GENERATING VERIFIABLE SERVICE
CHOREOGRAPHIES FROM SBVR MODELS

NURULHUDA A. MANAF

A THESIS
SUBMITTED FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
IN
COMPUTER SCIENCE
FROM THE
UNIVERSITY OF SURREY

DEPARTMENT OF COMPUTER SCIENCE
FACULTY OF ENGINEERING AND PHYSICAL SCIENCES
UNIVERSITY OF SURREY
GUIDFORD, SURREY GU2 7XH
UNITED KINGDOM

SUPERVISORS:
PRINCIPAL SUPERVISOR: DR. SOTIRIS MOSCHOYIANNIS
CO-SUPERVISOR: PROFESSOR PAUL J. KRAUSE

SEPTEMBER 2017
© NURULHUDA A. MANAF 2017
Declaration of Originality

This thesis and the work to which it refers are the results of my own efforts. Any ideas, data, images or text resulting from the work of others (whether published or unpublished) are fully identified as such within the work and attributed to their originator in the text, bibliography or in footnotes. This thesis has not been submitted in whole or in part for any other academic degree or professional qualification. I agree that the University has the right to submit my work to the plagiarism detection service TurnitinUK for originality checks. Whether or not drafts have been so-assessed, the University reserves the right to require an electronic version of the final document (as submitted) for assessment as above.

September 2017
Nurulhuda A.Manaf
Guaranteeing the correct coordination of distributed applications that are built up as networks of autonomous participants, e.g., software components, web services, online resources, software as a service (SaaS) peers, is inherently challenging. This is obvious when the current distributed applications involve a collaboration between loosely-coupled services on distinct providers; the ordering of interactions that may further affects the dependencies between different participants, including control flow dependencies (e.g., a given service invocation must occur before another one), time constraints, and transactional dependencies. This complexity of the development of distributed applications illustrates how important the techniques and approaches for designing and coordinating the service interactions between distinct participant services to ensure that the overall goal of the collaboration between participant services is achieved. Standardisation efforts to date have resulted in the Web Services Choreography Description Language (WS-CDL), a specification protocol advocated by W3C. WS-CDL and other modeling languages (e.g., UML2) provide various divergent semantics and less user-friendly graphical notation. On the other hand the formal approach would allow unambiguous specification and verification of the intended collaboration.

In this research work, a declarative approach was proposed for specifying coordination of distributed applications involving distinct participant services which is being able to verify that it is correct. The proposed approach could captures and describing the complex interactions that involves the ordering of service interaction based on the given global constraints.

A new model using a declarative approach, an OMG standard Semantics of Business Vocabulary and Rules (SBVR) model was introduced for specifying service choreography. This SBVR model is then formulated and transformed into Alloy model using Alloy Analyzer for verification. A fully automated SBVR2Alloy tool was implemented for transforming from the developed SBVR model into the Alloy model. This proposed model is targeted to enable the practitioners (business analysts, developers) to devise and set up the service choreographies that realise their collaborations by generating the automated verifiable choreography model.
There are a number of people without whom this thesis might not have been written, and to whom I am greatly indebted.

I would like to express my sincere gratitude to my principal supervisor, Dr. Sotiris Moschoyiannis for the continuous support of my PhD study, for his patience, motivation, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis. Without his precious support it would not be possible to conduct this research.

My sincere thanks also goes to my co-supervisor, Prof. Paul Krause for his insightful ideas, advice, comments and encouragement, especially from the business perspective to my research.

A very special gratitude goes out to Ministry of Higher Education Malaysia and also National Defence University of Malaysia (NDUM) for providing a fully-support of the funding throughout my PhD study.

To my dear husband, Mohd Zulkifli Ismail, I would like to express a very special thank you for your moral and emotional support in my life. Thank you for always taking a good care to our beloved children.

Last but not the least, thanks to my parents, parents-in-law, my siblings and my friends who have supported me along the way.
Dedication

This thesis is dedicated to:

my lovely husband

whose affection, endless love, encouragements and prayers of days and nights

make me able to get success and honour.

my beloved son and daughter,

Izz and Indah

whom I can’t force myself to stop loving...your crying and appealing,

"mummy, please don’t go work", "I miss you, mummy", gave me a strength

and a motivation to complete my PhD study.
List of Publications


# Contents

| Declaration | iii |
| Summary | v |
| Acknowledgements | vii |
| Dedication | ix |
| List of Publications | xi |
| Contents | xii |
| List of Figures | xv |
| List of Tables | xix |

## 1 Introduction
1.1 Motivation and Problem Description ........................................ 1
1.2 Problem Formulation .................................................. 3
1.3 PhD Research Objectives .................................................. 4
    1.3.1 Methodology .................................................. 5
1.4 Organisation ............................................................ 6

## 2 Literature Review
2.1 Service Oriented Computing (SOC) ........................................ 9
    2.1.1 Orchestration and Choreography .................................. 10
2.2 Choreography Coordination ............................................. 13
    2.2.1 Specification and verification of choreography coordination 14
2.3 An OMG standard *Semantics of Business Vocabulary and Rules* (SBVR) .............................................................. 17
    2.3.1 An overview of the OMG standard SBVR ............................. 18
    2.3.2 An overview of SBVR model for service choreography .......... 24
2.4 Verifying a choreography model using SBVR ............................. 63
## 2.5 Discussion ........................................ 66

### 3 Model for Service Choreography

#### 3.1 A framework for developing choreography model ............ 71
   3.1.1 A generic SBVR model .................................. 73
   3.1.2 A specific request on the generic SBVR model ............ 85

#### 3.2 Case Studies ........................................ 99
   3.2.1 Informal constraints of AT system ...................... 100
   3.2.2 Model development for the AT system .................. 101
   3.2.3 Specification of specific request ...................... 117

#### 3.3 Concluding remarks .................................... 120

### 4 Generation and Verification Choreography Model .............. 125

#### 4.1 An overview of Alloy model ............................ 126

#### 4.2 A transformation of Terms ............................. 126

#### 4.3 A transformation of Fact Types ........................ 128

#### 4.4 A transformation of SBVR Rules ........................ 132
   4.4.1 SBVR rules - complex interaction .................. 133
   4.4.2 SBVR rules - precedence (temporal operator) ....... 149
   4.4.3 SBVR rules - static constraints ................... 159

#### 4.5 Generating the choreography in Alloy .................. 164

#### 4.6 Verifying the choreography in Alloy ................... 169
   4.6.1 Verifying - realisability ............................ 169
   4.6.2 Verifying - static constraints ...................... 190

#### 4.7 Case Studies ........................................ 192

#### 4.8 Concluding remarks .................................... 207

### 5 SBVR2Alloy: an SBVR to Alloy Compiler ..................... 211

#### 5.1 Architecture of SBVR2Alloy ........................... 211

#### 5.2 Implementation of SBVR2Alloy ........................ 212
   5.2.1 SBVR2Alloy - the transformation between the SBVR and the
         Alloy models .......................................... 213
   5.2.2 SBVR2Alloy: illustration by example ................ 218

#### 5.3 Failure Model and Testing ............................ 220
   5.3.1 Syntax checking ...................................... 220
   5.3.2 Failure model ........................................ 220
   5.3.3 Testability Matrix ................................... 224

#### 5.4 Challenges ........................................... 230

#### 5.5 Concluding remarks .................................... 232

### 6 Conclusion and Future Work ............................. 233

#### 6.1 Main Achievements ..................................... 233

#### 6.2 Summary .............................................. 234

#### 6.3 Future Work .......................................... 235
A  The SBVR Model for the Online Photo Shop case study 241
B  The SBVR model for the Tuition Fee System case study 251
C  The Alloy Model for the Online Photo Shop case study 265
D  The Alloy Model for the Tuition Fee System case study 269
Bibliography 273
List of Figures

1.1 Generating verifiable service choreography from the SBVR model framework ........................................ 6
2.1 Orchestration and Choreography ........................................ 11
2.2 Collaboration of services ........................................ 13
2.3 An example of BPMN ........................................ 17
2.4 Part of an SBVR model for the Rental Car case study ........................................ 18
2.5 Designation concept in SBVR ........................................ 19
2.6 Business rule in SBVR ........................................ 22
2.7 SBVR model for service choreographies ........................................ 25
2.8 A part of participant set and event set in SBVR model for service choreographies ........................................ 29
2.9 The local behaviour model ........................................ 31
2.10 The composition of local behaviour models ........................................ 32
2.11 Logical formulations - modal, quantification, and atomic ........................................ 36
2.12 Illustration of the ordering of messages exchanged by the common event ........................................ 53
2.13 The illustration of time notion for the messages exchanged as in Rule 1 ........................................ 60
2.14 The illustration of time notion for the messages exchanged as in Rule 2 ........................................ 60
2.15 The illustration of immediately precedes notion for the ordering of the events in Rule 1 and Rule 2 ........................................ 61
3.1 Participant terms in the SBVR model for the Acme Travel system ........................................ 104
3.2 Event terms in the SBVR model for the Acme Travel system ........................................ 105
3.3 Static constraint terms and Time term in the SBVR model for the Acme Travel system ........................................ 106
3.4 Fact types for participant set in the SBVR model for the Acme Travel system ........................................ 107
3.5 Fact types for participant set in the SBVR model for the Acme Travel system ........................................ 108
3.6 Fact types for participant set in the SBVR model for the Acme Travel system ........................................ 109
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.7</td>
<td>Fact types for event set in the SBVR model for the Acme Travel system</td>
<td>110</td>
</tr>
<tr>
<td>3.8</td>
<td>Fact types for event set in the SBVR model for the Acme Travel system</td>
<td>111</td>
</tr>
<tr>
<td>3.9</td>
<td>Fact types for event set in the SBVR model for the Acme Travel system</td>
<td>112</td>
</tr>
<tr>
<td>3.10</td>
<td>Fact types describe the ordering of times in the SBVR model for the Acme Travel system</td>
<td>113</td>
</tr>
<tr>
<td>3.11</td>
<td>Fact types describing the ordering of events in the SBVR model for the Acme Travel system</td>
<td>114</td>
</tr>
<tr>
<td>3.12</td>
<td>Fact types for specifying the static constraints particularly on date constraints in the SBVR model for the Acme Travel system</td>
<td>115</td>
</tr>
<tr>
<td>3.13</td>
<td>SBVR rules in the SBVR model for the Acme Travel system</td>
<td>118</td>
</tr>
<tr>
<td>3.14</td>
<td>SBVR rules in the SBVR model for the Acme Travel system</td>
<td>119</td>
</tr>
<tr>
<td>3.15</td>
<td>SBVR rules in the SBVR model for the Acme Travel system</td>
<td>120</td>
</tr>
<tr>
<td>3.16</td>
<td>SBVR rules particularly for date constraints in the SBVR model for the Acme Travel system</td>
<td>121</td>
</tr>
<tr>
<td>3.17</td>
<td>Specific request in the SBVR model for the Acme Travel system</td>
<td>122</td>
</tr>
<tr>
<td>4.1</td>
<td>Abstract signatures and signatures for the participant set</td>
<td>193</td>
</tr>
<tr>
<td>4.2</td>
<td>Abstract signatures and signatures for the event set</td>
<td>194</td>
</tr>
<tr>
<td>4.3</td>
<td>Time declarations in Alloy for the SBVR model of the AT system</td>
<td>199</td>
</tr>
<tr>
<td>4.4</td>
<td>A signature for date’s declarations in Alloy for the SBVR model of the AT system</td>
<td>202</td>
</tr>
<tr>
<td>4.5</td>
<td>Predicates for constraining the rules for date constraints in Alloy for the SBVR model of the AT system</td>
<td>202</td>
</tr>
<tr>
<td>4.6</td>
<td>Predicates for generating the choreography for the SBVR model of the AT system</td>
<td>203</td>
</tr>
<tr>
<td>4.7</td>
<td>Graphical representation of the generated choreography in Alloy for the SBVR model of the AT system</td>
<td>204</td>
</tr>
<tr>
<td>4.8</td>
<td>Predicates for verifying the specific request on the generated choreography of the AT system</td>
<td>205</td>
</tr>
<tr>
<td>4.9</td>
<td>One possible realisation of the user request concreterequest on the generated choreography of the AT system</td>
<td>206</td>
</tr>
<tr>
<td>4.10</td>
<td>One possible realisation of the user request concreterequest on the generated choreography of the AT system</td>
<td>206</td>
</tr>
<tr>
<td>4.11</td>
<td>Assertions for verifying the static constraints in the SBVR model for the AT system</td>
<td>207</td>
</tr>
<tr>
<td>5.1</td>
<td>UML diagram of SBVR2Alloy tool</td>
<td>212</td>
</tr>
<tr>
<td>5.2</td>
<td>Stages process for the transformation between the SBVR and Alloy model in implementing the SBVR2Alloy tool</td>
<td>214</td>
</tr>
<tr>
<td>5.3</td>
<td>SBVR model input uploaded as a text file</td>
<td>219</td>
</tr>
<tr>
<td>5.4</td>
<td>Alloy code (right pane) produced for the SBVR model (left)</td>
<td>219</td>
</tr>
<tr>
<td>5.5</td>
<td>The SBVR2Alloy template</td>
<td>225</td>
</tr>
<tr>
<td>5.6</td>
<td>Uploading a .txt file</td>
<td>226</td>
</tr>
<tr>
<td>5.7</td>
<td>Display SBVR model</td>
<td>226</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>------</td>
</tr>
<tr>
<td>5.8</td>
<td>Convert the SBVR model</td>
<td>227</td>
</tr>
<tr>
<td>5.9</td>
<td>Save as .als file</td>
<td>227</td>
</tr>
<tr>
<td>5.10</td>
<td>Open .als file in Alloy Analyzer</td>
<td>228</td>
</tr>
<tr>
<td>5.11</td>
<td>The Alloy model in Alloy Analyzer</td>
<td>228</td>
</tr>
<tr>
<td>5.12</td>
<td>Execute the Alloy model</td>
<td>228</td>
</tr>
<tr>
<td>5.13</td>
<td>An instance of Alloy model</td>
<td>229</td>
</tr>
<tr>
<td>5.14</td>
<td>Verify the Alloy model</td>
<td>229</td>
</tr>
<tr>
<td>5.15</td>
<td>An instance of the verified Alloy model</td>
<td>229</td>
</tr>
<tr>
<td>A.1</td>
<td>Terms in the SBVR model for the online photo shop</td>
<td>243</td>
</tr>
<tr>
<td>A.2</td>
<td>Fact types for specifying the participant set (the local behaviour for each participant) in the SBVR model for the online photo shop</td>
<td>244</td>
</tr>
<tr>
<td>A.3</td>
<td>Fact types for specifying the participant set (the local behaviour for each participant) in the SBVR model for the online photo shop</td>
<td>245</td>
</tr>
<tr>
<td>A.4</td>
<td>Fact types for specifying the event set in the SBVR model for the online photo shop</td>
<td>246</td>
</tr>
<tr>
<td>A.5</td>
<td>Fact types for specifying the ordering of time associates with the same event in the SBVR model for the online photo shop</td>
<td>247</td>
</tr>
<tr>
<td>A.6</td>
<td>Fact types for specifying the ordering of events in the SBVR model for the online photo shop</td>
<td>247</td>
</tr>
<tr>
<td>A.7</td>
<td>Rules for specifying the global constraints in the SBVR model for the online photo shop</td>
<td>248</td>
</tr>
<tr>
<td>A.8</td>
<td>Rules for specifying the global constraints in the SBVR model for the online photo shop</td>
<td>249</td>
</tr>
<tr>
<td>A.9</td>
<td>Rules for specifying the specific request in the SBVR model for the online photo shop</td>
<td>250</td>
</tr>
<tr>
<td>B.1</td>
<td>Terms in the SBVR model for the tuition fee system</td>
<td>254</td>
</tr>
<tr>
<td>B.2</td>
<td>Fact types for specifying the participant set (the local behaviour for each participant) in the SBVR model for the tuition fee system</td>
<td>255</td>
</tr>
<tr>
<td>B.3</td>
<td>Fact types for specifying the event set in the SBVR model for the tuition fee system</td>
<td>256</td>
</tr>
<tr>
<td>B.4</td>
<td>Fact types for specifying the event set in the SBVR model for the tuition fee system</td>
<td>257</td>
</tr>
<tr>
<td>B.5</td>
<td>Fact types for specifying the ordering of time associates with the same event in the SBVR model for the tuition fee system</td>
<td>258</td>
</tr>
<tr>
<td>B.6</td>
<td>Fact types for specifying the ordering of events in the SBVR model for the tuition fee system</td>
<td>258</td>
</tr>
<tr>
<td>B.7</td>
<td>Fact types for specifying the static constraints particularly on date constraints in the SBVR model for the tuition fee system</td>
<td>259</td>
</tr>
<tr>
<td>B.8</td>
<td>Fact types for specifying the static constraints particularly on date constraints in the SBVR model for the tuition fee system</td>
<td>260</td>
</tr>
<tr>
<td>B.9</td>
<td>SBVR rules in the SBVR model for the tuition fee system</td>
<td>261</td>
</tr>
<tr>
<td>B.10</td>
<td>SBVR rules in the SBVR model for the tuition fee system</td>
<td>262</td>
</tr>
</tbody>
</table>
B.11 SBVR rules particularly for date constraints in the SBVR model for the tuition fee system ............................................................. 263
B.12 Specific request in the SBVR model for the tuition fee system ........ 264

