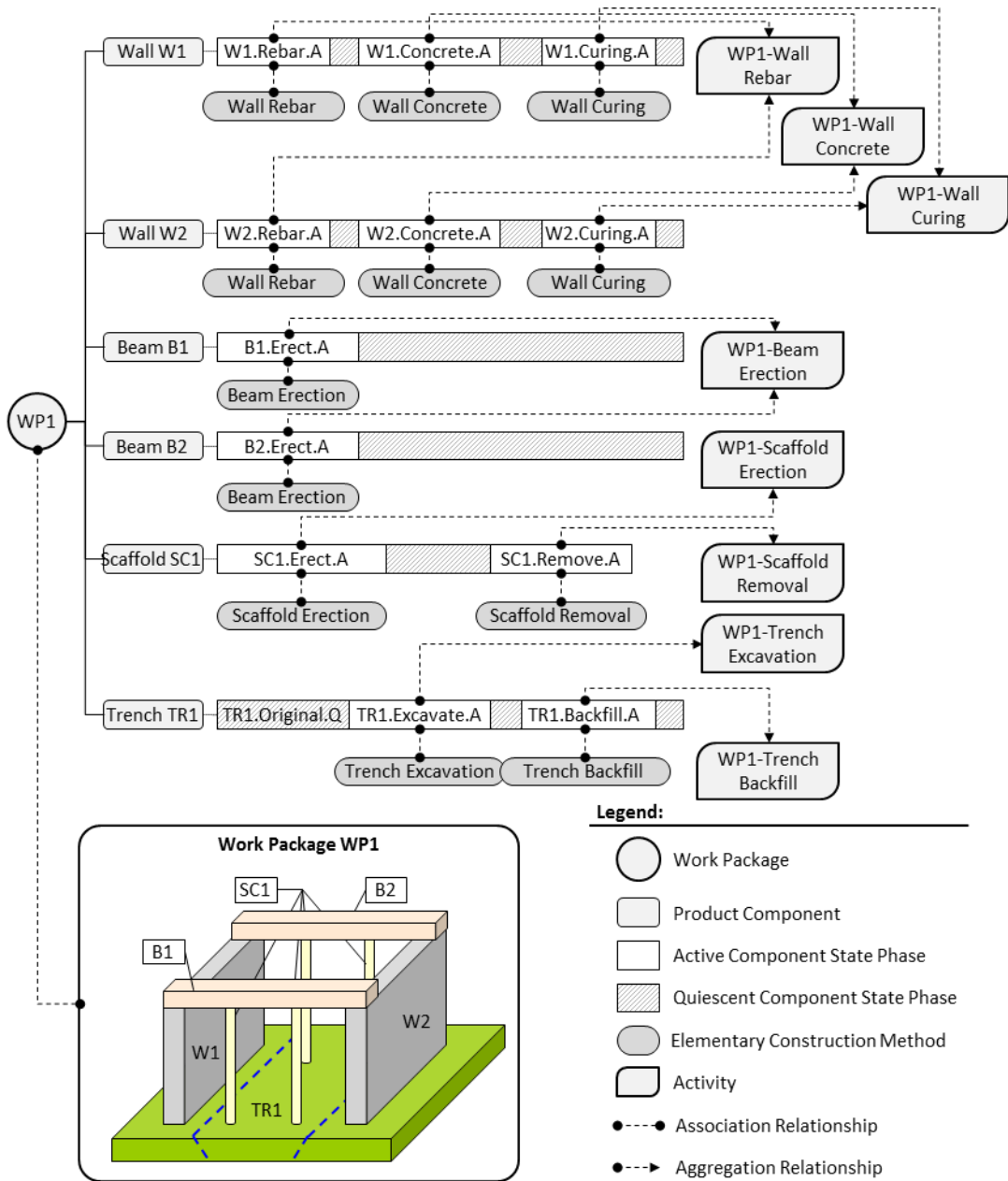


Figure 5.9. Illustrative example for activity generation mechanism

Figure 5.9 presents the application of this inference mechanism to a work package WP1 involving six product components: two cast-in-situ walls (W1 and W2), two precast beams (B1 and B2), one scaffold for erecting the beams (SC) and a trench (TC). There are altogether eight elementary construction methods defining 12 active component states. Hence, eight activities are generated in this case, each of which is

associated with one elementary construction method and consists of one or many component states. For example, activity WP1-Wall Rebar corresponds to the “Wall Rebar” method and comprises two component states: W1.Rebar.A and W2.Rebar.A. Similarly, activity WP1-Beam Erection involves two component states: B1.Erect.A and B2.Erect.A, which are both defined by the same elementary construction method “Beam Erection”.

### **5.5.2. Functional Requirement Sequence Reasoning Mechanism**

This mechanism is built upon the FReMAS model to convert functional requirements into temporal constraints between component states. Readers may wish to refer to Chapter 4 for a detailed description of the implementation of FReMAS.

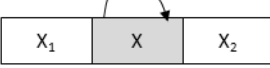
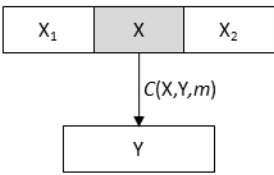
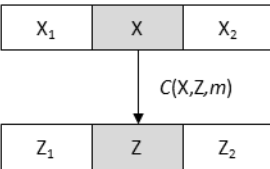
### **5.5.3. Constraint Transformation Mechanism**

The purpose of the constraint transformation mechanism is to aggregate temporal, resource and work space requirements at component state level to those at activity level. This reasoning mechanism is supported by the one-to-one equivalence between component states and elementary activities, so that the reasoning can be carried out directly from component states to activities. It also works under the assumption that key resource and work space requirements are associated with construction processes and thus related to only active component states. Without active states, there resources are not required.

To achieve the above purpose, the reasoning mechanism converts all temporal constraints related to quiescent state phases, into those of the associated active state phases based on the continuity nature of component state chains. Table 5.1 presents rules for converting twelve basic temporal constraints from quiescent states to active

states. In particular, a unary constraint of a quiescent state  $X$  can be converted into an equivalent unary constraint of its preceding or succeeding active state  $X_1/X_2$ . In this section, the subscript “1” denotes the active state phase immediately preceding a quiescent stage phase, and “2” denotes the active state phase immediately succeeding a quiescent stage phase. Similarly, a binary constraint between quiescent states  $X$  and an active phase  $Y$  can be represented by a binary constraint between  $X_1/X_2$  and  $Y$ . Finally, a binary constraint between two quiescent states  $X$  and  $Z$  can be transformed into a binary constraint between their preceding/ succeeding active states  $X_1/X_2$  and  $Z_1/Z_2$ .

Table 5.1. Rules for converting constraints from quiescent states to active states

Type of Constraint	Constraint of Quiescent Phase	Equivalent Constraint of Active Phase
<b>Unary Constraint of Quiescent State</b> $C(X,m)$ 	$X \text{ DB}(m)$	$X_2 \text{ SB}(m)$
	$X \text{ DA}(m)$	$X_2 \text{ SA}(m)$
	$X \text{ SB}(m)$	$X_1 \text{ DB}(m)$
	$X \text{ SA}(m)$	$X_1 \text{ DA}(m)$
<b>Binary Constraint between Quiescent and Active States</b> 	$X \text{ B}(m) \ Y$	$X_2 \text{ SS}(m) \ Y$
	$X \text{ B}(\sim m) \ Y$	$X_2 \text{ SS}(\sim m) \ Y$
	$X \text{ SS}(m) \ Y$	$X_1 \text{ B}(m) \ Y$
	$X \text{ SS}(\sim m) \ Y$	$X_1 \text{ B}(\sim m) \ Y$
	$X \text{ FF}(m) \ Y$	$X_2 \text{ SF}(m) \ Y$
	$X \text{ FF}(\sim m) \ Y$	$X_2 \text{ SF}(\sim m) \ Y$
	$X \text{ SF}(m) \ Y$	$X_1 \text{ FF}(m) \ Y$
	$X \text{ SF}(\sim m) \ Y$	$X_1 \text{ FF}(\sim m) \ Y$
<b>Binary Constraint between Quiescent States</b> 	$X \text{ B}(m) \ Z$	$X_2 \text{ SF}(m) \ Z_1$
	$X \text{ B}(\sim m) \ Z$	$X_2 \text{ SF}(\sim m) \ Z_1$
	$X \text{ SS}(m) \ Z$	$X_1 \text{ FF}(m) \ Z_1$
	$X \text{ SS}(\sim m) \ Z$	$X_1 \text{ FF}(\sim m) \ Z_1$
	$X \text{ FF}(m) \ Z$	$X_2 \text{ SS}(m) \ Z_1$
	$X \text{ FF}(\sim m) \ Z$	$X_2 \text{ SS}(\sim m) \ Z_1$
	$X \text{ SF}(m) \ Z$	$X_1 \text{ B}(m) \ Z_2$
	$X \text{ SF}(\sim m) \ Z$	$X_1 \text{ B}(\sim m) \ Z_2$

**Notes:**  Active state phase     Quiescent state phase

Furthermore, since any complex constraint like *Contains*, *Disjoint*, or *Overlaps* is represented by a conjunctive/ disjunctive combination of these basic constraints, the conversion rules presented in Table 5.1 can be applied for all complex constraints. For example, the complex constraint *Contains* between state Erect.Q of TR1 and Erect.Q of SC1 (see Figure 5.10) can be elaborated into two constraints: (Erectate.Q  $SS(0)$  Erect.Q), and (Erect.Q  $FF(0)$  Excavate.Q). According to the rules in Table 5.1, these constraints are respectively converted to (Erectate.A  $FF(0)$  Erect.A), and (Remove.A  $SS(0)$  Backfill.A) respectively. In addition, as described in chapter three, quiescent states in a state chain can be expressed as a precedent relationship between its two consecutive active phases. In this example, quiescent state Erect.Q of SC1 is expressed in the form of temporal constrain as (Erect.A  $B(0)$  Remove.A).

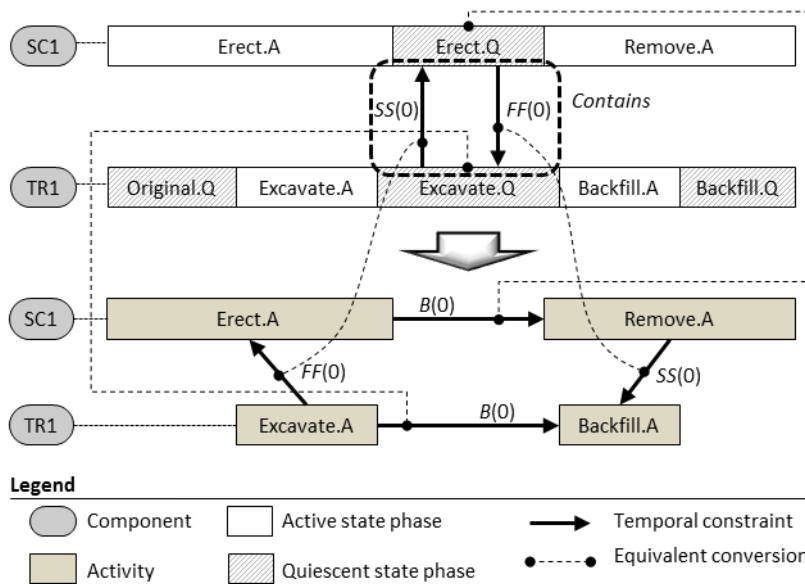


Figure 5.10. Converting quiescent state constraint to active state constraint

When all constraints related to quiescent states are removed, the constraint transformation mechanism proceeds by transferring all constraints (temporal, key resource, and work space) of active phases to the activity which they constitute. This

inference process (as illustrated Figure 5.11) in consists of two procedures: a One-to-One Mapping Procedure, and a Refining Procedure.

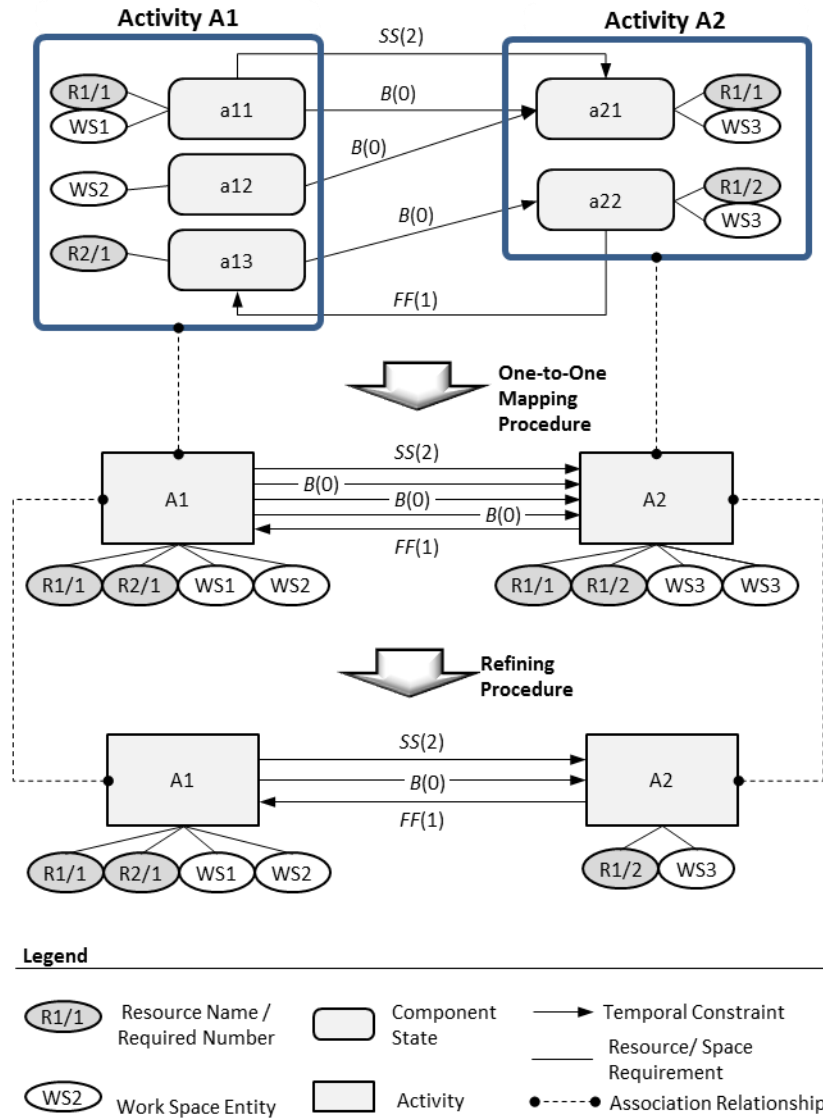


Figure 5.11. Converting state-based constraints to activity-based constraints

Initially, the One-to-One Mapping Procedure converts all constraints/requirements at state level to those at activity level using a one-to-one mapping rule. In detail, any constraint or requirement related to an active component state is converted into a similar constraint of its associated activity. For the example in Figure 5.11, the

constraints of components states (a11, a12, a13) constituting activity A1 and (a21, a22) constituting activity A2 are converted into constraints between A1 and A2 using a one-to-one mapping rule. This process eventually generates five temporal constraints between A1 and A2, two resource requirements (R1/1 and R2/1 from states a11 and a13 respectively) and two work space requirements (WS1 and WS2 from states a11 and a12 respectively) related to activity A1, and two resource requirements (R1/1 and R1/2) and two same work space requirements (WS3) of activity A2.

Then, the Refining Procedure is performed to remove replicated temporal/spatial constraints and to aggregate key resource requirements of the same type. The final output of this constraint transforming procedure lists of activity-based temporal constraints, key resource and space requirements. The resource and space requirements obtained in this step are then passed to the next reasoning mechanisms to reason into temporal constraints. In the above example, after the refining process, three precedence constraints (B(0)) between A1 and A2 is refined to one constraint. In a similar way, the resource requirement R1/1 of activity A2 is subsumed by R1/2 and is removed, and the two work space requirements of WS3 are also refined into one. Ultimately, the constraint transformation mechanism transforms 13 state-based requirements into 9 equivalent constraints between the corresponding activities.

#### **5.5.4. Key Resource Requirement Sequence Reasoning Mechanism**

In the context of this research, key resource refers to important equipment or specialized crews which must be available for the construction process to be carried out. In order to obtain the construction sequence providing the best project makespan, all possible sequences defined by this requirement type should be determined. In this

regards, this sequencing mechanism provides a general procedure to generate temporal constraints from key resource requirements.

Initially, all resource requirements of the same resource type are combined into one group. Each resource requirement group  $R$  is represented by two elements: a capacity limit ( $L_R$ ) and a list of activities with required numbers ( $a_i/r_i$ ), expressed as:

$$R(L_R, [a_1 / r_1, a_2 / r_2, \dots, a_n / r_n]) \quad (4.20)$$

The sequence reasoning procedure of a resource requirement group  $R$  consists of five main steps:

- Step 1: Generate all activity combinations,  $AC(A) = \mathcal{P}(A)$  where  $A = [a_1, a_2, \dots, a_n]$ . In fact,  $AC(A)$  is the powerset of the activity list  $A$  (denoted as  $\mathcal{P}(A)$ ) which comprises all non-empty subsets of  $A$ .
- Step 2: Identify all activity combinations that violate the capacity constraint,  $VC(A) = [VC_1, VC_2, \dots, VC_m]$ . These activity combinations are called violating activity sets, each containing a set of activities that cannot be altogether carried out concurrently due to resource capacity.
- Step 3: Identify the minimal violating activity sets by removing all violated combinations that have subset in the collection. For example, among two violated combinations  $VC_1 = [a_1, a_2]$  and  $VC_2 = [a_1, a_2, a_3]$ ,  $VC_2$  can be removed from analysis since  $VC_1 = [a_1, a_2]$  defines a stricter disjunctive constraint.
- Step 4: Convert capacity constraints into temporal constraints. In order to avoid the capacity violation, at least one activity of each violating combination  $VC_i$  must not be carried out concurrently with the another in  $VC_j$ . This reasoning

knowledge is represented by a disjunctive list of disjunctive constraints (*Disjoint*) between each pair of activities in the combination, shown as:

$$TC_i = \bigvee_{k,l} [(a_k \text{ Disjoint } a_l)] \forall a_k \neq a_l \in VC_i \quad (4.21)$$

- Step 5: Combine the constraints of all group  $TC_i$  to form final constraint list:

$$TC = \bigwedge_i (TC_i) \quad (4.22)$$

For illustration, consider the crane requirement of four activities  $[a_1, a_2, a_3, a_4]$ , in which activity  $a_1$  requires 2 cranes, activity  $a_2$  requires 1 crane, activity  $a_3$  requires 1 crane, and activity  $a_4$  requires 2 cranes. The crane availability is limited at 2 cranes. These requirements are presented as:  $crane(2, [a_1/2, a_2/1, a_3/1, a_4/2])$ . In order to incorporate them into the scheduling process, the proposed sequence reasoning mechanism is performed to convert them into a set of temporal constraints which equivalently ensures that the availability is fulfilled. The sequence reasoning results using the proposed mechanism are described as follows:

- Step 1: Generate all activity combinations (the powerset of the activity set)

$$AC(A) = [[a_1, a_2], [a_1, a_3], [a_1, a_4], [a_2, a_3], [a_2, a_4], [a_3, a_4], \\ [a_1, a_2, a_3], [a_1, a_2, a_4], [a_1, a_3, a_4], [a_2, a_3, a_4], [a_1, a_2, a_3, a_4]]$$

- Step 2: Identify all violating activity sets

$$VC = [[a_1, a_2], [a_1, a_3], [a_1, a_4], [a_2, a_4], [a_3, a_4], \\ [a_1, a_2, a_3], [a_1, a_2, a_4], [a_1, a_3, a_4], [a_2, a_3, a_4], [a_1, a_2, a_3, a_4]]$$

- Step 3: Identify minimal violating activity sets

$$VC = [[a_1, a_2], [a_1, a_3], [a_1, a_4], [a_2, a_4], [a_3, a_4]]$$

- Step 4: Determine capacity constraints in the form of temporal constraints



$$TC_1 = [(a_1 \text{ Disjoint } a_2)], TC_2 = [(a_1 \text{ Disjoint } a_3)], TC_3 = [(a_1 \text{ Disjoint } a_4)],$$

$$TC_4 = [(a_2 \text{ Disjoint } a_4)], TC_5 = [(a_3 \text{ Disjoint } a_4)]$$

- Step 5: Combine all temporal constraints into one set

$$TC = [(a_1 \text{ Disjoint } a_2) \wedge (a_1 \text{ Disjoint } a_3) \wedge (a_1 \text{ Disjoint } a_4) \wedge (a_2 \text{ Disjoint } a_4) \\ \wedge (a_3 \text{ Disjoint } a_4)]$$

The final temporal constraints reasoned from this resource constraint are five disjunctive constraints (*Disjoint*) represented five pairs of activities that cannot be carried out concurrently. The simultaneous satisfaction of these constraints can ensure that the resource availability constraint of this resource is always fulfilled.

#### 5.5.5. Work Space Requirement Sequence Reasoning Mechanism

Many researchers have agreed that spatial conflicts can also obstruct the concurrency of the construction processes using the space entities. This research only focus on the conflicts between work space entities since this conflict type can be avoided through sequencing and scheduling the related construction processes. Other types of spatial conflict between Interdiction Space and Dead Space Elements (Chua et al., 2010) are related to space assignment and cannot be resolved through scheduling, thus not being taken into account in this research. Reader may refer to **Error! Reference source not found.** for a complete set of topological relationships between two space entities. Moreover, although space is a special type of resource, the proposed reasoning mechanism for resource requirements cannot be applied to space constraints since the crucial element that defining the relationship between activities is not the availability but the spatial interference between space entities. Therefore, the

sequence reasoning mechanism for space requirements is built on the spatial interference, and stated in a rule form as:

“If the topological relationship between two work space entities  $ws_1$  and  $ws_2$  which are respectively required by two activities  $a_1$  and  $a_2$  is *intersection-conflict*, then  $a_1$  and  $a_2$  must be carried out disjunctively”.

```

Input: List of spatial requirements,  $SR = [sr_i]$ , with  $sr_i = (a_i, ws_i)$ 
          List of interference space entities,  $IS = [(ws_k, ws_l)]$ 
Output: List of temporal constraints,  $TC$ 

Initialize  $TC = []$ 
Generate List of pairs of spatial requirement,  $SRP = [srp_{ij}]$ 
For each  $srp_{ij} = (sr_i, sr_j) \in SRP$ 
    If  $(ws_i, ws_j) \in IS$ 
       $c_{ij} = (a_i \text{ Disjoint } a_j)$ 
       $TC \leftarrow [c_{ij}] \cup TC$ 
    End If
Next  $ij$ 
End

```

Figure 5.12. Pseudo code spatial requirement sequence reasoning mechanism

The procedure of the spatial requirement sequence reasoning mechanism based on the above rule is depicted in Figure 5.12. The final output of this reasoning process is a collection of disjunctive constraints (*Disjoint*) between activities requiring spatially conflicting work space entities.

## 5.6. Preemptive Constraint Analyzer

The outputs from the inference and reasoning kernel include two main elementary inputs for scheduling: list of activities and a list of temporal constraints. Since ASCoRe scheduler is a CLP-based scheduling system, its feasibility and

efficiency is thus affected by the redundancy/inconsistency relationships among temporal constraints. Therefore, constraint collection set should be pre-analyzed so that redundant and conflicting constraints can be identified and resolved before scheduling. In this regards, this section presents a reasoning framework for identifying redundant and conflicting constraints within single or pairs of activities in the pre-scheduling stage. The framework is implemented as a preemptive constraint analyzer which is performed prior to the schedule generation process to improve the feasibility and efficiency of scheduling.

### 5.6.1. Definition and Classification of Constraint Redundancies and Conflicts

#### 5.6.1.1. Definition

In the context of this research, a constraint is called redundant if it is overruled or subsumed by another constraint. A constraint is the subsumption of another constraint if any value of activity start times fulfilling it also satisfies the latter. Without loss of generality, the subsumption relationship between two binary constraint  $c_1$  and  $c_2$  involving activities  $X$  and  $Y$  where  $c_2$  subsumes  $c_1$  can be represented in the form of first order logic as shown Equation (4.23). This definition can be elaborated as: For any value of  $X^-$  and  $Y^-$  making  $c_2$  true also makes  $c_1$  true; consequently,  $c_1$  is subsumed by  $c_2$  and is a redundant constraint.

$$\forall X^-, \forall Y^-(c_2 \rightarrow c_1) \quad (4.23)$$

In terms of feasible values, constraint  $c_1$  is redundant when compared with  $c_2$  if the feasible ranges of  $X^-$  and  $Y^-$  defined by  $c_2$  is contained by that defined by  $c_1$ . It also means that every values of  $X^-$  and  $Y^-$  feasible for  $c_2$  is also feasible for  $c_1$ .

Consider, for example, two constraints between activities  $X$  ( $d_X = 3$ ) and  $Y$  ( $d_Y = 2$ ):

$c_1: X^- + 1 \leq Y^-$  referring to a constraint  $X$  SS(1)  $Y$ , and  $c_2: X^- + 3 + 3 \leq Y^- + 2$

representing a constraint  $X$  FF(3)  $Y$ . The feasible ranges of  $X^-$  defined by  $c_1$  and  $c_2$  are

$X_1^- = (-\infty, Y^- - 1]$  and  $X_2^- = (-\infty, Y^- - 4]$  respectively, and the feasible ranges of  $Y^-$

are  $Y_1^- = [X^- + 1, +\infty)$  and  $Y_2^- = [X^- + 4, +\infty)$ . As illustrated in Figure 5.13,  $X_2^-$  is

contained by  $X_1^-$ , and  $Y_2^-$  is contained by  $Y_1^-$ . Consequently,  $c_1$  is subsumed by  $c_2$ .

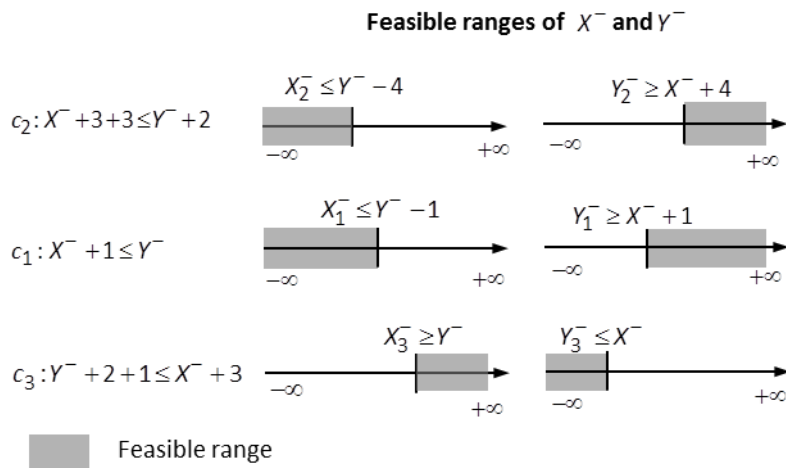


Figure 5.13. Example redundant and inconsistent constraints

In contrast, two constraints are called conflicting if they impose contradicting conditions on the activities involved. More specifically, two binary constraint  $c_1$  and  $c_2$  involving activities  $X$  and  $Y$  are conflicting if every value of activity start times fulfilling  $c_1$  makes  $c_2$  violated and vice versa. Alternatively, there is no value of either  $X^-$  or  $Y^-$  feasible for both  $c_1$  and  $c_2$ , and the feasible ranges of  $X^-$  (or  $Y^-$ ) defined by the two constraints do not overlap each other. Conflicting constraints are logically defined in the form of equation (4.24).

$$\forall X^-, \forall Y^- [(c_1 \rightarrow \neg c_2) \wedge (c_2 \rightarrow \neg c_1)] \quad (4.24)$$

For ease of notation, the definition of conflicting constraints is represented in a short form as shown in Equation (4.25)

$$\forall X^-, \forall Y^- (c_1 \times c_2) \quad (4.25)$$

An example of conflicting constraints is between constraints:  $c_1 : X^- + 1 \leq Y^-$  taken from the previous example and  $c_3 : Y^- + 2 + 1 \leq X^- + 3$  referring to the relationship  $Y \text{ FF}(1) X$ , with the same  $d_X = 3$  and  $d_Y = 2$ . The feasible ranges of  $X^-$  and  $Y^-$  are illustrated in Figure 5.13, where  $X_1^- = (-\infty, Y^- - 1]$  and  $X_3^- = [Y^-, +\infty)$ . Since  $X_1^-$  and  $X_3^-$  have no common value with any value of  $Y^-$ , there exist no value of  $X^-$  that simultaneously satisfies both constraints  $c_1$  and  $c_3$ . Hence, these constraints are inconsistent with each other.

### 5.6.1.2. Classification

In general, each temporal constraint involves two key parameters: lag time (denoted by  $m$ ) and activity durations. Together they define the feasible values of the associated activities' start times and the relationships between constraints. In terms of variation, there is a difference between lag time and activity duration. Since lag time is often dependent on construction technologies, codes and regulations, collaboration or contract issues, they are normally invariant with respect to a dynamic construction environment. For example in the construction of a cast-in-situ wall, a minimal lag time of 2 days is required between the finish of the concrete work and the start of the formwork removal for development of concrete strength. This lag time often remains

unchanged during construction unless there is some modification in the construction method like using concrete additives that allow rapid strength development. In contrast, activity durations are highly dependent on construction conditions such as productivity, resource adequacy, or weather condition. Hence, activity durations are more variant to changes in construction environment.