C.1 Abstract signatures and signatures for the participant set .............. 265
C.2 Abstract signatures and signatures for the event set ..................... 266
C.3 Abstract signature for Time and its extensions .......................... 267
C.4 Predicates for generating the choreography for the SBVR model of the online photo shop ...................................................... 267
C.5 Predicates for verifying the specific request on the generated choreography of the online photo shop ........................................ 267

D.1 Abstract signatures and signatures for the participant set .............. 269
D.2 Abstract signatures and signatures for the event set ..................... 270
D.3 Abstract signature for Time and its extensions .......................... 270
D.4 The declaration of date constraints ........................................... 271
D.5 Predicates for generating the choreography for the SBVR model of the Tuition Fee System ...................................................... 271
D.6 Predicates for verifying the specific request on the generated choreography of the Tuition Fee System ........................................ 272
List of Tables

3.1 A framework template for developing the SBVR model for service choreographies ................................................. 72
3.2 Participant terms in the SBVR model ................................. 74
3.3 Event terms in the SBVR model ......................................... 75
3.4 Static constraint terms in the SBVR model .......................... 75
3.5 Participant set and event set fact types in the SBVR model ...... 76
3.6 The messages exchanged fact types in the SBVR model ........ 77
3.7 The ordering of time and event fact types in the SBVR model ... 78
3.8 The date constraint fact types in the SBVR model .................. 79
3.9 Rules for specifying a messages exchanged by a single participant in the SBVR model ........................................... 80
3.10 Rules for specifying a messages exchanged by multiple participants (OR of participants) in the SBVR model ......................... 81
3.11 Rules for specifying a messages exchanged by multiple participants (AND of participants) in the SBVR model ................. 82
3.12 Rules for specifying a messages exchanged by multiple participants (XOR of participants) in the SBVR model ................. 83
3.13 Rules for specifying the ordering of events in the SBVR model .. 84
3.14 Rules for specifying the ordering of time intervals in the SBVR model ................................................................. 84
3.15 Rules for specifying static constraints of dates in the SBVR model ................................................................. 85
3.16 Specific request section in the SBVR model ......................... 85
3.17 An example of a specific request for the SBVR model for the Acme Travel case study ................................................. 86

5.1 Participant services and associated sets of events .................. 213
5.2 Testability Matrix .......................................................... 224
5.3 Testability Matrix .......................................................... 224
5.4 Testability Matrix .......................................................... 225
Chapter 1

Introduction

1.1 Motivation and Problem Description

Distributed applications involve a collaboration between a diverse number of autonomous participant services, such as software components and web services, on different providers. Coordinating service interactions is a vital part in developing distributed applications. Participant services interact with each other by exchanging messages through the invocations on the interfaces of participant services. This characterises the complexity of the development of distributed applications and the importance of the techniques and approaches required for designing and coordinating the interaction between participant services. These techniques will contribute to the understanding the result of coordinating service interactions. It is essential to ensure the overall goal of a collaboration between participant services is achieved.

However, the coordination of service interactions is inherently challenging. This is particularly the case when the collaboration is between loosely-coupled services (invocations on the interfaces of services) from different applications and providers. In this situation, each participant service plays their own role by contributing with their own services without any knowledge of the role of the other services, and it exploits the other services when it is required. This affects how to coordinate and assemble the participant services to capture dynamic interaction. Loose coupling becomes particularly important in the B2B context, where business entities must be able to interact independently and allow any change in their own business processes and the ways in which they interact with other business entities. This shows the importance of flexibility at the coordination level (specification). The design and coordination of service interactions should be able to be established without any effect on existing interactions and on the overall goal, by providing a coordination technique that could capture the dynamic service interactions and flexibility.

Other challenges arise for coordination of service interactions when the order in which the interactions between the underlying services take place is important as
it affects the dependencies between the participant services. This can be seen in
the online purchase scenario which involves the ordering of processes. A customer
is allowed to select the products before being required to make a payment. The
online shop then needs to give notification. However, if the coordination and design
service interaction allows the product selection again (more than once) and avoiding
the payment stage, the interaction between the participant services happens in an
uncontrolled way.

Service Oriented Computing (SOC) [1] is the principal paradigm for develop-
ing and coordinating such applications over a network. These service coordination
mechanisms are based on service orchestration and service choreography approaches.
The service orchestration approach works well when coordinating the multiple par-
ticipant services’s conversation from a specific services viewpoint, which is suitable
in a static environment. However, it is unable to capture the dynamic collaboration
from different participant services. In contrast, service choreography which focuses
on interactions between participant services from a global viewpoint encapsulates
the dynamic of service interactions by providing a looser way to coordinate service
interactions. This allows the collaboration between different participant services by
defining a set of rules that govern the ordering of exchanged messages as agreed be-
tween the interacting participants. Standardisation efforts in the web services com-
munity in this respect, have resulted in the Web Services Choreography Description
Language (WS-CDL) [2], a standardisation advocated by W3C. Choreography is
more appropriate in situations where the available participants can change at any
time as it provides a loosely coupled way to coordinate service interactions.

The difficulty in choreography design lies with the need to mediate between
conformance and realisability. On the one hand, all participating services must
respect the agreed global constraints (conformance) while on the other hand each
participant should be able to execute its part of the choreography as freely as
possible, preserving its local autonomy and replaceability of services (realisability).

To address these problems some researchers have proposed to provide a mech-
anism to let the participation from end-users (non-experts, e.g. stakeholder, busi-
ness analyst) to ensure the composition of service interactions becomes easier and
achieves the collaboration goal. To facilitate this mechanism, in terms of spec-
ification, the business requirements should be expressed declaratively in natural-
language sentences [3] to put the choreography into practice and to enable the
participation from the end-users. This perhaps explains the lack of adoption of
existing choreography languages by industry [4]. Enabling the participation of the
end-users in specification (coordinating and developing the services composition)
leads to provision of automated techniques for analysis and verifying in terms of
realisability and conformance of the choreography.
1.2 Problem Formulation

The service choreography approach [2] is concerned with coordinating the service interactions by describing the collaborations from a global perspective, effectively prescribing a contract that details the allowed interactions and the dependencies that arise, including control flow dependencies (e.g., a given service invocation must occur before another one), time constraints, and transactional dependencies. As the choreography provides a loosely coupled way to coordinate service composition, it is more appropriate in situations where the available participant services can change at any time.

A variety of modelling service interactions and approaches in service choreographies have been proposed (as described in the literature). However, most of them are focused on the graphical approach. For instance, there are choreography models providing a user-friendly graphical notation but introducing various divergent semantics, e.g. Business Process Model and Notation (BPMN) [5]. Other models use less user-friendly graphical notation, which practitioners are consequently reluctant to use [6]. Another approach to modelling choreography focuses more on the traditional imperative (or procedural approaches). However, it tends to be more restrictive and sometimes results in introducing artificial decision points for the practitioner, or over-specifying or under-specifying what actually happens in practice [6].

In comparison, work on a declarative approach to interaction-based service choreographies is limited. The focus often is on reasoning about the consistency of the rule sets and less on explicitly capturing the order of observable message exchanges or using standard notation.

Checking the consistency and verifying the reliability of a model allow problems to be detected in early stages of distributed system development. Various methods are introduced to automate model coordination. Nevertheless, most of them do not consider the interaction between autonomous participants and contract-based collaboration specifications and some of them are unable to capture the specification of the ordering of messages exchanges.

The research presented in this thesis tries to overcome the aforementioned limitations by proposing a declarative approach to modelling and generating service choreographies from informal requirements. Our proposed model is aimed to equip standard design practices of service interactions capturing and describing the complex interactions that involves the ordering of service interaction based on the given global constraints. Finally, the proposed model is targeted to enable the practitioners (business analysts, developers) to devise and set up the service choreographies that realise their collaborations by generating the automated verifiable choreography model. This allows the practitioners verify whether the developed choreography model is correct or not.
1.3 PhD Research Objectives

To achieve the aim, in order to guarantee distributed applications is behaved as expected, service choreography is targeted to be captured using specification languages which are clear to users, are expressive enough to capture the needs of complex protocols, can be parsed automatically and be analysed for the correctness.

The detailed objectives of my PhD research include:

- **To develop a new declarative approach for modelling service choreographies capturing the ordering of messages exchanged constraints and complex interactions.**
  
  To support the above objective, the following subobjectives are considered:
  
  - Determine the appropriate declarative approach towards specifying the service interactions which enables to capture the ordering of messages exchanged between different autonomous participant services, the complex interactions for handling alternative and concurrent interactions, and the specification for local behaviour and subsequently the global behaviour.
  
  - Develop a choreography model according to the standard (if available) of the approach and introduce a methodology for modelling service choreographies that encapsulates the aforementioned objective.
  
  - Derive the complete set of global behaviour for generating the choreography model.

- **To generate and verify the choreography model from the proposed model for service choreographies using constraint solvers.**
  
  To support the above objective, the following subobjectives are considered:
  
  - Recognise the syntax of the chosen constraint solver. The chosen constraint solver must be compatible with the feature of the proposed model and must be able to generate the choreography model from the correspond proposed model, and is allowed to verify the generated choreography model.
  
  - Formulate the model (based on the syntax of constraint solver) from the proposed SBVR model capturing the ordering of messages exchanged between different autonomous participant services, the complex interactions for handling alternative and concurrent interactions, and the specification for local behaviour and subsequently the global behaviours.
  
  - Generate the choreography model and verify the generated choreography model using the formulated model.
  
  - Develop a methodology of transformation between the proposed choreography model using declarative approach and the formulated model (the verifiable model) using the chosen constraint solver.
• To implement a tool that can automate the transformation between the choreography model and the verifiable model.

To support the above objective, the following subobjectives are considered:

– Determine the methodology for capturing the transformation between two models.
– Design a tool.
– Develop a tool.

1.3.1 Methodology

The framework presented in Figure 1.1 illustrates the methodology used for achieving the objectives in this PhD research.

As shown in the figure, the information from the informal constraints is used to develop the choreography model. There are several important steps taken to capture in the specification. This can be seen in details in Chapter 2.

At this stage the OMG standard SBVR model, a rule-based declarative language targeted and proposed for specifying the collaboration between autonomous participant services (services choreographies) regardless of the providers or programming model used for the implementation. As stated in [7,8], SBVR is usually used by business people because it is given in natural language to specify business objects and rules. Since the business processes or services are dynamic and heterogeneous, declarative specification languages remove the execution mechanisms by merging semantics and computation to find out the solutions. Therefore declarative language specification like SBVR approaches was taken into consideration for analysing the pattern of communication and interactions between services. In SBVR, activities can be freely executed, unless they are subject to constraints by providing a concept that manages to clearly encapsulate not only what is mandatory, but also what is forbidden. This is parallel with the view in [9] where the declarative approaches provide very effective methods for communication, interaction, and cooperation that can be used to implement choreographies and orchestrations.

The proposed SBVR model for specifying choreography is supplemented with the OMG standard Date-Time Vocabulary (DTV) to explore the possible orderings of service interactions and support the specifications for the ordering structures involving the precedence. The aim of proposing the SBVR model is to specify the complex interactions including sequential, concurrent and alternative interactions, the domain-specific constraints (static constraints) and subsequently the local behaviours of each participant service. The SBVR model, whose structure is closer to natural language formulation, is intended to provide an intuitive specification to the end-user and enable the SBVR model to be validated by the end-user.

The developed SBVR model is then be transformed according to generating formulation from the SBVR model into a chosen constraint solver model, that is Alloy model. The generated Alloy model is executed in a well known constraint
solver, namely *Alloy Analyzer* which is a first-order, declarative SAT based solver. The notion of relation in Alloy is closely related to SBVR structure. In general, expressions in Alloy are built using set theoretic relational operators and constants. This means it is an appropriate target for SBVR models which are declarative in nature.

Alloy is chosen since it is emphasised more on generating an automatic analysis and producing an instance structure of model. The analysis in Alloy is more powerful than the other approaches’ analyses since Alloy Analyzer does not require the test cases which need a initial conditions and inputs by an user, also it does not restrict the language to an executable subset, and it is more effective at uncovering subtle bugs.

In this research, *Alloy Analyzer* generates the choreography model from the corresponding SBVR model and perform verification on the generated choreography. Finally, an automated tool, the *SBVR2Alloy* translates the SBVR model into Alloy.

### 1.4 Organisation

This PhD research thesis is organised as follows:

*Chapter 2* gives an overview of *Service Oriented Computing* (SOC) and subsequently the *Service Choreography approach*. In addition, a review of the work done for choreograph modelling, transforming SBVR and analysing using Alloy, on its usefulness to this PhD research, are presented. Also, an overview of the OMG standard SBVR, an overview of the proposed SBVR model for
service choreographies, and *Alloy Analyzer*, are provided. The overview of the proposed SBVR model begins with a details description for developing the SBVR model includes the designation of terms, the construction of fact types and subsequently the development of the business rules in the SBVR model. The construction of business rules follow the semantic formulations that are used in SBVR standard is discussed afterwards.

**Chapter 3** presents the methodology for the development of the SBVR model rule-based language for service choreographies. A framework for developing the SBVR model as well as to specify the specific request on the developed SBVR model are introduced. The SBVR model is then illustrated by considering three case studies, specifically on the ACME Travel system case study.

**Chapter 4** provides the formulation and the methodology for transforming between the SBVR model and the Alloy model. The transformation is presented with respect to the corresponding components in the SBVR model, commencing with the transformation of terms, fact types and finally SBVR rules in the proposed SBVR model into Alloy. This Alloy is equipped with automatically generating all possible executions of the required service interactions with respect to the corresponding SBVR model. A details elaboration on how to produce the Alloy codes with respect to the methodology described in Chapter 3, for generating choreography and verifying on the generated choreography, are represented. To this end, the identical case study presented in Chapter 3 is taken to illustrate the implementation of formulating and transforming the models.

**Chapter 5** presents how the proposed transformation between the SBVR and Alloy models can be put into practice. The *SBVR2Alloy* tool for supporting the aforementioned transformation have been developed. The architecture and stages involved in developing this tool have been discussed. More specifically, there are details explanation on how the tool implements the translation based on a state machine paradigm. Finally, some challenges addressed during the implementation discussed.

**Chapter 6** summarises the results of this research and discusses possible future extensions and directions for further research work.
Chapter 2

Literature Review

In this Chapter, an overview of application domain in which interaction plays a central role: Service Oriented Computing (SOC), is provided. The topic is narrowed down further to service choreography as the main approach to design service composition. Next, the open challenges when dealing with coordination and service choreography are discussed. This is followed by a discussion on the advantage of a declarative versus a procedural approach for service coordination, motivating why the declarative approach should be supplemented in coordinating service composition. In addition to that, the other issues relating to specification and verification are presented. A literature of the work done so far for choreography modelling, transforming SBVR and analysing using Alloy, are presented.

2.1 Service Oriented Computing (SOC)

Service Oriented Computing (SOC) is a well-known computing paradigm towards developing distributed applications. According to Papazoglou et al. [1,10], SOC is a realm which provides an idea to develop distributed applications by employing services and coordinating them as networks of autonomous services in heterogeneous environments. The network of loosely coupled services provides a flexible and a dynamic composition of business processes of a system. The autonomous services which could be web services, online resources, software-as-a-service expose their own functionalities and exploit other services over a network in the form of contracts, protocols, rules or interface, producing a multi-tiered architecture. The contracts or rules are represented by means of standardised languages of specifications which can then be utilised by (e.g. business analysts, system analysts, etc) without a direct access to its concrete implementation [11].

Service Oriented Architecture (SOA) is a way of designing software applications to support service interactions distributed in a network through standard interfaces
and messaging protocols, for the construction of service model [1,12]. SOC has service layers: service management, service composition, and basic services where they act as an extended SOA. The service composition layer is most relevant to this research, with roles including coordination, monitoring, and conformance. The layer is responsible for:

- controlling the interaction between services, data flow within services and to the service output;
- arranging the events or information produced by the services and publish higher-level composite events;
- ensuring the integrity and the implementation of business rules of the composite services respectively.

In other words, service composition has an important role in shaping the SOC which assists the design, development, validation and execution of service-oriented applications. The composition of service applications is aimed at accomplishing some tasks by discovering and invoking interface through a network. Web services are the present technology based on the idea of SOC. It provides a platform for business processes to be implemented and distributed over a network of collaborating applications through a stack of Internet standards [10]. The stack of internet standards includes:

- HTTP for transmitting the messages;
- XML for representing the messages;
- Simple Object Access Protocol (SOAP) [13] as a transmission medium;
- Web Services Description Language (WSDL) [14] as a standard language for service description;
- Universal Description, Discovery and Integration (UDDI) [15] as a directory service that contains service publications and enables web service clients to locate candidate services and find their details.

### 2.1.1 Orchestration and Choreography

The composition of services involves business process regulations which apply to collaborating distributed business applications. Each service can be a human user or another service provider, and each of them can mutually benefit from each other by exposing their own functionalities and exploiting other services. Two different service composition approaches [16] are: service orchestration and service choreography.
2.1. SERVICE ORIENTED COMPUTING (SOC)

Service orchestration

Service orchestration refers to the collaboration from a single service’s perspective. Under this perspective, the single service composes an execution of business processes and coordinates the other services. This single service can interact with both internal and external services acting as a centralised controller, i.e a central coordinator. Because a central coordinator is used to coordinate the collaborations, it typically results in tightly coupled services. The standard specification language to compose service orchestration is Web Service Business Process Execution Language (WS-BPEL) [17] which uses several XML specifications, for instance Web Service Description Language (WSDL) [14] and others. It is responsible for describing the behaviour of the local business processes and the messages exchanged based on the collaboration (interaction) of the central coordinator with each participant service through web service interfaces.

Example (Service orchestration): There is an e-travel portal providing a Business-To-Consumer (B2C) web portal for creating an itinerary including making a reservation for hotel, airline, etc. The customer (user) is able to use the service from the e-travel portal by creating the itinerary, browsing, searching and selecting the prefered bookings, and asking for a quote. After getting a quote, the user needs to pay the booking or cancel the overall request. The perception of the user is of a single e-travel service. The e-travel acts as the central coordinator which is internally implemented as a workflow for coordinating with the internal services (web services) such as a smart cart service to place all bookings which are made by the user and a certified e-banking service for handling payment transactions in a secure manner.

As shown in Figure 2.1, let us say Participant Service 1 is the e-travel portal responsible for the orchestration of two internal services: the smart cart service and the e-banking service, and guides the interaction as a centralized controller. The orchestration is based on the steps contained in the workflow of the executable business processes.
Service choreography

Service choreography models the collaboration across the different participating services from a global perspective and allows each involved party to describe its part in the interaction [2, 10, 18]. Instead of describing the executable business processes, service choreography describes the collaboration capturing the ordering of public message exchanges between multiple participating services. The collaboration follows an agreed contract of a global set of rules which contains the messages exchanged, the control flow dependencies (e.g., a given service invocation must occur before another one), time constraints, to achieve the overall choreography objective. This contrasts with the service orchestration approach which focuses on the viewpoint of a single party. As can be seen in Figure 2.1, service choreography provides a looser way to compose the participating services by specifying each participant service interacting with the external service (another participant service) and message protocols between them.

This decentralised approach of choreography complements the statement made in the Web Services Choreography Description Language (WS-CDL) [2], a standard specification language for service choreography. WS-CDL which is developed by W3C stated that in real scenarios business, there are no organisations eager to lead and control the business collaboration. This facilitates the Business-To-Business (B2B) context where each organisation benefits from each other by sharing own service in order to achieve the collaboration goal [19, 20]. It is impossible for one stakeholder to determine how to reach the goal of collaboration with other services alone.

Example (Service choreography): Three participant services: e-shop online, customer, and bank, collaborate to facilitate online shopping. The choreography is triggered when the customer enters the e-shop online. The e-shop online needs a service from the customer before it can offer any products to be selected by the customer. The customer requires to register an account (check-in) at the e-shop. Once the customer has registered at the e-shop, she has an option either shop the products or check-out (exit) from the e-shop. If the customer chooses to shop, the e-shop offers all products. Once the products have been selected, the customer needs to check-out before the payment request is made to the e-shop. Right after the e-shop gives a response, the customer deploys a service from a bank to make a payment before the notification is given by the bank and the e-shop concurrently.

Figure 2.2 illustrates the global overview of service collaboration which shows that all the interacting services are equally responsible for carrying out the collaboration and there is no central coordinator. The choreography of this interaction must comply to the following global constraints which can be referred as a contract of choreography:

- the customer registers an account at the e-shop once before deciding to shop the products or to exit from the e-shop (exclusive-choice);
- the e-shop provides the products that can be selected by the customer;
2.2 Choreography Coordination

Modern distributed application results from coordination and composition of numerous distinct numbers of software services and are likely to increase in the future [21]. That distributed applications comprise of multiple components of services executing on multiple machines and interacting with each other by exchanging messages over the network. These system complexities will not surprisingly contribute to many complications of composition. However, Tevfik Bultan in [22] stressed that in order to face this issue there is a strong need for a good approach and techniques to establish the coordination in developing distributed applications. The complex interaction which encompasses diverse consumers and providers is necessarily be coordinated dynamically and be flexible. This is for allowing flexibility such as replacing of a service and changing the behaviour for a service [11, 21].

To address the above issues, the choreography approach is aimed to be applied for coordinating the interaction of multiple services in distributed applications. As mentioned in the previous section, service choreography approach describing the interaction across the different participating services effectively prescribing a contract that allowed interactions and the dependencies that arise, including control flow dependencies (e.g., a given service invocation must occur before another one). This enables the choreography coordinates and composes the distinct services dynamically. Also, as the choreography provides a loosely coupled way to coordinate service
composition, the choreography is suitable in situation where the available participating service can change at any time [21]. This impossible to be composed and coordinated using service orchestration approach as the composition is implemented in static environment and only minimal changing is allowed.

However, the service choreography approach faces some challenges in describing and verifying the interaction between distinct services towards producing an established choreography model that complies with the global constraints of the collaboration between the involved participating services.

2.2.1 Specification and verification of choreography coordination

As pointed out in [22] by Tevfik Bultan one of the important directions of research in current software development considers the best concept for specifying service composition, i.e., coordinating and composing choreography. This concept is targeted to contribute to and to improve the ability of the resulting distributed applications in delivering services as per specified and requested without any catastrophic failure when operating the applications (service interactions).

To address this issue, Tevfik Bultan [22] stated that the higher-level technique to specify and compose a complex interaction between participating services and support formal guarantees be required to develop. In addition to this, the technique should allow the service composition applicable to the end users to make the development of distributed applications easier and at the same time reducing an uncontrolled and an unintended interactions.

This has been previously emphasised by Montali et. al in [19], there are still some challenges in modeling service choreography representing the service interaction from a global viewpoint. Montali et. al stressed that it is important in the specification to permit the required behaviours and to prohibit the unwanted behaviours for avoiding specifying the unnecessary constraints.

However, the difficulty in choreography design lies with the need to differentiate between conformance and realizability [23]. On the one hand, all participating services must respect the agreed global constraints while on the other hand each participant service should be able to execute its part of the choreography as freely as possible, preserving its local autonomy and replaceability of services.

To facilitate these, an automated technique that could capture such as analysis and verification of composing service interactions is important [22].

The same concern also has been stressed in [21]. They claimed that in coordinating and prescribing the choreography specification and to ensure the interactions fulfill the specified global constraints, the automated mechanism in the specification towards realising service choreographies, by letting the participation of the end users, must be considered.
Declarative approach versus imperative approach

Both [22] and [21] pave the way for designing and specifying the service interactions in a declarative approach. The declarative approach focuses on what is the essential characteristics that need to be done rather than how to do it to accomplish the goal [8, 24–26]. Literally, the declarative language describes the particular information implicitly pertains to the overall conditions that produced the outcome and involves the logic to be used for solving problem but not necessarily the control [25]. This contributes to make any changing and modifying the business processes and the service interactions easy, to overcome the specifying an undesired interactions issue, to offer a clear understanding on a modelling process allowing the participation of end-users (non-experts) in the modelling process [25, 27, 28].

An imperative approach focuses on how - the possible steps taken to achieve the solution [24, 25]. The specification of interaction and business processes involve control (e.g., problem-solving strategies on how it is to be computed) and message flow. This means the imperative approach describe the information explicitly on how input conditions lead to a certain outcome sequentially, by considering all possible executions and adding possible alternative executions to the model [25, 28]. This mechanism may lead to produce an over-specified models and deliver larger and more complex process models [27].

Since the ability to understand the domain of process modelling is of importance and there is not only designers reading process models but stakeholders too, thus the declarative languages are worth considering [28] and deserve to be investigated in modelling service choreographies. According to [29], from the point of view of business people, the business rules represent core business logic and the declarative nature of whole sets of related rules. In the real world business, the implementer regardless of whether they are programmer or technical rule writer, they are left to make the decisions on the real meaning of the business rules, and the accuracy of the interpretation can only be determined at testing. Moreover, since there is no standard theory for a declarative model of business rules, the process model and business rule implementation is usually based on the opinion and preference of an analyst. Hence, it differs from project to project, analyst to analyst, process to process.

[9] claimed that the main cause why most of the model has been successfully developed using declarative approaches, is because it is suitable to manage the complexity of the service composition.

To facilitate the review of a specification, the business requirements should be expressed declaratively in natural-language sentences [3]. This perhaps explains the lack of adoption of existing choreography languages by industry [21].

Specification languages used by practitioners

Current proposals in modelling service choreographies such as WS-CDL [2] introduce the mechanism of message control-flow in the form of XML data format defi-
nition for specifying service interactions which causes the modeler to apply the imperative style of modelling. WS-CDL fails to tackle and come up with the method to verify conformance to choreography specification [19]. According to [30], in order to capture model verification and validation WS-CDL must be based on or related to a formal language. However, the connection to this or any other formalism is currently not comprehensive and not clearly established. Also, WS-CDL provides a lack of graphical notation which does not support user convenience in specifying and setting-up the interactions, and verifying the model, contributing a little interests to the practitioners [30].

The Business Process Model and Notation (BPMN) [5] became an ISO standard and choreography diagrams were included in its most recent specification. It provides a user-friendly graphical notations yet exhibits various different notations and focuses on the control-flow approach in defining the business processes. However, BPMN would produce errors when expressing the interactions which need a more complex service choreography, in the context of orderings of messages, of handling concurrency and may lead to deadlock.

Figure 2.3 is an example of a collaboration between participant services in BPMN. This BPMN refers to the global constraints in the given example for service choreographies previously (Figure 2.2). As shown in the figure, the interactions begin with a registration from the customer before making a selection either stays in the e-shop online by executing a selection product activity or exits the e-shop online by executing a check-out activity. For this case, the exclusive branches ("X") with two outgoing arrows are required. Once the product selection have been made by the customer, the activity payment request must be done by the customer to the e-shop. The customer makes a payment to a bank afterwards. The payment notification activity is given by both the e-shop and the bank concurrently. This is illustrated using the parallel and merging branches with two incoming and two outgoing arrows ("+").

However, this choreography might produce an undesired interactions. It can be seen when the customer completes the selection of products, the customer has to check-out to allow her for making a payment. However, Figure 2.3 shows the customer is allowed to select the product again again. This scenario illustrates the choreography becomes over-specified, affects the ordering of interaction and leads to deadlock resulting from the imperative style of modelling.

Another a well-known standard specification language for modelling service interactions is Unified Modeling Language (UML) [31]. It is a multi-purpose modelling language that can be applied to different applications areas such as in business, health, etc. Also, it provides a user-friendly graphical notations but as similar as BPMN it exhibits various divergent semantics leading to models that would be difficult to interpret [25]. Specifically, UML collaboration diagrams [32] and UML 2 sequence diagrams [33], [34], [35] have been used for specifying service choreographies, but drawing the diagrams typically requires integration with a UML tool.

Work on formal semantics in this area has focused more on the imperative (or
2.3 An OMG standard Semantics of Business Vocabulary and Rules (SBVR)

A promising approach to expressing complex business requirements in a declarative manner is the OMG standard SBVR [7]. As stated in [7,8], SBVR is usually used by business people because it is given in natural language to specify business objects.
CHAPTER 2. LITERATURE REVIEW

and rules. As defined in [39], "SBVR provides a way to capture specifications in natural language and represent them in formal logic so they can be machine processed".

2.3.1 An overview of the OMG standard SBVR

The OMG standard SBVR [7] provides means for specifying business vocabularies and rules for business purposes of all different types of business activities. The specification defines the business vocabularies and rules declaratively, unambiguously and expresses the structure of the meaning of business concepts and rules in the natural language for the use of business purposes by business people. The OMG standard SBVR is composed by a set of rules, formed by Terms and Fact Types that follows the doctrine: "Rules build on facts, and facts build on concepts as expressed by terms. Terms express business concepts; facts make assertions about these concepts; rules constrain and support these facts" [7].

SBVR is a meta-model with models natively expressed as logical formulations. As argued in [40] and elsewhere, its most common serialization is SBVR Structured English (SBVR - SE) [7]. The Structured English provides a standardized representation to formalize the syntax of natural language representation. An example of (part of) an SBVR model can be seen in Figure 2.4. It refers to the Rental Car case study included in its OMG document [7].

![Figure 2.4: Part of an SBVR model for the Rental Car case study](http://sbvr.co)

The rule in Figure 2.4 is written using our web-based SBVR editor [41] maintained by Rulemotion\(^1\) and is a representation of higher-level facts that use the deontic constraint, obligatory on the constraint defined by the rule. The quantifications, each and at least one show the restriction of rental car’s ownership. Furthermore, is owned by is the designation for the Fact Type. Fact Type is constructed based on identified Terms (a noun concept, such as rental car and branch). Thus, the combination of deontic constraint, quantification (cardinality), Terms and Fact Types will yield a constructive rule (this type of rule can be used by domain specialists (e.g., business analysts) in defining the business model or activity to be performed with a choreography).

In the next sections, the details of all ground facts involved in building a set of business concepts and rules as defined by OMG’s standard SBVR are provided. It begins with a description of concepts which include Term and Fact Type (Verb

\(^1\)With thanks to Rulemotion, the editor SBVR Lab 2.0 is available at http://sbvr.co
2.3. AN OMG STANDARD SEMANTICS OF BUSINESS VOCABULARY AND RULES (SBVR)

A concept (word) in SBVR refers to a business concept that may be used to construct a rule later. According to Figure 2.5, a concept encompasses a noun concept and a verb concept.

**Noun Concept**

In [7], the noun concept tends to be used to refer to the meaning of a noun or noun phrase. It is either a general concept, or an individual concept, which are defined as follows:

- A **general concept** is ”a noun concept that classifies things (anything perceivable or conceivable) on the basis of their common properties”. For example, **car, customer, hotel**, etc.

- An **individual concept** is ”a noun concept that corresponds to at most one thing in all possible worlds”. In other words it is unique. For example, **Kendall cars Guildford Head Office** is an individual concept whose one and only instance of an individual branch in Guildford.
Figure 2.5 shows that thing, which is an instance of concept, is represented by term and name. Term is applied as the designation for a general concept. It is defined using lower case letters (a proper noun is excepted) and explicitly in singular form. While name is a proper noun, which is a designation of an individual concept.

Notice that the general concept has a categorisation scheme of that general concept. For example, the general concept person can be categorised by gender. So categories of person are man and woman. This categorisation scheme is used later in SBVR model for service choreography to categorise the term in representing the interaction between participant services.