Due to the above distinction between lag times and activity durations, constraint redundancies and conflicts are divided into two categories: *primary* and *secondary*. Primary conflicts and redundancies are those dependent only on lag times and independent of activity durations. With any activity duration, the existence of a primary conflict or redundancy remains unchanged, and their respective logic definitions are presented in Equations (4.26) and (4.27). Without loss of generality, activity durations and lag time are assumed to have non-negative values in all cases ( $d_X \geq 0, d_Y \geq 0, m \geq 0$ ).

$$\forall X^-, \forall Y^-, \forall d_X, \forall d_Y (c_2 \rightarrow c_1) \quad (4.26)$$

$$\forall X^-, \forall Y^-, \forall d_X, \forall d_Y (c_1 \times c_2) \quad (4.27)$$

In contrast, secondary conflicts and redundancies, as defined in Equations (4.28) and (4.29), depend on both lag times and activity durations.

$$\forall X^-, \forall Y^-, \exists d_X, \exists d_Y (c_2 \rightarrow c_1) \quad (4.28)$$

$$\forall X^-, \forall Y^-, \exists d_X, \exists d_Y (c_1 \times c_2) \quad (4.29)$$

Although primary and secondary constraint redundancies/conflicts have similar impacts to a schedule solution, they have different significance to planners and project

managers. Primary redundancies are invariant with activity durations and could be “completely” ignored if lag time remains unchanged. On the other hand, under some conditions of activity duration, secondary redundancies may no longer exist. This commonly happens when activities are prolonged due to productivity issues or shortened to expedite. Consequently, project managers still need to pay special attention to secondary redundant constraints as they may become significant to the schedule. Primary conflicts are independent of activity durations and thus, can only be resolved when either of the constraints is removed or the lag times are modified. This will require some change in construction method, collaboration with related parties, or contractual agreements. In contrast, secondary conflict can be resolved by changing activity durations. Since changing activity durations are normally easier than modifying lag time values, primary conflicts can be considered more “severe”, and need more management attention than secondary ones.

### **5.6.2. Pre-emptive Constraint Analysis Framework**

The preemptive constraint analysis framework identifies the primary and secondary redundancy and inconsistency constraints occurring in one activity or between two activities. The reasoning basis is represented in the form of comparison rules between two temporal constraints. Temporal constraints can be classified into two groups: simple and complex. Simple constraints are represented by only one mathematical inequality constraint. This group includes 4 unary constraints and 8 binary constraints (with minimal and maximal lag requirements), forming the basic constructs which can be used to represent complex constraints. In contrast, complex constraints are mathematically represented by either a combination of inequality constraints. Accordingly, set of basic rules to compare simple constraints is first

developed as the foundation of the whole framework and the development of rules for comparing complex constraints.

In addition, since definitions of temporal constraints relate to early starts of  $X$  and  $Y$ ,  $X^-$  and  $Y^-$ , it is better to transform them to the form of Equation (4.30) as

$$c: (Y^- - X^-) \Psi f(d_X, d_Y, m), \quad \Psi \in \{\leq, \geq\} \quad (4.30)$$

where  $f(d_X, d_Y, m)$  is a time function of the durations of  $X$  and  $Y$ , and associated lag time  $m$ . In addition,  $\Psi$  represents the nature of the lag time, with “ $\geq$ ” referring to a minimal lag constraint, and “ $\leq$ ” to a maximal lag constraint. This representation format essentially recasts a binary constraint in terms of its feasible range of  $(Y^- - X^-)$  as a relation of two main parameters  $\Psi$  and  $f$ , so that two constraints can be directly compared. Specifically, the redundancy/inconsistency relationship between a pair of constraints associated with two same activities can be inferred from a comparison of  $\Psi$  and  $f$  as established in the rules presented in the next section.

### 5.6.2.1. Redundancy Rules of Simple Constraints

Two rules are developed to identify the primary and secondary redundant constraint (if one exists) between two simple constraints  $c_1$  and  $c_2$  associated with lag types  $\Psi_1$ ,  $\Psi_2$  and functions  $f_1$  and  $f_2$  respectively. Redundancies can only exist when  $\Psi_1$  and  $\Psi_2$  are of the same lag type.

**Primary Redundancy Rule (PR):** *If  $\{\Psi_1, \Psi_2\} \in \{\geq\}$  and if  $(f_2 \geq f_1 \forall d_X, \forall d_Y)$  then  $c_1$  is a primary redundant constraint; else if  $(f_1 \geq f_2 \forall d_X, \forall d_Y)$  then  $c_2$  is a primary redundant constraint; or conversely, if  $\{\Psi_1, \Psi_2\} \in \{\leq\}$  and if  $(f_2 \geq f_1 \forall d_X, \forall d_Y)$*



*then  $c_2$  is a primary redundant constraint; else if  $(f_1 \geq f_2 \forall d_X, \forall d_Y)$  then  $c_1$  is a primary redundant constraint.*

As an example, consider two constraints  $c_1: A \text{ FF}(2) B$  and  $c_2: A \text{ FF}(3) B$ . Thus,  $c_1: B^- - A^- \geq 2 + d_B - d_A$  and  $c_2: B^- - A^- \geq 3 + d_B - d_A$ . It is apparent that with any value of  $d_A$  and  $d_B$ , the condition  $f_2 \geq f_1$  is always satisfied. Consequently, as illustrated in Figure 5.14a, the feasible range of  $B^- - A^-$  defined by  $c_1$  contains the one defined by  $c_2$ , showing that  $c_1$  is subsumed by  $c_2$  with any value of activity durations and thus, identified as a primary redundant constraint.

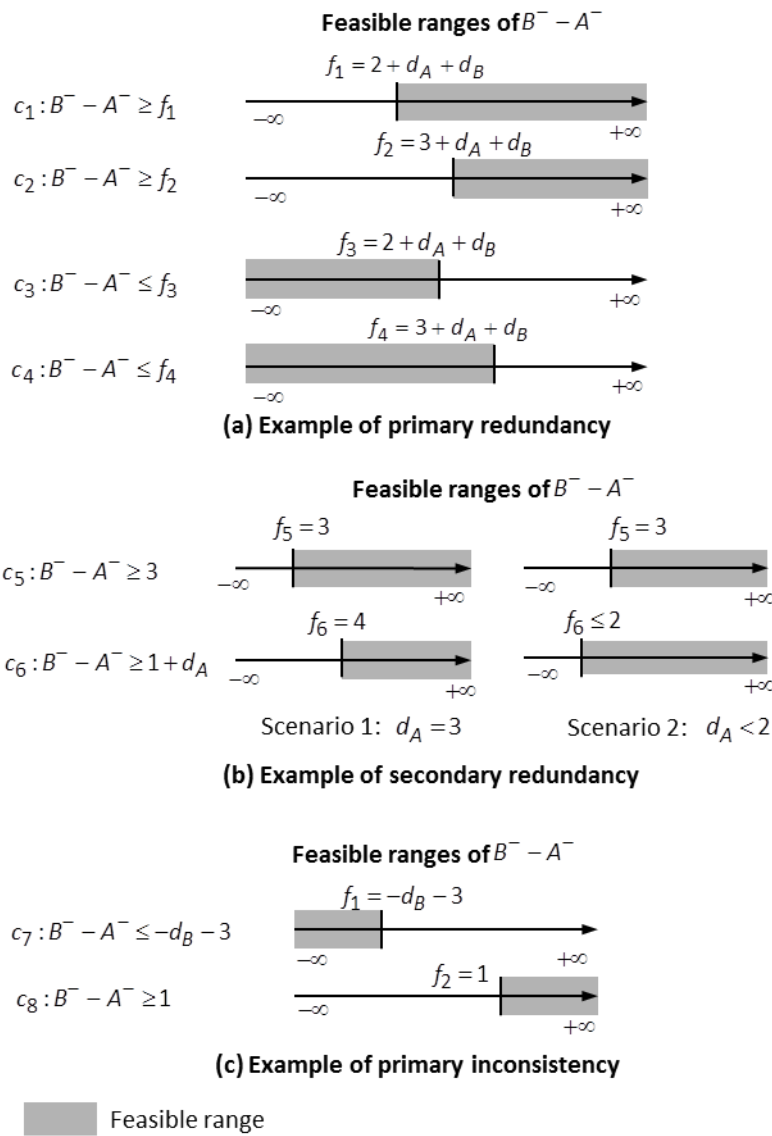


Figure 5.14. Examples redundancy and inconsistency rules for simple constraints

Similarly, consider two other constraints with maximal lags  $c_3: A \text{ FF}(\sim 2) B$  (or  $c_3: B^- - A^- \leq 2 + d_B - d_A$ ) and  $c_4: A \text{ FF}(\sim 3) B$  (or  $c_4: B^- - A^- \leq 3 + d_B - d_A$ ). The feasible ranges of  $B^- - A^-$  are described in Figure 5.14a, showing that  $c_4$  is subsumed by  $c_3$  regardless of the values of  $d_A$  and  $d_B$  and thus, is a primary redundant constraint.

**Secondary Redundancy Rule (SR):** *If  $\{\Psi_1, \Psi_2\} \in \{\geq\}$  and if  $(f_2 \geq f_1)$  then  $c_1$  is a secondary redundant constraint; else if  $(f_1 \geq f_2)$  then  $c_2$  is a secondary redundant constraint; or conversely, if  $\{\Psi_1, \Psi_2\} \in \{\leq\}$  and if  $(f_2 \geq f_1)$  then  $c_2$  is a secondary redundant constraint; else if  $(f_1 \geq f_2)$  then  $c_1$  a secondary redundant constraint.*

For example, consider two constraints  $c_5: A \text{ SS}(3) B$  and  $c_6: A \text{ B}(1) B$ , with  $c_5: B^- - A^- \geq 3$  and  $c_6: B^- - A^- \geq 1 + d_A$ . With  $d_A = 3$ , as shown in Figure 5.14b, any values of  $A^-$  and  $B^-$  satisfying  $c_6$  also fulfill  $c_5$ . Consequently,  $c_5$  is redundant when compared with  $c_6$ . However, with  $d_A < 2$ ,  $f_2 = 1 + d_A \leq 2 < f_1$ , so that  $c_6$  now becomes redundant. Thus they are secondary redundant constraints being contingent on the activity duration.

### 5.6.2.2. Conflict Rules for Simple Constraints

The potential of a conflict occurs when the constraints are of a different nature, i.e.  $\Psi_1 \neq \Psi_2$ . A primary/secondary conflict occurs under two scenarios as defined in the following rules.

**Primary Conflict Rule (PC):** *If  $(\Psi_1 \in \{\leq\}, \Psi_2 \in \{\geq\}, f_1 < f_2) \forall d_X, \forall d_Y$  or  $(\Psi_1 \in \{\geq\}, \Psi_2 \in \{\leq\}, f_1 > f_2) \forall d_X, \forall d_Y$  then a primary conflict is detected.*

An example of a conflict arises between two constraints involving  $A$  and  $B$ ,  $c_7: B \text{ B}(3) A$  (or  $c_7: B^- - A^- \leq -d_B - 3$ ) and  $c_8: A \text{ SS}(1) B$  (or  $c_8: B^- - A^- \geq 1$ ). Regardless of the durations of  $A$  and  $B$ , it is evident that  $f_1 < f_2$ , resulting in the scenario shown in Figure 5.14(c). This is the condition given by the first part of the rule thus defining a primary conflict between  $c_7$  and  $c_8$ .

**Secondary Conflict Rule (SC):** *If*  $(\Psi_1 \in \{\leq\}, \Psi_2 \in \{\geq\}, f_1 < f_2)$  *or*

$(\Psi_1 \in \{\geq\}, \Psi_2 \in \{\leq\}, f_1 > f_2)$  *then a secondary conflict is detected.*

A potential conflict also exists for another two constraints  $c_9: A \text{ SS}(1) B$  (or  $c_9: B^- - A^- \geq 1$ ) and  $c_{10}: FF(1) A$  (or  $c_{10}: B^- - A^- \leq d_A - d_B - 1$ ) with  $d_A = 3$  and  $d_B = 4$ . However, it is a secondary conflict because the conflict no longer occurs for any durations fulfilling  $d_A - d_B \geq 2$ .

In summary, there are three possible outcomes when comparing two constraints: (1) a constraint is subsumed by the other, resulting in a redundancy relationship, (2) they contradict each other, indicating an inconsistency relationship, and (3) neither of the constraints subsumes the other and they do not impose contradicting conditions. In the first scenario, the redundant constraint can be removed from the scheduling process. In contrast, the conflict in the second case must be resolved in order to achieve a feasible solution, while in the last scenario, both constraints need to be considered for scheduling and no special action is required.

Mathematically, these scenarios are identified through the relationships of two basic parameters  $(\Psi_1, f_1)$  and  $(\Psi_2, f_2)$  as summarized in Table 5.2. The redundancy rules PR and SR handle 6 scenarios (green) where either of the constraints is subsumed by the other. Inconsistency rules PC and SC on the other hand represent 2 scenarios (red) of conflict in which the constraints contradict each other. The final 4 scenarios (grey) refer to the situation to the last case.

Table 5.2. Constraint relationships in according with  $\Psi$  and  $f$

	$\Psi_1 = \{\geq\}$			$\Psi_1 = \{\leq\}$		
	$f_1 > f_2$	$f_1 = f_2$	$f_1 < f_2$	$f_1 > f_2$	$f_1 = f_2$	$f_1 < f_2$
$\Psi_2 = \{\geq\}$	$c_2$ is redundant	$c_1$ or $c_2$ is redundant	$c_1$ is redundant	$c_1$ and $c_2$ are significant	$c_1$ and $c_2$ are significant	Conflict
$\Psi_2 = \{\leq\}$	Conflict	$c_1$ and $c_2$ are significant	$c_1$ and $c_2$ are significant	$c_1$ is redundant	$c_1$ or $c_2$ is redundant	$c_2$ is redundant

### 5.6.2.3. Redundancy and Inconsistency Rules of Complex Constraints

The rules described in the previous section for simple constraints form the fundamental constructs for the reasoning of conjunctive/disjunctive constraints. These constraints are necessary for capturing complex construction requirements such as work concurrency, continuity or disjunction. The redundancy and inconsistency relationship among such constraints are also more intricate, and prone to error with manually reasoning.

For this section, consider two complex constraints  $C_1 = (c_{1,1} \wedge c_{1,2} \wedge \dots \wedge c_{1,p})$  and  $C_2 = (c_{2,1} \wedge c_{2,2} \wedge \dots \wedge c_{2,q})$ , and a simple constraint  $c_3$ .  $c_{j,k}$  is a simple constituent constraint of a complex constraint  $C_j$ . Note that capital notation is used for complex constraints and the lower case notation for simple constraints. The interaction of complex constraints is built upon the following rule which defines the subsumption relationship of a conjunctive constraint  $C_1$  over a simple constraint  $c_3$ . In addition, although the following rules are developed for basic complex constraints which involve only simple constraints, they can still be applied to more complex constraints by either decomposing the constraints using distribution laws or performing the comparison in a hierarchy procedure.

### a) Redundancy and Inconsistency Rules of Conjunctive Constraints

**Rule C1:** *If*  $\exists c_{1,k} \in C_1, c_{1,k} \rightarrow c_3$  *then*  $C_1 \rightarrow c_3$

Essentially, if there is at least one constituent constraint  $C_{i,k}$  that subsumes  $c_3$ , then  $C_1$  also subsumes  $c_3$ . Consider for example two constraints involving  $A$  and  $B$ ,  $C_1$ :  $A$  *Overlaps(3)*  $B$  comprising two simple constraints:  $c_{1,1}: B^- - A^- \leq d_A - 3$  and  $c_{1,2}: B^- - A^- \geq 3 - d_B$ , and  $c_3: A$  *SF(2)*  $B$  or  $c_3: B^- - A^- \geq 2 - d_B$ , with  $d_A = 4$  and  $d_B = 5$ . It can be evaluated that  $f_{1,2} = -2$ , and  $f_3 = -3$ . Thus  $c_{1,2}$  subsumes  $c_3$  following rule SR and  $c_3$  is redundant when compared with  $C_1$ . In other words, as illustrated in Figure 5.15a, any value of  $A^-$  and  $B^-$  fulfilling  $C_1$  automatically satisfies  $c_3$ .

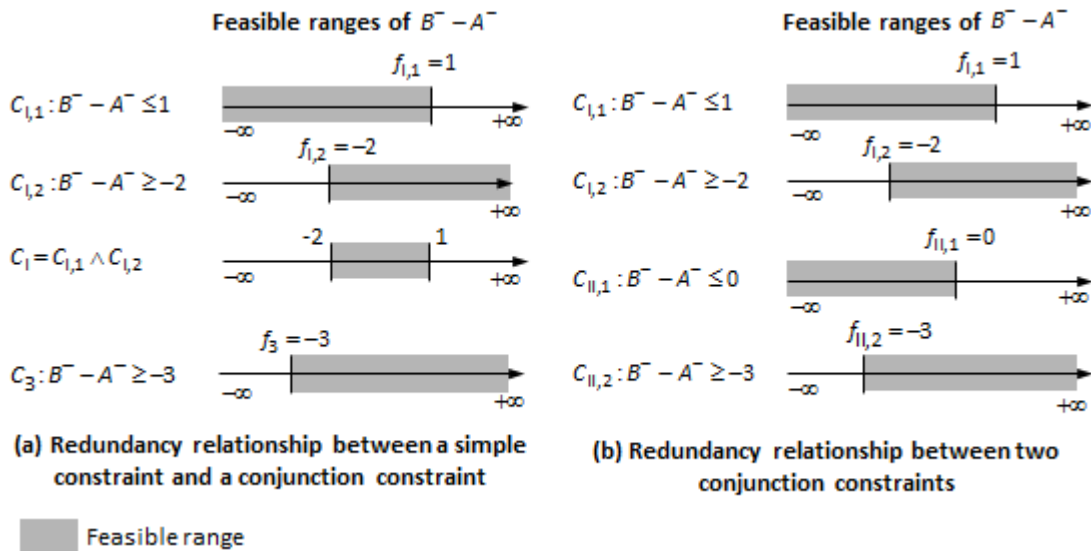


Figure 5.15. Examples of redundancy rules of conjunction constraints

In the case of two conjunctive constraints  $C_1$  and  $C_2$ , the subsumption relationship may be determined using the following rule.

**Rule C2:** *If*  $\forall c_{2,j} \in C_2, C_1 \rightarrow c_{2,j}$  *then*  $C_1 \rightarrow C_2$

Essentially, if every constituent constraint of  $C_2$  is subsumed by  $C_1$ , then  $C_1$  subsumes  $C_2$  so that  $C_2$  is redundant. Extending the example in Figure 4(a) to consider  $C_1$  with  $C_2: A \text{ } SO(2) \text{ } B$ , comprising two simple constraints  $c_{2,1}: B^- - A^- \leq 0$  and  $c_{2,2}: B^- - A^- \geq 2 - d_B$ . Figure 5.15b shows the relevant  $f_{1,1}$ ,  $f_{1,2}$  of  $C_1$  and  $f_{2,1}$ ,  $f_{2,2}$  of  $C_2$ , and that  $C_2$  is redundant when compared with  $C_1$ .

On the other hand, two complex constraint  $C_1$  and  $C_2$  are conflicting when any constituent constraint  $c_{1,k}$  of  $C_1$  contradicts any  $c_{2,k}$  of  $C_2$  as represented in Rule C3.

**Rule C3.** *If*  $\exists c_{1,k} \in C_1, \exists c_{2,j} \in C_2, (c_{1,k} \times c_{2,j})$  *then*  $C_1 \times C_2$

As an example, consider two constraints  $C_1: A \text{ } Meets \text{ } B$  and  $C_2: A \text{ } O(3) \text{ } B$ . Constraint  $C_1$  is a combination of two constraints:  $c_{1,1}: B^- - A^- \leq 0$  and  $c_{1,2}: B^- - A^- \geq 0$ , and constraint  $C_2$  is described as  $c_{2,1}: B^- - A^- \leq d_A - 3$ , and  $c_{2,2}: B^- - A^- \geq 3 - d_B$ . With  $d_A = 4$  and  $B \text{ } d_B = 5$ ,  $f_{2,1} \leq 1$  and  $f_{2,2} \geq -2$ . Thus,  $c_{1,2}$  contradicts  $c_{2,1}$  according to rule SC, showing that  $C_1$  and  $C_2$  cannot be simultaneously satisfied, and are conflicting constraints.

### b) Redundancy and Inconsistency Rules of Disjunctive Constraints

In construction schedules, disjunctive constraints represent construction requirements that could be fulfilled by different ways of sequencing construction processes. They therefore result in different alternative schedules, providing planners with multiple planning options. With a large number of disjunctive constraints, the number of backtrackings or branches is commonly huge and a scheduling problem

may become computationally intractable. The following rules can be used to reduce the number of disjunctive constraints as well as identify inconsistent constraints.

If any disjunct  $c_{1,k}$  of a disjunctive constraint  $C_1$  contradicts a simple or conjunctive constraint  $C_2$ , then  $c_{1,k}$  can be removed without affecting the schedule solution. This could be depicted in Rule D1.

**Rule D1.** *If  $\exists c_{1,k} \in C_1, (c_{1,k} \times C_2)$  then remove  $c_{1,k}$  from  $C_1$ .*

This is possible because in a disjunctive constraint, each disjunct refers to one alternative branch which may lead to a feasible solution. When a disjunct contradicts any other constraint, the associated branch becomes infeasible and thus ignored.

On the other hand, a conflict occurs when all disjunct  $c_{1,k}$  constituting a disjunctive constraint  $C_1$  is inconsistent with a simple or conjunctive constraint  $C_2$  as stated in Rule D2.

**Rule D2.** *If  $\forall c_{1,k} \in C_1, (c_{1,k} \times C_2)$  then  $C_1 \times C_2$*

Essentially, the existence of such a conflict means that no feasible branch can be found, and thus an infeasible schedule ensues.

### 5.6.3. Identifying Feasible Duration Range

Variations of activity durations are common in construction schedules. They could happen incidentally due to variations of different construction factors such as weather conditions, productivity, or resource availability. Activity duration could also be modified for different management aims like expediting delays or resolving schedule inconsistencies. As each activity is possibly involved in many constraints,



changing the duration of one activity may resolve one inconsistency but at the same time cause new conflicts. In addition, in the execution stage, changes in activity durations can result in conflicts and make the baseline plan infeasible. Due to constraints, there is a range of values for the duration of an activity, beyond which the schedule will be infeasible. Therefore, identifying the feasible duration range of an activity would provide planners with opportunities for modifying activity durations as required in both planning and control stages.

The feasible duration range of an activity is defined as the range of values that the activity duration can take without causing any conflict among all the constraints associated with it while maintaining the duration of other activities and lag times. The feasible duration range of an activity  $X$  is denoted by  $FD_X = [FD_X^L, FD_X^U]$ , where  $FD_X^L$  and  $FD_X^U$  are the lower and upper bounds of the range respectively.

When both constraints are of the same lag type, there exist a redundancy between two constraints as in Rules PR and SR, and no value of activity durations could lead to a conflict between them. Therefore, there is no bounds to  $d_X$  and  $d_Y$  and their feasible duration ranges with regard to constraints  $c_1$  and  $c_2$  are specified as  $FD_X^{1,2} = FD_Y^{1,2} = [0, +\infty)$ .

A conflict may occur when two constraints are of different lag types. Without loss of generality, assume that  $\Psi_1 \in \{\leq\}$  and  $\Psi_2 \in \{\geq\}$ . From rules PC and SC, in order to ensure there is no conflict between them, the condition  $f_1 \geq f_2$  needs to be satisfied. From this, the feasible duration ranges of activities  $X$  and  $Y$  may be derived.

Consider two constraints  $c_1: B \text{ FF}(1) A$  or  $c_1: B^- - A^- \leq d_A - d_B - 1$  and  $c_2: A \text{ SS}(2) B$  or  $B^- - A^- \geq 2$ . By applying condition  $f_1 \geq f_2$ , the feasible duration ranges of  $A$  and  $B$  are determined as  $d_A \geq d_B + 3$  and  $d_B \leq d_A - 3$  respectively. Thus, with predefined  $d_A = 7$  and  $d_B = 3$ , their respective feasible duration ranges in regard to constraints  $c_1$  and  $c_2$  are identified as  $d_A \geq 6$  or  $FD_A^{1,2} = [6, +\infty)$ , and  $d_B \leq 4$  or  $FD_B^{1,2} = [0, 4]$ .

The overall feasible duration range of an activity  $X$  (denoted as  $FD_X$ ) is the combination of the intervals computed from the pair-wise comparisons of all constraints involving that activity, as shown in Equation (4.31) where  $(i, j)$  refer to any pair of constraints  $i$  and  $j$  involving  $X$ .

$$FD_X = \bigcap^{i,j} (FD_X^{i,j}) \quad (4.31)$$

The feasible duration range could also be employed to analyze the consistency of all constraints related to an activity. If the feasible range of an activity  $X$  become empty ( $FD_X = \emptyset$ ), there is no feasible value of  $d_X$  that simultaneously satisfies all constraints related to  $X$ . In other words, there exists an inconsistency within the associated constraints which cannot be resolved with any value of  $d_X$ . Hence, in order to remove such a conflict, planners have to choose other strategies such as removing conflicting constraints or modifying the duration of other related activities.

The application of the framework to PDM++ constraints is presented in Appendix 1 so that readers and planners can directly and manually apply the outcomes

of the rules without resorting to complex computing (albeit computing will automate the process and enable the conflicts and redundancies to be readily identified).

#### 5.6.4. Preemptive Constraint Analyzer

The preemptive constraint analyzer is built upon the Constraint Integration Reasoning Framework as depicted in Figure 5.16. The analysis starts with an initialization process, which essentially elaborates complex constraints into combinations of simple constraints and generates a constraint pair collection  $\mathbb{C}$  and finally, initializes all outputs.

The reasoning process is divided into two main parts. The first handles simple constraints by sequentially examining every constraint pair  $(C_i, C_j)$  in  $\mathbb{C}$  using rules PR, SR, PC, and SC, and determining the feasible duration ranges of the associated activities. The second part analyzes the complex constraints. Based on the redundancies and conflicts found in the first part, the relationship between each complex constraint and simple constraints are identified using rules C1 and D1. The comparison of complex constraints then proceeds using rules C2, C3 and D2. In general, in a worst-case scenario, the reasoning of simple constraints runs in  $O(|\{S\}|^2)$  polynomial time where  $|\{S\}|$  is the number of simple constraints, while the run time complexity for complex constraints is  $O((2|\{S\}|)|\{C\}|^2|\{j\}|^2)$  where  $|\{C\}|$  is the number of complex constraints, and  $\{j\}$  the maximum number of constituting simple constraint of a complex constraint.