**Verb Concept**

Another concept as depicted in Figure 2.5 is a verb concept. The verb concept is a fact, which is a proposition taken to be true by the business. In this thesis, a verb concept is referred to a Fact Type (FT) (a set of possible ground facts) which can be formulated based on verb symbols that make rules relevant to the business [7]. A Verb symbol is used to represent FT, which is demonstrated by a verb concept wording. The verb concept wording describes a meaning of FT by relations between terms in the form term verb symbol (verb) term”. Each FT may consist of at least one, or exactly two (binary), or more terms.

For example, the binary FT, that is extracted from our model for Online Travel System - Acme Travel case study (a full case study can be seen in Chapter 3), 'customer sends reservation request’ shows customer as a term to represent customer role (the participant who plays a role as one of participant service entities involved in the business collaboration for Online Travel System), which specifically characterises another term, reservation request that plays a role as the occurrence of events performed by the participant. Whereas sends represents the verb denoting the FT of the factual relationship between the two terms.

Besides representing a general concept, term that is in a FT also is ”a noun concept that corresponds to things based on their playing a part, assuming a function or being used in some situation”. For example, the term reservation request of the above FT has a role that ranges over the general concept reservation which corresponds to reservation (booking) that are requested.

In addition, the verb concept wording could act as a noun rather than a FT. For example, the FT that is extracted from Acme Travel case study, 'airline reservation has outbound date’ can be designated as a noun, 'outbound date of airline reservation’. This is useful in describing a static constraint in SBVR model for service choreography later.

Furthermore, a set is defined as a group of zero or more things regardless its order and repetition. There are three type of FT structures that can be advocated to represent a set in the following form:

Fact Type: thing is in set
Synonymous Form: \textit{set includes thing}

Synonymous Form: \textit{set has element}

The representation of FT using the \textit{set} definition is vital to denote a \textit{term} set which contains a number of types. For example, \textbf{car} is a \textit{‘car’} set that consists of many different \textit{car’} model, such as \textbf{Honda}, \textbf{BMW}, and \textbf{Porsche}. Hence, to specify those different car models in a \textbf{car} set, the \textit{set} definition as above is applied. We will have the FT:

\textbf{Fact Type: car includes Honda}
\textbf{Fact Type: car includes BMW}
\textbf{Fact Type: car includes Porsche}

In the Acme Travel case study of Online Travel System, \textbf{accommodation} set contains distinct types of accommodation as an option for a customer in considering a request for accommodation. The following FT describes the \textbf{accommodation} set:

\textbf{Fact type: accommodation includes hotel}
\textbf{Fact type: accommodation includes apartment}
\textbf{Fact type: accommodation includes hostel}

The FT describes that an \textbf{accommodation} set has several choices of accommodation for a customer. It includes \textbf{hotel}, \textbf{apartment}, and \textbf{hostel}.

\section*{Business Rules}

A rule is often to be understood as a restriction of something and a reduction in the degree of freedom [7]. In the OMG standard SBVR, a rule is classified as a business rule, which is defined as a rule that is under the authority of an official organisation(s). The organisation who responsible and able to decide on amending and removing the rule at its own discretion. The business rule is dissimilar to laws of physics, or legislation and regulation since they might be imposed on a company. Based on the company’s perspective those mentioned are not the business rules as the company is not in the position to change them.

Figure 2.6 depicts the type of business rules in SBVR. The SBVR standard defines two distinct business rules; structural business rules and operative (behavioural) business rules. The structural rule (i.e. alethic) specifies about the organisations’ understanding of concepts on how the organisation takes things to be [7], [8]. It can be expressed in the form of \textit{necessity, impossibility,} or \textit{restricted possibilities} statements. For instance, a structural rule from [7], "It is necessary that the pick-up branch of a one-way rental is not the return branch of that rental", is a rule on the understanding concept of the rental car organisation as agreed by the members of the community (a branch).

Nonetheless, the operative (behavioural) business rules (i.e. deontic) expresses
and describes the business process of the organisation(s). It concerns on the behaviour of organisation(s) which is directly controlled, influenced, or regulated by the element of guidance. The element of guidance is detailed of behaviour that organisation(s)’ own. That behaviour can be applied effectively and consistently by the organisation(s). Also, it can be put into practice successfully and useful to the organisation(s) (people in that organisation(s)) by applying the element of guidance [7].

The operative behavioural rule can be expressed using either obligation, prohibition, or restricted permission modalities. For example, the operative behavioural rule from [7], “It is prohibited that the duration of a rental be more than 90 rental days” describes the guidance that controlling the business process or behaviour of the rental car organisation. Another example, in Figure 2.4, the obligation statement is used to restrict each rental car to be owned by at least one branch.

Semantic Formulation

Semantic formulations are a means that are used in SBVR standard to structure the meaning of rules. Without the semantic formulation, both type of business rules as stated before, either structural or operative (behaviour) business rules will not be specified meaningfully. The semantic formulations describe the formal semantic structures of the meaning of rules that underlying business discourses of concepts, propositions and questions. Semantic formulations are categorised into two:

- logical formulation and proposition - the combination of logical formulation and proposition are specialised to formulate the rules involving modal formulation, logical operations, quantifications, atomic formulations based on FT and other formulations;
2.3. AN OMG STANDARD SEMANTICS OF BUSINESS VOCABULARY AND RULES (SBVR)

- projecting formulation - this is used to formulate the rules involving definitions, aggregations, and questions.

The SBVR rules describe the meaning by composing the logical formulation [7] which is a semantic formulation that formulates a proposition.

The semantic or logical formulations and propositions that are used in SBVR rule to structure the meaning of rules are defined below. A below n and m represent a whole number, and p and q represent expressions of propositions:

- **Atomic Formulation** - Is based on a FT, where the FT binds to each *term* (noun concept) of FT. For example, the extracted FT from SBVR model for Rental Car case study in [7]: EU-Rent purchases from General Motors Company. This FT is formulated by an atomic formulation, which is based on the generic FT company purchases from vendor. The first role binding is of the role 'company', is the first term of generic FT. This first role binding binds to the term 'EU-Rent' of FT. The atomic formulation also has a second role binding is of the role of the second term 'vendor' of the FT. This second role binding binds to the term 'General Motors Company' of FT.

- **Modal Formulation** - The modal formulations include obligation formulation, permissibility formulation and possibility formulation. This formulation is vital to formulate those two type of business rules; structural (alethic) business rules by exploiting the modal operations: it is possible that p; it is impossible that p (possibility formulation embedding a logical negation), or operative (deontic) business rules by using the modal operations: it is obligatory that p; it is prohibited that p (obligation formulation embedding a logical negation); and it is permission that p. These modal formulations formulate the meaning by embedding them in another logical formulation. For example, refer to Figure 2.4, that rule is formulated by an obligation formulation that embed the atomic formulation 'rental car is owned by branch'.

- **Logical Operations** - The logical operations include conjunction, disjunction (inclusive disjunction), and exclusive disjunctions. These logical operations are only advocated in SBVR model for service choreography. The logical operations based on only the truth or falseness of the meanings of its logical operands, p proposition, or q proposition. Each logical operation has at least one logical operand.

- **Quantification formulation** - The quantification encompasses each universal quantification, some existential quantification, at least one existential quantification, at most one quantification, and exactly one quantification. Only these quantifications will be considered in SBVR model for service choreography. Each quantification formulates the meaning by introducing exactly
one variable that ranges over the concept, *term* of FT and scopes over at most one logical formulation.

- **Objectification** - An objectification formulates the meaning by considering a proposition (FT) that binds to a bindable target, a state of affairs. A state of affairs is defined as any event, situation, activity, or circumstances which corresponds to any representation of proposition statement. For example, the FT, *company reviews account at place*, is constructed from a combination of two binary FT. There are *company reviews account* and *state of affairs at place*. So, the proposition statement, *company reviews account* binds to a bindable target, *state of affairs*.

The constructive rule will be yielded once the rule uses the semantic formulation or the composition of the logical formulation to structure the meaning of rule.

### 2.3.2 An overview of SBVR model for service choreography

The SBVR model for service choreographies which advocates the OMG standard SBVR (as described in the previous section) [7], describes the collaboration between different participating services which refer to stand-alone services from different business applications on the web. Collaboration here means the messages exchanged across different participating services (services interaction) by the invocations on the interfaces of services e.g. web services, as agreed between the interacting participants.

Since the proposed SBVR model is based on the service choreography approach, the SBVR model describes the service interaction from a global perspective. The SBVR model prescribes a set of global constraints (global behaviours) from informal requirements, namely an agreed contract of common rules that govern the allowed interactions and the ordering of services interaction. In the proposed SBVR model, each participating service has its own local behaviour that prescribes how it contributes to their own specific task for the overall business collaborative goal while adhering to a set of global constraints e.g. due to processes/rules/regulations/policies.

The proposed SBVR model for service choreography that is built on the OMG standard SBVR, is rule-based. So, a set of global constraints (global behaviours) refers to a set of business rules that describes service interactions. In the OMG standard SBVR, the rules consist of *Terms* and *Fact Types*. Therefore, to model service choreographies, a set of business rules are built on a set of *terms* and a set of *fact types*. The structure of business rules in the proposed SBVR model is based on the semantic formulations that are used in SBVR standard.

Figure 5.7 represents a picture of the SBVR model for service choreographies. In the proposed SBVR model, two main components play an important role in modelling the choreographies. They are the participating services and the messages exchanged between them, where they are called *participants* and *events*, respectively. These will be represented as *Terms* in the Vocabulary of the SBVR model. Since
2.3. AN OMG STANDARD SEMANTICS OF BUSINESS VOCABULARY AND RULES (SBVR)

the participants and the events that the participants perform are both diverse, they are denoted as a participant set and an event set in the SBVR model. The participant set as shown in the figure, represents multiple terms in the SBVR model to denote the participants. Similarly, the event set denotes several different terms that indicate the event, while the static constraints terms designate all the terms that are used to specify the domain-specific constraints (if any) for each participant and for any events.

In the SBVR standard, Fact Types are built over Terms, which are interconnected by verbs. The same approach is applied for constructing Fact Types in the SBVR model. The Fact Types in the Vocabulary of the SBVR model for service choreographies will be presented as “term verb term”, as in the SBVR standard. term is used to represent a participant, an event, or a static constraint. The fact types in the SBVR model are grouped into specific roles:

the participant and event set These fact types specify a participant set and an event set, which are involved in the services interaction. The fact types also specify a nesting of participants and events (if any). The fact types are constructed based on the specific definition defined in SBVR standard.

the messages exchanged and static (domain-specific) constraints for each participant These fact types describe the messages exchanged (the sending and the receiving of a message) between different participants and the static (domain-specific) constraints,

the static constraints for any events These fact types represent the static constraints for any required events.
The construction of each fact type is discussed in the next section. Subsequently, the business rules in the SBVR model are developed based on the terms and the fact types, defined in the SBVR model. The construction of business rules follow the semantic formulations that are used in SBVR standard. These business rules capture the specification for services interactions such as the complex interactions in services coordination, that are concurrent and alternatives interaction; and the ordering of services interaction. A set of business rules in the SBVR model are then characterises the global behaviours with respect to the overall business collaborative goal between different online business applications (interacting participants).

**Terms for participants, events, and static constraints**

The development of a SBVR model for service choreographies starts with the designation of terms to capture the participants, the events, as well as the static constraints involved in the services interaction. As discussed previously, the terms are the vocabulary of the SBVR model. In Section 2.3.1, it is stated that the rules in OMG standard SBVR apply term and specific name to designate a noun concept. Similarly, terms in the SBVR model for choreography are used to represent a noun, as in the following:

- **participant** - represents participating service entities involved in the services interaction;
- **event** - characterises the occurrence of event that is messages exchanged performed by the participant(s).
- **static constraint** - denotes the terms that are used to specify the domain-specific constraints for each participant and event (if any).

The SBVR model for service choreographies consists of multiple distinct participants, events, and static constraints. This means that each represent multiple terms. To specify all terms of participant (participant set) and terms of event (event set), categorisation schemes (as explained in Section 2.3.1) advocated in OMG standard SBVR [7] employed. The participant is classified into multiple different participants, such as participant\_1, participant\_2 until n number of participants, participant\_n; and likewise the event comprises event\_1, event\_2 up to n number of events, event\_n. On the other hand, static constraints terms can be any term, which later can be associated with any participant and event as fact types to represent static constraints.

For example, the participant set in the SBVR model, which is under the Term: participant category can be denoted as Term: customer, Term: airline. Similar to the event set, all different terms under the Term: event category can be specified as Term: reservation request, Term: airline reservation. The static constraints terms (if any) designate all terms to specify the domain-specific constraints for each participant and for any events. For instance, Term: name is used
to show that the customer has the name, and Term: outbound date is used to show that the event airline reservation, which is requested by the customer has the outbound date.

**Fact Types for specifying participant set, event set, and local model**

As discussed previously, to develop an SBVR model for choreographies, fact types are employed to specify (i) the participant and event sets, which are involved in the service interactions, (ii) the messages exchanged and the static constraints for each participant, and (iii) the static constraint for any events.

- **Fact types for specifying participant and event set**
  
  After all the terms for participants and events are designated, the participant set and the event set are specified. In order to preserve the semantics, the specific definition, namely Sets definition found in the latest SBVR specification document [7] is adopted (detailed see Section 2.3.1).

  The Sets definition is defined in the fact types as "set includes thing". Hence, participant set that contains participant\(_1\), participant\(_2\) until \(n\) number of participants, participant\(_n\) for service choreography is specified in the SBVR model in the following form:

  - participant includes participant\(_1\);
  - participant includes participant\(_2\);
  - ........;
  - participant includes participant\(_n\).

  Similarly, the same structure is applied to specify event set which consists of event\(_1\), event\(_2\) up to \(n\) number of events, event\(_n\):

  - event includes event\(_1\);
  - event includes event\(_2\);
  - ........;
  - event includes event\(_n\).

  It is often the case that in some circumstances, the participants and events in a business activity are grouped together. For instance, the participant might be the university students, where they could be grouped as either an undergraduate student or a postgraduate student. Similarly, the event might be the reservation response (a response for any travel reservation that has been made) is either a successful reservation or an unsuccessful reservation. In order to specify this nesting of participant and event, the Sets definition is used as well.
Assume that the participant\(_1\) (event\(_1\)) has its own group. Which own group describes several distinct participants (events) that involved in the interactions between the participants. For example, participant\(_1\) comprises participant\(_a\) and participant\(_b\). The same approach applies to specify a group of events. For instance, the event\(_1\) be composed of event\(_a\) and event\(_b\). In general, the specification for nesting of participant\(_1\) and event\(_1\) in the SBVR model is as follows:

- A nesting (grouping) of participant\(_1\):
  *
  - participant\(_1\) includes participant\(_a\)
  - participant\(_1\) includes participant\(_b\)

- A nesting of event\(_1\):
  *
  - event\(_1\) includes event\(_a\)
  - event\(_1\) includes event\(_b\)

As an example, a part of SBVR model for one of our case studies, Acme Travel is illustrated in Figure 2.8. It is written using our web-based SBVR editor [41]. It illustrates the designating of terms and specifying of fact types that relate with the specification of participant set, event set and their nesting.

Terms in the figure designate all the interacting participants and the occurrence of events are involved in the collaboration of Online Travel System for Acme Travel. Subsequent to terms, there are fact types which show participant set by using set definition: participant includes customer; participant includes accommodation. Therefore, it can be understood that the participants that collaborate in Acme Travel Online Travel System consists of customer and accommodation. However, the accommodation has been classified into two groups of accommodation that includes hotel and apartment. The accommodation nesting is specified by applying the set definition as described previously. The same method is employed to specify event set which consists of reservation request and accommodation reservation, and a group of accommodation reservation which includes hotel reservation and apartment reservation.

- Fact types for specifying messages exchanged and static constraints for each participant

The previous fact types are constructed based on the particular definition in the SBVR standard. Conversely, the fact types in the form of “term\(_1\) verb term\(_2\)” are adapted to capture the sending and the receiving of a message. For modelling choreographies in the SBVR model, we assume each participant provides a service (performing sending or receiving a message) in the multi-party conversation, is associated with any events in the event set. Based
on that assumption, \( \text{term}_1 \) specifies the participant and \( \text{term}_2 \) specifies the event, which are interconnected by the verb. The verb can be any verb to demonstrate the message that is performed by the given participant is either the sending or the receiving of the messages. Therefore, the generic fact type for specifying the messages exchanged is in the form of "\( \text{participant} \ \text{verb} \ \text{event} \)". We refer to fact types of a given participant that involve sent events as export messages and those involving received events as import messages. For example, in our case study on Online Travel System (Acme Travel (AT) case study), there is a fact type, \text{customer} sends reservation request indicating that the customer sends a reservation request. This fact type is referred as the export message of the customer participant. It is then followed by the fact type, \text{AT} receives reservation request, illustrating the import message of Acme Travel participant.
The identical form of fact type, "term₁ verb term₂" is applied to specify the properties that belong to each participant (called the static (domain-specific) constraints). term₁ denotes the participant term (as declared in the participant set terms), while term₂ designates the static constraint terms (as in the terms’ declaration). The verb can be any verb that shows belonging, such as has. For example, customer has name demonstrates that the customer has his own name.

- Local behaviour model
  The existence of the import and the export messages as well as the domain-specific constraints allows each participant is overlayed a local behaviour model for each participant. This concept is reminiscent of the local model used in [19], although there it is defined more formally. The local behaviour model, which is designated for every single participant expresses its behavioural constraints. The construction of the local behaviour models (particularly in specifying the export and import messages), contribute to develop a part of an overall interactions as predefined global behaviours, by combining their import and export messages (this will be discussed later). The local behaviour model contains a set of messages exchanged and the domain-specific constraints, which is performed by the intended participant:

  a set of messages exchanged is classified into:

  * a set of fact types containing an export message
    The messages demonstrate the participant provides/delivers a potential service (the intended event) to participants of the other local models;

  * a set of fact types containing an import message
    The messages (the intended events) that are provided by the other participants

  * a set of fact types containing domain-specific constraints
    The fact types demonstrate the belonging properties of the participant.

  The example of local behaviour model can be seen as depicted in Figure 2.9. The local behaviour model for the participant, customer comprises the export message, customer sends reservation request to describe the customer provides a request; the import message, customer receives notification characterises the customer requires the event notification from the other participant who will deliver that intended event; the domain specific constraint, customer has name shows that the customer has a name.

- Composition of local behaviour models
The local behaviour models are composed of the basis of common events appearing in the respective sets of import and export messages of the participants. It captures the activities of messages exchanged, that is the sending and the receiving of the message (event). The idea to compose the local behaviour models is an adaptation of the composition local model used in [19]. Montali et al. composed the local model by combining all the business activities and the constraints (e.g., cardinality, temporal, and alternative interaction constraints, etc) of the local models. In our approach, the composition of local behaviour models combining the messages exchanged (business activities) of common event between participants only. The purpose of the composition process is to complement the import message of each participant with a corresponding export message of the other participant(s). When a local behaviour model is composed with other local behaviour models, the complete messages exchanged of common event is produced.

It is essential that the import message is provided by other participants, and that the export message is required by the other participant(s). This is important to ensure that the composition plays a part in realising the overall interactions (global behaviours) as specified in a set of business rules. Business goals are subsequently achieved.

In general, the following fact types encapsulate the composition of local
CHAPTER 2. LITERATURE REVIEW

Figure 2.10: The composition of local behaviour models

behaviour models which are exposed by different participants to capture the interactions between them. The export message of participant
1 in the fact type is participant1 sends event1 and the import message of participant2 in the fact type is participant2 receives event1. Note that sends and receives are the example of that verb, which is used to describe the participants executing sending or receiving a message (event).

The composing of local behaviour models is illustrated in Figure 2.10. The combining of messages exchanged between the export message of customer and the import message of AT, both of which are steered by the event reservation request, complement each other. Similarly, the composition of sending the notification by AT and receiving the notification by the customer realises the interaction between them.

We do not have any means of deriving the ordering between them (if any). In order to deal with the ordering of messages exchanged by the common event from different participants, we need to further elaborate on this topic, dealing also with precedence and a notion of time that draws from [42].

- Fact types for specifying static constraints for event

It is possible that in some circumstances an event has its own constraints elaborating its unique properties (called as static constraints for event). The static constraints for any intended event must be specified by exploiting the fact type in the form, "term1 verb term2". term2 denotes the static constraint term, which is associated with its intended event in term1, and both are connected with any verb that expresses the belongings of the event. The general form of fact types for specifying static constraint for event is "event verb static constraint term". The fact type in the Acme Travel case study is considered. Fact type airline reservation has outbound date demonstrates the airline reservation that is requested by the customer must be provided with the outbound date. Hence, the static constraint of the event airline reservation is outbound date.

- Static constraints for event - on the Dates

In most situations, there is a need to specify certain constraints on the dates, for instance, the dates of travel arrangements, the dates of
payment fees arrangements, etc. Dates can be treated like participants and events earlier, so there is a Date set declared as Term: date.

Previously, in the Acme Travel case study, the outbound date was declared as the static constraint of the airline reservation. Other dates such as start date and end date of reservation request, check-in date of accommodation reservation, are also declared as static constraints terms. Those terms can be specified as members of the Date set using the set definition in [7] and the construct is in which declares belonging, in the corresponding fact types:

Fact Type: start date is in date
Fact Type: end date is in date
Fact Type: check-in date is in date

Hence, generally any dates that are associated with any events as the static constraints must be specified in a Date set as the following form:

Fact Type: date\_1 is in date
Fact Type: date\_2 is in date

..............
Fact Type: date\_n is in date,

where date can be defined up to n number of dates in date set.

The constraints on the dates are usually related to the comparability of the dates such as the equality of two dates occurrences; one date must start before or after another date, or one date is occurred between two other dates. Towards specifying and then verifying those constraints, the standard form of fact type, ”term\_1 verb term\_2” is used to initialise the start and end date in date set. term\_1 denotes any date in date set as the initial (end) date, while term\_2 is specifically to characterise that initial (end) date. It is shown as the following form:

Fact Type: date\_1 is initial date
Fact Type: date\_2 is final date.

**SBVR rules semantics**

As discussed in Section 2.3.1, the OMG standard SBVR classifies a rule as a business rule, which is under the authority of an official organisation(s). In our approach, the business rules are the global behaviours with respect to a set of business processes (activities) being controlled by the different participants, i.e. different business applications (organisations), in the business collaboration.

We concentrate only on the operative (behavioural) business rules as they describe the constraints of service interactions that involve business activities across multiple organisations. The constraints of services interaction relate to the global
behaviours that must comply with a set of business processes to achieve the overall goal of the collaboration as agreed between the interacting participants. This contrasts with the structural rules, which specify the structure of an organisation, "what the organisation takes things to be and how do the members of the community agree on the understanding of the domain" [8], which is beyond the scope of this study.

The operative business rules are defined as a claim of obligation [7]. They can be expressed in the form of obligation, prohibition, or restricted permission statements.

In developing the SBVR model for choreographies, after the participant terms, event terms, and all the intended fact types have been defined, a set of business rules describing the choreography of the system is constructed. The construction emphasises the semantic for formulating the rules and the constraints, i.e. complex interactions constraints and time constraints, pertaining to the specification of the global behaviours.

• Semantic formulation

Semantic formulations are used in the SBVR model to structure the meaning of rules. Semantic formulations are categorised into logical formulation and proposition, and projecting formulation. Since the projecting formulation formulates the business rules involving definitions, aggregations, and questions, this is not of our concern as we focus only on the logical formulation and proposition of semantic formulation. The constructive rule is formulated by considering the combination of the following logical formulations and propositions.

1. Modal formulation

The SBVR standard [7] defines two distinct business rules; structural business rules and operative (behavioural) business rules. The structural rule (alethic) specifies about the organisations’ understanding of concepts on how the organisation takes things to be [7], [8]. It can be expressed in the form of necessity, impossibility, or restricted possibilities statements. For instance, the structural rule from [7], "It is necessary that the pick-up branch of a one-way rental is not the return branch of that rental", is a rule on the understanding concept of the rental car organisation as agreed by the members of the community (a branch).

On the other hand, the operative (behavioural) business rules (deontic) expresses and describes the business process of the organisation(s). It concerns on the behaviour of organisation(s) which is directly controlled, influenced, or regulated by the element of guidance. The element of guidance is detailed of behaviour that organisation(s)’ own. That behaviour can be applied effectively and consistently by the organisation(s). Also, it can be put into practice successfully and useful to the organisation(s) (people in that organisation(s)) by applying the element of guidance [7].
The operative behavioural rule can be expressed using either *obligation*, *prohibition*, or *restricted permission* modalities. For example, the operative behavioural rule from [7], "It is prohibited that the duration of a rental be more than 90 rental days" describes the guidance that controlling the business process or behaviour of the rental car organisation.

Since the focus of the SBVR model is on the operative business rules, the business rules are defined as the business rules that are a claim of obligation. This is expressive enough for our purpose which is to generate and verify service choreographies as we do not consider erroneous transmission or faulty channels, therefore all messages sent are received. What we verify is the ordering of messages exchanged. In terms of prohibition, we only consider the prohibition statement in expressing "equality" for specifying static constraints particularly for dates. In future work, we plan to address prohibited behaviour more general. The permission is not included in our model as the obligation is enough for choreography modelling and verifying and the permission statement (e.g. 'a' might happen or 'b' might happen) is captured in our model by using the logical operation inclusive or (OR) and exclusive or (XOR).

2. Quantifications formulation

The quantifications which capture multiplicity and participation constraints in the SBVR model are: *at least one* existential quantification, *at most one* quantification, and *exactly one* quantification. Only these quantifications will be considered for constructing the business rules in the SBVR model. Each quantification formulates the meaning by introducing exactly one *variable* that scopes over at most one logical formulation (i.e. atomic formulation) and ranges over the *term* of fact type as follows.

*participant terms* participant terms refer to any participants, which are defined as the member of *participant set*. For instance, in the Acme Travel case study, *customer* is a participant term as customer is included in the *participant set*.

*event terms* event terms are based on any events as specified in the SBVR model and included in the *event set*. For example, *reservation request* is an event term.

In view of fact that each participant plays a role as a single service entity who sends or receives the events (captured in the local behaviour model of import and export sets of messages exchanged), the quantifications *exactly one* or *the* as defined in OMG standard [7] are employed and ranged over each participant term. For instance, *the* which ranges over the term *customer* represents each customer who sending the travel requests in the online travel system.

The quantifications *at least one*, *at most one*, and *exactly one* are spe-
cialised to formulate the meaning for the event terms. We assume each occurrence of event as the general situation kind as defined in OMG standard SBVR [7] and Date Time Vocabulary (DTV) [42]. Situation kind is an occurrence that is probably a type of situation, event or activity that may occur, and the actual situation that may be planned for, etc. In our choreography model, situation kind is considered as the event (messages exchanged between participants) that has more than one occurrence in the possible system (application); even when the event cannot have more than one occurrence, it is chosen to be the universe of discourse in the possible system. This refers to general situation kind [42].

For instance, customer sends reservation request describes a general situation kind as it represents the messages exchanged that could be occurred more than once in the online travel system.

In addition, the aim of each occurrence event in the SBVR model is to resemble the multiplicity constraints of the Unified Modeling Language (UML) [31] which represents either the minimal, the exact or the maximum required in the executions of activities. Thus, those three types of quantifications are enough to capture the messages exchanged.

![Figure 2.11: Logical formulations - modal, quantification, and atomic](image)

With reference Figure 2.11, the rule is formulated by the obligation formulation and two quantification formulations. The first quantification the introduces the first variable that ranges over the term, customer. The second quantification exactly one introduces the second variable, which ranges over the term, reservation request.

3. **Atomic formulations**

Atomic formulations refer to fact types as discussed previously. In formulating the rules, each atomic formulation has a role binding that bind to each term in fact types. The fact type customer sends reservation request in Figure 2.11 is formulated by an atomic formulation, based on the generic fact type as discussed previously, participant sends verb. The first role binding is of the role participant which binds to the term customer of fact type. The atomic formulation also has a second role binding which is of the role event that binds to the second term of fact type reservation request.

4. **Objectification**

The objectification formulation is essential when there are more than one
2.3. AN OMG STANDARD SEMANTICS OF BUSINESS VOCABULARY AND RULES (SBVR)

fact types exist in the rule. In our approach, most of the rules consist of multiple fact types. Those rules are constructed to specify the messages exchanged associating with time, the ordering of events (immediately precedes notion), the static constraints, and the complex interactions. The meaning of the rule is formulated by the objectification formulation with considering the fact type that binds to a bindable target, a state of affairs (situation kind). For instance, the rule: It is obligatory that the customer sends reservation request at T1 (time interval 1) is constructed from a combination of two binary fact types. There are customer sends reservation request and state of affairs at T1. So, the fact type customer sends reservation request binds to a bindable target, state of affairs.

5. Logical operations In the SBVR model, the logical operations for exclusive disjunction (XOR), conjunction (AND), and inclusive disjunction (OR) are used on participation constraints (participant terms) and messages exchanged constraints (event terms), i.e. alternative and concurrent messages exchanged between participants and the ordering of messages exchanged, in forming SBVR rules. The logical operations have at least one logical operand (e.g., p proposition, or q proposition), where the meaning of each logical operand is either true or false. AND, OR, and XOR logical operations are defined in the SBVR model as in the following:

- AND operation - p and q, the meaning of each of its logical operand is true;
- OR operation - p or q, the meaning of at least one of its logical operands is true;
- XOR operation - p or q but not both the meaning of exactly one logical operand is true, while the meaning of another logical operand is false.

The above definitions are based on the binary logical operands [7]. In our approach, we consider also the n-ary logical operands as in the real scenario of services interaction, there is a need to execute some events (messages exchanged) of belonging to a set of activities, by more than one participants.

The definition and the specification of AND and OR logical operations for n-ary logical operands are identical as defined above, provided that there are n number of logical operands, \( p_1, p_2, \ldots, p_n \). The definition of XOR for n-ary logical operands is based on [43]. We notice that generally, n-ary logical operands for XOR are true just in case an odd numbers of the propositions (logical operands) \( p_1, p_2, \ldots, p_n \) are true, which is called the odd counting function of adicity n [43]. However, in the case of natural language, it is absolutely clear that XOR is exploited to express exactly
one sense of XOR, not as the definition defined for the odd counting function of adicity n. [43] addressed that the logical operation called real variable-adicity exclusive or is the one that relevant to formal accounts of natural language. The real variable-adicity exclusive or defines n-ary logical operands for XOR in the SBVR model is true if exactly one of logical operands $p_1, p_2, ..., p_n$ is true. Based on that definition, n-ary logical operands for XOR is expressed as follows:

$$p_1 \text{ or } p_2 \text{ or } .... \text{ or } p_n \text{ but not all}$$

The logical operations pertaining to the topic of complex interactions and temporal operator for immediately precedes in the SBVR model.

Finally, the accurate meaning of rule will be produced whenever all the introduced variables in the rule constructing by combining the logical formulations (e.g. modality, quantification, logical operation (if any), objectification (if any), and atomic formulation) are bound.

**Developing SBVR rules pertaining to complex interaction constraints**

The complex interaction constraints reflect the business constraints (business rules) of the system which encapsulate the participation constraints and the messages exchanged (events) constraints.

- **Participation constraints**

  In the SBVR model, the participation constraints focus on the participant terms constraining multiple participants refer to: n number of participants, $participant_1, participant_2, ..., participant_n$, or a nesting of participants, e.g. $participant_1$ comprises $participant_{a1}, participant_{a2}$, up to n number of participants $participant_{a_n}$ perform the messages exchanged.

  The participation constraints on participant terms are grouped into three logical operations; inclusive disjunction (OR), conjunction (AND), and exclusive disjunction (XOR) on participants. The illustrating of specifying the participation constraints refer to the generic rule is as follows:

  **It is obligatory that the** **participant** **verb exactly one** **event**

  The given rule is constructed according to the semantic formulation described in Section 2.3.2, by combining the appropriate logical formulations such as modal formulation, atomic formulation, and quantification. **participant** in the rule refers to the interacting participant sending or receiving exactly one event. Note that **verb** could be any **verb** as long as it gives the same meaning, e.g. **sends, receives**, etc. **event** in the rule represents exactly one event, however it could be several events (further explanation on this topic in Section 2.3.2). In order to specify the constraints on the participant terms, term
2.3. AN OMG STANDARD SEMANTICS OF BUSINESS VOCABULARY AND RULES (SBVR)

**participant** must be replaced with a collection of participant terms together with the logical operations over those terms as described in the following sections.