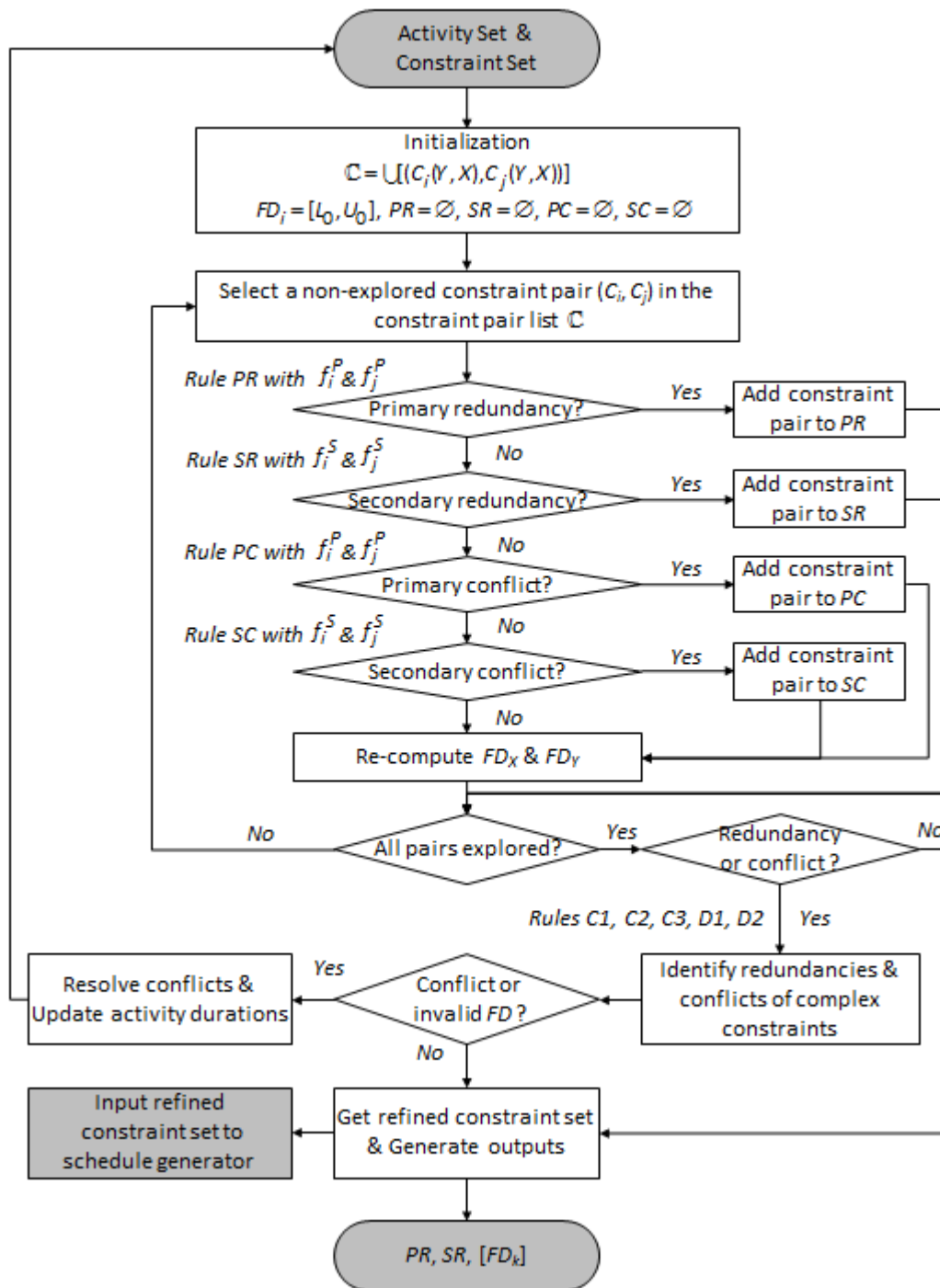


Figure 5.16. Flowchart for implementing preemptive constraint analysis

The output of the preemptive constraint analyzer are sets of primary and secondary redundant constraints, primary and secondary conflicting constraint pairs, and the corresponding feasible duration ranges of all associated activities. Planners

need to take suitable actions to resolve the identified conflicts. Common strategies include: (1) Change the duration of one or some activities within their feasible ranges, (2) Examine to see if a conflicting constraint can be removed, or (3) Examine to see if construction method can be changed so that the lag time of a conflicting constraint can be modified. When conflicting constraints are resolved within the feasible ranges, no new conflicts would arise. The objective, therefore, of the preemptive constraint analyzer is to obtain a refined constraint set without redundant constraints (although remaining in the database) for efficient scheduling using the scheduler.

In summary, with the existence of disjunctive constraints, the scheduling problem is generally a NP problem. For  $n$  activities and  $m$  constraints with  $k$  disjuncts each, the worst case run-time complexity of this scheduler is of an exponential order as  $O(n^2mk^m)$  (Tsamardinos and Pollack, 2003), so that by removing the redundant disjuncts via the preemptive constraint analyzer, the computational time of the solver can be significantly reduced. The benefit of applying the proposed framework in the pre-scheduling stage is twofold. The scheduling process will not start until all basic conflicts are resolved since it is known that no solution is obtained if such a conflict still exists, and redundant constraints are removed from the constraint set to improve computational efficiency.

## **5.7. Schedule Generator**

The scheduling problem is modeled as a CSP with activity start times as variables and a set of constraints containing both conjunctive and disjunctive constraints, including the Makespan constraints. It is solved using the Schedule Generator which is built on PDM++ model. The major goal of this scheduler is to

generate all feasible schedule solutions based on the refined constraint set obtained from the preemptive constraint analysis process using constraint propagation and backtracking search techniques provided by *ECLIPSe*. The computation procedure (depicted in Figure 5.16) consist of four stages: (1) Initialization, (2) Constraint Propagation, (3) Backtracking Search, and (4) Output Finalization.

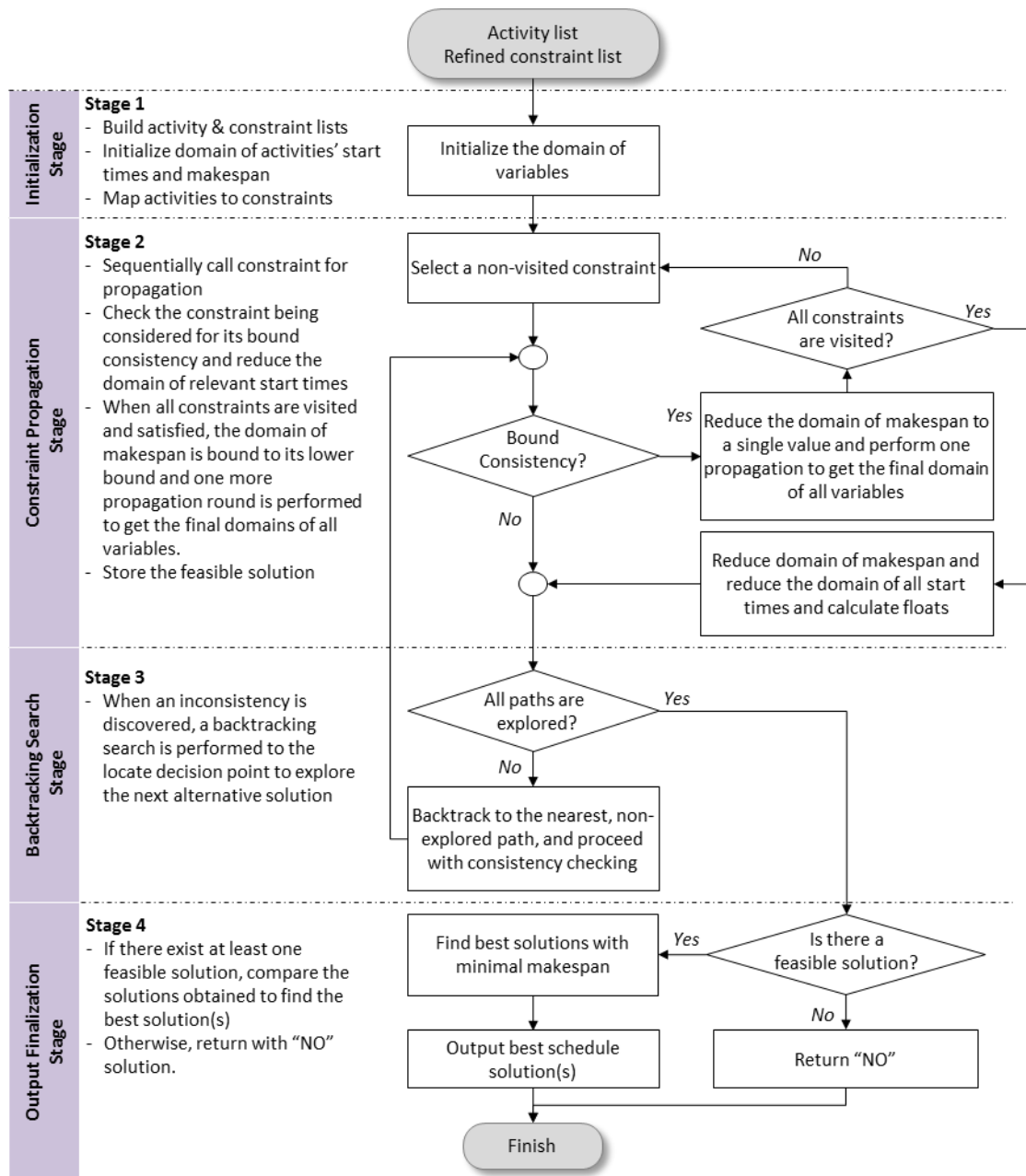


Figure 5.17. Flowchart of scheduling process

- **Stage 1 (Initialization):** Input activities and constraints are added to pre-defined data structures. The domain of all variables (activities' start times) is initialized. Then, activities are mapped the relevant constraints for constraint propagation.
- **Stage 2 (Constraint Propagation):** The constraint propagation is facilitated by the BCSolver Algorithm developed by Yeoh (2012), which has been adapted from the Bounds Consistency Algorithm (Jaffar et al., 1994). In particular, the definition for Bounds Consistency proposed by Choi et al. (2006) stating “A constraint is Bounds Consistent if for each bound of the domain of a variable there is an integer support for the values of the domain of the other variables occurring in the same constraint.” is adopted in BCSolver Algorithm. When all constraints are visited and satisfied, the domain of makespan is bound to the lower bound and one more propagation process is performed to get the final domains variables. The lower and upper bounds of a domain respectively represent the early and late start time of the corresponding activity.
- **Stage 3 (Backtracking Search):** When a path is fully explored or an inconsistency occurs, backtracking search is perform to the nearest non-explored path to examine new constraint combinations. The constraint propagation is then performed to identify a feasible solution. The backtracking process is iterated until the search space is entirely explored.
- **Stage 4 (Output Finalization):** When the backtracking search process is finished, the identified feasible schedule solutions (if any exists) are compared to find and output the best solutions with the minimal makespan. If there is no feasible solution, a “NO” result is returned.

## 5.8. Concluding Remarks

This chapter introduces the foundational knowledge necessary for implementing the ASCoRe framework. To summarize, the system architecture for implementing ASCoRe based on .NET framework and *ECLIPSe* platforms integrates the advantageous features of three modules for alternative auto-scheduling from construction methods and requirements. As a whole, the proposed system architectural framework contains necessary tools and mechanisms for auto-scheduling from both product and process perspectives, and also allows for more flexibility in representing construction methods and requirements as well as updating their changes to schedules.

In particular, the construction knowledge modeling module provides templates for systematically formalizing basic construction methods and requirements. These templates support rapid gathering and unambiguous representation of construction knowledge, so that major construction knowledge can be passed on through the scheduling generation and analysis phases for better traceability of changes and project management. Secondly, the inference and reasoning kernel incorporates inference algorithms for automatically deriving activities and temporal constraints from project data. Especially, activities are not pre-defined as in existing planning systems, but generated from directly construction methods and product components. By this, changes in construction methods and/or the design model can be steadily updated to activities and schedules. Finally, the schedule generation engine based on the PDM++ model provides a computational model for generating all best alternative schedules. In addition to the extensional features inherited from the PDM++ model, it contains a preemptive constraint analysis module to further improve the scheduling efficiency.



Another vital contribution of this chapter is the preemptive constraint analysis framework. By identifying redundant and conflicting constraints between single or pairs of activities in the pre-scheduling stage, this framework helps identify the infeasibility and/or eliminate unnecessary searching space of the scheduling problem, thus improving schedule efficiency. In essence, the classification of constraint redundancies/inconsistencies based on the impact of activity durations and lags provide planners with better understandings of the nature of the redundancies/conflicts and useful strategies for resolving conflicts. Moreover, the feasible range of an activity duration computed from the framework provides planners with useful guidelines for solving conflicts, and also allows them to verify the validity of an activity duration when changes happen.

Due to the existence of disjunctive constraints, the scheduling problem is generally NP-hard. The proposed schedule generation algorithm which is currently based on basic constraint propagation and branch and bound techniques can be incorporated with more efficient search approaches, such as the hybrid conflict-directed backjumping, semantic branching and no-good based reasoning (Tsamardinos and Pollack, 2003) to further improve the scheduling efficiency.

# **CHAPTER 6. CRITICALITY ANALYSIS OF CONSTRUCTION REQUIREMENTS FOR SCHEDULE CHANGE MANAGEMENT**

## **6.1. Introduction**

As discussed in chapter three, construction requirements represent construction knowledge and practice from which schedule constraints and alternative schedules are derived. Therefore, this research highlights their governing role for schedule and proposes that construction schedules should be analysed and managed from the perspective of construction requirements. In this regard, this chapter presents an innovative concept for analysing the criticality of constraints and construction requirements with respect to multiple alternative schedules. The proposed concept will provide the fundamental basis for constraint-based methodology for schedule change analysis and management.

A qualitative classification of constraint criticality to a single schedule is proposed to provide a broader definition of criticality. This classification schema forms the basis for identifying the criticality of complex requirements over multiple alternative schedules. Subsequently, a systematic procedure to identify constraint criticality is developed using two constraint criticality indicators. These indicators are further employed for analysing the impact of constraint variations on schedule makespan. The concept of constraint criticality also allows for developing a new approach to schedule management which is based on constraints and construction requirements. The application of the proposed concept and methodology is demonstrated via an illustrative example.

## **6.2. Constraint Criticality**

Generally, construction requirement is represented by one single or a set of simple constraints, and its criticality can be determined from that of its constituent constraints. Therefore, identifying the criticality of individual simple constraint in a schedule is fundamental for identifying the overall criticality of complex construction requirements. In addition, constraints constituting disjunctive requirements may not be involved or active in some schedules. Thus, from the perspective of a single schedule, a constraint can be characterized from two aspects: Existence and Criticality. The existence of a constraint refers to its presence within a specific schedule, while its non-existence means that the constraint is not involved in the solution of that schedule. As such, the criticality of a constraint is always determined with its existence condition.

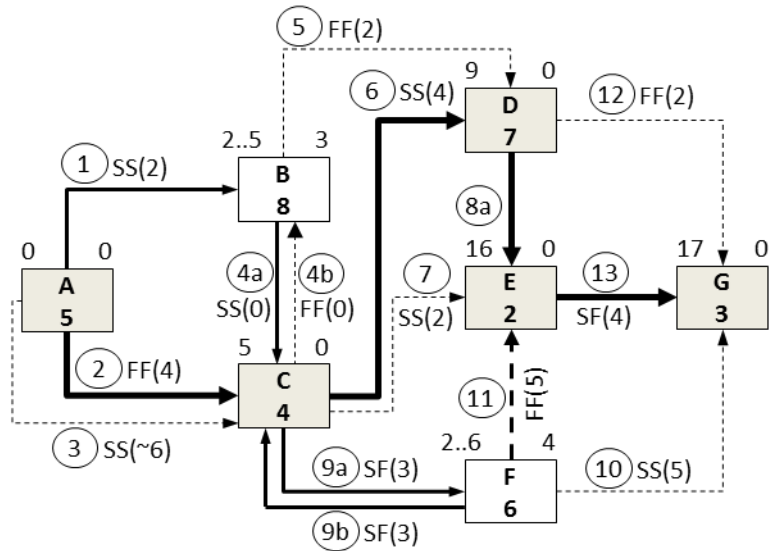
### **6.2.1. Definition and Classification**

Due to the complex nature of some constraints, a critical constraint may affect not only project duration or the start/finish times of activities but also the sequence of activities in a project plan. In contrast, non-critical constraints are redundant ones, which can be removed without causing any change to the schedule. From this perspective, constraints can be classified into four groups: project-critical, activity-critical, sequence-critical, and redundant, described as follows.

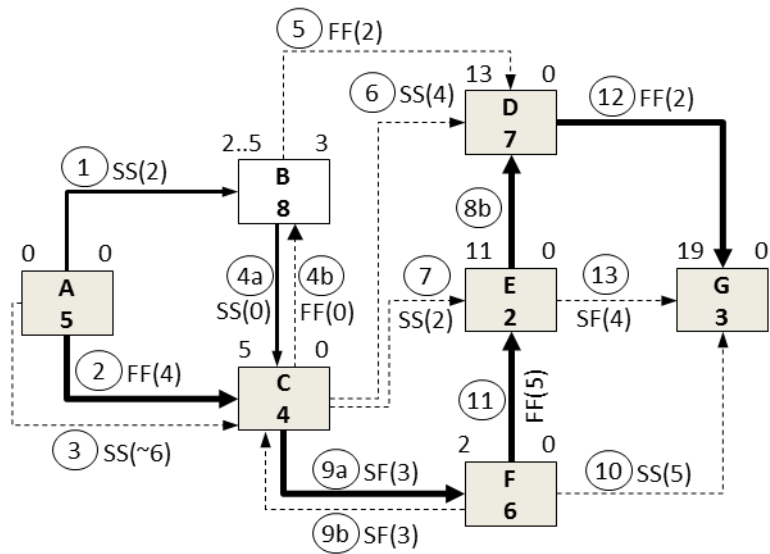
#### ***6.2.1.1. Project-critical Constraint***

A constraint is project-critical if it controls the start/finish times of a critical activity and thus governs the project duration. As such, a project-critical constraint path implies a critical activity path and vice versa. More precisely, any critical activity

path has an associated project-critical constraint path, which links all constraints governing the start/finish times of the critical activities involved.



(a) Alternative Schedule 1



(b) Alternative Schedule 2

**Legend**

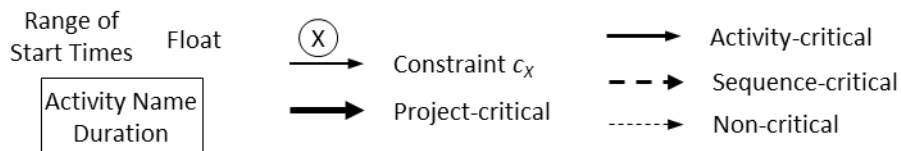


Figure 6.1. Example schedule network

For illustration, Figure 6.1 presents two alternative schedules of a schedule network resulting from the disjunction requirement between activities  $D$  and  $E$  ( $C_8: D$  *Disjoint*  $E$ ) so that  $D$  is scheduled before  $E$  as in Figure 6.1a, or  $D$  is scheduled after  $E$  as in Figure 6.1(b). This requirement is captured by the disjunctive combination of two constraints  $c_{8a}$  and  $c_{8b}$ , expressed as  $C_8:(c_{8a} \vee c_{8b})$ . The concurrent relationship between  $B$  and  $C$  ( $C_4: B$  *Contains*  $C$ ) is represented by a conjunctive combination of two constraints  $c_{4a}$  and  $c_{4b}$ , as  $C_4=(c_{4a} \wedge c_{4b})$ . Similarly, the overlapping relationship between  $C$  and  $F$  ( $C_9: C$  *Overlaps(3)*  $F$ ) is a conjunction of  $c_{9a}$  and  $c_{9b}$  as  $C_9:(c_{9a} \wedge c_{9b})$ . The respective durations of Schedules 1 and 2 are 20 and 22 days. It is also noted that short-form notation will be used for simple constraints while long-form notation for complex constraints for easy reading and consistency with previous chapters. Readers may wish to refer to Figure 2.1 for a full description of PDM++ constraints in both long and short form notations.

In Schedule 1, constraints  $c_2$ ,  $c_6$ ,  $c_{8a}$ , and  $c_{13}$  are project-critical, since they define the times of the critical activities  $A$ ,  $C$ ,  $D$ ,  $E$  and  $G$ , as well as the schedule makespan. If for example constraint  $c_6$  is modified to SS(5), the start time of activity  $D$  is delayed by 1 day, and the schedule makespan is prolonged to 21 days accordingly.

### **6.2.1.2. Activity-critical Constraint**

Similar to critical activities, the start/finish times of every non-critical activity are also controlled by at least one constraint which is classified as activity-critical. Any change or deletion of such a constraint can cause activity times to be changed while the schedule makespan remains unchanged. An activity-critical constraint becomes project-critical when its associated activities become critical.

In Schedule 1 (Figure 6.1a), activity  $B$  is non-critical and its start/finish times are controlled by two constraints  $c_1$  and  $c_{4a}$ . In detail,  $c_1$  defines its early start/finish times while  $c_{4a}$  governs its late start/finish times. If  $c_1$  is changed to SS(3), early start/finish times of  $B$  will change to 3 and 9 respectively, whereas the project duration remains unchanged as 20 days. When  $c_1$  is changed to SS(5), activity  $B$  turns to be critical and  $c_1$  becomes project-critical.

### **6.2.1.3. Sequence-critical Constraint**

When a constraint does not control start/finish times of any activity, it is intuitively considered redundant, and removal of such a constraint may seem not to cause any change to project makespan. Yet under some scenarios, removing a “non-critical” constraint allows for the re-sequencing of some activities so that a better project duration is achieved. These sequences may be originally infeasible and only made feasible by the removal of such a constraint. Due to this distinctive characteristic, this type of constraints is classified as “sequence-critical” in this paper. It refers to those constraints whose existence has no impact on the schedule but affects the sequence which defines the best project duration.

As shown in Figure 6.1, the project duration is 20 days following the sequence defined in Schedule 1 where  $D$  is before  $E$ . In this schedule, constraint  $c_{11}$  is found to be non-critical. However, if this constraint is deleted, the makespan of Schedule 2 where  $E$  is before  $D$  is reduced to 18 days thus improving the overall project duration, while that of Schedule 1 remains at 20 days. In another example shown in Figure 6.2, with the existence of constraint  $c_4$ , there is only one feasible sequence in which activity  $B$  is before activity  $C$ , giving the project makespan of 20 days (Figure 6.2a). However, when this constraint is removed, the alternative sequence in which activity  $C$  is before

activity *B* become feasible, producing a shorter project duration of 17 days (Figure 6.2b) while the makespan of the original sequence remains unchanged.

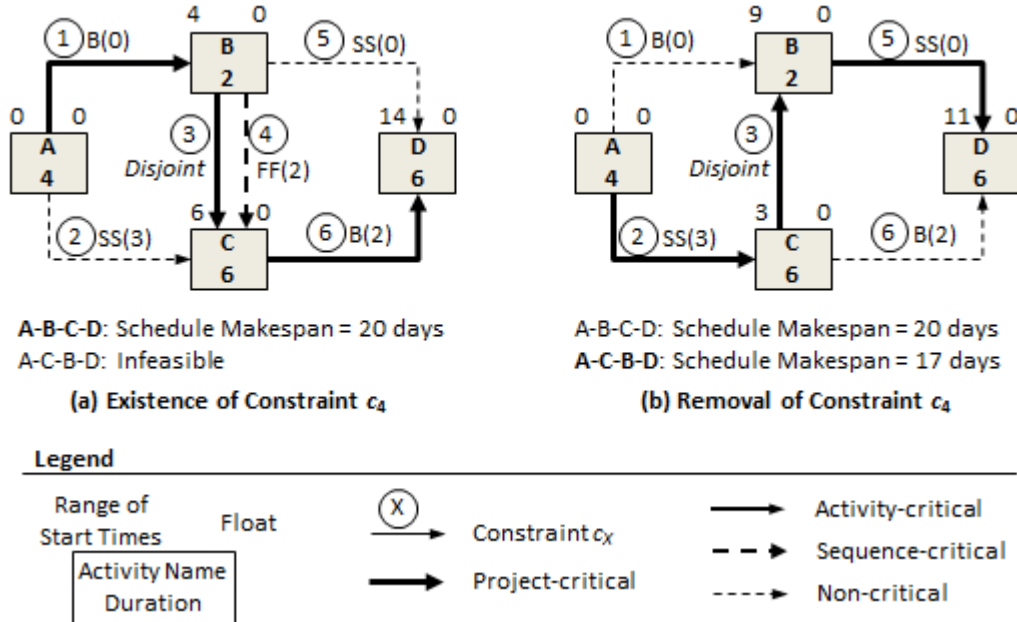


Figure 6.2. Example of sequence-critical constraint

Due to their special nature, sequence-critical constraints should be carefully examined to see if they can be removed to achieve better project makespan.

#### 6.2.1.4. Redundant Constraint

The last category of constraint is named “redundant”, which refers to constraints whose change and existence have no impact on start/finish time of any activity, project duration and the activity sequence defining the overall project duration.

### 6.2.2. Order of Constraint Criticality

Constraint criticality in a schedule can be ordered as follows:

$$Project\text{-}critical \succ Activity\text{-}critical \succ Sequence\text{-}critical \succ Redundant$$

Project-critical constraints are apparently the most crucial since not satisfying them can prolong the schedule makespan. Secondly, activity-critical constraints also need to be well-managed in order to maintain activity's times as planned. Although not crucial to the schedule times, sequence-critical constraints cannot be ignored since their removal can allow for project improvement.

### 6.3. Identifying Constraint Criticality

Similar to activity criticality, constraint criticality may be determined based on criticality indicators. The criticality of a constraint is closely related to whether it may prolong schedule makespan or may reduce the feasible ranges of activities' start times when the constraint becomes more obstructive to project performance or more tightened. In other words, if a constraint has less room by being tightened or conversely, less relaxed, it yields a higher degree of criticality. Accordingly, two types of relaxation time are proposed to characterize the criticality of a constraint, defined as follows:

- **Aggregate Relaxation Time:** The Aggregate Relaxation Time (*ART*) of a constraint  $c$ , denoted as  $ART_c$ , is the total amount of time that its lag time and/or associated activities' times (start time and/or duration) can be varied without violating the constraint, and thus without increasing schedule makespan.
- **Intrinsic Relaxation Time:** The Intrinsic Relaxation Time (*IRT*) of a constraint  $c$ , denoted as  $IRT_c$ , is the total amount of time that its lag time can be varied without reducing the feasible start time ranges of all activities involved while activity durations remain unchanged.



Table 6.1. Relationship of criticality and relaxation times

Criticality	Relaxation Times
Project-critical	$ART = IRT = 0$
Activity-critical	$ART > 0$ $IRT = 0$
Sequence-critical	$ART > 0$ $IRT > 0$ Removal provides better project duration
Non-critical	$ART > 0$ $IRT > 0$ Removal has no impact on project duration

The distinction of the two relaxation times is in two aspects. The first refers to the impact of change: on makespan for *ART* and activity times for *IRT*. The second is directed to the scope of change: both lag and activity times for *ART* while merely lag time for *IRT*. The relationship between *ART* and *IRT* and the criticality of a constraint is shown in Table 6.1. A project-critical constraint will cause project delay if it is further tightened, and thus its relaxation times are zero. On the other hand, an activity-critical constraint still can be tightened without affecting schedule makespan, yet affecting the feasible time range of the activities involved. Hence, an activity-critical constraint has zero *IRT* and non-zero *ART*.

### 6.3.1. Determining Constraint Relaxation Times

The *ART* and *IRT* of a constraint may be computed by the introduction of flexibility measures of the activities involved. The flexibility of an activity with regard to a constraint can be measured based on the amount of time that its start time can be changed (pushed forward or pulled backward) from the original start time range without violating the constraint while the times (duration and start time) of its

successor/predecessor involved in the constraint, lag time and its duration remain unchanged. Similar to relaxation times, two types of flexibility times are introduced as follows:

- **Aggregate Forward/Backward Flexibility Time:** The Aggregate Forward/Backward Flexibility Time of an activity  $k$  with regarding to a constraint  $c$ , denoted as  $AFT_{k,c}^{FW} / AFT_{k,c}^{BW}$ , is the amount of time that  $k$  can be moved forward/backward without violating  $c$ , while its duration, the feasible time range of its successor/predecessor involved in  $c$  and lag time remain unchanged.
- **Intrinsic Forward/Backward Flexibility Time:** The Intrinsic Forward/Backward Flexibility Time of an activity  $k$  with regarding to a constraint  $c$ , denoted as  $IFT_{k,c}^{FW} / IFT_{k,c}^{BW}$ , is the amount of time that  $k$  can be moved forward/backward beyond its original feasible range without violating  $c$ , while its duration, the feasible time range of its successor/predecessor involved in  $c$  and lag time remain unchanged.

Flexibility times of an activity with respect to a constraint show how flexible the activity can be scheduled without affecting the constraint's satisfaction. As such, less flexibility times indicate that the activity is less flexible or more constrained. In addition,  $AFT$  refers to the flexibility of an activity taking into account its total float time while  $IRT$  does not involve float time.

Let  $k^- = [L_k .. U_k]$  denote the original feasible start time of activity  $k$  obtained from schedule computation result where  $L_k$  and  $U_k$  are early and late start time of  $k$ , and  $k_c^- = [L_{k,c} .. U_{k,c}]$  the start time range of  $k$  governed only by constraint  $c$ . The

intrinsic and aggregate flexibility times of activity  $k$  with respect to constraint  $c$  can be determined from the difference between  $k_c^-$  and  $k^-$  as described in equations (4.32) and (4.33) respectively.

$$\begin{aligned} IFT_{k,c}^{FW} &= U_{k,c} - U_k \\ IFT_{k,c}^{BW} &= L_k - L_{k,c} \end{aligned} \quad (4.32)$$

$$\begin{aligned} AFT_{k,c}^{FW} &= U_{k,c} - L_k \\ AFT_{k,c}^{BW} &= U_k - L_{k,c} \end{aligned} \quad (4.33)$$

It can be inferred from the above definitions that  $IFT_{k,c}^{BW}$  and  $IFT_{k,c}^{FW}$  respectively refer to the flexibility of early and late times of activity  $k$  with respect to constraint  $c$ . Hence, zero  $IFT_{k,c}^{BW} / IFT_{k,c}^{FW}$  indicates that  $c$  defines the early/late time of  $k$ . Moreover, the relationship between  $IFT$  and  $AFT$  can be described as

$$\begin{aligned} AFT_{k,c}^{FW} &= IFT_{k,c}^{FW} + TF_k \\ AFT_{k,c}^{BW} &= IFT_{k,c}^{BW} + TF_k \end{aligned} \quad (4.34)$$

where  $TF_k = U_k - L_k$  is the total float of activity  $k$ .