1. **OR on participants**

   OR on participants defines that *at least one* of participants performs the messages exchanged concurrently. The specification of OR constraints on participant are divided into two structures:

   a) exactly one $participant_1$ or exactly one $participant_2$ or ... or exactly one $participant_n$

   b) exactly one $participant_1$ that includes exactly one $participant_{a1}$ or exactly one $participant_{a2}$ or ... or exactly one $participant_{an}$

   With reference to the given generic rule, the following generic rules illustrate how each structure of OR participation constraint is employed.

   a) It is obligatory that exactly one $participant_1$ or exactly one $participant_2$ or ... or exactly one $participant_n$, verb exactly one event

   b) It is obligatory that exactly one $participant_1$ that includes exactly one $participant_{a1}$ or exactly one $participant_{a2}$ or ... or exactly one $participant_{an}$, verb exactly one event

   Both rules describe at least one of $n$ distinct participants sending or receiving the event. However, the first rule describes exactly one $participant_1$ sends (receives) event or exactly one $participant_2$ sends (receives) event or up to exactly one $participant_n$ sends (receives) event, or probably some of participants performing sending or receiving the event, or all participants executing the activity concurrently. The quantification *exactly one* for each participant emphasises the rule definition.

   On the other hand, the second rule relates to a group of $participant_1$, where at least one of it’s members be obligated to execute the sending or the receiving of the event. The quantification *exactly one* is employed for $participant_1$ as it refers to exactly one participant set containing participants targeting to achieve the same goal. The quantification *exactly one* is employed for its members as well. The rule describes at least one members of $participant_1$, either exactly one $participant_{a1}$ or exactly one $participant_{a2}$, or until exactly one $participant_{an}$, or some participants, or all participants performing sending or receiving the event.

2. **AND on participants**

   The logical operation AND over participant terms is used to specify each $n$ number of participants are obligatory to perform the messages exchanged. The specification for AND constraints on participants are grouped into two structures:
a) exactly one \( \text{participant}_1 \) and exactly one \( \text{participant}_2 \) and ... and exactly one \( \text{participant}_n \)

b) exactly one \( \text{participant}_1 \) that includes exactly one \( \text{participant}_{a_1} \) and exactly one \( \text{participant}_{a_2} \) and ... and exactly one \( \text{participant}_{a_n} \)

By replacing the above structures of AND constraint on participant terms to the given generic rule, the following general forms of rules are obtained.

a) It is obligatory that exactly one \( \text{participant}_1 \) and exactly one \( \text{participant}_2 \) and ... and exactly one \( \text{participant}_n \), verb exactly one \( \text{event} \)

b) It is obligatory that exactly one \( \text{participant}_1 \) that includes exactly one \( \text{participant}_{a_1} \) and exactly one \( \text{participant}_{a_2} \) and ... and exactly one \( \text{participant}_{a_n} \), verb exactly one \( \text{event} \)

The first structure and the second structure indicate all the specified participants, either distinct participants or participants that are grouped together having the identical purpose in executing the messages exchanged of the \( \text{event} \). The first rule represents non-related participants performing the corresponding activity concurrently at the same time interval. For the second rule, each distinct participant from the same group performing sending or receiving the same event. The quantification exactly one refers to the particular participant who is obligated executing the activity.

3. XOR on participants

The specification of XOR constraint on participants focuses on an explicit choices of participants, which are grouped together performing the interactions. Two structures of specification for XOR over participant terms; namely binary and \( n \)-ary participants as defined in Section 2.3.2 are as follows.

a) exactly one \( \text{participant}_1 \) that includes exactly one \( \text{participant}_{a_1} \) or exactly one \( \text{participant}_{a_2} \) but not both

b) exactly one \( \text{participant}_1 \) that includes exactly one \( \text{participant}_{a_1} \) or exactly one \( \text{participant}_{a_2} \) or ... or exactly one \( \text{participant}_{a_n} \) but not all

The following rules illustrate the XOR constraints on participants based on the given generic rule.

a) It is obligatory that exactly one \( \text{participant}_1 \) that includes exactly one \( \text{participant}_{a_1} \) or exactly one \( \text{participant}_{a_2} \) but not both, verb exactly one \( \text{event} \)

b) It is obligatory that exactly one \( \text{participant}_1 \) that includes exactly one \( \text{participant}_{a_1} \) or exactly one \( \text{participant}_{a_2} \) or ... or exactly one \( \text{participant}_{a_n} \) but not all, verb exactly one \( \text{event} \).
The first rule expresses the binary participants of \textit{participant}_1 set, where exactly one of \textit{participant}_1 collection is chosen to perform the messages exchanged of the \textit{event}. It is similar to the second specification for \(n\)-ary participants, exactly one of \(n\) number of \textit{participant}_1 is decided to execute the \textit{event}. The quantification exactly one emphasises the intended participant set and only the particular participant is selected to send or receive the event, not all the members of \textit{participant}_1 set.

- Messages exchanged constraints

  The focal point of messages exchanged constraints is to capture the specification of messages exchanged relating to the concurrent and alternative interactions. Constraining is on several event terms exploit the logical operations over them. The events refer to either \(n\) number of distinct events, \textit{event}_1, \textit{event}_2, \ldots, \textit{event}_n, or different events that are grouped together, e.g. \textit{event}_1 comprises \textit{event} \_a_1, \textit{event} \_a_2, up to \(n\) number of events \textit{event} \_a_n. The events characterise the occurrence of sending or receiving of the events which are performed by the participant(s).

  The messages exchanged constraints for each concurrent and alternative interaction have their own structures. Each structure is illustrated by two kind of rules: i). a single participant sending (receiving) multiple events, ii). multiple participants (if any) (as specified in Section 2.3.2) sending (receiving) multiple events (if any). The same generic rule expressed in Section 2.3.2 is applied in this section.

  - Concurrent Interactions

    Based on the logical operation definition in Section 2.3.2, logical operation AND over event terms specifies concurrent interactions by assuming that each event takes place, in no particular order. There are two structures constraining event terms using logical operation AND over them.

    1. exactly one \textit{event}_1 and exactly one \textit{event}_2 and \ldots and exactly one \textit{event}_n
   
    2. exactly one \textit{event}_1 that includes exactly one \textit{event}_a and exactly one \textit{event}_b and \ldots and exactly one \textit{event}_n

    Each structure is expressed in the given generic rule by replacing the event term with an appropriate given structure. Two general forms of rules for each structure of AND constraints on event are specified as in the following:

    1. A single participant performing sending or receiving of the intended events concurrently:
       a) It is obligatory that the \textit{participant}_1 \textit{verb} exactly one \textit{event}_1 and exactly one \textit{event}_2 and \ldots and exactly one \textit{event}_n
b) It is obligatory that the participant verbs exactly one event that includes exactly one event and exactly one event and ... and exactly one event.

Both rules concern when n number of unordered events take place in the interaction by a single participant. The first rule refers to each distinct event, while the second rule represents each event under the same category of event, which may be delivered or required by the participant. The quantification exactly one for each event emphasises the purpose of the rules, where each event are obligated to execute the activities.

2. Multiple participants sending or receiving of the intended events concurrently:

a) OR/AND of participants

It is impossible to specify the rule involving at least one of participants (refers to OR constraints on participant) and each participant (refers to AND constraints on participant) executing the messages exchanged of multiple events concurrently. For example, the rule describing messages exchanged
It is obligatory that the airline and (or) the hotel sends exactly one airline reservation response and exactly one hotel reservation response is unable to be achieved. In this situation, it is not possible to the airline sending the hotel reservation response since that event is describe as the local behaviour of the airline. In other words, the sending of that event is not under the airline’s responsibility. Similarly, the sending of the airline reservation response is not under the responsibility of the hotel. Hence, separate rules expressing the role of the hotel and the airline for sending the response are worth to consider. Likewise, each participant (AND over participant terms) or at least one of participants (OR over participant term) which are grouped under the same participant are not possible executing multiple events concurrently.

b) An explicit choices (XOR) of participants

The following general forms of rules are especially useful when the SBVR model contains the constraints which require an exclusive choice of binary participants or n number of participants, performing the messages exchanged of unordered events (AND constraints on events).

i. Binary participants:

A. It is obligatory that exactly one participant that includes exactly one participant or exactly one participant but
B. It is obligatory that exactly one \( \text{participant}_1 \) that includes exactly one \( \text{participant}_{a_1} \) or exactly one \( \text{participant}_{a_2} \) but not both, verb exactly one \( \text{event}_1 \) that includes exactly one \( \text{event}_a \) and exactly one \( \text{event}_b \) and ... and exactly one \( \text{event}_n \)

ii. \( n-\text{ary} \) participants:

A. It is obligatory that exactly one \( \text{participant}_1 \) that includes exactly one \( \text{participant}_{a_1} \) or exactly one \( \text{participant}_{a_2} \) or ... or exactly one \( \text{participant}_{a_n} \) but not all, verb exactly one \( \text{event}_1 \) and exactly one \( \text{event}_2 \) and ... and exactly one \( \text{event}_n \)

B. It is obligatory that exactly one \( \text{participant}_1 \) that includes exactly one \( \text{participant}_{a_1} \) or exactly one \( \text{participant}_{a_2} \) or ... or exactly one \( \text{participant}_{a_n} \) but not all, verb exactly one \( \text{event}_1 \) that includes exactly one \( \text{event}_a \) and exactly one \( \text{event}_b \) and ... and exactly one \( \text{event}_n \)

For instance, the rule It is obligatory that the \textit{accommodation} that includes the \textit{hotel} or the \textit{hostel} but not both, sends exactly one \textit{accommodation reservation response} and exactly one \textit{reservation notification} asserting that exactly one of participant, either the hotel or the hostel executing the sending of both events reservation notification and accommodation reservation response concurrently to the other intended participant.

**Alternative Interactions**

In the SBVR model, alternative interaction constraints concern on the interacting participants performing the messages exchanged of the events by selecting them inside a set of possible choices. This is useful to the participants who having the same business goal as they leave to choose the most suitable event as free as possible. The alternative interaction constraints asserting that there are \( n-\text{ary} \) events that must be chosen by a single participant or \( n \) distinct participants. The participant(s) are able to choose either at least one of possible choices (called inclusive-disjunction (OR)) of events or exactly one of them (called exclusive-disjunction (XOR)).

OR and XOR constraints on events have their own structures that can be used in the rules to express a single participant or multiple participants (based on the participation constraints definition) performing the alternative interaction. In order to illustrate the specification of rule for the alternative interactions, the given generic rule in Section 2.3.2 is used by replacing a single event with \( n \) distinct events (OR or XOR constraints on events). Two categories of rules specification are included for each constraint on events: i) for a single participant executing the
alternative interaction, ii). for multiple participants (if any) performing the alternative interaction.

1. Inclusive-choice (OR) constraints on events

OR constraint on events refers to the interaction must necessarily performed by participant(s) via selecting at least one of the distinct events inside a set of possible choices of event. In our approach, we consider two kinds of structures describing the choices of $n$-ary events as in the following.

a) exactly one $\text{event}_1$ or exactly one $\text{event}_2$ or ... or exactly one $\text{event}_n$

b) exactly one $\text{event}_1$ that includes exactly one $\text{event}_a$ or exactly one $\text{event}_b$ or ... or exactly one $\text{event}_n$

The first structure shows the choices from several distinct events and the second structure represents the choices from a collection of events that are interconnected under the same purposes. Each structure can be used to express the rules encapsulate a single participant or multiple distinct participants executing the messages exchanged of OR constraints on events.

a) A single participant performing sending or receiving of the inclusive choices of events:

i. It is obligatory that the $\text{participant}_1$ verb exactly one $\text{event}_1$ or exactly one $\text{event}_2$ or ... or exactly one $\text{event}_n$

ii. It is obligatory that the $\text{participant}_1$ verb exactly one $\text{event}_1$ that includes exactly one $\text{event}_a$ or exactly one $\text{event}_b$ or ... or exactly one $\text{event}_n$

The general form of first rule specifies that the $\text{participant}_1$ is able to choose at least one of the given events. The quantification exactly one for each event captures the meaning of only one of the event is chosen, or some of them are selected, or each event is required in the interaction.

This alternative interaction can be seen clearly in the rule extracted from the Acme Travel case study, "It is obligatory that the $\text{Acme Travel}$ requests for exactly one $\text{airline reservation}$ or exactly one $\text{accommodation reservation}$ or exactly one $\text{transport reservation}". This rule asserting that Acme Travel may send a request of one of the given events, or some combination of the given events (e.g. $\text{accommodation reservation}$ or $\text{transport reservation}$), or all events.

The general form of second rule captures the specification when the participant intends to send or receive the events, which are grouped under the identical category and objective. The rule, "It is obligatory that the $\text{customer}$ makes exactly one
**product request** that *includes* exactly one **photo request** or exactly one **album request** or exactly one **poster request**, is one of the rules from our case study, Online Photo Shop. With reference to that rule, each distinct event (i.e. **photo request**, **album request**, **poster request**) are grouped together under **product request** event. Each event plays a role as kind of products that are offered to the customer. However at least one of them could be chosen.

b) Multiple participants sending or receiving of the inclusive choices of events:

i. OR/AND constraints on participants

It is worth noting that the specifications for at least one of participants (OR constraint on participant) or each participant (AND constraint on participant) executing at least one of distinct events or at least one of the interconnected events (that are grouped together) are impossible to be done. As described in Section 2.3.2, it is impossible that multiple participants executing more than one event together, mostly when one of those events are not related to the intended participant(s).

ii. XOR of participants

The general forms of rules describing the messages exchanged of the inclusive choices of events by exactly one of participants (XOR constraint on participant) are divided into two categories as in the following.

**Binary participants**

* It is obligatory that exactly one **participant\(_1\)** that *includes* exactly one **participant\(_{a1}\)** or exactly one **participant\(_{a2}\)** but not both, **verb** exactly one **event\(_1\)** or exactly one **event\(_2\)** or ... or exactly one **event\(_n\)**

* It is obligatory that exactly one **participant\(_1\)** that *includes* exactly one **participant\(_{a1}\)** or exactly one **participant\(_{a2}\)** but not both, **verb** exactly one **event\(_1\)** that *includes* exactly one **event\(_a\)** or exactly one **event\(_b\)** or ... or exactly one **event\(_n\)**

**n-ary participants**

* It is obligatory that exactly one **participant\(_1\)** that *includes* exactly one **participant\(_{a1}\)** or exactly one **participant\(_{a2}\)** or ... or exactly one **participant\(_{an}\)** but not all, **verb** exactly one **event\(_1\)** or exactly one **event\(_2\)** or ... or exactly one **event\(_n\)**

* It is obligatory that exactly one **participant\(_1\)** that *includes* exactly one **participant\(_{a1}\)** or exactly one **participant\(_{a2}\)** or ... or exactly one **participant\(_{an}\)** but not all, **verb** exactly
one \( \text{event}_1 \) that \textit{includes} exactly one \( \text{event}_a \) or exactly one \( \text{event}_b \) or ... or exactly one \( \text{event}_n \).

Each general form of rules represents the constraints when exactly one of two distinct participants (binary participants) or of \( n – ary \) interacting participants in the \textit{participant}_1 set may selecting at least one of the given events.