Since the criticality of a constraint corresponds to how flexibly its associated activities can be changed without violating it, the relaxation time of a constraint is then defined as the minimal flexibility time of all associated activities, given by:

$$\begin{aligned} ART_c &= \underset{k}{\text{Min}}(AFT_{k,c}^{FW}, AFT_{k,c}^{BW}) \quad \forall \text{ activity } k \text{ involved in } c \\ IRT_c &= \underset{k}{\text{Min}}(IFT_{k,c}^{FW}, IFT_{k,c}^{BW}) \quad \forall \text{ activity } k \text{ involved in } c \end{aligned} \quad (4.35)$$

For illustration, consider constraint  $c_5: B \text{ FF}(2) D$ , or  $c_5: B^- + d_B + 2 \leq D^- + d_D$  in Figure 6.1a, with  $d_B = 8$ ,  $d_D = 7$ ,  $B^- = [2..5]$  (or  $L_B = 2$ ,  $U_B = 5$ ) and  $D^- = [9]$  (or  $L_D = U_D = 9$ ). Note that  $D$  is a critical activity. The start times of  $B$  and  $D$  as defined by only  $c_5$  are  $B_5^- = (-\infty..6]$  (or  $L_{B,5} = -\infty$ ,  $U_{B,5} = 6$ ), and  $D_5^- = [5..\infty)$  (or  $L_{D,5} = 5$ ,  $U_{D,5} = \infty$ ). Accordingly, the flexibility times of  $B$  and  $D$  can be determined as follows:

$$IFT_{B,5}^{FW} = U_{B,5} - U_B = 6 - 5 = 1, \quad AFT_{B,5}^{FW} = U_{B,5} - L_B = 6 - 2 = 4,$$

$$IFT_{B,5}^{BW} = L_B - L_{B,5} = \infty, \quad AFT_{B,5}^{BW} = U_B - L_{B,5} = \infty, \text{ and similarly}$$

$$IFT_{D,5}^{BW} = AFT_{D,5}^{BW} = 9 - 5 = 4, \quad IFT_{5,D}^{FW} = AFT_{5,D}^{FW} = \infty.$$

Consequently, the relaxation times of constraint  $c_5$  are determined as

$$IRT_5 = \text{Min}(IFT_{B,5}^{FW}, IFT_{B,5}^{BW}, IFT_{D,5}^{FW}, IFT_{D,5}^{BW}) = 1 \text{ determined by } IFT_{B,5}^{FW}, \text{ and}$$

$$ART_5 = \text{Min}(AFT_{B,5}^{FW}, AFT_{B,5}^{BW}, AFT_{D,5}^{FW}, AFT_{D,5}^{BW}) = 4 \text{ by } AFT_{B,5}^{BW} \text{ and } AFT_{D,5}^{FW}.$$

The significance of these relaxation times can be perceived in this way. With  $IRT_5 = 1$ , the lag time of  $c_5$  ( $m_5 = 2$ ) could be increased by  $\Delta T_5 = 1$  to  $m_5 = 3$  without affecting the start times and floats (i.e. time ranges) of activities  $B$  and  $D$ . However, if  $m_5$  is increased by  $\Delta T_5 = 2$  to from  $m_5 = 2$  to  $m_5 = 4$ , the feasible time range of activity  $B$  is reduced to  $B^- = [2..4]$  to satisfy  $c_5$  since this change exceeds the forward intrinsic flexibility time of active  $B$ , ( $IFT_{B,5}^{FW} = 1$ ). On the other hand,  $ART_5 = 4$  indicates that the  $\Delta T_5 = 2$  change does not affect project duration. Specifically, it can accommodate

changes in lag time or activity times (start time and/or duration) not totaling more than 4 days without affecting project completion. In particular, at the extremities, it is still satisfied if its  $m_5$  can be increased by  $\Delta T_5 = 4$  to  $m_5 = 6$  and activity  $B$  is carried out on Day 2 ( $B^- = 2$ ), or if activity  $B$  can be delayed by  $\Delta T_5 = 4$  days ( $B^- = 6$ ) and its lag remains as  $m_5 = 2$ . However, if lag time is increased to  $m_5 = 7$ , constraint  $c_5$  is violated and the schedule makespan needs to be increased to resolve the violation. The impact of constraint variation on schedule makespan will be examined in section 6.4.

### 6.3.2. Interpreting Constraint Relaxation Times

Each constraint comprises two principle elements: lag time and activities' temporal attributes (start times and durations). Relaxation times of a constraint indicate the temporal magnitude in which these two elements can be varied while maintaining the satisfaction of the constraint. As illustrated in the previous example, the *IRT* of a constraint represents the maximal time amount that its lag time can be increased (for minimal lag) or decreased (for maximal lag) without causing any change to the feasible time ranges of the associated activities when activity durations are unchanged. If lag time change exceeds *IRT*, activities' feasible time ranges will be reduced accordingly. The *ART* of a constraint on the other hand refers to the maximal total time amount that its lag time and activities could be changed without affecting its satisfaction, and thus not delaying the schedule makespan.

*IRT* could be considered as "free" relaxation time of a constraint to be analogous to the free float from the activity perspective. Since any change within *IRT* does not reduce the feasible time range of activities or their total float, such a variation only happens within the constraint and does not affect the *IRT* of other constraints. *ART* on

the other hand represents the relaxation time of both lag time and activities' times. Thus, it is shared among the constraints involving the same activities and could be considered as "total" relaxation time to be analogous to the total float from the activity perspective. Changes in the *ART* of one constraint will lead to variations in the *ART* of other constraints, while changes in the *IRT* will not lead to variations in other constraints. Accordingly, *IRT* and *ART* are used to capture the inherent and aggregated changes of a constraint.

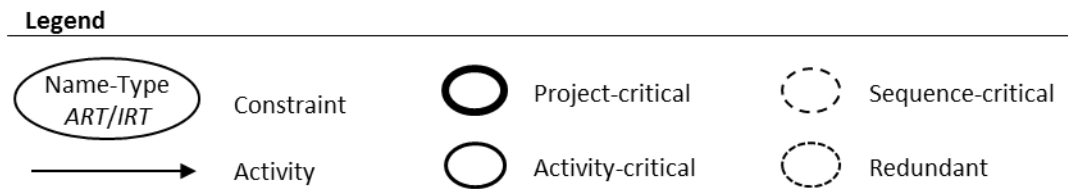
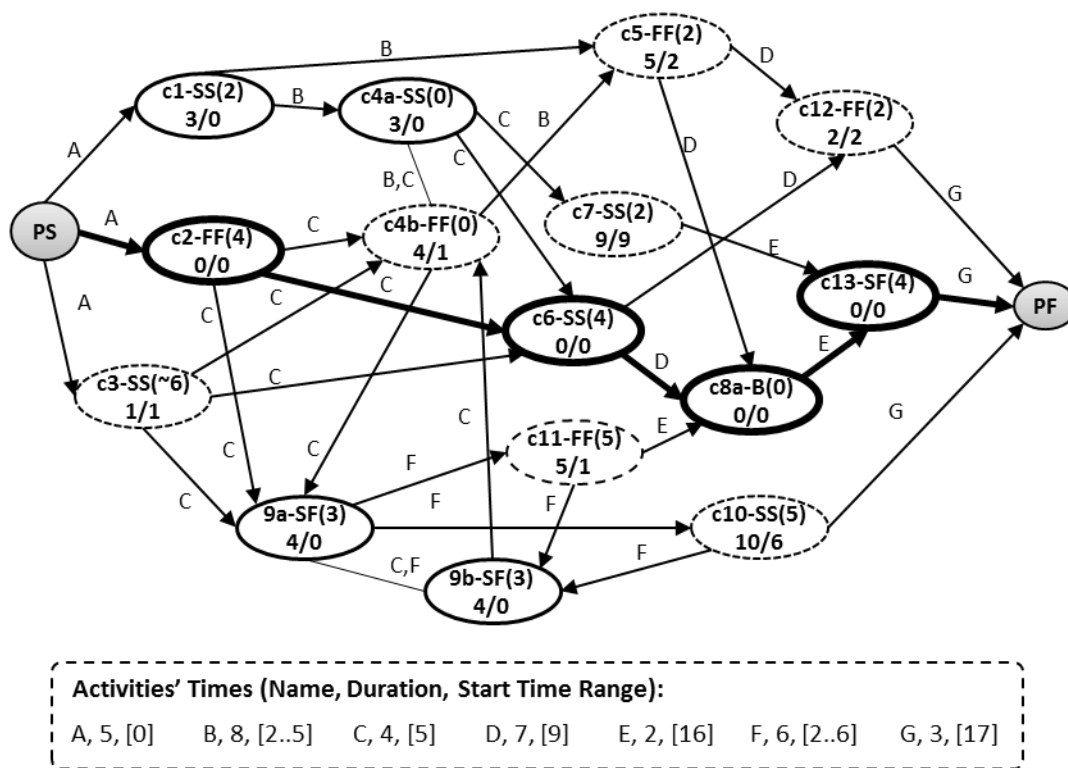


Figure 6.3. Example constraint network showing relaxation times

Figure 6.3 depicts an equivalent constraint network of the schedule shown in Figure 6.1a where nodes represent constraints and the edges denotes activities. With this representation from the constraint perspective, the relationship between the relaxation times of associated constraints can be better conveyed. In essence, changes in both activity and lag times can be reflected as changes of the *ART* and *IRT* of the associated constraint(s), and then propagated throughout the downstream network. For example, the *ART* of constraint  $c_1$  ( $ART_1 = 3$ ) is shared among all non-project-critical constraints involving non-critical activities. Consequently, if the start time of activity  $B$  is delayed by 2 days ( $\Delta T_1 = 2$ ) from  $B^- = [2..5]$  to  $B^- = [4..5]$ ,  $ART_1$  is reduced from  $ART_1 = 3$  to  $ART_1 = 1$  correspondingly, and the *ART*s of the constraints related to activity  $B$  also decreased similarly, shown as:  $ART_{4a} = 1$ ,  $ART_{4b} = 2$ , and  $ART_5 = 3$ . Especially, the *IRT* of these constraints remains unchanged (as  $IRT_5 = 2$  and  $IRT_{4b} = 1$ ) since they original change is from an activity (activity  $B$ ).

#### **6.4. Criticality of Construction Requirements**

Construction requirements can be seen as conjunctive and disjunctive combinations of one or many simple constraints. For generality, a construction requirement comprising only one simple constraint is considered as a conjunctive combination. The criticality of a construction requirement is therefore derived from that of its constituent constraints. In addition, since some constraints may not be involved in some schedules due to some disjunctive requirements, the existence aspect must be taken into account for identifying the criticality of construction requirements.

From the perspective of a single alternative schedule, the criticality of a conjunctive group of active constraints is characterized by the highest degree of criticality of its constituent constraints, expressed as:

$$\mathbb{C}_{(c_1 \wedge c_2 \wedge \dots \wedge c_n)} = \sup(\mathbb{C}_1, \mathbb{C}_2, \dots, \mathbb{C}_n) \quad (4.36)$$

where  $\mathbb{C}_i$  denotes the criticality of constraint  $c_i$ .

For instance, consider  $C_4: B \text{ Contains } C$  in Figure 6.1a comprising a conjunctive combination of two simple constraint as  $C_4: (c_{4a} \wedge c_{4b})$ . Constraint  $c_{4a}$  defining the start time of activity B is activity-critical while constraint  $c_{4b}$  is non-critical. Consequently, according to the characterization given by Equation (4.36),  $C_4$  is identified as activity-critical.

On the other hand, the criticality of a disjunctive requirement with respect to an alternative schedule is defined by the criticality of its constituent constraints which are active or existent in that schedule. Consider the disjunctive requirement  $C_8: (c_{8a} \vee c_{8b})$  in Figure 6.1, for example. In Schedule 1, constraint  $c_{8a}$  exists while  $c_{8b}$  is not active. Therefore, the criticality of requirement  $C_8$  in Schedule 1 is determined by the criticality of constraint  $c_{8a}$  as project-critical.

From the perspective of multiple alternatives, the criticality of construction requirements overall multiple schedules can be characterized as follows:

- **Super-critical Requirement:** A construction requirement is classified as “Super-critical” if it is project-critical in all alternative schedules. This class of requirements should receive more attention from managers since their delays or



violations will invariably affect project duration. As an example, requirement  $C_8$  shown in Figure 6.1 is super-critical since it is identified as project-critical in both alternative schedules. Consequently, no matter which alternative schedule is selected for execution, this requirement requires careful attention.

- **Alternative-critical Requirement:** A construction requirement is classified as “alternative-critical” if it is project-critical in at least one alternative schedules. Identifying them allows for plan flexibility when unforeseen circumstances occur which perturb the plan. Hence, when an “alternative-critical” requirement is perturbed, a possible mitigation may be to proceed with an alternative schedule where the affected constraint is no longer critical to the schedule duration. For example, in Figure 6.1, the start requirement between activities  $C$  and  $D$ , represented by constraint  $c_4$  is alternative-critical since it is identified as project-critical in Schedule 1 but not in Schedule 2. If Schedule 1 is chosen for execution and if this requirement is subsequently perturbed, alternative Schedule 2 could be considered and put into action.

The identification of “super-critical” and “alternative-critical” requirements allows managers to determine the driving construction requirements of the project, so that appropriate managerial action may be taken when necessary. From the alternative-critical requirements, managers can then identify those requirements which if violated could allow for alternative schedules to be considered. For completeness, the criticality classification of requirement also includes another two following types.

- **Quasi-critical Requirement:** A requirement is classified as “quasi-critical” if it is not project-critical in any alternative schedules and there is at least one alternative schedule in which it is not identified as non-critical. By this, the

quasi-criticality is a mixture of activity-criticality, sequence-criticality, and redundancy. When the criticality of a construction requirement are “activity-critical” or “sequence-critical” in all alternative schedules, it will be classified as “*quasi-activity-critical*” and “*quasi-sequence-critical*” respectively.

- **Redundant Requirement:** A construction requirement is classified as “non-critical” if it is identified as non-critical in all alternative schedules. Changes in such requirements will have no impact on project completion time.

Some complex requirements may be made up of hierarchical (or nested) disjunctive and conjunctive operators. Under these circumstances, the criticality of requirements is evaluated hierarchically as illustrated in Table 6.2. In this example for an arbitrary problem with four alternative schedules, the criticality of requirement  $R_1$  in each alternative schedule is defined by the criticality of either a conjunctive combination ( $c_1 \wedge c_2$ ) or  $c_3$ . In Schedule 1, the criticality of ( $c_1 \wedge c_2$ ) is project-critical given by the supreme of that of  $c_1$  and  $c_2$  following Equation (4.36), while  $c_3$  is non-existent. As a result, the criticality of requirement  $R_1$  in Schedule 1 is project-critical. On the other hand in Schedule 2 in which only  $c_3$  is active, the criticality of  $R_1$  is defined by that of  $c_3$  to be activity-critical. On the whole, requirement  $R_1$  is classified as alternative-critical since it is not project-critical in all alternative schedules.

Table 6.2. Criticality of complex and simple construction requirements

					Complex Construction Requirement	Simple Construction Requirement
	$c_1: A \text{ SS}(2) B$	$c_2: B \text{ B}(2) C$	$(c_1 \wedge c_2)$	$c_3: C \text{ SS}(3) B$	$R_1 = [(c_1 \wedge c_2) \vee c_3]$	$c_4: C \text{ SS}(3) D$
Schedule 1	Project-critical	Redundant	Project-critical	Non-existent	Project-critical	Activity-critical
Schedule 2	Non-existent	Non-existent	Non-existent	Activity-critical	Activity-critical	Activity-critical
Schedule 3	Project-critical	Redundant	Project-critical	Non-existent	Project-critical	Redundant
Schedule 4	Non-existent	Non-existent	Non-existent	Redundant	Redundant	Redundant
Overall					Alternative-critical	Quasi-critical

This classification approach can also be applied to simple construction requirements which comprise only one constraint. As shown in the last column of Table 6.2, the overall criticality of constraint  $c_4$  can be defined from its criticality in all alternative schedules as quasi-critical.

Similar to constraint criticality, the criticality of construction requirements from the perspective of multiple alternative schedules can be ordered as follows:

$$\textit{Super-critical} \succ \textit{Alternative-critical} \succ \textit{Quasi-critical} \succ \textit{Non-critical}$$

Super-critical requirements are the most important since they are project-critical in all alternative schedules and thus invariantly govern project completion time. Secondly, alternative-critical requirements also govern project makespan but not in all alternatives; hence, they allow for plan flexibility and should also receive special attention. Thirdly, although not defining project duration, quasi-critical requirements cannot be simply considered redundant as they have impact on activities' time or sequence in some alternative schedules. Finally, redundant requirements govern neither activities' time nor construction sequence in all alternative schedules.

## **6.5. Schedule Change Analysis from the Perspective of Construction Requirements**

This section presents an approach for analyzing schedule change from the perspective of construction requirements. Due to the dynamic environment, construction projects are subjected to numerous changes from different sources and by various causes. Project changes have apparent impacts on different aspects of construction process including schedule, cost, and project's performance (Hanna et al., 1999; Ibbs et al., 2001). Change is also a major cause of delay, disruption and disputes among construction parties (Motawa et al., 2007; Zhao et al., 2010). Therefore, analysing impact of project changes is necessary for project management.

From the viewpoint of scheduling, project changes can be reflected in variations of schedule constraints which can be categorized in two groups: (1) variation (decrease/increase) of relaxation times caused by changes in activity times (start/finish time or duration) and lag time, and (2) introduction of a new or removal of an existing constraint. In general, constraint variations could have beneficial, neutral or disruptive impact on schedule makespan. They may also lead to an inconsistency in the constraint set, which cannot be resolved by changing the schedule makespan. The inconsistent constraint group can be identified using the preemptive constraint analysis approach presented in chapter five. Accordingly, the proposed approach aims at analyzing the impact of a constraint variation on the makespan of a schedule when any inconsistency caused by such a variation can be resolved with a new schedule makespan.

### **6.5.1. Schedule Makespan Change by Variations of Relaxation Times**

The *ART* of a constraint is increased when the constraint is relaxed and conversely decreased if the constraint is tightened. Table 6.3 depicts the causes of

constraint tightening of 4 unary and 4 simple minimal-lag binary constraints. Variations can originate from changes in lag time ( $m$ ), activities' start times ( $X^-$  and  $Y^-$ ) and durations ( $d_x$  and  $d_y$ ). Up and down arrows respectively denote value increase and decrease, while a dash sign refers to an invariant relationship between lag/activity times and  $ART$ . Conversely, changes in the opposite direction will lead to constraint relaxation. The impact of changes of lag and activities' times on  $ART$  of maximal-lag constraints is converse to that of the corresponding minimal-lag constraints.

Table 6.3. Changes of lag and activities' time leading to constraint tightening

Constraint	Mathematical Definition	$m$	$X^-$	$d_x$	$Y^-$	$d_y$
$X SB(m)$	$X^- \leq m$	↓	↑	-	-	-
$X SA(m)$	$X^- \geq m$	↑	↓	-	-	-
$X DB(m)$	$X^- + d_x \leq m$	↓	↑	↑	-	-
$X SA(m)$	$X^- + d_x \geq m$	↑	↓	↓	-	-
$X SS(m)Y$	$X^- + m \leq Y^-$	↑	↑	-	↓	-
$X B(m)Y$	$X^- + d_x + m \leq Y^-$	↑	↑	↑	↓	-
$X FF(m)Y$	$X^- + d_x + m \leq Y^- + d_y$	↑	↑	↑	↓	↓
$X SF(m)Y$	$X^- + m \leq Y^- + d_y$	↑	↑	-	↓	↓

A constraint that is not project critical becomes project-critical when its  $ART$  is reduced to zero, while relaxing such a constraint has no impact on schedule makespan. On the other hand, the relaxing or tightening of a project-critical constraint can directly affect schedule makespan, and this depends on the lag type as depicted in Table 6.4. For minimal-lag type, relaxing a project-critical constraint can relax the entire schedule and allow the makespan to be shortened while tightening a project-critical constraint will make it violated and the makespan must be prolonged to resolve the constraint

violation. Conversely, for a maximal-lag type, relaxing a project-critical constraint causes no change to schedule makespan while tightening it can lead to inconsistency, and thus an infeasible schedule.

Table 6.4. Impact of variation of project-critical constraint on schedule makespan

Lag type	Change of project-critical constraint	
	Relaxed ( $ART \uparrow$ )	Tightened ( $ART \downarrow$ )
Minimal-lag	Shortened makespan	Lengthened makespan
Maximal-lag	Unchanged makespan	Infeasible schedule

#### 6.5.1.1. Change in Schedule Makespan through Constraint Tightening

In general, tightening a minimal-lag constraint beyond its  $ART$  will violate the constraint and lead to schedule delay. Besides, the underlying requirements in which all activities must be carried out within the project timespan, from project start time ( $PS$ ) to project end time ( $PT$ ) are explicitly expressed by assigning two constraints, *Start-After*( $PS$ ) and *Due-Before*( $PT$ ) (or in short form as  $SA(PS)$  and  $DB(PT)$ ) to all activities, so that the analysis method can be applied to all activity changes without checking if the activity is the first or the last in the network. The delay amount  $\Delta T_p$  is dependent on how much a constraint say  $c_i$  is tightened beyond its  $ART$ , and is determined by the difference between its total amount of tightening in time unit (denoted by  $\Delta T_i$ ) and its  $ART$ , given as:

$$\Delta T_p = \max\{0, (\Delta T_i - ART_i)\} \quad (4.37)$$

Consider the simple network with three constraints and an original makespan of 14 days (see Figure 6.4) for example.  $c_{k,s}$  and  $c_{k,f}$  denote two implied constraints  $k$   $SA(0)$  and  $k$   $DB(14)$  the added to every activity  $k$ . The relaxation times  $ART$  and  $IRT$  of

all constraints are also presented in the figure for easy reading. When the duration of activity  $B$  is reduced by 2 days ( $d_B = 6$ ), constraint  $c_1$  is violated according to Table 6.3, and thus the schedule is delayed by  $\Delta T_p = \Delta T_1 - ART_1 = 2 - 0 = 2$  days. The same value is obtained by re-computing the schedule with the new value of  $d_B$ .

Similarly, if the duration of activity  $C$  is increased from  $d_C = 10$  to  $d_C = 13$ , none of the original constraints are violated yet the underlying constraint  $c_{C,f}$  is violated, and consequently the makespan is prolonged by  $\Delta T_p = \Delta T_{C,f} - ART_{C,f} = 3 - 0 = 3$  days. In another scenario, although  $c_2$  is non-critical with  $ART_2 = 2$ , if its lag time is increased from  $m_2 = 0$  to  $m_2 = 4$ ,  $c_2$  is violated and consequently the makespan is increased by  $\Delta T_p = \Delta T_2 - ART_2 = 4 - 2 = 2$  days.

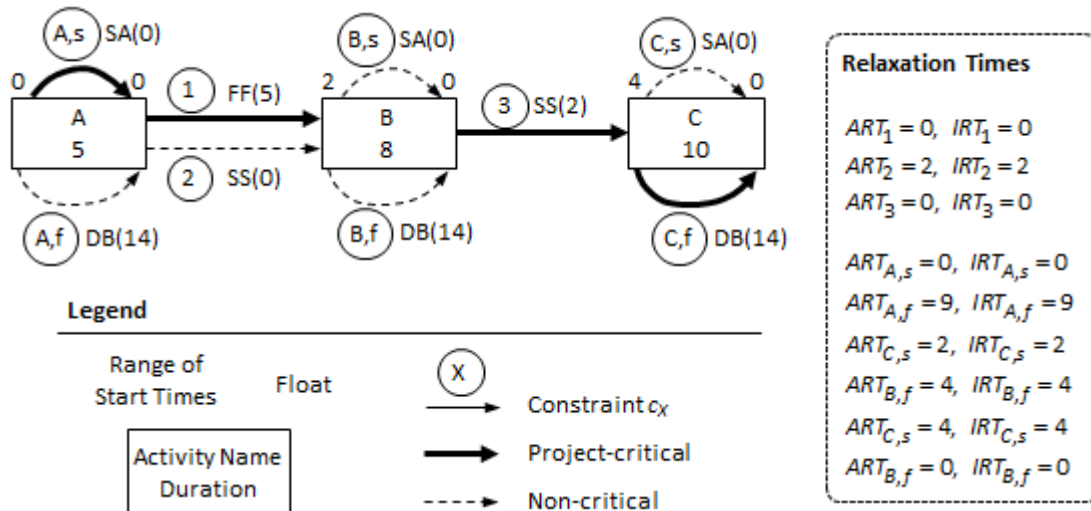


Figure 6.4. Example schedule for analyzing schedule change

### 6.5.1.2. Change in Schedule Makespan through Constraint Relaxation

Relaxing a project-critical constraint will allow schedule shortening if the constraint is involved in all critical paths. Let  $N$  be the entire activity network,  $N_C =$

$(X_1, X_2, \dots, X_k, \dots, X_n)$  be the sub-network of critical activities. When a project-critical constraint  $c_i$  between two critical activities  $(X_k, X_l)$ , expressed as  $c_i(X_k, X_l)$ , is relaxed, the relaxation is propagated throughout downstream sub-network (called relaxed sub-network and denoted as  $N_R$ ) which includes all critical activities in  $(X_l, \dots, X_n)$  and their successors as illustrated in Figure 6.5. Then, schedule can be shortened, and the shortening amount is defined by the relaxation amount ( $\Delta T_i$ ) and the following ARTs:

- (i)  $ART_{(I)}$  of constraint  $c_{l,s}: X_l$  SA(PS), which requires  $X_l$  to start on or after project start time under any condition. This constraint is taken into account to ensure that the relaxed sub-network  $N_D$  starts on or after project start time after being shortening.
- (ii)  $ART_{(II)}$  of all non-project-critical constraints  $c_j(X_p, X_l)$  linking to  $X_l$ ,  $\forall X_p$  be the precedent of  $X_l$ .
- (iii)  $ART_{(III)}$  of all constraints  $c_j(X_p, X_q)$  involving at least one non-critical activity not belonging to the relaxed sub-network, expressed as:

$$c_j(X_p, X_q) \text{ with } \begin{cases} X_p \notin N_R, \text{ or} \\ X_q \notin N_R, \text{ or} \\ X_p \notin N_R, X_q \notin N_R \end{cases} \quad (4.38)$$



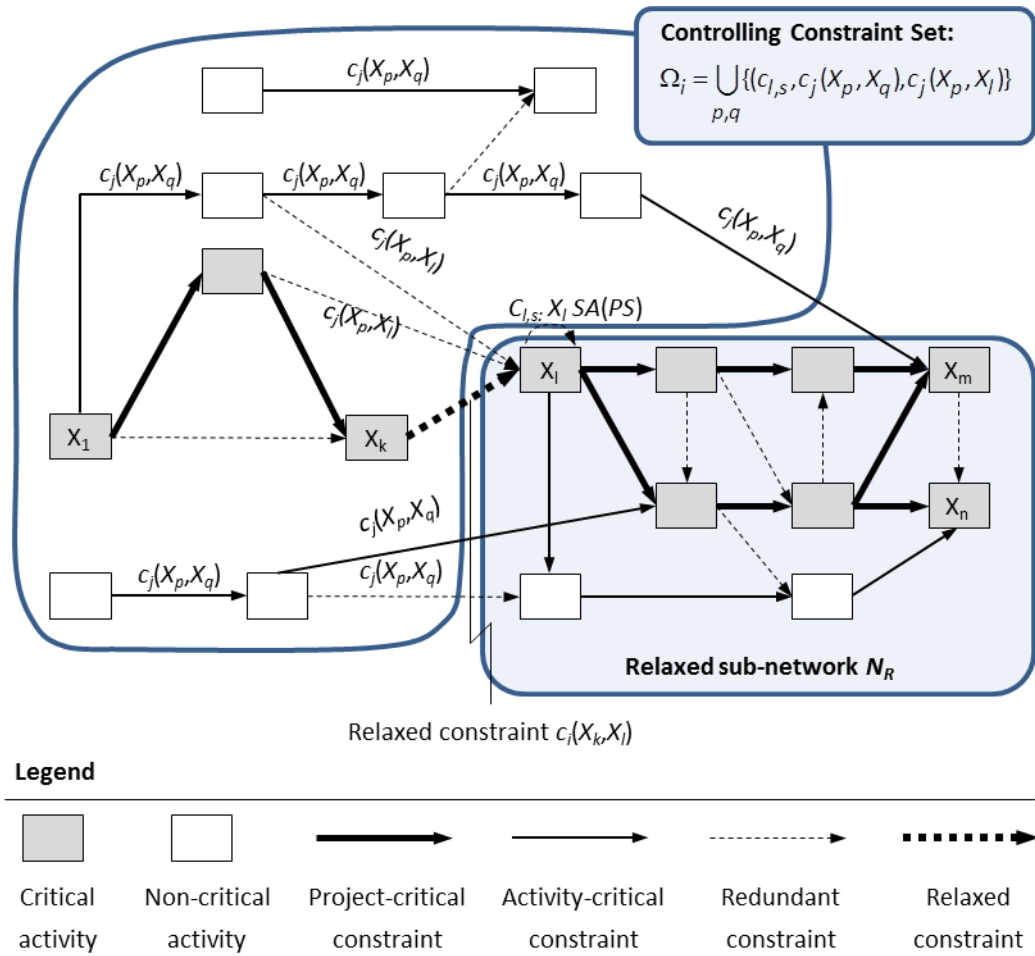


Figure 6.5. Schedule change analysis under constraint relaxation

The makespan shortening time ( $\Delta T_P$ ) by relaxing a constraint  $c_i$  is determined as the minimal among the relaxation in time unit ( $\Delta T_i$ ) and the original ARTs of constraints belonging to three groups above, given by:

$$\Delta T_P = \text{Min}(\Delta T_i, \text{ART}_{(I)}, \text{ART}_{(II)}, \text{ART}_{(III)}) \quad (4.39)$$

The collection of all constraints belonging to groups (i), (ii), and (iii) when relaxing a constraint  $c_i$  is called controlling constraint set and denoted as  $\Omega_i$  as:

$$\Omega_i = \bigcup_{p,q} \{(c_{l,s}, c_i(X_p, X_q), c_j(X_p, X_l))\} \quad (4.40)$$

Then,  $\Delta T_P$  can be alternatively expressed as:

$$\Delta T_P = \text{Min}(\Delta T_i, ART_j) \quad \forall c_j \in \Omega_i \quad (4.41)$$

When the shortening amount of the makespan is not defined by  $\Delta T_i$  ( $\Delta T_P < \Delta T_i$ ), the critical constraint path is changed to another. New project-critical constraints can be identified through the new relaxation time of those in the controlling constraint set ( $ART_j'$ ), given by:

$$ART_j' = ART_j - \Delta T_P \quad \forall c_j \in \Omega_i \quad (4.42)$$

The originally project-critical constraints in the sub-network ( $N_R$ ) will no longer be project-critical. In general, the updated relaxation time of all constraints  $c_k$  in the sub-network  $N_R$  is increased an amount of  $(\Delta T_i - \Delta T_P)$ , given by:

$$ART_k' = ART_k + (\Delta T_i - \Delta T_P) \quad \forall c_k \in N_R \quad (4.43)$$

For example, when the duration of activity  $B$  in Figure 6.4 is increased by 1 day ( $d_B = 9$ ), constraint  $c_1: A \text{ FF}(5) B$  is relaxed according to Table 6.3. Controlling constraint set  $\Omega_1$  includes two constraints  $c_{B,f}$  and  $c_2$  belonging to groups (2) and (3), respectively. The schedule makespan is reduced by 1 day, determined as

$$\Delta T_P = \text{Min}(\Delta T_1, ART_2, ART_{B,s}) = \text{Min}(1, 2, 2) = 1$$

Similarly, if constraint  $c_1$  is relaxed from FF(5) to FF(2), the schedule can be shortened by only 2 days due to constraints  $c_2$  and  $c_{B,s}$ , shown as:

$$\Delta T_P = \text{Min}(\Delta T_1, ART_2, ART_{B,s}) = \text{Min}(3, 2, 2) = 2$$

The new relaxation times of these constraints are identified as:

$$\begin{aligned} ART_2' &= ART_2 - \Delta T_p = 2 - 2 = 0, \\ ART_{B,s}' &= ART_{B,s} - \Delta T_p = 2 - 2 = 0, \\ ART_1' &= ART_1 + (\Delta T_1 - \Delta T_p) = 0 + (3 - 2) = 1 \end{aligned}$$

Therefore, constraint  $c_2$  and the implicit constraint  $c_{B,s}$  become project-critical while  $c_1$  becomes non-critical.

### 6.5.2. Change in Schedule Makespan through Adding/Removing a Constraint

New requirements may arise along the project's lifecycle, resulting in new schedule constraints. If the new constraint  $c_i$  is violated ( $ART_i < 0$ ) based on the original schedule, the schedule makespan will be prolonged to accommodate the new constraint by  $\Delta T_p$  determined as:

$$\Delta T_p = |ART_i| \quad (4.44)$$

If a new constraint  $c_4: A \text{ B}(2) \text{ C}$  (or  $c_4: A^- + 5 + 2 \leq C^-$ ) is added to the schedule network in Figure 6.4. With the original start times of  $A$  ( $A^- = [0]$ ) and  $C$  ( $C^- = [4]$ ),  $c_4$  is violated with  $ART_4 = -3$ . Hence, the schedule is delayed by  $\Delta T_p = |-3| = 3$  days.

In contrast, removing a project-critical constraint will shorten the schedule makespan if the constraint belongs to all critical paths. Similar to the case of constraint relaxation, the shortening amount ( $\Delta T_p$ ) when a constraint  $c_i(X_k, X_l)$  is removed is governed by the  $ARTs$  of non-project-critical constraints  $c_p$  related to activities in the downstream sub-network (as characterized in section 6.5.1.2) and given by

$$\Delta T_P = \text{Min}(ART_j) \quad \forall c_j \in \Omega_i \tag{4.45}$$

For example, when constraint  $c_1$  in the example schedule (shown in Figure 6.4) is removed, the schedule makespan can easily be recalculated as 12 days as

$$\Delta T_P = \text{Min}(ART_2, ART_{B,s}) = \text{Min}(2, 4) = 2$$

### 6.6. Illustrative Example

An example schedule project (depicted in Figure 6.6) is used to demonstrate application of the proposed concepts. This example project consists of 7 activities and 16 simple constraints. Four construction requirements (denoted as R1 to R4 in the figure) have been identified for the project. Requirement R1 defines conjunctive relationships between constraints ( $c_2$  and  $c_3$ ). Requirements R2 to R4 are disjunctive combinations of constraints ( $c_{5a}$  and  $c_{5b}$ ), ( $c_{8a}$  and  $c_{8b}$ ) and ( $c_{9a}$  and  $c_{9b}$ ) respectively.

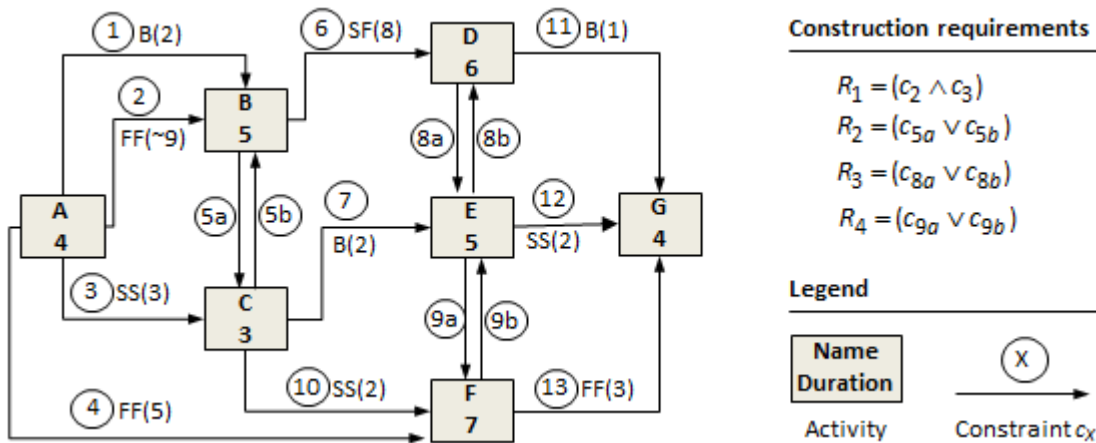


Figure 6.6. Illustrative example for criticality analysis

Table 6.5. Result from criticality analysis

Constraint/ Requirement	S1 (29)			S2 (24)			S3 (30)			S4 (34)			Overall
	ART	IRT	Type	ART	IRT	Type	ART	IRT	Type	ART	IRT	Type	
C <sub>1</sub>	0	0	PC	0	0	PC	0	0	PC	0	0	PC	Super-critical
C <sub>2</sub>	5	5	SC	5	5	SC	5	5	SC	5	5	SC	Quasi-sequence-critical
C <sub>3</sub>	6	6	RE	6	6	RE	6	6	RE	6	6	RE	Redundant
C <sub>4</sub>	17	17	RE	9	9	RE	18	17	RE	9	9	RE	Redundant
C <sub>5a</sub>	0	0	PC	0	0	PC	0	0	PC	0	0	PC	Super-critical
C <sub>5b</sub>	-	-	-	-	-	-	-	-	-	-	-	-	-
C <sub>6</sub>	2	0	AC	6	0	AC	13	13	RE	17	17	RE	Quasi-critical
C <sub>7</sub>	0	0	PC	4	4	RE	0	0	PC	4	4	RE	Alternative-critical
C <sub>8a</sub>	2	0	AC	6	0	AC	-	-	-	-	-	-	Quasi-activity-critical
C <sub>8b</sub>	-	-	-	-	-	-	0	0	PC	0	0	PC	Super-critical
C <sub>9a</sub>	0	0	PC	-	-	-	1	0	AC	-	-	-	Alternative-critical
C <sub>9b</sub>	-	-	-	0	0	PC	-	-	-	0	0	PC	Super-critical
C <sub>10</sub>	8	8	RE	0	0	PC	9	8	-	0	0	PC	Alternative-critical
C <sub>11</sub>	12	10	RE	7	1	RE	0	0	PC	0	0	PC	Alternative-critical
C <sub>12</sub>	9	9	SC	0	0	PC	10	10	SC	10	10	RE	Alternative-critical
C <sub>13</sub>	0	0	PC	3	3	RE	1	0	AC	13	13	RE	Alternative-critical
$R_1 = (c_2 \wedge c_3)$			SC			SC			SC			SC	Quasi-sequence-critical
$R_2 = (c_{5a} \vee c_{5b})$			PC			PC			PC			PC	Super-critical
$R_3 = (c_{8a} \vee c_{8b})$			AC			PC			PC			PC	Alternative-critical
$R_4 = (c_{9a} \vee c_{9b})$			PC			PC			AC			PC	Alternative-critical

**Note:** "-" denotes non-existent constraints  
 PC = project-critical                      SC = sequence-critical  
 AC = activity-critical                      RE = redundant

Four alternative schedules (named S1 to S4) are obtained, in which S2 is the best schedule with the shortest makespan of 24 days. The result from criticality analysis is presented in Table 6.5 with shaded columns representing the best schedule (S2), and all alternatives are graphically demonstrated in Figure 6.7.

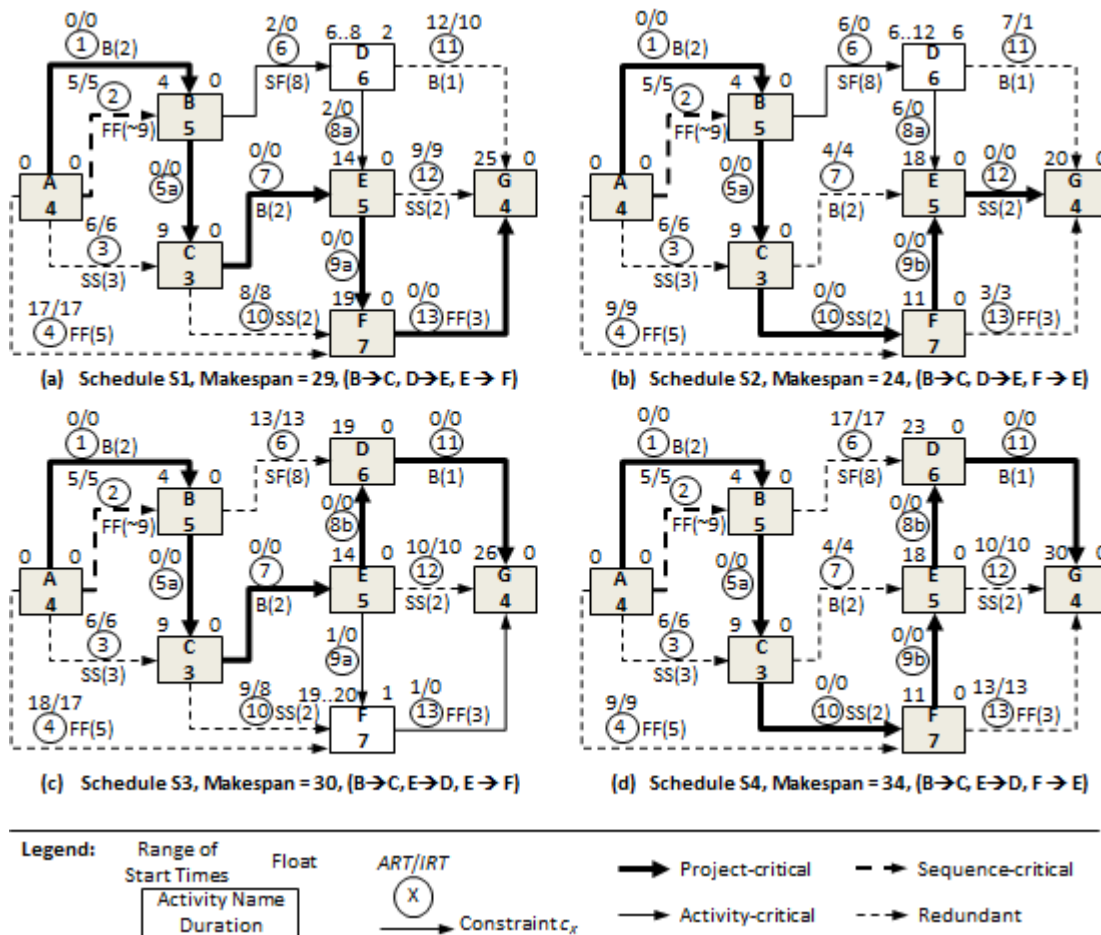


Figure 6.7. Alternative schedules demonstrating constraint criticality

The criticality analysis shows that constraints  $c_3$  and  $c_4$  are redundant in all alternative schedules. Constraints  $c_{5a}$  and  $c_{9b}$  are super-critical as they are project-critical in all alternative schedules in which they exist. Especially, constraint  $c_2$  is found to be sequence-critical in all alternatives. Removing it makes the originally infeasible schedules (in which constraint  $c_{5b}$  is active and allows activity C to be scheduled before activity B) become feasible, providing a shorter project makespan of 20 days (see Figure 6.8).

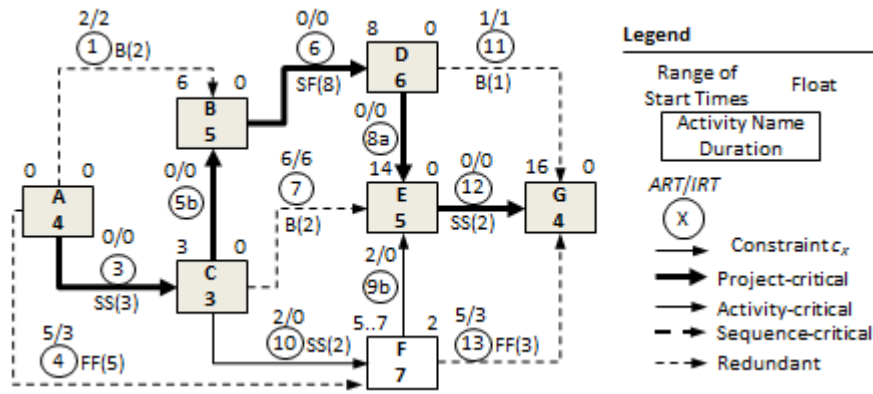


Figure 6.8. Best alternative schedule when constraint  $c_2$  is removed

The criticality of requirements is determined from that of their constituent simple constraints. Requirement R1 is identified as sequence-critical in all schedules and thus is quasi-sequence-critical. Requirement R2 is super-critical due to the project-criticality of constraint  $c_{5a}$  while the alternate constraint  $c_{5b}$  is not active in all alternative schedules. The rest two requirements R3 and R4 are alternative-critical, since they are not project-critical in all alternatives.

From the criticality analysis of construction requirements, some interesting observations can be made about the change of the preferable alternative schedule when variations happen. Firstly, the super-criticality of constraint  $c_{5a}$  does not allow for any plan flexibility when this constraint is violated. In other words, violating this constraint will increase the makespan of all alternative schedules accordingly and the construction sequence option defined in schedule S2 remains the most preferable.

Secondly, the preference of alternative schedule S2 may be impacted when a change happens to a constraint which is not super-critical. Two scenarios are presented for illustration. The first scenario demonstrates a change impacting all schedule makespans but schedule S2 remains as the most preferable. Consider for example a

change of  $\Delta T_6 = 7$  (from SF(8) to SF(15)) of constraint  $c_6$ :  $B$  SF(8)  $B$ . Since tightening is greater than the  $ART$  of  $c_6$  in schedules S1 and S2, the makespan of these schedules will be prolonged by  $\Delta T_6 - ART_6$ , and become 34 and 25 days respectively. Hence, schedule S2 remains as the best schedule in this case.

The second scenario demonstrates a change impacting the preference among the alternatives. Consider constraint  $c_{12}$ :  $E$  SS(2)  $G$  which is project-critical to S2 but not to other alternative schedules for example. If due to some site condition the lag time requirement of  $c_{12}$  is increased from  $m_{12} = 2$  to  $m_{12} = 8$ , constraint  $c_{12}$  is tightened by  $\Delta T_{12} = 6$  days, and thus the makespan of schedule S2 will be increased by  $\Delta T_p = 6$  days into 30 days accordingly. However, this tightening of  $c_{12}$  does not prolong the makespan of other alternative schedules since the tightening amount is less than the relaxation time of this constraint in the other alternatives ( $ART_{12}(S1) = 9$ ,  $ART_{12}(S3) = 10$ , and  $ART_{12}(S4) = 10$ ). In this scenario schedule S1 with a makespan of 29 days becomes the most preferable alternative.

## 6.7. Concluding Remarks

This chapter presents a criticality analysis approach for schedule constraints and construction requirements. A detailed criticality classification was developed to provide a better understanding of the role of constraints to a schedule. In particular, a constraint could be project-critical, activity-critical, sequence-critical or non-critical depending on how it could affect activities' start/finishes times and/or project duration. This classification forms the foundation to classify the behavior of criticality of a set of constraints under the effects of combinations of conjunction and disjunction. This also



allows the criticality of complex construction requirements under the context of alternative schedules to be classified.

Constraint criticality analysis is achieved through two new concepts, *Aggregate Relaxation Time (ART)* and *Intrinsic Relaxation Time (IRT)*. These relaxation times refer to the maximal temporal magnitude a constraint can be varied without affecting schedule makespan or activities' times. They are devised as criticality indicators and provide the basis for analyzing schedule changes from the requirement perspective. From these *ART* and *IRT*, the impact of changes can be determined.

Accordingly, this chapter presents a constraint-based method for schedule change analysis using constraint relaxation times. The analysis is based on and the nature of change and the impact of change on the *ART* of the associated constraint to identify the impact on schedule makespan. In particular, schedule delay could result from constraint tightening or introduction while schedule shortening can be achieved when relaxing or removing a project-critical constraint. By this, construction schedules can be analyzed and managed from a constraint perspective including changes from both activity's times and lag times. With the capability of handling a larger scope of changes from a broader perspective, the proposed concept allows project management to be raised from the process (as activity) level in traditional approaches to a higher level of construction knowledge.

## **CHAPTER 7. CASE STUDIES**

### **7.1. Introduction**

This chapter presents two case studies to demonstrate the application of the schedule generation approaches concepts and the criticality analysis concept from previous chapters. The first case study demonstrates the application of the ASCoRe framework and criticality concept for schedule generation and analysis. The second case study describes the application of the preemptive constraint analyzer to improve the feasibility and efficiency of CLP-based scheduling approach.

### **7.2. Case Study 1: Schedule Generation and Analysis of a Covered Walkway Project**

A simplified example based on the construction of the covering structure of a covered walkway project is presented to demonstrate the proposed scheduling frameworks and schedule analysis methodology described in previous chapters. The covering structure is divided into 3 sections for construction with Sections 1 and 3 spanning a length of 20m and a height of 2.5m, and Section 2 spanning 20m with a slope of 0.25 as shown in Figure 7.1. All footings are precast concrete while beams, columns, and roof structures are steel.

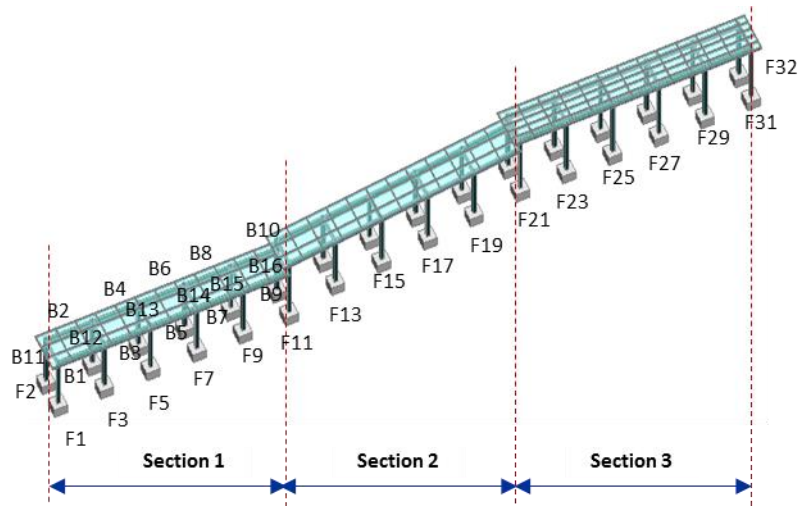


Figure 7.1. 3D perspectives of the covered walkway structure

### 7.2.1. Product Hierarchy and Component State Chain

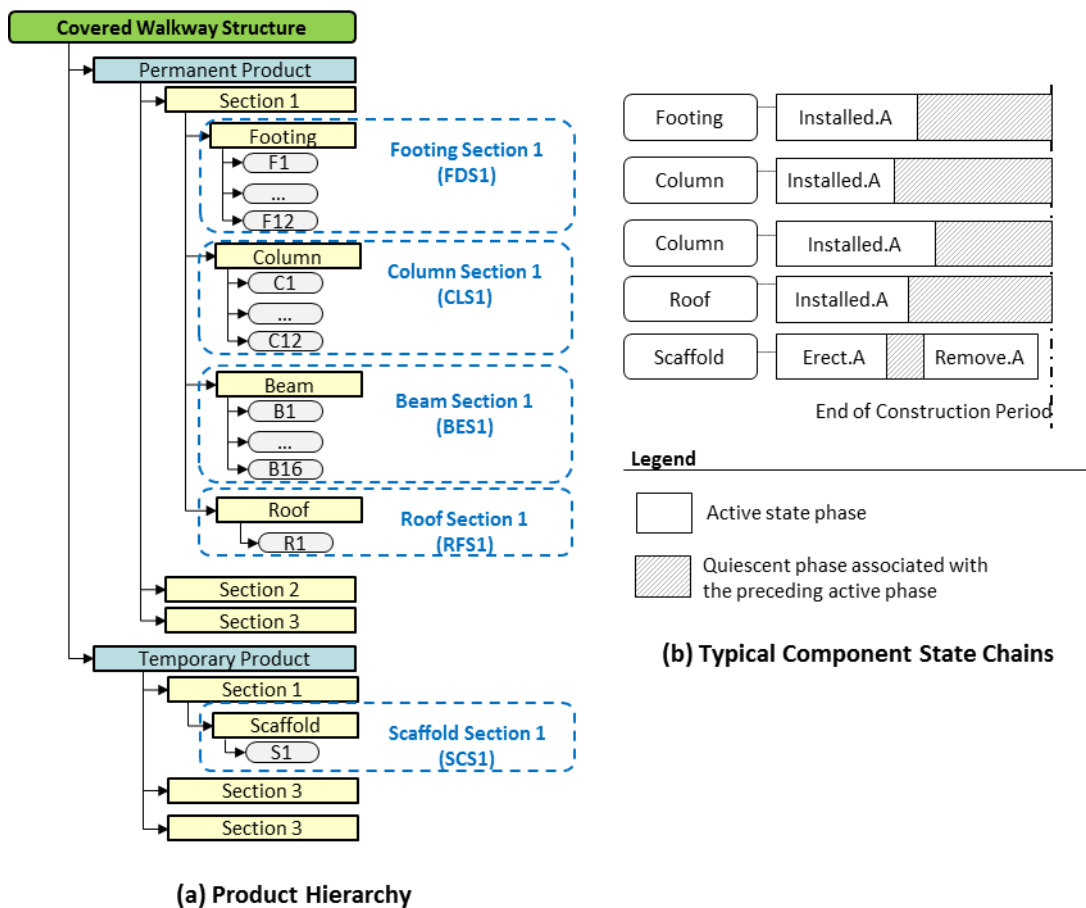


Figure 7.2. Product hierarchy and component state chain

The entire structure is decomposed into systems and subsystems as shown in Figure 7.2 (a). Components in the lowest subsystems are grouped into work packages. For instance, work package “Footing Section 1” includes 12 footing elements (F1 to F12). Similarly, work package “Column Section 1” involves 12 columns (C1 to C12); work package “Beam Section 1” contains 16 beams (B1 to B16), while work packages “Roof Section 1” and “Scaffold Section 1” consists of only one element, R1 and S1 respectively.

The state chain of typical component is depicted in Figure 7.2 (b). Since all permanent components are either steel or precast concrete, their construction state chain only consists of one state representing the installation process. On the other hand, the state chain of scaffold components consists of two states related to the erection and removal processes respectively.

With the defined work packages and component state chains, construction activities of the project are generated as depicted in Figure 7.3.

## **7.2.2. Construction Requirements and Constraint Network**

### **7.2.2.1. Functional Requirements**

Typical functional requirements applied to this project are defined at the section level as follows:

- F1: Footing structure support column structure  
`support([[Column.Installed.A]], [[Footing.Installed.Q]],E)`
- F2: Column structure support beam structure  
`support([[Column.Installed.Q]], [[Beam.Installed.Q]],E)`
- F3: Scaffold structure support the erection of beam and roof structures

```
support ([Beam.Installed.A], [Roof.Installed.A]),  
[[Scaffold.Erected.Q]], E)
```

### 7.2.2.2. *Non-functional Requirements*

Six major non-functional requirements have been identified for this project:

- R1. The foundation of sections 1 and 3 cannot be done concurrently due to routing condition, expressed as:  $R1:(C_1 \vee C_2)$ .
- R2. The foundation of section 2 must be carried out after either that of sections 1 or 3 due to site restriction, shown as:  $R2:(C_3 \vee C_4)$ .
- R3. The foundation of section 2 must be start before day 15, shown as:  $R3:C_{29}$
- R4. The column of section 2 must be installed after that of section 1, represented by one precedence constraint as:  $R4:C_8$
- R5. Due to a design requirement, there must be overlap time of at least 1 day between the beam installation of sections 1 and 2, and sections 2 and 3. This requirement requires two complex constraints, (BES1 *Overlaps*(1) BES2) and (BES2 *Overlaps*(1) BES3), and is represented by a conjunctive combination of four simple constraints as:  $R5:(C_{16} \wedge C_{17} \wedge C_{18} \wedge C_{19})$ .
- R6. Due to routing issues, the roof must be installed sequentially from either section 1 or 3, expressed as:  $R6:[(C_{22} \wedge C_{23}) \vee (C_{24} \wedge C_{25})]$ .

The identified construction requirements are reasoning into temporal constraints between activities and resulted schedule network of this project is depicted in Figure 7.3. The temporal constraints are indicated on the directed arcs.

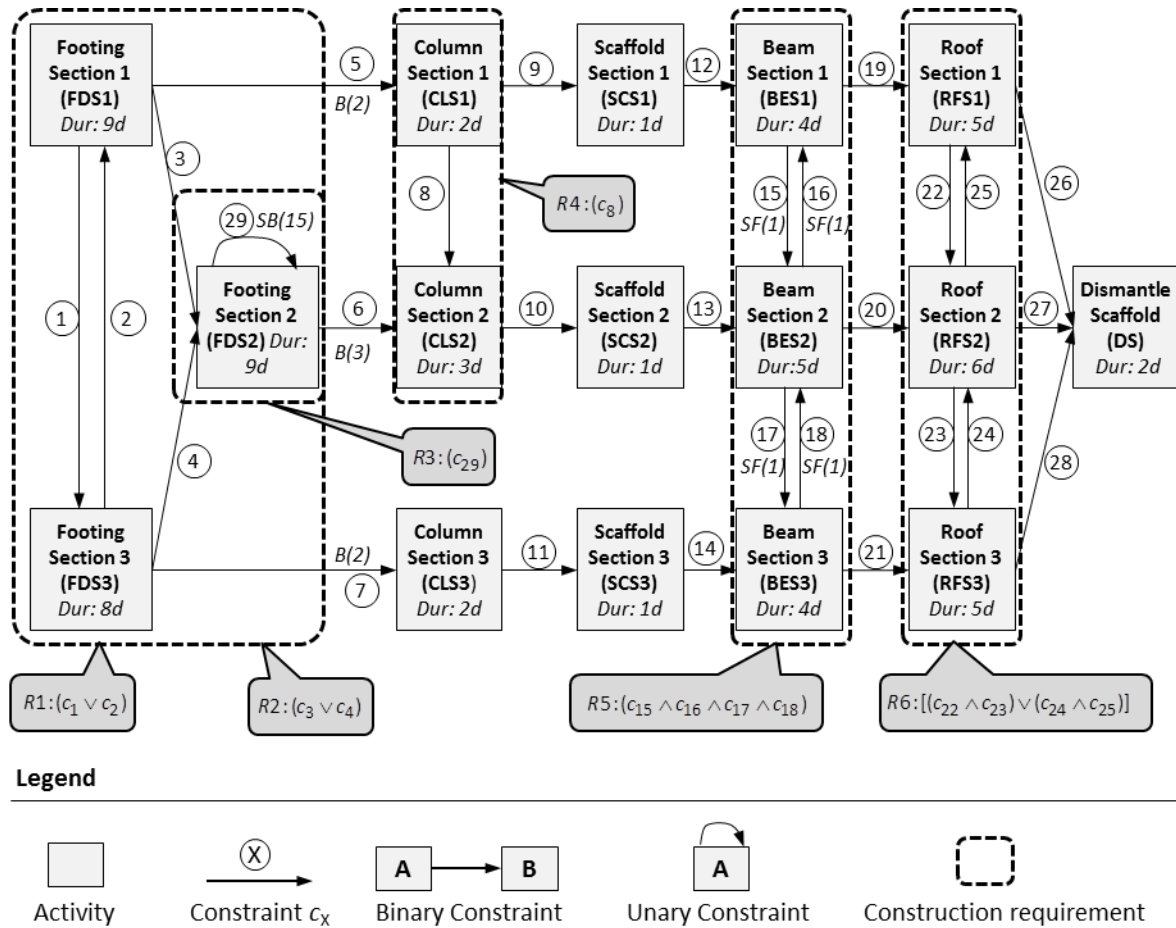


Figure 7.3. Schedule network of covered walkway project

### 7.2.3. Schedule Generation

By applying the proposed scheduling approach, four alternative schedules with a minimal makespan of 44 days (as summarized in Table 7.1) have been generated for this project. These schedules refer to different construction sequence options of footing and roof structures that the contractor can be implemented to achieve the best project completion time. In particular, in Schedule 1 and 2, the footing of Section 1 is done before Section 3, while in Schedule 3 and 4, the footing of Section 3 is done before Section 1. Similarly, in Schedule 1 and 3, the roof structure is installed sequentially from Section 1 to Section 3, while it is done from Section 3 to Section 1 in Schedule 2 and 4. In addition, it is found from four alternatives that besides the traditional

sequence (which is sequentially done from Section 1 to Section 3 for all structures), other sequences can also lead to the same best makespan. Therefore, applying alternative scheduling can provide contractor with more planning flexibility.