2. Exclusive-choice (XOR) constraints on events

The alternative interaction constraints for XOR over events term are not limited to two distinct choices of the events only, it concerns more than two-exclusive choices of events as well. The structures for both; two-exclusive choices of events and \( n \)-exclusive choices of events are as follows:

a) exactly one \( \text{event}_1 \) that \textit{includes} exactly one \( \text{event}_a \) or exactly one \( \text{event}_b \) but not both

b) exactly one \( \text{event}_1 \) that \textit{includes} exactly one \( \text{event}_a \) or exactly one \( \text{event}_b \) or ... or exactly one \( \text{event}_n \) but not all

Both structures are adopted to the following general forms of rules.

a) A single participant performing sending or receiving of the exclusive choices of events:

i. It is obligatory that the \textit{participant}_1 \textit{verb} exactly one \( \text{event}_1 \) that \textit{includes} exactly one \( \text{event}_a \) or exactly one \( \text{event}_b \) but not both

ii. It is obligatory that the \textit{participant}_1 \textit{verb} exactly one \( \text{event}_1 \) that \textit{includes} exactly one \( \text{event}_a \) or exactly one \( \text{event}_b \) or ... or exactly one \( \text{event}_n \) but not all

The given rules representing exactly one participant performs the messages exchanged of the given events by selecting exactly one of them. In the Acme Travel case study, the rule It is obligatory that the \textit{airline} \textit{verb} exactly one \( \text{airline reservation response} \) that \textit{includes} exactly one \textit{successful airline reservation} or exactly one \textit{unsuccessful airline reservation} but not both showing the participant \textit{airline} is responsible to inform the intended participant either the reservation for airline is successful or not.

b) Multiple participants sending or receiving of the exclusive choices of events

In some circumstances, multiple distinct participants having the same objective in executing activities, by selecting the same event of the given two or \( n \) distinct choices. Multiple distinct participants refer to each participant (AND constraints on participants), or at least one of participants (OR constraints on participants), or only the particular participant (XOR
constraints on participants), take(s) the responsibilities to execute exactly one of the possible choices of the events. The general forms of rules for those situations are illustrated as in the following:

**OR of participants**

i. It is obligatory that exactly one \(\text{participant}_1\) or exactly one \(\text{participant}_2\) or ... or exactly one \(\text{participant}_n\), verb exactly one \(\text{event}_1\) that includes exactly one \(\text{event}_a\) or exactly one \(\text{event}_b\) but not both.

ii. It is obligatory that exactly one \(\text{participant}_1\) or exactly one \(\text{participant}_2\) or ... or exactly one \(\text{participant}_n\), verb exactly one \(\text{event}_1\) that includes exactly one \(\text{event}_a\) or exactly one \(\text{event}_b\) or ... or exactly one \(\text{event}_n\) but not all.

iii. It is obligatory that exactly one \(\text{participant}_1\) that includes exactly one \(\text{participant}_{a1}\) or exactly one \(\text{participant}_{a2}\) or ... or exactly one \(\text{participant}_{an}\), verb exactly one \(\text{event}_1\) that includes exactly one \(\text{event}_a\) or exactly one \(\text{event}_b\) but not both.

iv. It is obligatory that exactly one \(\text{participant}_1\) that includes exactly one \(\text{participant}_{a1}\) or exactly one \(\text{participant}_{a2}\) or ... or exactly one \(\text{participant}_{an}\), verb exactly one \(\text{event}_1\) that includes exactly one \(\text{event}_a\) or exactly one \(\text{event}_b\) or ... or exactly one \(\text{event}_n\) but not all.

**AND of participants**

i. It is obligatory that exactly one \(\text{participant}_1\) and exactly one \(\text{participant}_2\) and ... or exactly one \(\text{participant}_n\), verb exactly one \(\text{event}_1\) that includes exactly one \(\text{event}_a\) or exactly one \(\text{event}_b\) but not both.

ii. It is obligatory that exactly one \(\text{participant}_1\) and exactly one \(\text{participant}_2\) and ... or exactly one \(\text{participant}_n\), verb exactly one \(\text{event}_1\) that includes exactly one \(\text{event}_a\) or exactly one \(\text{event}_b\) or ... or exactly one \(\text{event}_n\) but not all.

iii. It is obligatory that exactly one \(\text{participant}_1\) that includes exactly one \(\text{participant}_{a1}\) and exactly one \(\text{participant}_{a2}\) and ... and exactly one \(\text{participant}_{an}\), verb exactly one \(\text{event}_1\) that includes exactly one \(\text{event}_a\) or exactly one \(\text{event}_b\) but not both.

iv. It is obligatory that exactly one \(\text{participant}_1\) that includes exactly one \(\text{participant}_{a1}\) and exactly one \(\text{participant}_{a2}\) and ... or exactly one \(\text{participant}_{an}\), verb exactly one \(\text{event}_1\) that includes exactly one \(\text{event}_a\) or exactly one \(\text{event}_b\) or ... or exactly one \(\text{event}_n\) but not all.
CHAPTER 2. LITERATURE REVIEW

XOR of participants

i. Binary participants

* It is obligatory that exactly one participant<sub>1</sub> that includes exactly one participant<sub>a_1</sub> or exactly one participant<sub>a_2</sub> but not both, verb exactly one event<sub>1</sub> that includes exactly one event<sub>a</sub> or exactly one event<sub>b</sub> but not both

* It is obligatory that exactly one participant<sub>1</sub> that includes exactly one participant<sub>a_1</sub> or exactly one participant<sub>a_2</sub> but not both, verb exactly one event<sub>1</sub> that includes exactly one event<sub>a</sub> or exactly one event<sub>b</sub> or ... or exactly one event<sub>n</sub> but not all

ii. n-ary participants

* It is obligatory that exactly one participant<sub>1</sub> that includes exactly one participant<sub>a_1</sub> or exactly one participant<sub>a_2</sub> or ... or exactly one participant<sub>a_n</sub> but not all, verb exactly one event<sub>1</sub> that includes exactly one event<sub>a</sub> or exactly one event<sub>b</sub> but not both

* It is obligatory that exactly one participant<sub>1</sub> that includes exactly one participant<sub>a_1</sub> or exactly one participant<sub>a_2</sub> or ... or exactly one participant<sub>a_n</sub> but not all, verb exactly one event<sub>1</sub> that includes exactly one event<sub>a</sub> or exactly one event<sub>b</sub> or ... or exactly one event<sub>n</sub> but not all

– Special concerns on messages exchanged constraints

In the SBVR model, we consider the situation where a participant executing the messages exchanged by selecting particular event(s) in a nesting of event set (using fact types as discussed in Section 2.3.2), or some participants who are specified as a nesting of group having dissimilar goal in executing the messages exchanged of the event(s) (even though those events are specified under a nesting of group using fact type). The following rules are extracted from our two case studies showing this situation.

1. Rules in the Acme Travel case study

   a) It is obligatory that the accommodation that includes the hotel receives exactly one accommodation reservation that includes exactly one hotel reservation

   b) It is obligatory that the accommodation that includes the apartment receives exactly one accommodation reservation that includes exactly one apartment reservation

Based on the given rules, we notice that hotel and apartment are grouped under the same set, namely accommodation, while
hotel reservation and apartment reservation are classified under the identical set, namely accommodation reservation. However, both hotel and apartment intend to receive the different events, hotel reservation and apartment reservation, respectively.

2. Rules in the Tuition Fees case study
   a) It is obligatory that the self-funding student that includes the visa prerequisite student makes exactly one tuition fee payment that includes exactly one full payment
   b) It is obligatory that the self-funding student that includes the non-visa prerequisite student makes exactly one tuition fee payment that includes exactly one full payment or exactly one instalment payment but not both

The given rules constrain that each student who is categorised under the same type of student self-funding student executing the same activity of the tuition fee payment, that is full payment. However the non-visa prerequisite student has the additional choice of tuition fee payment, namely instalment payment.

In this kind of situation, only the intended participant(s) or/and the intended event(s) defined in a nesting of group are specified, regardless of participation constraints and messages exchanged constraints in rules, i.e. a single participant, or inclusive-choices (OR) of participants/events, or conjunction (AND) of participants/events, or exclusive-choices (XOR) of participants/events.

We illustrate the general forms of rules for this situation by assuming there are the fact types defining participant_1, participant_2, participant_3, and participant_4 as the participants in the SBVR model. A nesting of participant_2 comprises participant_a, participant_b, and participant_c; a nesting of participant_3 encompasses participant_d and participant_e. The fact types are as follows:

* participant includes participant_1
* participant includes participant_2
   · participant_2 includes participant_a
   · participant_2 includes participant_b
   · participant_2 includes participant_c
* participant includes participant_3
   · participant_3 includes participant_d
   · participant_3 includes participant_e
* participant includes participant_4

Also, we assume there are fact types for the events event_1, event_2, event_3, and event_4. The event_3 has its nesting event_a, event_b, and event_c, while event_4 has its nesting event_d and event_e.
CHAPTER 2. LITERATURE REVIEW

* event includes event\(_1\)

* event includes event\(_2\)

* event includes event\(_3\)
  - event\(_3\) includes event\(_a\)
  - event\(_3\) includes event\(_b\)
  - event\(_3\) includes event\(_c\)

* event includes event\(_4\)
  - event\(_4\) includes event\(_d\)
  - event\(_4\) includes event\(_e\)

The general forms of rules are constructed based on the given fact types as defined. These rules illustrate the instances of rules for the aforementioned situation. The rules are categorised into two types as in the following.

1. A single participant executing the messages exchanged of the event(s) by selecting the intended event inside a nesting of group of event.

   * It is obligatory that the participant\(_1\) verb exactly one event\(_3\) that includes exactly one event\(_a\)
     This rule considers when the participant\(_1\) selects event\(_a\) only. event\(_b\) and event\(_c\) are not selected.

   * It is obligatory that the participant\(_1\) verb exactly one event\(_3\) that includes exactly one event\(_a\) or exactly one event\(_b\)
     This rule describes that event\(_a\) or event\(_b\) (OR constraints on events) are chosen from event\(_3\) set, by the participant\(_1\).

   * It is obligatory that the participant\(_1\) verb exactly one event\(_3\) that includes exactly one event\(_b\) and exactly one event\(_c\)
     This rule expresses the goal of the participant\(_1\) which executes the messages exchanges of the event\(_b\) and event\(_c\) (AND constraints on events) only.

   * It is obligatory that the participant\(_1\) verb exactly one event\(_4\) that includes exactly one event\(_d\) or exactly one event\(_e\) but not both

It is obligatory that the participant\(_4\) verb exactly one event\(_4\) that includes exactly one event\(_d\)

These two rules show different participants aiming to execute the event(s) from the event\(_4\) set which contains two distinct choices of events. The participant\(_1\) decides to choose exactly one of the events event\(_d\) or event\(_e\) but not both(XOR constraints on events), however the participant\(_4\) opts to execute the event\(_d\) only. In this case, there is probably the participant\(_1\) chooses only one event event\(_d\), or the same event with the participant\(_4\), that is event\(_d\) only. It depends on the situation.
2. The intended participant(s) inside a nesting of group executing the messages exchanged of the single event or the intended event(s) as specified in a nesting of group.

* It is obligatory that the \textit{participant}_2 that \textit{includes} \textit{participant}_a or the \textit{participant}_b, \textit{verb} exactly one \textit{event}_1

This rule expresses the \textit{participant}_a or \textit{participant}_b (OR constraints on participants) are selected from \textit{participant}_2 set performing the sending or the receiving of the \textit{event}_1. The \textit{participant}_c is not be chosen. Notice that the specification of the event in this rule could be replaced with XOR of events from the \textit{event}_4 set as well, showing that at least of \textit{participant}_a or \textit{participant}_b executing the given events.

* It is obligatory that the \textit{participant}_2 that \textit{includes} the \textit{participant}_a and the \textit{participant}_b, \textit{verb} exactly one \textit{event}_1

This rule shows both \textit{participant}_a and \textit{participant}_b (AND constraints on participants) in the \textit{participant}_2 set aiming to perform the same \textit{event}_1. The \textit{participant}_c is not be selected. Notice that the specification of the event in this rule could be replaced with XOR of events from the \textit{event}_4 set describing each \textit{participant}_a and \textit{participant}_b executing the given events.

* It is obligatory that the \textit{participant}_2 that \textit{includes} the \textit{participant}_c \textit{verb} exactly one \textit{event}_2

In this rule, only \textit{participant}_c inside the \textit{participant}_2 set, takes the responsibility to execute the \textit{event}_2.

* It is obligatory that the \textit{participant}_2 that \textit{includes} the \textit{participant}_c \textit{verb} exactly one \textit{event}_3 that \textit{includes} exactly one \textit{event}_a or exactly one \textit{event}_b

The \textit{participant}_c from the \textit{participant}_2 set is selected for executing the messages exchanged of OR constraints on events \textit{event}_a or \textit{event}_b from the \textit{event}_3 set. The \textit{event}_c is not selected.

* It is obligatory that the \textit{participant}_2 that \textit{includes} the \textit{participant}_c \textit{verb} exactly one \textit{event}_4 that \textit{includes} exactly one \textit{event}_a and exactly one \textit{event}_b

The \textit{participant}_c inside the \textit{participant}_2 set has a goal to execute the unordered events \textit{event}_a and \textit{event}_b concurrently from a nesting of \textit{event}_3. Similarly, the \textit{event}_c is not the intention of the given participants.

* It is obligatory that the \textit{participant}_2 that \textit{includes} the \textit{participant}_c \textit{verb} exactly one \textit{event}_4 that \textit{includes} exactly one \textit{event}_a or exactly one \textit{event}_b but not both

The \textit{participant}_c inside the \textit{participant}_2 set is obligated to execute exactly one of the \textit{event}_4, either \textit{event}_a or \textit{event}_b but not both.
It is worth noting that as explained in Section 2.3.2, the participants involving the logical operations OR and AND over them are impossible executing the sending or the receiving of the events involving OR and AND constraints over the event terms.

Temporal operator in SBVR

The temporal operators play a central role in the underlying service choreographies to be able to derive the allowed orderings between the messages exchanged when the multi-party conversation takes place. In the SBVR model, the DTV [42] supplement to the SBVR specification is used to incorporate the temporal operators as a notion of precedence.

The principal concept in the DTV is time interval because it is used in specifying many of business terms [42]. DTV defines that time interval has a linear time structure under segment of time axis, where each time interval is finite and the structure is bounded. In other words, each time interval has a start, an end, and a duration, even if not known. Also, it worth noting that, time interval may be defined by reference to events that occur for a time interval. In our approach, as described previously, we assume each occurrence of event as the general situation kind. It means that the event (messages exchanged between participants) in the SBVR model is the event that may occur perhaps more than once in an individual instance of the process in the system.

We focus here only on the most relevant aspects for our purposes: i). time notion - a notion of time which encapsulates the ordering of message exchanges from different participants; ii). immediately precedes notion - a notion of time encapsulating the ordering of messages exchanged of dissimilar events, where each event refers to the messages exchanged in i).

1. SBVR rules pertaining to time notion

With reference to the composition of local behaviour models as described previously, we note that there is no indication to inform the ordering of messages exchanged by the common event from different participants. Note that the composing local behaviour models involve the export and import message that engaged with the correspond event(s) of different participants such as the export message of participant$_1$ in the fact type is “participant$_1$ sends event$_1$”, and the import message of participant$_2$ in the fact type is ”participant$_2$ receives event$_1$”. From this composition, there is no indication showing which interaction is performed initially and which interaction is performed immediately after. To overcome this we incorporate a notion of time understood as in the construct situation kind at time interval which is consistent with [42].

Situation kind refers to the occurrence occurs more than once or the universal discourse in the possible system. And situation kind occurs within the given time interval, which is expressed in the form of proposition. The
2.3. AN OMG STANDARD SEMANTICS OF BUSINESS VOCABULARY AND RULES (SBVR)

proposition is represented by fact type to describe the export or the import of messages by the participants.

By using the notion *situation kind at time interval*, the composition of local behaviour models could be represented meaningfully. The general forms of rules which apply this notion are illustrated as in the following:

a) The rule for describing the export message of *participant$_1$*:
   
   It is obligatory that the *participant$_1$ sends* exactly one *event$_1$* at exactly one *T$_1$*

b) The rule for describing the import message of *participant$_2$*:
   
   It is obligatory that the *participant$_2$ receives* exactly one *event$_1$* at exactly one *T$_2$*

Note that *sends* and *receives* are exploited in the above rules to illustrate the export and the import message in the interaction (in the real situation, they could be any *verb* demonstrating the sending or receiving of the event), and *T$_1$* and *T$_2$* refer to time interval 1 and time interval 2. The time notion captures the ordering of messages exchanged between *participant$_1$* and *participant$_2$*, which is steered by the same event *event$_1$*. Figure 2.12 depicts a picture of how the ordering of messages exchanged takes place within the given time interval. It shows that the initial interaction is performed by the *participant$_1$* throughout the time interval *T$_1$*, then followed by the *participant$_2$* which performs the interaction within the given time interval *T$_2$*.

![Figure 2.12: Illustration of the ordering of messages exchanged by the common event](image)

This time notion also encapsulates the specification of the complex interactions that involve the participation constraints or/and the messages exchanged constraints. It means that the time notion could describe the ordering of messages exchanged which consists of a combination activities that involves sequence, parallel, or choice of events. For specifying the rules showing this situation, the time interval is added at the end of each rule as shown in the following general forms of rules with reference to the rules as discussed before. Note that the general forms demonstrate the rules which specify up to binary participants and events only, however in the real situation the specification for participants and events allow up to *n* number of participants and events. The
logical operation or in the rules could be replaced with the other logical operations (if any) and and or, but not both. *T* denotes a generic time interval that can be designated as *T*₁..*T*ₙ.

a) The general forms of rules represent a single participant sending/receiving multiple distinct events or the events inside a nesting of group.