Table 7.1. Alternative schedules

Activity	Short Form	Duration	Schedule 1		Schedule 2		Schedule 3		Schedule 4	
			Start	Finish	Start	Finish	Start	Finish	Start	Finish
Footing Section 1	FDS1	9	0	9	0	9	8	17	8	17
Footing Section 2	FDS2	9	9	18	9	18	8..9	17..18	8..9	17..18
Footing Section 3	FDS3	8	9..16	17..24	9	18	0	8	0	8
Column Section 1	CLS1	2	11..19	13..21	11..19	13..21	19	21	19	21
Column Section 2	CLS2	3	21	24	21	24	21	24	21	24
Column Section 3	CLS3	2	19..26	21..28	19	21	10..26	12..28	10..19	12..21
Scaffold Section 1	SCS1	1	13..21	24..22	13..18	14..19	21	22	21..28	22..29
Scaffold Section 2	SCS2	1	24	25	23	24	24	25	24	25
Scaffold Section 3	SCS3	1	21..28	22..29	21	22	12..28	13..29	12..21	13..22
Beam Section 1	BES1	4	22	26	22..29	26..33	22	26	22..29	26..33
Beam Section 2	BES2	5	25	30	25	30	25	30	25	30
Beam Section 3	BES3	4	22..29	26..33	22	26	22..29	26..33	22	26
Roof Section 1	RFS1	5	26	31	37	42	26	31	37	42
Roof Section 2	RFS2	6	31	37	31	37	31	37	31	37
Roof Section 3	RFS3	5	37	42	26	31	37	42	26	31
Dismantle Scaffold	DS	2	42	44	42	44	42	44	42	44

#### 7.2.4. Criticality Analysis

The criticality analysis results of constraints and requirements with respect to individual and all alternative schedules are presented in Table 7.2, and demonstrated in Figure 7.4.

Table 7.2. Criticality of simple constraints in four alternative schedules

Constraint/ Requirement	S1			S2			S3			S4			Overall
	ART	IRT	Type	ART	IRT	Type	ART	IRT	Type	ART	IRT	Type	
C1	7	0	AC	0	0	PC	-	-	-	-	-	-	Alternative-critical
C2	-	-	-	-	-	-	0	0	PC	0	0	PC	Supper-critical
C3	0	0	PC	0	0	PC	-	-	-	-	-	-	Supper-critical
C4	-	-	-	-	-	-	1	0	AC	1	0	AC	Quasi-activity-critical
C5	8	0	AC	8	0	AC	0	0	PC	0	0	PC	Alternative-critical
C6	0	0	PC	0	0	PC	1	0	AC	1	0	AC	Alternative-critical
C7	7	0	AC	0	0	PC	16	0	AC	9	0	AC	Alternative-critical
C8	8	0	AC	8	0	AC	0	0	PC	0	0	PC	Alternative-critical
C9	8	0	AC	15	0	AC	0	0	PC	7	0	AC	Alternative-critical
C10	0	0	PC	0	0	PC	0	0	PC	0	0	PC	Supper-critical
C11	7	0	AC	0	0	PC	16	0	AC	9	0	AC	Alternative-critical
C12	8	0	AC	7	0	AC	0	0	PC	7	0	AC	Alternative-critical
C13	0	0	PC	0	0	PC	0	0	PC	0	0	PC	Supper-critical
C14	7	0	AC	0	0	PC	16	0	AC	9	0	AC	Alternative-critical
C15	7	7	RE	7	0	AC	7	7	RE	7	0	AC	Quasi-critical
C16	0	0	PC	7	0	AC	0	0	PC	7	0	AC	Alternative-critical
C17	7	0	AC	0	0	PC	7	0	AC	0	0	PC	Alternative-critical
C18	7	0	AC	7	7	RE	8	0	AC	7	7	RE	Quasi-critical
C19	0	0	PC	11	7	SC	0	0	PC	11	4	SC	Alternative-critical
C20	1	1	RE	1	1	RE	1	1	RE	1	1	RE	Redundant
C21	11	4	SC	0	0	PC	4	4	SC	0	0	PC	Alternative-critical
C22	0	0	PC	-	-	-	0	0	PC	-	-	-	Supper-critical
C23	0	0	PC	-	-	-	0	0	PC	-	-	-	Supper-critical
C24	-	-	-	0	0	PC	-	-	-	0	0	PC	Supper-critical
C25	-	-	-	0	0	PC	-	-	-	0	0	PC	Supper-critical
C26	11	11	SC	0	0	PC	11	11	SC	0	0	PC	Alternative-critical
C27	5	5	RE	5	5	RE	5	5	RE	5	5	RE	Redundant
C28	0	0	PC	11	11	SC	0	0	PC	11	11	SC	Alternative-critical
C29	6	6	RE	7	6	RE	7	6	RE	7	6	RE	Redundant
R1:(c <sub>1</sub> ∨ c <sub>2</sub> )			AC			PC			PC			PC	Alternative-critical
R2:(c <sub>3</sub> ∨ c <sub>4</sub> )			PC			PC			AC			AC	Alternative-critical
R3:(c <sub>29</sub> )			RE			RE			RE			RE	Redundant
R4:(c <sub>8</sub> )			AC			AC			PC			PC	Alternative-critical
R5:(c <sub>15</sub> ∧ c <sub>16</sub> ∧ c <sub>17</sub> ∧ c <sub>18</sub> )			PC			PC			PC			PC	Supper-critical
R6:[(c <sub>22</sub> ∧ c <sub>23</sub> ) ∨ (c <sub>24</sub> ∧ c <sub>25</sub> )]			PC			PC			PC			PC	Supper-critical

**Note:** "-" denotes non-existent constraints

PC = project-critical

SC = sequence-critical

AC = activity-critical

RE = redundant



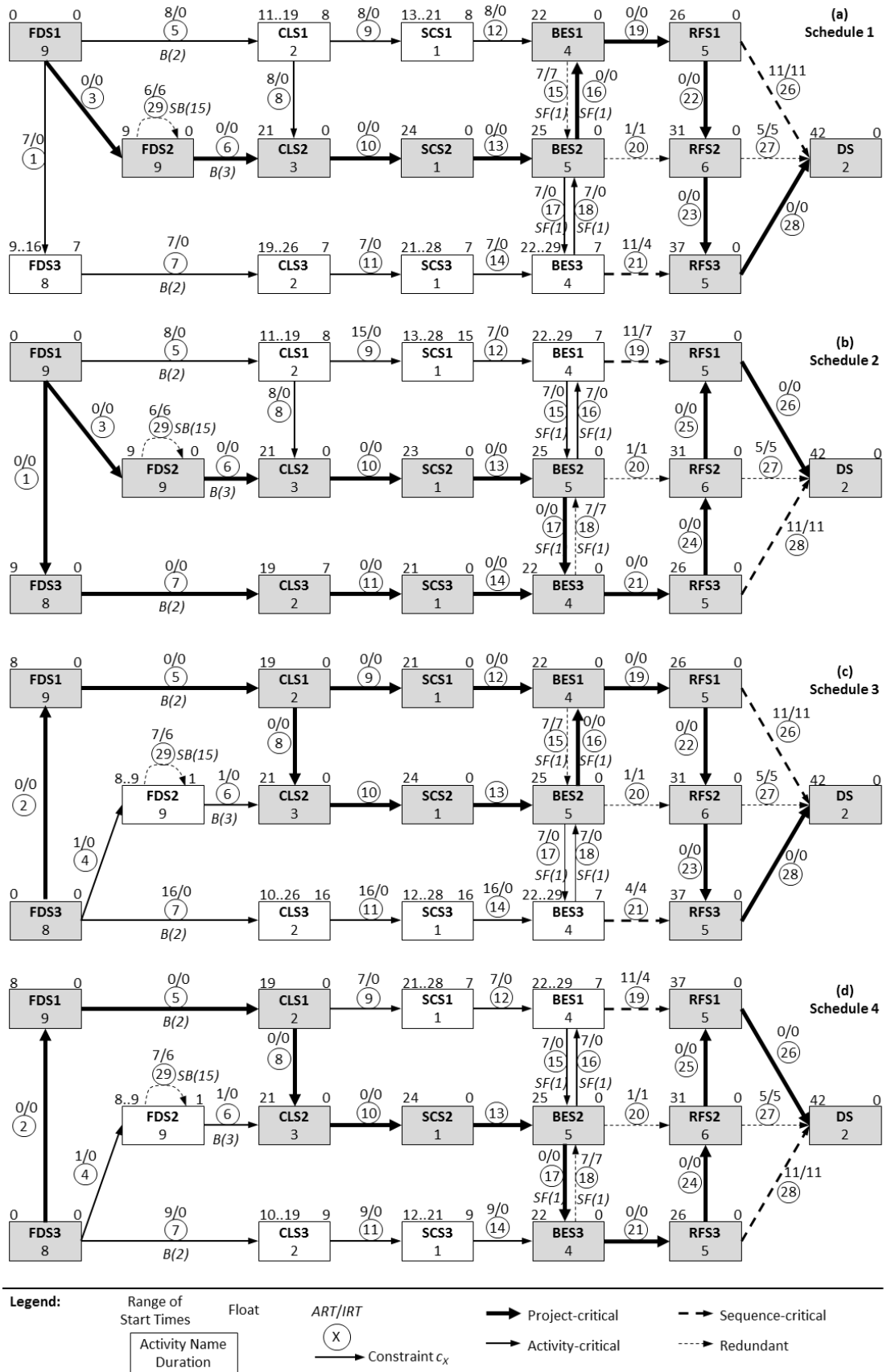


Figure 7.4. Alternative schedules indicating critical constraints

### 7.2.4.1. Criticality Analysis of Single Constraints

The use *ART* and *IRT* allows for a more effective method for identifying constraint criticality in all alternative schedules. These criticality indicators also provide useful information for planners to understand the role of constraints for all alternative schedules. Consider Schedule 1 (Figure 7.4a) for illustration. Firstly, constraints  $c_3$ ,  $c_6$ ,  $c_{10}$ ,  $c_{16}$ ,  $c_{19}$ ,  $c_{22}$ ,  $c_{23}$  and  $c_{28}$  are project-critical; hence tightening them will cause schedule delay. If for instance constraint  $c_6$  is tightened by  $\Delta T_6 = 1$  from  $B(3)$  to  $B(4)$ , the makespan of this schedule will be increased by 1 day to 45 days.

Constraints  $c_1$ ,  $c_5$ ,  $c_7$ ,  $c_8$ ,  $c_{11}$ ,  $c_{17}$ , and  $c_{18}$  are activity-critical, and tightening them will reduce the feasible time ranges of the associated activities but not the schedule makespan. If for example, due to some site condition the columns at section 1 must be installed at least 3 days after the foundation is completed, constraint  $c_5$  is tightened by 1 day (from  $B(2)$  to  $B(3)$ ), decreasing its *ART* from  $ART_5 = 8$  to  $ART_5 = 7$ , and consequently reducing the start time range of Column Installation Section 1 to [12..19], yet not impacting the schedule makespan. The change in activity start time of Column Section 1 (CLS1) will be propagated to downstream network, and the *ART*s of the related constraints  $c_9$  and  $c_{12}$  are decreased by a similar amount ( $\Delta T = 1$ ).

Constraints  $c_{21}$  and  $c_{26}$  are identified as sequence-critical in Schedule 1, indicating that a better project makespan can be obtained by removing these constraints. Although sequence-critical constraints are intuitively “redundant” to this alternative schedule, identifying them provide planners with useful strategies on sequence selection when these constraints can be eliminated. If the roof structure can be redesigned so that its erection does not require a scaffolding structure, constraint  $c_{26}$  can be removed. Under such a scenario, the makespan of Schedule 1 remains

unchanged, yet that of Schedules 2 and 4 is improved from 44 days to 42 days. With this change, planners should proceed with either Schedule 2 or Schedule 4 for a better project completion time.

Finally, constraints  $c_6$ ,  $c_{15}$ ,  $c_{20}$  and  $c_{27}$  are found to be redundant in Schedule 1. Tightening, relaxing or removing them will neither change activities' feasible start times nor impact the schedule makespan. Although both sequence-critical and redundant constraints have non-zero *IRT*s, the key difference between them is the impact of their existence to the overall project makepan. For instance, if constraint  $c_{27}$  is deleted, the makespan of all four schedules still remains at 44 days, while removing constraint  $c_{21}$  (a sequence-critical constraint) will allow the makespan of Schedules 2 and 4 to be reduced to 42 days.

#### ***7.2.4.2. Criticality Analysis from the Perspective of Construction Requirements***

The criticality of constraints lays the foundation to classify the criticality of construction requirements. For the simplest requirements R3 and R4 comprising only one constraint, their criticality in each schedule is similar to that of the constituent constraint. The criticality of a disjunction construction requirement in each schedule is defined by its active disjunct. For instance, R1 is identified as activity-critical in Schedule 1 due to constraint  $c_1$  while it is project-criticality in Schedules 2, 3, and 4 due to  $c_2$ . Similarly, requirement R2 is project-critical in Schedules 1 and 2 based on  $c_3$  while activity-critical in Schedules 3 and 4 on  $c_4$ . The criticality of a conjunction requirement is defined by all constituent constraints due to their co-existence in every schedule. As such, requirement R5 is identified as project-critical in all schedules, due to constraint  $c_{16}$  in Schedules 1 and 3, and  $c_{17}$  in Schedules 2 and 4.

For the complex requirement R6 with each disjunct comprising a conjunctive combination of constraints, its criticality in each alternative schedule is defined by that of the active disjunct, which in turn is specified by the criticality of all constraints constituting the disjunct. In detail, the project-criticality of R6 in Schedules 1 and 3 is defined by the conjunctive combination of  $c_{22}$  and  $c_{23}$ , while in Schedules 2 and 4 by  $c_{24}$  and  $c_{25}$ .

From the perspective of multiple schedules, requirements R5 and R6 are super-critical in all alternative schedules, implying that they dictate the project makespan. Changes in these requirements will affect all alternative schedules. Therefore, they should receive the highest management priority. Requirements R1, R2, and R3 are identified as alternative-critical, allowing planners to anticipate for switching among alternative schedules to mitigate their impact to schedule makespan when changes happen despite the super-criticality of some activities. For example, in this case example, the “Foundation Section 1” is critical under the consideration of all alternative schedules, and a Planner may choose to proceed with Schedule 2 with “Foundation Section 1” starting on the first day of the project. However, if this activity is anticipated not be delayed and not carried out on Day 1, then alternative Schedule 3 or 4 may be chosen, with “Foundation Section 3” commencing first, and “Foundation Section 1” can be carried out on Day 8, thus not delaying the project. Finally, requirement R4 is redundant in all schedules, and hence the analysis of this requirement may not be necessary if changes are within the relaxation times of the constituent constraint.

In addition, constraint criticality would provide a new perspective for evaluating alternative schedules. Evaluating alternative schedules is commonly based on some

robustness indicators which are often functions of total free floats (Ghezail et al., 2010); these slack-based criteria may however not be representative if a construction requirement is the major consideration. The requirement perspective may present deeper insight into the choice of alternative. For example, if the routing condition (defined in Requirement R1) is the major consideration, then Schedule 1, in which R1 is not project-critical, the most is preferable, and should be chosen for execution. In contrast, with the total free float of the four alternative schedules respectively as 8, 8, 9, and 9, Schedules 3 and 4 in which R1 is project-critical are more preferable than Schedules 1 and 2.

### **7.2.5. Analyzing Schedule Changes**

From the criticality analysis of construction requirements, we can draw some interesting conclusions for schedule change management. Two scenarios are presented for illustration. The first scenario demonstrates schedule change resulting from constraint tightening while the second examines the impact of removing a construction requirement upon schedule makespan.

Firstly, if the Foundation Section 2 is prolonged to from 9 to 11 days ( $\Delta T_6 = 2$ ), constraint  $c_6$  is violated in all alternative schedules but will cause different impacts to the schedule makespan due to its different  $ART$ s. Specifically, the makespan of Schedules 1 and 2 with  $ART_6 = 0$  will be prolonged by  $\Delta T_P = 2$  days, while that of Schedules 3 and 4 with  $ART_6 = 1$  will be increased by only 1 day.

Secondly, if the site condition can be modified so that the foundation at section 2 can be carried out when the project starts, requirement R2 comprising both constraints  $c_3$  and  $c_4$  can be deleted. Since Schedule 1 only has one critical path, deleting the

project-critical constraint  $c_3$  will shorten the makespan of this schedule. However, removing this constraint does not improve the makespan of Schedule 2 since this schedule has two critical paths. According to section 6.5.1.2, the shortening time Schedule 1 is governed by the following  $ART$ s:

- $ART_{(I)} = [ART_{FDS2,s}]$  with  $c_{FDS2,s}$ : FDS2 SA(0)
- $ART_{(II)} = [ART_1]$ , and
- $ART_{(III)} = [ART_5, ART_7, ART_8, ART_9, ART_{11}, ART_{12}, ART_{14}, ART_{18}, ART_{21}]$

The shortening time is determined according to Equation (4.45) as:

$$\Delta T_p = \min(ART_{(I)}, ART_{(II)}, ART_{(III)}) = \min(9, 7, 8, 7, 8, 8, 7, 8, 7, 7, 11) = 7$$

With the impact of this makespan shortening, the  $ART$  of all constraints  $\Omega_i$  is correspondingly decreased by  $\Delta T_p = 7$ . Consequently, constraints  $c_1$ ,  $c_7$ ,  $c_{11}$ ,  $c_{14}$ , and  $c_{18}$  with zero updated  $ART$ s become project-critical. In addition, since both  $c_{14}$  and  $c_{18}$  become project-critical, activity BES3 also becomes critical with its updated start time ranges as  $BES3^- = [22]$ , and consequently constraint  $c_{17}$  becoming project-critical.

In summary, the case project has demonstrated the application of the proposed criticality concept to analyzing and managing construction schedules with the existence of multiple alternative schedules. The proposed criticality concept allows Planners to determine the super-critical construction requirements which always dictate the project makespan. The identification of alternative-critical requirements indicates some plan flexibility enables Planners to anticipate switching between alternative schedules when some changes happen. Furthermore, the relaxation times  $ART/IRT$  quantitatively represent the criticality of the constraints and provide the

fundamentals for an innovative approach to schedule change analysis, which is carried out from the construction requirement perspective.

### 7.3. Case Study 2: Application of the Preemptive Constraint Analysis Framework to a Pipeline Installation Project

An illustrative case example based on a simplified gas pipeline installation project is presented to demonstrate the application of the preemptive constraint analyzer in alternative scheduling. The piping structure stretches over 300 meters and is divided into 5 sections for construction as shown in . Sections 1 and 5 represent the construction of two concrete pipe bridges crossing existing water channels with their associated foundations followed by pipeline installation phases. Sections 2 to 4 refer to the main pipeline which is installed on steel pipe racks on shallow foundations.

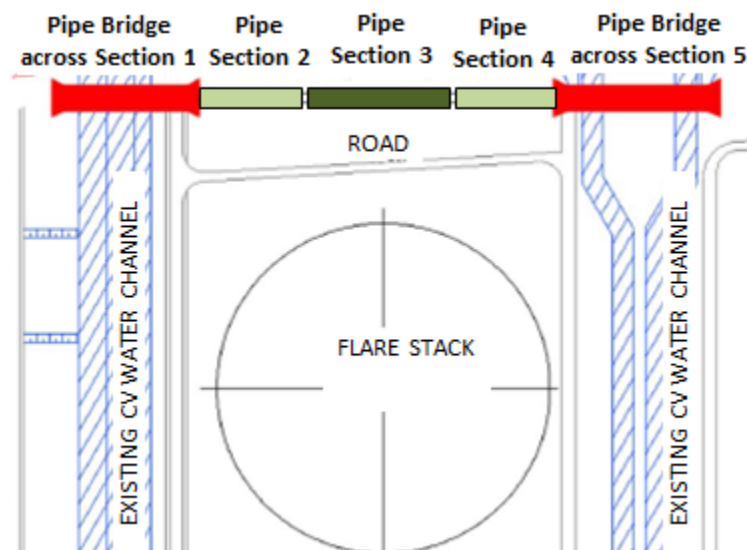


Figure 7.5. Pipeline installation layout

#### 7.3.1. Construction Requirements

Major construction requirements have been considered for this project, described as follows:

- R1. The bridge foundation work of sections 1 and 5 shares one common micropiler. Consequently, they must happen disjunctively.
- R2. There is only one crew working on the foundation work of sections 2 to 4.
- R3. The construction of shallow foundations must be sequentially started with a minimal lag time requirement of 3 day, and sequentially finished with a maximal lag time requirement of 1 day due to design requirement.
- R4. There is only one crew working on the pipe installation of sections 1 to 5.
- R5. The pipeline installation work at section 1 must be finished before that at other sections can start.
- R6. Special technical constraints require that the pipeline installation must be continuous from sections 2 to 3 and from sections 4 to 5.
- R7. Pipe installation of sections 4 and 5 have to be finished at least 2 days after the completion of sections 2 and 3 respectively.

The identified construction requirements are converted into temporal constraints as shown in Figure 7.6. Since the major focus of this case example is the constraint preemptive analysis, the conversion from requirements into temporal constraints is not presented in this section. Temporal constraints are indicated on the directed arcs. Directed arcs without any indications are assumed to depict the  $B(0)$  constraint. The *All-Disjoint* constraint is used to model key resource requirement where only one key machine or crew available for the activities. It includes a set of disjunctive constraints (*Disjoint*) between every pair of activities sharing the resource, for example three constraints: PF2 *Disjoint* PF3, PF2 *Disjoint* PF4, and PF3 *Disjoint* PF4 representing Requirement R2. Constituting temporal constraints of the imposed requirements are summarized in Table 7.3.





Together the 3 requirements R1, R2, and R4 result in 14 disjunctive constraints (*Disjoint*), resulting in  $2^{14}$  backtrackings, which would not be possible to examine in total. In this case, planners often need to employ some priority rules in sequencing the related activities possibly leading to many infeasible solutions. The remaining constraints involve 31 simple constraints and 2 conjunctive constraints as a result of the *Meets* requirement in R6.

### 7.3.2. Preemptive Constraint Analysis and Schedule Generation

By applying the proposed constraints analyzer, 10 of the 14 disjunctive constraints (*Disjoint*) comprising conflicting constraints are identified as shown in . Column 2 of the Table indicates the constituent simple constraints of the disjunctive in column 1 while column 3 presents the conflicting constraints corresponding to the constraints in column 2. The identified constraint redundancies and inconsistencies are grouped into two groups: primary and secondary according to the classification presented in chapter five. In detail, primary conflicts/redundancies are those dependent only on lag times and independent of activity durations. In contrast, secondary conflicts/redundancies depend on both lag times and activity durations. Accordingly primary constraint redundancies are invariant with activity durations, while the secondary ones may no longer exist under some conditions of activity duration. Primary conflicts are independent of activity durations and thus, can only be resolved when the lag times are modified or either of the constraints is removed, while secondary conflict can be resolved by changing activity durations.

Of these 10 are primary conflicting pairs which can be removed because of the existence of the other disjunct in the disjunctive constraint, for example, PI4 B(0) PI2

is ignored while  $PI_2 B(0) PI_4$  remains so that the conflict with  $PI_2 FF(2) PI_4$  can be resolved. As a result, the number of branches is dramatically reduced from  $2^{14}$  to  $2^4$ . This huge reduction makes it possible to analyze all 16 remaining sequencing options to obtain the globally optimal solutions.

Table 7.4. Conflicting constraints

Disjunctive Constraints	Disjunct/ Simple Constraint	Conflicting Constraints	Conflict Type
$PF_2$ Disjoint $PF_3$	$PF_2 B(0) PF_3$	$PF_2 FF(\sim 1) PF_3$	SC
	$PF_3 B(0) PF_2$	$PF_2 SS(3) PF_3$	PC
$PF_3$ Disjoint $PF_4$	$PF_3 B(0) PF_4$	$PF_3 FF(\sim 1) PF_4$	SC
	$PF_4 B(0) PF_3$	$PF_3 SS(3) PF_4$	PC
$PI_1$ Disjoint $PI_2$	$PI_1 B(0) PI_2$		
	$PI_2 B(0) PI_1$	$PI_1 B(0) PI_2$	PC
$PI_1$ Disjoint $PI_3$	$PI_1 B(0) PI_3$		
	$PI_3 B(0) PI_1$	$PI_1 B(0) PI_3$	PC
$PI_1$ Disjoint $PI_4$	$PI_1 B(0) PI_4$		
	$PI_4 B(0) PI_1$	$PI_1 B(0) PI_4$	PC
$PI_1$ Disjoint $PI_5$	$PI_1 B(0) PI_5$		
	$PI_5 B(0) PI_1$	$PI_1 B(0) PI_5$	PC
$PI_2$ Disjoint $PI_3$	$PI_2 B(0) PI_3$		
	$PI_3 B(0) PI_2$	$PI_2 Meets PI_3$	PC
$PI_2$ Disjoint $PI_4$	$PI_2 B(0) PI_4$		
	$PI_4 B(0) PI_2$	$PI_2 FF(2) PI_4$	PC
$PI_3$ Disjoint $PI_5$	$PI_3 B(0) PI_5$		
	$PI_5 B(0) PI_3$	$PI_3 FF(2) PI_5$	PC
$PI_4$ Disjoint $PI_5$	$PI_4 B(0) PI_5$		
	$PI_5 B(0) PI_4$	$PI_4 Meets PI_5$	PC
	$PF_2 SS(3) PF_3$	$PF_2 FF(\sim 1) PF_3$	SC
	$PF_3 SS(3) PF_4$	$PF_3 FF(\sim 1) PF_4$	SC

Note: PC = Primary Conflict; SC = Secondary Conflict

The remaining four conflicts are between two pairs of activities ( $PF_2, PF_3$ ) and ( $PF_3, PF_4$ ). Since they are secondary conflicts they can be resolved with another combination of activity durations, and the feasible duration ranges will provide some

useful references if such a resolution strategy can be applied. However, the feasible duration ranges of these activities corresponding to the secondary conflicts are determined as  $FD_{PF2} = [0, +\infty)$ ,  $FD_{PF3} = [0, 1]$ , and  $FD_{PF4} = [0, 1]$ . Changing activity durations of PF3 and PF3 may not be applicable since it is generally impossible to finish pipe installation in 1 day. Instead the pipe design has to be revised so that the requirements on the finish times of installation work (PF2  $FF(\sim 1)$  PF3 and PF3  $FF(\sim 1)$  PF4) can be removed, simultaneously resolving all four conflicts.

In addition, four constraints are found to be secondary redundant as shown in , and can be excluded from the scheduling process.

Table 7.5. Redundant constraints

Redundant Constraint	Subsuming Constraint	Redundancy Type
PF2 $SS(3)$ PF3	PF2 $B(0)$ PF3	Secondary
PF3 $SS(3)$ PF4	PF3 $B(0)$ PF4	Secondary
PI2 $FF(2)$ PI4	PI2 $B(0)$ PI4	Secondary
PI3 $FF(2)$ PI5	PI3 $B(0)$ PI5	Secondary

With the refined constraint set the scheduler generates 2 best alternative solutions with a project makespan of 79 days as presented in . The two alternative schedules refer to two ways of sequencing the bridge foundation work of sections 1 and 5 (BF1 and BF5), arising from the resource requirement constraint defined in R1. In Schedule 1, BF1 precedes BF5, while in Schedule 2 it succeeds BF5. From a management perspective, the construction of the bridge foundations has sequencing flexibility without affecting project duration. However, in both schedules, PF2, PF3 and PF4 are carried out sequentially due to resource and the modified design

constraints defined in R2. Similarly, the pipe installation must be sequentially done from sections 1 to 5 to fulfill technical and resource constraints defined in R4 in either alternative schedules.

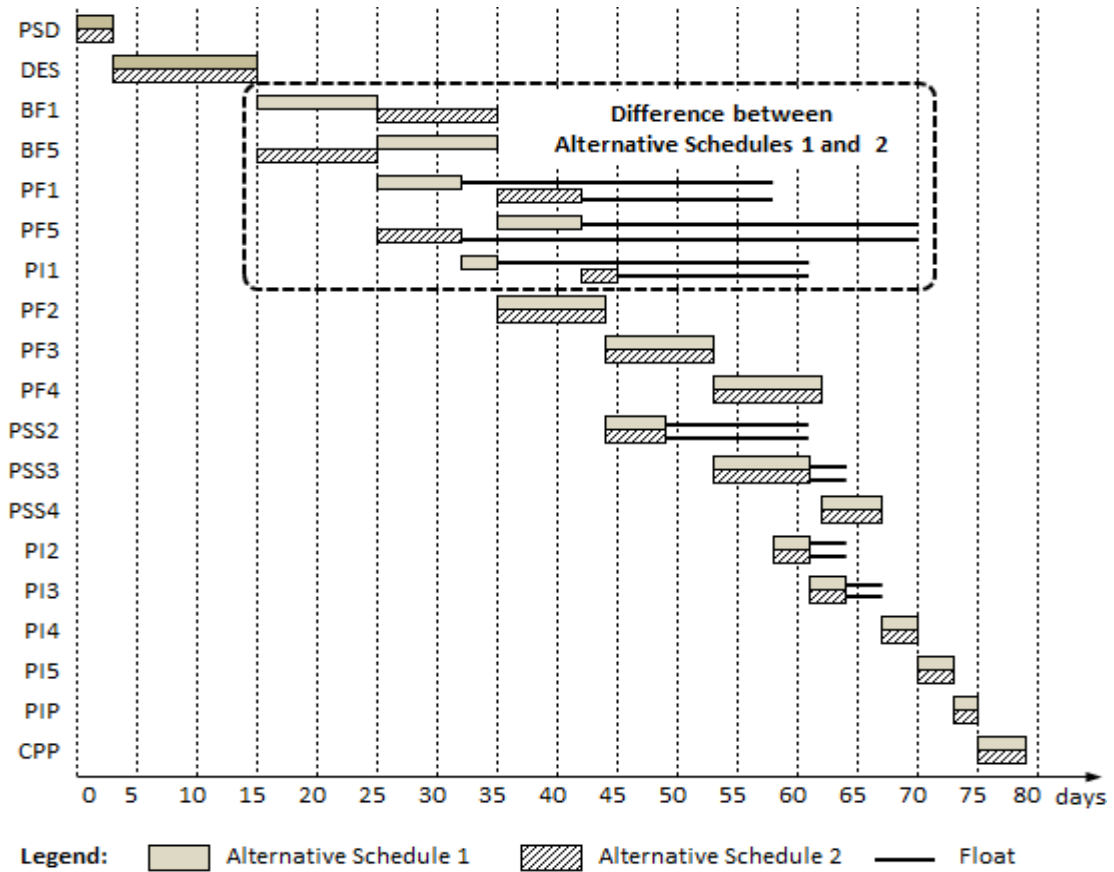


Figure 7.7. Alternative schedules

In summary, this example demonstrates how the proposed preemptive constraint analyzer can be applied to improve the efficiency of construction scheduling when multiple construction sequences are available. The framework is built on a set of comparison rules of constraints pairs, and is performed in the pre-scheduling stage without requiring complete constraint propagation and backtracking. The identified constraint redundancies and conflicts are thus independent of the constraint ordering. Moreover, the classification of constraint redundancy/inconsistency and the feasible

duration ranges also provide useful strategies for resolving the conflicts. By this, planners can have a deeper insight into the nature of the redundancies/inconsistencies so that more appropriate resolution approaches can be carried out. Beyond that, the framework helps eliminate unnecessary 10 unnecessary disjunctive constraints, and thus the search space can be dramatically reduced from  $2^{14}$  to  $2^4$  backtrackings. This reduction allows the application of a complete search technique to determine all feasible schedule solutions.

#### **7.4. Concluding Remarks**

In this chapter, three case studies are presented to demonstrate how the scheduling generation and analysis can be carried out from the perspective of construction requirements through the application of the proposed methodologies. The first case study is used to illustrate the application of the ASCoRe scheduling framework and the FReMAS model into automated construction sequencing and scheduling from construction requirements. This case study serves as a validation that FReMAS is capable of capturing multiple engineering solutions through the provider co-functionality attribute so that all alternative schedules can be obtained. Especially, the use of provider co-functionality provider attribute allows planners to examine different collaboration scenarios to improve project time.

In the second case study, the application of the proposed preemptive constraint analysis framework described in chapter five is illustrated. The case study demonstrates the capability and usefulness of the proposed framework in identifying and removing redundant and conflicting constraints in the pre-scheduling stage. In particular, applying this framework helps remove 10 out of 14 disjunctive constraints

and thus reduces the search space from  $2^{14}$  to  $2^4$  backtrackings. The framework also classifies constraint redundancy/inconsistency into primary and secondary classes so that planners can have a better understanding of the role of these relationships to the schedules. In addition, the identified feasible duration ranges provide useful strategies for resolving the secondary conflicts.

The final case study presents the application of the proposed concept and methodology for analyzing the criticality of construction requirements and their application to schedule change analysis. By considering multiple alternative schedules, the proposed method provides a deeper insight to the role of constraints and requirements for not only an individual schedule but for the entire project. It is also highlighted in this case study that despite the super-criticality of some activities, identifying alternative-critical requirements helps planner determine implicit sequencing flexibility which can be exploited to mitigate some anticipated variations. Especially, the criticality of constraint can be quantitatively represented using two new criticality indicators *ART/IRT*. These criticality indicators also allow the schedule to be analyzed and managed from a construction requirement perspective which encompasses both constraint and activity changes.

## **CHAPTER 8. CONCLUSION AND RECOMMENDATIONS**

### **8.1. Introduction**

The main purpose of this research is to develop the necessary methodologies and concepts for automated schedule generation and analysis from the perspective of construction requirements to improve the efficiency and feasibility of construction schedules. For this purpose, this dissertation proposes an overarching framework to integrate, interpret, and analyze construction requirements for schedule auto-generation, criticality analysis and change management.

The outline of the overarching framework follows the structure of this dissertation. It includes a generalized framework for automated scheduling from construction requirements (ASCoRe) which provides the core modeling tools for formalizing construction methods and requirements and the main scheduling processes for automatically generating alternative schedules from construction requirements. A generalized functional requirement model (FReMAS) is then developed to formalize and convert complex functional requirements into temporal constraints. To improve the efficiency and feasibility of scheduling from complex requirement, the framework utilizes a preemptive constraint analysis framework which allows basic constraints redundancies/inconsistencies to be identified and removed in the pre-scheduling stage. Finally, the framework proposes a new perspective for schedule analysis which is based on the criticality of constraints and construction requirements. Three industrial case projects are used to demonstrate key features of the overarching framework and verify the research findings.



This chapter summarizes the significant research results, discusses the key contributions of the research, identifies the main limitations and finally recommends directions for future studies.

## **8.2. Conclusion and Research Contributions**

### **8.2.1. Generalized Framework for Automated Scheduling from Construction**

#### **Requirement (ASCoRe)**

The ASCoRe approach proposed in this dissertation addresses the current limitations of incorporating construction methods and requirements into automated scheduling, which have been discussed earlier in section 1.2.1. In particular, ASCoRe develops four core knowledge models: Product, Construction Method, Construction Requirement and Schedule to describe the immutable core characteristics of building data and construction knowledge necessary for scheduling. In particular, construction requirement model allows construction requirements to be explicitly captured and managed in their original existence form from both product and process perspectives and at both component state and activity levels. Such a clear elicitation allows construction requirements to be passed on through the project phases, enhancing the traceability of changes for better schedule management.

Another contribution of the ASCoRe approach is its generalized framework for automated BIM-based scheduling which comprises four main procedures: (P) to generate an extended product hierarchy, (R) to identify construction requirements, (S) to create a schedule model by generating activities and temporal constraints, and finally (A) to compute for alternative schedules. One key advantageous feature of this framework is the usage of component states as the key construct to integrate

construction method, product and activity perspectives. The product-process attribute of component states allows the direct generation of activity from construction method which enables the consideration of multiple construction methods. This allows changes in design and construction methods to be steadily updated to schedule.

A system architectural framework with sequence reasoning and scheduling algorithms for implementing ASCoRe is then proposed as part of this dissertation. The key extensions of this scheduling system from the existing model-based schedulers include the Construction Knowledge Modeling Module together with different knowledge templates to formalize necessary knowledge and data for scheduling, the Inference and Sequence Reasoning Kernel to automatically derive activities and temporal constraints from construction methods and requirements, and the Schedule Generation Engine to generate all alternative schedules. In essence, the knowledge templates facilitate the development of knowledge libraries, and further accelerate the scheduling. Furthermore, the developed sequence reasoning mechanisms allows construction requirements defined at different levels to be automatically converted into temporal constraints for scheduling at activity level. Such built-in generic sequence reasoning knowledge also enables the system to be applicable to different project types and from both product and process perspectives.

### **8.2.2. Functional Requirement Model for Automated Sequencing (FReMAS)**

In order to overcome the research limitations on construction sequencing from functional requirements described in section 1.2.2, this research developed a generalized framework called FReMAS to capture functional requirements and convert them into temporal constraints. FReMAS overcomes the limitations through the

extension of the Intermediate Function Concept (Song, 2006) via the use the provider co-functionality and meta-provider constructs to capture both intermediate and final functional requirements with multiple users and providers as well as different provider combinations. This not only allows for greater generality expression from the model to capture complex functional requirements but also allows the complete identification of all possible construction sequences for the fulfilment of the requirements. In particular, three key advantageous features of FReMAS are summarized as follows.

Firstly, FReMAS provides a generalized format with four modeling elements: function user, function provider, function type, and provider co-functionality to determine any complex functional requirement with multiple users and providers. The key advantage of this representation format lies in the use of the provider co-functionality construct to represent different functionality relationships between providers. As illustrated in the case study in section 7.2, this construct allows FReMAS to explicitly capture multiple engineering solutions for the requirement which are often resulting from different collaboration scenarios among project parties.

Secondly, FReMAS provides a hierarchical structure to systematically define the temporal attributes of a functional requirement. In essence, the concept of Requirement/Availability Time Windows (*RTW/ATW*) proposed by (Song and Chua, 2011) is redefined at two levels: the elementary *RTW/ATW* of an individual user/provider determined from the time intervals of their constituting component states, and the aggregate *RTW/ATW* of all users/providers determined from the elementary *RTWs/ATWs*. This hierarchical structure clearly depicts the time window of individual and the combined set of users/providers, and thus provides planners with a deeper insight to the temporal nature of each user/provider. In addition, a new

construct called “meta-provider” is introduced to define a group of providers which can share their functionalities to jointly satisfy the requirement. The concept of meta-provider is necessary for alternative scheduling from functional requirements, as it enables all provider combinations – representing all possible engineering solutions for the requirement – to be simultaneously considered in the planning stage.

Finally, FReMAS incorporates sequence reasoning knowledge in a three-level framework to convert the necessary condition in the form of a functional relationship between function user and function provider into a temporal constraint between the *RTW/ART*. This constraint is further elaborated into a disjunctive set of constraints between component state intervals based on the hierarchical relationships between the *RTW/ART* and the component state intervals captured in the temporal model. The resultant constraint set of this reasoning process represents all alternative construction sequences making the requirement satisfied. Especially, with the sequence reasoning knowledge built at component state level which is the key construct linking product and process data, FReMAS can be used for scheduling from both product and process perspectives at this lowest level of detail.

### **8.2.3. Preemptive Constraint Analysis Framework**

To address the research needs of improving the solution feasibility and computational efficiency of CSP/CLP-based schedulers presented in section 1.2.3, this dissertation develops a preemptive constraint analysis framework to identify the primary and secondary conflicts and redundancies among the constraints in single and pairs of activities in the pre-scheduling stage. This framework surpasses the existing approaches with the following aspects.

Firstly, the proposed framework can identify all redundancies/inconsistencies between constraints of a single or a pair of activities. Such a complete result is independent of the constraint ordering pattern and thus can help planners identify the optimal conflicting sets to resolve. Effectively, as described in the case study, by identifying redundant disjunctive constraints, the framework helps eliminate unnecessary search space, allowing for a complete search strategy.

The second contribution of this preemptive constraint analysis framework is that the analysis is carried on from the construction management perspective by classifying constraint redundancies/inconsistencies into primary and secondary classes based on impact of activity durations and lags. In particular, to resolve primary constraint redundancy/inconsistency requires a change in lag or constraint type which involves construction method and technical considerations, while the latter can be resolved by a change in activity duration which often involves resource consideration. As such, the primary and secondary distinctions of redundancies/inconsistencies provide useful information for planners to resolve conflicts, and facilitate a more elaborate strategy to manage the constraints.

Finally, to further support the planner in managing the constraints, a method for the computation of the feasible range of an activity duration is embedded in the framework to identify the feasible range of an activity duration considering all associated constraints. This parameter allows planners to verify the validity of an activity duration when changes happen.

From a project management perspective, the framework can practically benefit planners in many ways. Firstly, since temporal constraints are derived from

construction requirements, conflicts among construction requirements can be inferred from any inconsistency among their associated temporal constraints. From this, resolution strategies can be carried out at a higher level. Secondly, when new constraints are introduced to the schedule, its conflicting/redundancy relationships with other constraints of the same activities can be readily identified, and its impact on the schedule can be predicted before implementation. As such, unnecessary rescheduling may be eliminated. Similarly, from the feasible duration ranges, planners can anticipate an inconsistency when an activity duration has to change. Effectively, the proposed approach helps planners and project managers gain a deeper insight on the rationale of the plan so that they may better control the project from the perspective of constraints or construction methods.

#### **8.2.4. Criticality Concept and Schedule Change Analysis Methodology from the Perspective of Constraints and Construction Requirements**

To overcome the research gaps in schedule analysis discussed in section 1.2.4, this dissertation introduces a new criticality concept which is built from the perspective of constraint and construction requirements. This new criticality concept provides a deeper understanding on the role of constraints and requirements to the schedule and forms the foundation for an innovative approach to schedule change analysis using constraint relaxation times.

The first contribution of this concept is a detailed and complete classification of constraint criticality with four categories: project-critical, activity-critical, sequence-critical and redundant. Accordingly, the traditional “non-critical” constraint class is distinguished into three different categories to concisely convey the role of a “non-

critical” constraint to a schedule. Most importantly, the identification of sequence-critical constraints is important for planners since their removal helps achieve a better project makespan.

The second contribution of the proposed concept is the qualitative approach for analyzing the criticality of construction requirements as conjunctive and disjunctive combinations of simple constraints. With the capability to determine the criticality of high-level requirements, this qualitative approach provides project managers with a clearer understanding of the responsibility of associated parties to the overall project schedule, so that better collaboration and management strategies could be employed for good schedule performance. In addition, as illustrated in the case study, the criticality of construction requirements can present deeper insight into the choice of alternative, thus assisting planners in selecting the most suitable schedule from management intentions and/or anticipations for variation.

The constraint criticality indicators, *Aggregate Relaxation Time (ART)* and *Intrinsic Relaxation Time (IRT)* are also key research contributions, and provide the mechanism to determine constraint criticality through activity times. Since changes in the *IRT* will not lead to variations in other constraints, *IRT* can be considered as “free” relaxation time of a constraint. On the other hand, *ART* is shared among the constraints involving the same activities and thus changes in the *ART* of one constraint will lead to variations in the *ART* of other related constraints. Therefore, *ART* could be considered as “total” relaxation time. These distinctions between *ART* and *IRT* clearly demonstrate the impacts of a change in a constraint to others as well as to the entire schedule.

Finally, an innovative approach for schedule change analysis is developed on the proposed constraint relaxation times, allowing schedules to be analyzed and managed from the constraint and requirement perspectives. One significant advantage of this approach is that it provides an insight into how constraint variations could affect schedule makespan including new constraints and removing of constraints, which may not be well conveyed from the activity perspective. Moreover, it can be applied to variations related to both activities times (durations and start times) and lag times, thus encompassing current activity-based analysis approaches. Essentially, this approach enables schedules to be analyzed and managed at different levels of management: activity (duration), constraint (lag), requirement (combination of constraints) and aspect of construction (origin of requirement). Accordingly, planners could choose the most appropriate management policy for each constraint and requirement to achieve better project performance.

### **8.3. Limitations**

In the course of the present research, some limitations have been observed and the major limitations are summarized as follows.

#### **8.3.1. Incorporating Practice Considerations into Automated Scheduling**

The ASCoRe framework provides mechanisms to automatically generate alternative schedules from four basic types of construction requirements: functional requirements, key resource requirements, workspace constraints and temporal constraints. One basic assumption of this scheduling approach is that activities are continuous, and thus calendar constraints have been excluded from the scheduling. To improve the practical advantages of ASCoRe, the framework will have to be extended



to handle activity spitting by implementing a new representation format for activities. This extension also allows for calendar consideration and progress-related constraints.

### **8.3.2. Modeling and Reasoning Nonstandard Complex Functional Requirements**

Engineering solutions for functional requirements are represented as multiple providers in FReMAS. The co-functionality types used in FReMAS capture only standard relationships among providers, in which all providers are either mutually exclusive or compatible. From a practical perspective, the providers of a functional requirement can be combined in different ways to fulfill the requirement, i.e. some of the providers are mutually exclusive while others are compatible. Therefore, FReMAS should be extended to capture such nonstandard functional requirements. This would help improve the practical application of FReMAS to large scale and complex projects.

### **8.3.3. Analyzing Non-Temporal Constraints in the Pre-Scheduling Stage**

The proposed preemptive constraint analysis framework has been demonstrated to be useful in identifying basic redundant constraints in the pre-scheduling stage. However, the major emphasis of the present framework is on conflicts and redundancies of temporal constraints. Future work could extend the framework to include non-temporal constraints such as resource or budget. This will further enhance the feasibility and efficiency of the scheduling. In addition, since activity splitting and calendar constraints are common in construction, the framework would be extended to incorporate these conditions so that its practical benefits could be enhanced.

## **8.4. Recommendations for Future Work**

The methodologies and concepts proposed in this research have opened a new direction to project planning and management which is carried out from the perspective of construction requirements. Some of the potential extensions from this research are summarized as follows.

### **8.4.1. Time-cost Tradeoff Using ASCoRe**

In addition to time, cost is a key indicator for project performance. Incorporating cost into the ASCoRe framework is thus a potential research extension. This multi-objective optimization problem could be solved using a hybrid solving strategy combining the strength of both CLP and heuristic search approaches. In addition, since cost is greatly determined by applied construction methods, time-cost tradeoff analysis would allow planners to specify which method is more attractive in terms of both time and cost, and also analyze possible alternative combinations based on choice.

### **8.4.2. Using Constraint Criticality and Alternative Schedules for Dynamic Schedule Management**

One key advantage of generating alternative schedules is to provide planners with more flexibility in planning and controlling the project. Accordingly, one potential direction for future research is the development of an approach for dynamic schedule control taking into account the existence and significance of multiple alternative schedules. The major emphasis of such an approach is on dynamically analyzing schedule changes to identify if any other alternative schedules would produce a shorter project makespan. Switching among alternative schedules could potentially be a good resolution strategy for mitigating project changes.

Moreover, the existence of some requirements may be uncertain under some scenarios. The criticality concept would be extended to consider this feature. For this purpose, a modeling tool based on fuzzy set theory to capture the uncertain existence of construction requirements along the project lifecycle would be developed. This extension would help improve the practical benefits of the proposed criticality concept.

#### **8.4.3. Prototyping a BIM-based System for Automated Project Planning and Dynamic Control**

The proposed frameworks and concepts for schedule generation and analysis proposed have been validated through proof of concept implementation and small scale case projects. One important future research task is thus to develop a more complex prototype for BIM-based automated planning and control to expand their applications to large scale projects so that the proposed concepts can benefit both researchers and practitioners. The extended prototype would include the following main features:

- Incorporating structural knowledge to automatically derive functional requirements from topological relationships among components. Topological relationships can be extracted from BIM/IFC models and mapped in a graph data model (GDM) (Khalili and Chua, 2012). GDM reorganizes building data in a systematic structure which will be able to run rule-based queries. As such, structural knowledge could be incorporated with GDM to performing structural analysis and deriving functional requirements from BIM models. This facilitates the effective generation of extended product model and spatial interference matrix proposed in process P as depicted in section 3.4.1.

- Improving the present knowledge modeling tools to capture more complex construction methods and requirements to create more comprehensive knowledge libraries. In particular, taxonomies for methods and requirements would be created, and then a knowledge language would be developed. The interpretation of construction knowledge is facilitated by a language parser. These modeling tools can be built upon construction method and requirement models described in sections 3.3.2 and 3.3.3. Based on these, knowledge libraries would be easily created and maintained, allowing planning knowledge to be reused for different projects. These libraries also assist planners to rapidly generate scheduling input.
- Integrating schedule generation and analysis functions into a dynamic project planning and controlling system. The core reasoning and solving engine of such a system is established on the proposed frameworks and algorithms. To support this integration, a comprehensive project database should be designed to enable easy tracing the status of construction requirements and their constituting temporal constraints as well as their impacts on project completion along the project life cycle.
- Importing schedule data and results to Microsoft Project (MSP) application for printing and reporting. The friendly user interface in MSP would be useful for presenting the scheduling results in a familiar format. Special add-ons can be developed in MSP for displaying construction requirements, PDM++ relationships and analysis results such as criticality of constraints, and *ART/IRT*. The integration with MSP would also allow planners consider practical constraints such as calendar, pool resource constraints or activity splitting.

## REFERENCES

- Abeyasinghe, M. C. L., et al. (2001). "An efficient method for scheduling construction projects with resource constraints." *International Journal of Project Management* **19**(1): 29-45.
- Akinci, B., et al. (2002). "Automated generation of work spaces required by construction activities." *Journal of Construction Engineering and Management* **128**(4): 306-315.
- Allen, J. F. (1984). "Towards a general theory of action and time." *Artificial Intelligence* **23**(2): 123-154.
- Alshawi, M. and D. Jagger (1991). "Expert system to assist in generating and scheduling construction activities." *Computers and Structures* **40**(1): 53-58.
- Apt, K. R. (2007). Constraint logic programming using eclipse. Cambridge, Cambridge University Press.
- Babic, N. C., et al. (2010). "Integrating resource production and construction using BIM." *Automation in Construction* **19**(5): 539-543.
- Baptiste, P. and C. Le Pape (2000). Constraint propagation and decomposition techniques for highly disjunctive and highly cumulative project scheduling problems, Netherlands, Kluwer Academic Publishers.
- Barták, R. (2008). Principles of Constraint processing. Artificial Intelligence for Advanced Problem Solving Techniques. I. Vlahavas and V. Dimitris, IGI Global.
- Beck, J. C. and M. S. Fox (2000). "Constraint-directed techniques for scheduling alternative activities." *Artificial Intelligence* **121**(1-2): 211-250.
- Bessiere, C. (2006). Constraint Propagation. Handbook of Constraint Programming. F. Rossi et al, Elsevier.
- Bouchlaghem, D., et al. (2004). "Integrating product and process information in the construction sector." *Industrial Management and Data Systems* **104**(3): 218-233.
- Bowers, J. A. (1995). "Criticality in resource constrained networks." *The Journal of Operational Research Society* **46**(1): 80.
- Bowers, J. A. (2000). "Interpreting float in resource constrained projects." *International Journal of Project Management* **18**(6): 385-392.
- Brailsford, S. C., et al. (1999). "Constraint satisfaction problems: Algorithms and applications." *European Journal of Operational research* **119**(3): 557-581.
- Caseau, Y. and F. Laburthe (1994). Improved CLP scheduling with task intervals. *Proceedings of the eleventh international conference on Logic Programming*, MIT Press.
- Cesta, A., et al. (2002). "A constraint-based method for project scheduling with time windows." *Journal of Heuristics* **8**(1): 109-136.
- Chan, D. W. M. and M. M. Kumaraswamy (1997). "A comparative study of causes of time overruns in Hong Kong construction projects." *International Journal of Project Management* **15**(1): 55-63.
- Chan, W.-T., et al. (1996). "Construction resource scheduling with genetic algorithms." *Journal of Construction Engineering and Management* **122**(2): 125-132.

- Cherneck, J., et al. (1991). "Integrating CAD with construction-schedule generation." *Journal of Computing in Civil Engineering* **5**(1): 64-84.
- Chevallier, N. and A. D. Russell (1998). "Automated schedule generation." *Canadian Journal of Civil Engineering* **25**: 1095-1123.
- Choi, C., et al. (2006). Finite Domain Bounds Consistency Revisited. *LNCS AI 2006: Advances in Artificial Intelligence*. a. B.-h. K. A. Sattar, eds., Springer Berlin / Heidelberg: 49-58.
- Choo, H. J., et al. (1999). "WorkPlan: Constraint-based database for work package scheduling." *Journal of Construction Engineering and Management* **125**(3): 151-160.
- Chua, D. K. H., et al. (1999). "Critical success factors for different project objectives." *Journal of Construction Engineering and Management* **125**(3): 142-150.
- Chua, D. K. H., et al. (2013). "Automated construction sequencing and scheduling from functional requirements." *Automation in Construction* (**in press**).
- Chua, D. K. H. and L. J. Shen (2005). "Key constraints analysis with integrated production scheduler." *Journal of Construction Engineering and Management* **131**(7): 753-764.
- Chua, D. K. H. and K. W. Yeoh (2011). "PDM++: Planning framework from a construction requirements perspective." *Journal of Construction Engineering and Management* **137**(4): 266-274.
- Chua, D. K. H., et al. (2010). "Quantification of spatial temporal congetion in four-dimensional computer-aided design." *Journal of Construction Engineering and Management* **136**(6): 641-649.
- Dechter, R. (2003). *Constraint processing*. Morgan Kaufmann Publishers, San Francisco.
- Dorndorf, U., et al. (2000). "Constraint propagation techniques for the disjunctive scheduling problem." *Artificial Intelligence* **122**(1): 189-240.
- Dzeng, R.-J. and I. D. Tommelein (2004). "Product modeling to support case-based construction planning and scheduling." *Automation in Construction* **13**(3): 341-360.
- Echeverry, D., et al. (1991). "Sequencing knowledge for construction scheduling." *Journal of Construction Engineering and Management* **117**(1): 118-130.
- El-Bibany, H. (1997). "Parametric Constraint Management in Planning and Scheduling: Computational Basis." *Journal of Construction Engineering and Management* **123**(3): 348-353.
- El-Diraby, T. E. and K. F. Kashif (2005). "Distributed ontology architecture for knowledge management in highway construction." *Journal of Construction Engineering and Management* **131**(5): 591-603.
- El-Gohary, N. M. and T. E. El-Diraby (2010). "Domain ontology for processes in infrastructure and construction." *Journal of Construction Engineering and Management* **136**(7): 730-744.
- Elmaghraby, S. and J. Kamburowski (1992). "The analysis of activity networks under generalized precedence relations (GPRs)." *Management Science* **38**(9): 1245-1263.
- Fan, S.-L. and H. P. Tserng (2006). "Object-oriented scheduling for repetitive projects with soft logics." *Journal of Construction Engineering and Management* **132**(1): 35-48.

- Faris, R. K. (1991). Role of scheduling in computer integrated construction. *Construction Congress '91, April 13, 1991 - April 16, 1991*, Cambridge, MA, USA, Publ by ASCE.
- Feng, C.-W., et al. (2010). "Using the MD CAD model to develop the time-cost integrated schedule for construction projects." *Automation in Construction* **19**(3): 347-356.
- Fischer, M. and F. Aalami (1996). "Scheduling with computer-interpretable construction method models." *Journal of Construction Engineering and Management* **122**(4): 337-347.
- Fischer, M. A. (1993). "Automating constructibility reasoning with a geometrical and topological project model." *Computing Systems in Engineering* **4**(2-3): 179-192.
- Froese, T. (1996). "Models of construction process information." *Journal of Computing in Civil Engineering* **10**(3): 183-193.
- Fromherz, M. P. J. (2001). Constraint-based scheduling, Arlington, VA, United states, Institute of Electrical and Electronics Engineers Inc.
- Ghezail, F., et al. (2010). "Analysis of robustness in proactive scheduling: a graphical approach." *Computers & Industrial Engineering* **58**(2): 193-198.
- Goedert, J. D. and P. Meadati (2008). "Integrating construction process documentation into building information modeling." *Journal of Construction Engineering and Management* **134**(Compendex): 509-516.
- Goltz, H. J. (1995). Reducing domains for search in CLP(FD) and its application to job-shop scheduling. *Proceedings of Principles and Practice of Constraint Programming, 19-22 Sept. 1995*, Berlin, Germany, Springer-Verlag.
- Gominuka, T. and F. Sadeghpour (2008). A framework for activity-based identification of space requirements for construction equipment, Quebec City, QC, Canada, Canadian Society for Civil Engineering.
- Gray, C. (1986). "Intelligent construction time and cost analysis." *Construction Management and Economics* **4**(2): 135-150.
- Halfawy, M. M. R. and T. M. Froese (2007). "Component-based framework for implementing integrated architectural/ engineering/construction project systems." *Journal of Computing in Civil Engineering* **21**(6): 441-452.
- Halpin, D. W. (1985). Financial and cost concepts for construction management, Wiley, Newyork.
- Hanna, A., et al. (1999). "Impact of Change Orders on Labor Efficiency for Mechanical Construction." *Journal of Construction Engineering and Management* **125**(3): 176-184.
- Harris, R. B. (1978). Precedence and arrow networking techniques for construction. Wiley, New York.
- Hartmann, V., et al. (2012). Model-based Scheduling for Construction Planning. *14th International Conference on Computing in Civil and Building Engineering*, Moscow, Russia.
- Hegazy, T. (1999). "Optimization of resource allocation and leveling using genetic algorithms." *Journal of Construction Engineering and Management* **125**(3): 167-175.
- Hendrickson, C., et al. (1987). An expert system architecture for construction planning, Southampton, UK, Comput. Mech. Publications.
- Ibbs, C., et al. (2001). "Project Change Management System." *Journal of Management in Engineering* **17**(3): 159-165.

- Iyer, K. C. and K. N. Jha (2006). "Critical factors affecting schedule performance: Evidence from indian construction projects." *Journal of Construction Engineering and Management* **132**(8): 871-881.
- Jaafari, A. (1984). "Criticism of CPM for project planning analysis." *Journal of Construction Engineering and Management* **110**(2): 222-233.
- Jaafari, A. (1996). "Time and priority allocation scheduling technique for projects." *International Journal of Project Management* **14**(5): 289-299.
- Jaffar, J. and M. J. Maher (1994). "Constraint logic programming: a survey." *Journal of Logic Programming* **19-20**(Copyright 1994, IEE): 503-581.
- Jaffar, J., et al. (1994). Beyond finite domains. *Proceedings of the Second International Workshop on Principles and Practice of Constraint Programming*, Springer-Verlag, 86-94.
- Jung, Y. and M. Joo (2011). "Building information modelling (BIM) framework for practical implementation." *Automation in Construction* **20**(2): 126-133.
- Kartam, N. and R. E. Levitt (1990). "Intelligent planning of construction projects." *Journal of Computing in Civil Engineering* **4**(2): 155-175.
- Kataoka, M. (2008). "Automated generation of construction plans from primitive geometries." *Journal of Construction Engineering and Management* **134**(8): 592-600.
- Kelley, J. E. J. (1961). "Critical-path planning and scheduling: Mathematical basis." *Operations Research* **9**(3): 296-320.
- Khalili, A. and D. K. H. Chua (2012). "An IFC-based framework to move beyond individual building elements towards configuring higher level prefabrication." *Journal of Computing in Civil Engineering* **27**(3): 243-253.
- Kim, H., et al. (2013). "Generating construction schedules through automatic data extraction using open BIM (building information modeling) technology." *Automation in Construction*(0).
- Kim, K. and J. M. d. I. Garza (2005). "Evaluation of the resource-constrained critical path method algorithms." *Journal of Construction Engineering and Management* **131**(5): 522-532.
- Koo, B., et al. (2007). "Formalization of construction sequencing rationale and classification mechanism to support rapid generation of sequencing alternatives." *Journal of Computing in Civil Engineering* **21**(Compendex): 423-433.
- Laborie, P. (2003). "Algorithms for propagating resource constraints in AI planning and scheduling: Existing approaches and new results." *Artificial Intelligence* **143**(2): 151-188.
- Lecoutre, C. (2009). *Constraint Networks: Techniques and Algorithms*. London : ISTE ; Hoboken, NJ : Wiley.
- Leu, S.-S. and C.-H. Yang (1999). "GA-based multicriteria optimal model for construction scheduling." *Journal of Construction Engineering and Management* **125**(6): 420-427.
- Levitt, R. E. and J. C. Kunz (1985). Knowledge-based system for updating engineering project schedules, Miami Beach, FL, USA, ASME.
- Liu, S.-S. and K.-C. Shih (2009). "A framework of critical resource chain for project schedule analysis." *Construction Management and Economics* **27**(9): 857-869.
- Liu, S.-S. and C.-J. Wang (2008). "Resource-constrained construction project scheduling model for profit maximization considering cash flow." *Automation in Construction* **17**(8): 966-974.



- Lock, D. (2003). Project management. Gower Publishing Ltd., Hampshire, England.
- Lorterapong, P. and M. Ussavadiokrit (2012). "Construction Scheduling Using the Constraint Satisfaction Problem Method." *Journal of Construction Engineering and Management*.
- Lorterapong, P. and M. Ussavadiokrit (2013). "Construction Scheduling Using the Constraint Satisfaction Problem Method." *Journal of Construction Engineering and Management* **139**(4): 414-422.
- Lu, M. and H.-C. Lam (2008). "Critical path scheduling under resource calendar constraints." *Journal of Construction Engineering and Management* **134**(1): 25-31.
- Lu, M. and H.-C. Lam (2009). "Transform schemes applied on non-finish-to-start logical relationships in project network diagrams." *Journal of Construction Engineering and Management* **135**(9): 863-873.
- Lu, M. and H. Li (2003). "Resource-activity critical-path method for construction planning." *Journal of Construction Engineering and Management* **129**(4): 412-420.
- Marriott, K., et al. (2006). Constraint Logic Programming. Handbook of constraint programming. F. Rossi et al. Amsterdam ; Boston, Elsevier: 409-452.
- McKinney, K. and M. Fischer (1998). "Generating, evaluating and visualizing construction schedules with CAD tools." *Automation in Construction* **7**(6): 433-447.
- Mikulakova, E., et al. (2010). "Knowledge-based schedule generation and evaluation." **24**: 389-403.
- Moder, J. J., et al. (1983). Project management with CPM, PERT and precedence diagramming. 3rd edition. Wokingham, Berks., UK, Van Nostrand Reinhold.
- Morad, A. A. and Y. J. Beliveau (1994). "Geometric-based reasoning system for project planning." *Journal of Computing in Civil Engineering* **8**(1): 52-71.
- Moselhi, O. and M. J. Nicholas (1990). "Hybrid expert system for construction planning and scheduling." *Journal of Construction Engineering and Management* **116**(2): 221-238.
- Motawa, I. A., et al. (2007). "An integrated system for change management in construction." *Automation in Construction* **16**(3): 368-377.
- Navinchandra, D., et al. (1988). "GHOST: Project network generator." *Journal of Computing in Civil Engineering* **2**(3): 239-254.
- Nguyen, T.-H. and A. A. Oloufa (2002). "Spatial information: Classification and applications in building design." *Computer-Aided Civil and Infrastructure Engineering* **17**(4): 246-255.
- Nguyen, T.-H., et al. (2005). "Algorithms for automated deduction of topological information." *Automation in Construction* **14**(1): 59-70.
- Nguyen, T. Q. and D. K. H. Chua (2012). Criticality of Schedule Constraints – Classification and Identification. 3rd International Conference of Engineering, Product and Project Management. Brighton, UK.
- Plotnick, F. L. (2006). RDM - Relationship Diagramming Method. 2006 ACE International Transactions - 50th Annual Meeting, Jun 19 - 22 2006, Las Vegas, NV, United states, Association for the Advancement of Cost Engineering.
- Porkka, J. and K. Kähkönen (2007). Software Development Approaches and Challenges of 4D Product Models. Proceedings of the CIB-W78 Conference, 2007.
- Rahman, S. U. (1998). "Theory of constraints: a review of the philosophy and its applications." *International Journal of Operations & Production Management* **18**(4): 336-355.

- Rivera, F. A. and A. Duran (2004). "Critical clouds and critical sets in resource-constrained projects." *International Journal of Project Management* **22**(6): 489-497.
- Ryu, H.-G., et al. (2007). "Construction planning method using case-based reasoning (CONPLA-CBR)." *Journal of Computing in Civil Engineering* **21**(6): 410-422.
- Shaked, O. and A. Warszawski (1992). "Consched: expert system for scheduling of modular construction projects." *Journal of Construction Engineering and Management* **118**(3): 488-506.
- Shaked, O. and A. Warszawski (1995). "Knowledge-based system for construction planning of high-rise buildings." *Journal of Construction Engineering and Management* **121**(2): 172-182.
- Shih-Chung, K. and E. Miranda (2008). "Computational methods for coordinating multiple construction cranes." *Journal of Computing in Civil Engineering* **22**(4): 252-263.
- Singh, V., et al. (2011). "A theoretical framework of a BIM-based multi-disciplinary collaboration platform." *Automation in Construction* **20**(Compendex): 134-144.
- Song, Y. (2006). Intermediate function analysis for improving constructability. Doctoral Dissertation, National University of Singapore.
- Song, Y. and D. K. H. Chua (2006). "Modeling of Functional Construction Requirements for Constructability Analysis." *Journal of Construction Engineering and Management* **132**(12): 1314-1326.
- Song, Y. and D. K. H. Chua (2011). "Requirement and availability time-window analysis of intermediate function." *Journal of Construction Engineering and Management* **137**(11): 967-975.
- Sriprasert, E. and N. Dawood (2002). Requirement Identification for 4D constraint-based construction planning and control system. *International Council for Research and Innovation in Building and Construction*.
- Staub-French, S., et al. (2003). "An ontology for relating features with activities to calculate costs." *Journal of Computing in Civil Engineering* **17**(4): 243-254.
- Stumpf, A. L., et al. (1996). "Object-oriented model for integrating construction product and process information." *Journal of Computing in Civil Engineering* **10**(3): 204-212.
- Tah, J. H. M., et al. (1999). "Information modelling for case-based construction planning of highway bridge projects." *Advances in Engineering Software* **30**(7): 495-509.
- Tamimi, S. and J. Diekmann (1988). "Soft logic in network analysis." *Journal of Computing in Civil Engineering* **2**(3): 289-300.
- Tauscher, E., et al. (2009). Automated generation of construction schedules based on the IFC object model, Austin, TX, United states, American Society of Civil Engineers.
- Thabet, W. Y. and Y. J. Beliveau (1997). "SCaRC: Space-Constrained Resource-Constrained scheduling system." *Journal of Computing in Civil Engineering* **11**(1): 48-59.
- Tsamardinos, I. and M. E. Pollack (2003). "Efficient solution techniques for disjunctive temporal reasoning problems." *Artificial Intelligence* **151**(1-2): 43-89.
- Valls, V. and P. Lino (2001). "Criticality analysis in activity-on-node networks with minimal time lags." *Annals of Operations Research* **102**: 17-37.
- Van Hentenryck, P. (1989). Constraint Satisfaction in Logic Programming. MIT Press.

- Vozzola, M., et al. (2009). BIM use in the construction process. *2009 International Conference on Management and Service Science (MASS)*, 20-22 Sept. 2009, Piscataway, NJ, USA, IEEE.
- Vries, B. D. and J. M. J. Harink (2007). "Generation of a construction planning from a 3D CAD model." *Automation in Construction* **16**(1): 13-18.
- Wallace, M. (2002). Constraint Logic Programming. Computational Logic: Logic Programming and Beyond. A. C. Kakas and F. Sadri. New York, Springer: 512-532.
- Waugh, L. M. (1989). Knowledge-based construction scheduling, New York, NY, USA, Publ by ASCE.
- Weise, M. and T. Liebich (2009). IFC support for model-based scheduling. Managing It in Construction/Managing Construction for Tomorrow. A. Dikbasetal, CRC Press, Taylor & Francis Group, Istanbul, Turkey. **26**: 75-83.
- Wiest, J. D. (1964). "Some properties of schedules for large projects with limited resources." *Operations Research* **12**: 395-418.
- Wiest, J. D. (1981). "Precedence diagramming method: some unusual characteristics and their implications for project managers." *Journal of Operations Management* **1**(3): 121.
- Wiest, J. D. and F. K. Levy (1977). A management guide to PERT/CPM with GERT/PDM/DCPM and other networks. Printice-Hall.
- Winstanley, G., et al. (1993). "An integrated project planning environment." *Intelligent Systems Engineering* **2**(2): 91-106.
- Winstanley, G., et al. (1993). "Model-based planning: scaled-up construction application." *Journal of Computing in Civil Engineering* **7**(2): 199-217.
- Woodworth, B. M. and S. Shanahan (1988). "Identifying the critical sequence in a resource constrained project." *International Journal of Project Management* **6**(2): 89-96.
- Yamazaki, Y. (1995). "An integrated construction planning system using object-oriented product and process modelling." *Construction Management & Economics* **13**(5): 417:426.
- Yeoh, K. W. (2012). Construction Requirements Driven Planning and Scheduling. Doctoral Dissertation, National University of Singapore.
- Zhang, S., et al. (2012). "Build information modeling (BIM) and safety: Automatic safety checking of construction models and schedules."
- Zhao, Z. Y., et al. (2010). "Prediction system for change management in construction project." *Journal of Construction Engineering and Management* **136**(6): 659-669.
- Zozaya-Gorostiza, C., et al. (1989). Knowledge-based construction project planning, New York, NY, USA, Publ by ASCE.
- Zweben, M. and M. Fox (1994). Intelligent Scheduling, Morgan Kaufman.

## APPENDIX

### A1. Pairwise Constraint Integration Tableaux

This appendix presents the application of the preemptive constraint analysis framework proposed in chapter five to PDM++ constraints. The result is described in a table format. These tables are useful for planners to perform instant or manual constraint check. Moreover, primary conflicts (displayed as shaded cells) are further distinguished into “hard” and “soft”. A conflict is considered “hard” when it always happens regardless of the value of lag times, while a “soft” conflict can be removed in some very specific conditions of lag types. For example, constraints  $X DB(m)$  and  $X SA(n)$  are conflicting with any value of  $m$  and  $n$ ; therefore this is a hard constraint. In contrast, constraints  $X SA(m)$  and  $X SA(n)$  are inconsistent with any value of  $m$  and  $n$  where  $m \neq n$ . This conflict however will no longer exist when  $m$  is equal to  $n$ . Therefore, it is classified as a soft conflict. This differentiation provides planners with a deeper insight into primary constraint conflicts as well as alternative strategies to resolve such inconsistencies.

Table A.1. Pairwise integration of unary constraints

$C2 \backslash C1$	Short Form	$DB(n)$	$DA(n)$	$SB(n)$	$SA(n)$
Due-Before( $m$ )	DB	<b>DB(<math>m</math>)</b>	$m \neq n$	<b>DB(<math>m</math>)</b>	
Due-After( $m$ )	DA		<b>DA(<math>n</math>)</b>		<b>SA(<math>n</math>)</b>
Start-Before( $m$ )	SB		$d_x \geq n - m$	<b>SB(<math>m</math>)</b>	$m \neq n$
Start-After( $m$ )	SA	$d_x \leq n - m$			<b>SA(<math>n</math>)</b>

Table A.2. Pairwise integration of non-lag type binary constraints

$C1 \backslash C2$	Short Form	M	MB	C	CB	D
Meets	M	<b>M</b>				<b>M</b>
Met-By	MB		<b>MB</b>			<b>MB</b>
Contains	C			<b>C</b>	$d_x = d_y$	
Contained -By	CB			$d_x = d_y$	<b>CB</b>	
Disjoint	D	<b>M</b>	<b>MB</b>			<b>D</b>

Table A.3. Pairwise Integration of Lag Type and Non-lag Type Binary Constraints

$C2 \backslash C1$	B(m)	B(~m)	A(m)	A(~m)	S(m)	S(~m)	SB(m)	SB(~m)	F(m)	F(~m)	FB(m)	FB(~m)
M	$m \neq 0$	<b>M</b>			$d_x \geq m$	$d_x \leq m$			$d_y \geq m$	$d_y \leq m$		
MB			$m \neq 0$	<b>MB</b>			$d_y \geq m$	$d_y \leq m$			$d_x \geq m$	$d_x \leq m$
C					$d_x - d_y \geq m$		$m \neq 0$	$m \neq 0$	$m \neq 0$	$d_y - d_x \leq m$	$d_x - d_y \geq m$	
CB					$m \neq 0$	$d_x - d_y \leq m$	$m \neq 0$	$m \neq 0$	$m \neq 0$		$m \neq 0$	$d_x - d_y \leq m$
D	<b>B(m)</b>	<b>B(~m)</b>	<b>A(m)</b>	<b>A(~m)</b>	<b>S(m)</b> <b>B(0)</b>		<b>SB(m)</b> <b>A(0)</b>		<b>F(m)</b> <b>B(0)</b>		<b>FB(m)</b> <b>A(0)</b>	

$C2 \backslash C1$	SF(m)	SF(~m)	ISF(m)	ISF(~m)	O(m)	O(~m)	OB(m)	OB(~m)	SO(m)	SOB(m)	EO(m)	EOB(m)
M	$d_x + d_y \geq m$	$d_x + d_y \leq m$			$m \neq 0$	<b>M</b>	$n \neq 0$	<b>M</b>		$m \neq 0$	$m \neq 0$	
MB			$d_x + d_y \geq m$	$d_x + d_y \leq m$	$m \neq 0$	<b>MB</b>	$n \neq 0$	<b>MB</b>	$m \neq 0$			$m \neq 0$
C	$d_x \geq m$	$d_y \leq m$	$d_x \geq m$	$d_y \leq m$	<b>C</b>	$d_y \leq m$	<b>C</b>	$d_y \leq m$	$d_x \geq m$	$d_x \geq m$	$d_x \geq m$	$d_x \geq m$
CB	$d_y \geq m$	$d_x \geq m$	$d_y \geq m$	$d_x \geq m$	<b>CB</b>	$d_x - d_y \geq m$	<b>CB</b>	$d_x \leq m$	$d_y \geq m$	$d_y \geq m$	$d_y \geq m$	$d_y \geq m$
D					$m \neq 0$	<b>D</b>	$m \neq 0$	<b>D</b>	$m \neq 0$	$m \neq 0$	$m \neq 0$	$m \neq 0$

Table A.4. Pairwise Integration of Lag Type Binary Constraints

C2 \ C1	B(n)	B(~n)	A(n)	A(~n)	S(n)	S(~n)	SB(n)	SB(~n)	F(n)	F(~n)	FB(n)	FB(~n)	SF(n)	SF(~n)	ISF(n)	ISF(~n)	O(n)	O(~n)	OB(n)	OB(~n)	SO(n)	SOB(n)	EO(n)	EOB(n)	
Before(m)	<b>B(n)</b>					$d_x \leq n-m$				$d_y \leq n-m$				$d_x+d_y \leq n-m$			$m \neq 0$	$m \neq 0$	$m \neq 0$	$m \neq 0$		$m \neq 0$	$m \neq 0$		
Before(~m)	$m \neq 0$	<b>B(~m)</b>			$d_x \geq n-m$				$d_y \geq n-m$				$d_x+d_y \geq n-m$				$n \neq 0$	<b>M</b>	$n \neq 0$	<b>M</b>	$n \neq 0$	$n \neq 0$	$n \neq 0$		
After(m)			<b>A(n)</b>					$d_y \leq n-m$				$d_x \leq n-m$			$d_x+d_y \leq n-m$		$m \neq 0$	$m \neq 0$	$m \neq 0$	$m \neq 0$	$m \neq 0$			$m \neq 0$	
After(~m)			$m \neq 0$	<b>A(~m)</b>				$d_y \geq n-m$			$d_x \geq n-m$				$d_x+d_y \geq n-m$		$n \neq 0$	<b>MB</b>	$n \neq 0$	<b>MB</b>	$n \neq 0$			$n \neq 0$	
Starts(m)	<b>B(n)</b>				<b>S(n)</b>				$d_y-d_x \leq n-m$	$d_x-d_y \geq n+m$				$d_y \leq n-m$	$d_x \geq n+m$		$d_x \geq n+m$					$d_x \geq n+m$	$d_x \geq n+m$		
Starts(~m)					$m \neq 0$	<b>S(~m)</b>			$d_x-d_y \geq n-m$		$d_x-d_y \leq n+m$		$d_y \geq n-m$		$d_x \leq n+m$	$d_y \geq n-m$		$d_y \geq n-m$							
Started-By(m)			<b>A(n)</b>				<b>SB(n)</b>		$d_x-d_y \geq n+m$			$d_x-d_y \leq n-m$		$d_x \geq n+m$	$d_x \geq n-m$		$d_y \geq n+m$				$d_y \geq n+m$			$d_y \geq n+m$	
Started-By(~m)							$m \neq 0$	<b>SB(~m)</b>		$d_x-d_y \leq n+m$	$d_x-d_y \geq n-m$			$d_y \leq n+m$	$d_x \geq n-m$		$d_x \geq n-m$				$d_x \geq n-m$				
Finishes(m)	<b>B(n)</b>					$d_x-d_y \leq n-m$	$d_x-d_y \leq n-m$		<b>F(n)</b>					$d_x \leq n-m$	$d_y \geq n+m$		$d_y \geq n+m$					$d_y \geq n+m$	$d_y \geq n+m$		
Finishes(~m)					$d_x d_y \geq n-m$			$d_x-d_y \leq n+m$	$m \neq 0$	<b>F(~m)</b>				$d_x \geq n-m$		$d_x-d_y \leq n+m$							$d_y-d_x \leq m$	$d_y-d_x \leq m$	
Finished-By(m)			<b>A(n)</b>			$d_x+d_y \geq n+m$					<b>FB(n)</b>			$d_x \geq n+m$		$d_y \leq n-m$	$d_x \geq n+m$			$d_x \geq n+m$				$d_x \geq n+m$	
Finished-By(~m)						$d_x-d_y \leq n+m$	$d_y-d_x \geq n-m$				$m \neq 0$	<b>FB(~m)</b>		$d_x \leq n+m$	$d_y \geq n-m$		$d_y \geq n-m$				$d_x-d_y \leq m$			$d_x-d_y \leq m$	
Start-Finish(m)	<b>B(n)</b>				<b>S(n)</b>			$d_y \geq n+m$	<b>F(n)</b>		$d_x \geq n+m$	<b>SF(m)</b>	<b>SF(n)</b>		$d_x+d_y \geq n-m$		$d_x+d_y \geq n+m$				$d_y \geq m$	$d_x+d_y \geq n+m$	$d_x+d_y \geq n+m$	$d_x \geq m$	
Start-Finish(~m)								$d_y \leq n+m$	<b>SF(~m)</b>		$d_x \leq n+m$	$m \neq 0$	<b>SF(~m)</b>		$d_x+d_y \leq n+m$							$d_y \leq m$	$d_x \leq m$		
Inv_SF(m)	$m \neq 0$	$m \neq 0$			$d_x \geq n+m$				$d_y \geq n+m$					$d_x+d_y \geq n+m$		<b>ISF(n)</b>		$d_x+d_y \geq n+m$		$d_x+d_y \geq n+m$		$d_x+d_y \geq n+m$	$d_x \geq m$	$d_y \geq m$	
Inv_SF(~m)	$n \neq 0$	<b>M</b>			$d_x \leq n+m$				$d_y \leq n+m$					$d_x+d_y \leq n+m$	$m \neq 0$	<b>ISF(~m)</b>					$d_x \leq m$		$d_y \leq m$		
Overlaps(m)	$m \neq 0$	$m \neq 0$	$m \neq 0$	$m \neq 0$	$d_x \geq n+m$		$d_y \geq n+m$		$d_y \geq n-m$		$d_x \geq n+m$			$d_x+d_y \geq n+m$	$d_y-d_x \leq n-m$	$d_x+d_y \geq n-m$		<b>O(n)</b>		<b>OB(n)</b>		<b>SO(n)</b>	<b>SOB(n)</b>	<b>EO(n)</b>	<b>EOB(n)</b>
Overlaps(~m)	$n \neq 0$	<b>M</b>	$n \neq 0$	<b>MB</b>													$m \neq 0$	<b>O(~m)</b>	$m \neq 0$	<b>O(~m)</b>	$m \neq 0$	$m \neq 0$	$m \neq 0$	$m \neq 0$	
Overlaped-By(m)	$m \neq 0$	$m \neq 0$	$m \neq 0$	$m \neq 0$	$d_x \geq n+m$		$d_y \geq n+m$		$d_y \geq n-m$		$d_x \geq n+m$			$d_x+d_y \geq n+m$	$d_y-d_x \leq n-m$	$d_x+d_y \geq n-m$		<b>O(n)</b>		<b>OB(n)</b>		$m \neq 0$	$m \neq 0$	$m \neq 0$	
Overlaped-By(~m)	$n \neq 0$	<b>M</b>	$n \neq 0$	<b>MB</b>													$m \neq 0$	<b>OB(~m)</b>	$m \neq 0$	<b>OB(~m)</b>	$m \neq 0$	$m \neq 0$	$m \neq 0$		
Start-Overlap(m)			$m \neq 0$	$m \neq 0$			$d_y \geq n+m$				$d_x \geq n+m$	$d_x-d_y \leq n$	$d_y \geq n$		$d_x+d_y \geq n-m$	$d_x \leq n$	<b>SO(n)</b>		<b>SO(n)</b>		<b>SO(n)</b>			<b>EOB(n)</b>	
Start-O-By(m)	$m \neq 0$	$m \neq 0$			$n \neq 0$				$d_y \geq n+m$	$d_y-d_x \leq n$				$d_x+d_y \geq n+m$	$d_y \leq n$	$d_x \geq n$	<b>SOB(n)</b>		<b>SOB(n)</b>			<b>SOB(n)</b>	<b>EOB(n)</b>		
End-Overlap(m)	$m \neq 0$	$m \neq 0$			$d_x \geq n+m$	$d_x-d_y \leq n$			$d_y \geq n+m$					$d_x+d_y \geq n+m$	$d_x \leq n$	$d_y \geq n$	<b>EO(n)</b>		<b>EO(n)</b>		<b>SO(n)</b>		<b>EO(n)</b>		
End-O-By(m)			$m \neq 0$	$m \neq 0$			$d-d_x \geq n$				$d_x \geq n+m$	$d_x-d_y \leq n$	$d_x \geq n$		$d_x+d_y \geq n-m$	$d_x \leq n$	<b>EOB(n)</b>		<b>EOB(n)</b>		<b>SO(n)</b>			<b>EOB(n)</b>	

# CURRICULUM VITAE

---

## NGUYEN THI QUI

### 1. EDUCATION

- 2001-2006      B.Eng (Civil and Industrial Structures),  
1<sup>st</sup> Class Upper Honors,  
Faculty of Civil Engineering  
Hochiminh City University of Technology, Vietnam
- 2006-2008      M.Eng (Construction Technology and Management),  
Hochiminh City University of Technology, Vietnam
- 2008-2013      Research Scholar, Infrastructure System Group (IS),  
Department of Civil and Environmental Engineering,  
National University of Singapore, Singapore

### 2. EXPERIENCE

- Nov, 2005 –      Structural Engineer  
Aug, 2006      Structures Vietnam, Vietnam
- Aug, 2006 –      Lecturer  
Aug, 2008      Division of Construction Engineering and Management  
Faculty of Civil Engineering  
Hochiminh City University of Technology, Vietnam

### 3. LIST OF PUBLICATIONS

#### 3.1. Journal Papers

1. David K. H. Chua, T. Q. Nguyen, K. W. Yeoh, (2013) “Automated construction sequencing and scheduling from functional requirements.” *Automation in Construction*, 35, 79-88.
2. T. Q. Nguyen, David K. H. Chua, “Preemptive analysis of temporal constraints in construction schedules.” *Journal of Computing in Civil Engineering*, ASCE (in press).

3. T. Q. Nguyen, David K. H. Chua, “Criticality of schedule constraints – classification and identification for project management.” *Journal of Engineering, Project, and Production Management*, 4(1), 17-25.

### 3.2. Conference Papers

1. Nguyen Thi Qui, David K.H. Chua, and Ker-Wei, Yeoh (2010). *Functional Requirement Oriented Framework for Schedule Generation*, 6<sup>th</sup> International conference on Innovation in Architecture, Engineering and Construction (AEC), Pennsylvania, USA, June 2010.
2. T.Q. Nguyen, David K.H. Chua, and K.W. Yeoh (2010). *Extended Functional Requirement Model for Construction Schedule Computation*, 23<sup>rd</sup> KKCNN Symposium on Civil Engineering, Taipei, Taiwan, November 2010.
3. Qui T. Nguyen and David K.H. Chua (2012), *Criticality of Schedule Constraints – Identification and Classification*, 3<sup>rd</sup> International Conference on Engineering, Project and Production Management, Brighton, United Kingdom, September 2012, (Best paper award).