- It is obligatory that the *participant₁* verb exactly one *event₁* or exactly one *event₂*, at exactly one *T*

- It is obligatory that the *participant₁* verb exactly one *event₁* that includes exactly one *eventₐ* or exactly one *event₇*, at exactly one *T*

b) The general forms of rules represent multiple participants, which are specified as distinct participants or interconnected participants in a nesting of group. The participants sending/receiving a single event, multiple distinct events or the events inside a nesting of group.

- It is obligatory that the *participant₁* or the *participant₂* verb exactly one *event₁* at exactly one *T*
  
  Note that this rule is applicable for OR and AND constraints on participant only.

- It is obligatory that the *participant₁* that includes the *participantₐ* or the *participant₇*, verb exactly one *event₁* at exactly one *T*
  
  This rule is applicable for XOR and AND constraints on event inside a nesting of group.

- It is obligatory that the *participant₁* or the *participant₂*, verb exactly one *event₁* that includes exactly one *eventₐ* or exactly one *event₇*, but not both, at exactly one *T*
  
  Note that this rule is not applicable to XOR participation constraints. Each rule that specifies OR and AND constraints on participants is impossible to express multiple events involving OR and AND constraints on events.

- It is obligatory that the *participant₁* that includes the *participantₐ* or the *participant₇*, verb exactly one *event₁* that includes exactly one *eventₐ* or exactly one *event₇*, but not both, at exactly one *T*
  
  The similar concern as in the previous rule, where each rule that involves OR and AND constraints on participants is impossible to express multiple events involving OR and AND constraints on events.

- It is obligatory that the *participant₁* that includes the *participantₐ* or the *participant₇*, but not both, verb exactly one *event₁* or exactly one *event₂*, at exactly one *T*

- It is obligatory that the *participant₁* that includes the *participantₐ* or the *participant₇* but not both, verb exactly one *event₁* that
includes exactly one \textit{event}_a or exactly one \textit{event}_b, at exactly one \text{T}

These two rules are appropriate only to explicit choice of participants (XOR) executing the messages exchanged at the given time interval.

2. SBVR rules pertaining to immediately precedes notion

The purpose of \textit{immediately precedes} notion is to prescribe i). the ordering of time interval associating with the same event, ii). the ordering of messages exchanged of different events. The notion of \textit{immediately precedes} "\textit{time interval 1 immediately precedes time interval 2}" which a synonymous form of "\textit{time interval 1 meets time interval 2}" is advocated from DTV [42]. This notion is based on one of Allen [44] thirteen ways in which an ordered pair of time intervals can be related, namely mutually exclusive relation "\textit{meet}". It defines that \textit{time interval 1} is before \textit{time interval 2}, where there is no another time interval \textit{time interval 3} after \textit{time interval 1} and before \textit{time interval 2}.

The significants of the \textit{immediately precedes} notion are illustrated using the following example of rules.

a) Rule 1: It is obligatory that the \textit{participant}_1 sends exactly one \textit{event}_1 at exactly one \text{T}_1

Rule 2: It is obligatory that the \textit{participant}_2 receives exactly one \textit{event}_1 at exactly one \text{T}_2

b) Rule 3: It is obligatory that the \textit{participant}_3 sends exactly one \textit{event}_2 at exactly one \text{T}_1

Rule 4: It is obligatory that the \textit{participant}_4 receives exactly one \textit{event}_2 at exactly one \text{T}_2

All rules employ the time notion. Rule 1 and Rule 2 demonstrate that the messages exchanged of the \textit{event}_1 is initiated by \textit{participant}_1, then followed by the \textit{participant}_2. Similarly, the \textit{participant}_3 initiates the interaction of the \textit{event}_2 immediate before the \textit{participant}_4 executing the same event. However, there is no indication to show:

a) \text{T}_1 which associates with \textit{event}_1 occurs immediate before \text{T}_2 relating to the same event \textit{event}_1. It is similar to time interval associating to \textit{event}_2.

b) which activities of exchanging messages are initially been performed, either the executing of \textit{event}_1 between \textit{participant}_1 and \textit{participant}_2 or the executing of \textit{event}_2 that is performed by \textit{participant}_3 and \textit{participant}_4.
Hence, the notion of immediately precedes is essential for determining the ordering of i) \(T_1\) and \(T_2\), ii) \(event_1\) and \(event_2\) for the messages exchanged occur in Rule 1, Rule 2, Rule 3, and Rule 4.

a) **The ordering of time interval associating with the same event**

In order to sequence the possible time intervals relating to the same event, the identical definition of time interval is applied. Therefore, we advocate the immediately precedes notion to order time interval, i.e. \(T_1\), \(T_2\), ..., \(T_{n-1}\), \(T_n\) which associate with the same event, e.g. \(event_1\). The following fact types represent the general form of fact types, which are used to express "\(T_1\) immediately precedes \(T_2\)" or any time intervals, e.g. "\(T_{n-1}\) immediately precedes \(T_n\)", which associate with the same event, e.g. \(event_1\):

i. \(event_1\) at \(T_1\) immediately precedes \(event_1\) at \(T_2\)

ii. \(event_1\) at \(T_{n-1}\) immediately precedes \(event_1\) at \(T_n\)

The above fact types are specified in the following general form of rules, respectively.

i. It is obligatory that exactly one \(event_1\) at exactly one \(T_1\) immediately precedes exactly one \(event_1\) at exactly one \(T_2\)

ii. It is obligatory that exactly one \(event_1\) at exactly one \(T_{n-1}\) immediately precedes exactly one \(event_1\) at exactly one \(T_n\)

With reference to the aforementioned example of rules; Rule 1, Rule 2, Rule 3, and Rule 4, the rules describing the ordering of time interval associating with \(event_1\) and \(event_2\) are as follows:

- The ordering of \(T_1\) and \(T_2\) relating to \(event_1\) in Rule 1 and Rule 2. It is obligatory that exactly one \(event_1\) at exactly one \(T_1\) immediately precedes exactly one \(event_1\) at exactly one \(T_2\)

- The ordering of \(T_1\) and \(T_2\) associating to \(event_2\) in Rule 3 and Rule 4. It is obligatory that exactly one \(event_2\) at exactly one \(T_1\) immediately precedes exactly one \(event_2\) at exactly one \(T_2\)

It is worth to note that the same time \(T_1\), \(T_2\), ..., \(T_n\) are used for each event to preserve the meaning of \(T_1\) as the initial time interval in the messages exchanged of the particular event, next is \(T_2\), up until \(T_n\).

b) **The ordering of messages exchanged concerning a different event**

Since the intention is to order the messages exchanged of different events, the immediately precedes notion refers time interval as the event(s) that occurs for a time interval.
The general form of rule for the immediately precedes notion, which is the make-up of the fact type "event_1 immediately precedes event_2" and the other logical formulations, is as follows:

It is obligatory that exactly one event_1 immediately precedes exactly one event_2

This rule describes that any interaction that relates to event_1 occurs immediate before any interaction that associates with event_2. By exploiting this general form of rule to the given example of Rule 1, Rule 2, Rule 3, and Rule 4 previously, it expresses that the interaction between participant_1 and participant_2 occurs initially, then followed by the interaction between participant_3 and participant_4.

It is worth to note that there is the possibility where the ordering structure of messages exchanged of the events such as sequence, concurrency, or alternative combine activities with other ordering structures of the other events in a nested structure to express the ordering rules of actions performed within a choreography [2]. The immediately precedes notion encapsulates the rule expressing that combination of activities. The same approach for specifying the messages exchanged constraints on events in Section 2.3.2 is applied to specify the aforementioned ordering structures.

The following rules show exactly one event occurs immediate before another events in the ordering structure namely, AND for concurrent, OR and XOR for choices (alternatives), where the logical operation is exploited over event terms. Note that the following rules express for binary events, the same approach and structures are applied for n-ary events.

i. The rule showing exactly one event occurs immediate before another events that use the logical operation AND over event terms.

A. It is obligatory that exactly one event_1 immediately precedes exactly one event_2 and exactly one event_3

This rule is formed by the fact types; i). event_1 immediately precedes event_2, ii). event_1 immediately precedes event_3.

This rule asserts that the messages exchanged that steered by the event_1 occurs immediate before the messages exchanged of the unordered events event_2 and event_3 concurrently.

B. It is obligatory that exactly one event_1 immediately precedes exactly one event_2 that includes exactly one event_a and exactly one event_b

This rule is formed by the fact type event_1 immediately precedes event_2 only, provided that there is a fact type defines i). event_2 includes event_a, ii). event_2 includes event_b. This rule expresses a nesting of event_2, which is the unordered events
**event\textsubscript{a}** and **event\textsubscript{b}** occur concurrently right after the occurrence of the messages exchanged of the **event\textsubscript{1}**.

ii. The rule showing exactly one event occurs immediate before another events that use the logical operation OR over event terms.

A. **It is obligatory that exactly one **event\textsubscript{1}** immediately precedes exactly one **event\textsubscript{2}** or exactly one **event\textsubscript{3}**.**

This rule is the make-up of fact types; i). **event\textsubscript{1}** immediately precedes **event\textsubscript{2}**, ii). **event\textsubscript{1}** immediately precedes **event\textsubscript{3}**.

This rule is employed when there is an inclusive-choice of exchanging the messages of either the **event\textsubscript{2}** or the **event\textsubscript{3}**, or both of them occur concurrently immediate after the occurrence of **event\textsubscript{1}**.

B. **It is obligatory that exactly one **event\textsubscript{1}** immediately precedes exactly one **event\textsubscript{2}** that includes exactly one **event\textsubscript{a}** or exactly one **event\textsubscript{b}**.**

This rule is the make-up of fact type **event\textsubscript{1}** immediately precedes **event\textsubscript{2}** only. This rule has a similar intention as the previous rule although the inclusive-choices of events are specified as nesting events of the **event\textsubscript{2}**.

iii. The rule showing exactly one event occurs immediate before another events that use the logical operation XOR over event terms.

A. **It is obligatory that exactly one **event\textsubscript{1}** immediately precedes exactly one **event\textsubscript{2}** that includes exactly one **event\textsubscript{a}** or exactly one **event\textsubscript{b}** but not both**

This rule is constructed using the fact type **event\textsubscript{1}** immediately precedes **event\textsubscript{2}** only. This rule is benefit to specify the interaction involving the mutually exclusive choices of events that must be performed by the participants occur immediate after the interaction involving the messages exchanged of the **event\textsubscript{1}**.

All preceding rules could be specified in the other way around structure such as **It is obligatory that exactly one **event\textsubscript{1}** that includes exactly one **event\textsubscript{a}** or exactly one **event\textsubscript{b}** but not both immediately precedes exactly one **event\textsubscript{2}**.** This rule expresses that the messages exchanged of the exclusive-choices of the events occurs immediately before the messages exchanged of the **event\textsubscript{2}**.

The additional benefit of the **immediately precedes** notion is the combination of any ordering structure regardless of the messages exchanged constraints, OR, XOR, or AND on event terms (as specified previously), is possible to specify. For instance, the rule:
It is obligatory that exactly one \textit{event}_1 that \textit{includes} exactly one \textit{event}_a or exactly one \textit{event}_b but not both \textit{immediately precedes} exactly one \textit{event}_2 and exactly one \textit{event}_3.

This rule illustrates the combination of the ordering structure; XOR choices on events and AND on events. This rule is useful when there are participants sending or receiving the messages of the \textit{event}_a or the \textit{event}_b inside the \textit{event}_1 set are executed before the executing of the concurrent interaction, \textit{event}_2 and \textit{event}_3. All the possible rules that contain the combination of the ordering structure could be refer in Section 3.1.

The following example shows the interactions as the aforementioned rule illustrating the activities combine the ordering structure XOR choices on events and AND on events.

i. Rule 1:

The export message of the nesting of \textit{participant}_1 sending the \textit{event}_1.

It is obligatory that the \textit{participant}_1 that \textit{includes} exactly one \textit{participant}_a or exactly one \textit{participant}_b but not both \textit{sends} exactly one \textit{event}_1 that \textit{includes} exactly one \textit{event}_a or exactly one \textit{event}_b but not both, \textit{at} exactly one \textit{T1}.

The import message of \textit{participant}_2 and \textit{participant}_3 steered by the same event the nesting of \textit{event}_1.

It is obligatory that the \textit{participant}_2 \textit{receives} exactly one \textit{event}_3 that \textit{includes} exactly one \textit{event}_a \textit{at} exactly one \textit{T2}.

It is obligatory that the \textit{participant}_3 \textit{receives} exactly one \textit{event}_1 that \textit{includes} exactly one \textit{event}_b \textit{at} exactly one \textit{T2}.

Figure 2.13 depicts the \textit{time notion} of this Rule 1; the ordering of messages exchanged between a nesting of \textit{participant}_1, either the \textit{participant}_a or the \textit{participant}_b throughout time interval \textit{T1} and the concurrent interaction between \textit{participant}_2 and the \textit{participant}_3 throughout time interval \textit{T2}.

ii. Rule 2:

The export message of \textit{participant}_4 sending the \textit{event}_2 and \textit{event}_3.

It is obligatory that the \textit{participant}_4 \textit{sends} exactly one \textit{event}_2 and exactly one \textit{event}_3, \textit{at} exactly one \textit{T1}.

The import message of \textit{participant}_5 receiving the same \textit{event}_2 and \textit{event}_3.

It is obligatory that the \textit{participant}_5 \textit{receives} exactly one \textit{event}_2 and exactly one \textit{event}_3, \textit{at} exactly one \textit{T2}.

The \textit{time notion} of Rule 2 in Figure 2.14 shows the interaction starts with the sending of the messages of the unordered events, the \textit{event}_2 and the
CHAPTER 2. LITERATURE REVIEW

Figure 2.13: The illustration of *time notion* for the messages exchanged as in Rule 1

Figure 2.14: The illustration of *time notion* for the messages exchanged as in Rule 2

*event3*, which is executed by the *participant4* throughout time interval *T1*. It is then followed by the receiving of the same unordered events throughout time interval *T2*, which is performed by the *participant5*.

The ordering of the messages exchanged in Rule 1 and Rule 2, as described in the previous rule (the combination of the ordering structure; XOR choices on events and AND on events) can be seen in Figure 2.15. The occurrence of the *event1*, either the *event40* or the *event41* as described in Rule 1 occurs immediate before the occurrence of the *event2* and the *event3* as prescribed in Rule 2.

Developing SBVR rules pertaining to static constraints - Date constraints

Previously, there is the elaboration of designating the fact types for specifying the static constraints for event specifically on Date. In the proposed SBVR model
for service choreographies, the static constraints rule is only considered to specify the Date constraints. However, the idea proposed for the Date constraints can be applied for some other similar domains.

In the SBVR model, all involved dates in the choreography are defined in date set to assemble all the specified dates as it is essential to managing the structure of dates in date set later (see Chapter 4). In date set, there are numerous distinct (unique) dates belong to the particular event, so the identification of an initial and a final date of choreography in the SBVR model is vital for managing the dates in date set purposes. The description on how to initialise the initial and the final date was described previously. The other important aspect for managing the structure of dates is on how to constrain the comparability of the dates in date set specifically when two dates of events must be occurred one after another, or one date of event must equal to another date of event or in opposite way both dates must not equal, or one date of event must be occurred within two other dates. In order to highlight date constraints in a rule of the SBVR model, the supplement to OMG SBVR standard; the DTV [42] is advocated in defining the appropriate vocabulary to the corresponding constraints. This is because of date correlates to time.

Previous, the specification of the static constraints for event was used the general form of fact type "event verb static constraint term" i.e. "airline reservation has outbound date". For date constraints purposes, the synonymous form of the general fact type, namely "date of event" is used to construct a rule constraining date constraints. static constraint term in general form of fact type is replaced to any dates as specified in date set. This fact type is then associates with the appropriate verb (vocabulary) as defined in the DTV and the other fact type.

The general form of rule illustrating date constraints is as follows:

It is obligatory that exactly one date1 of exactly one event verb exactly one date2 of exactly one event.

verb is restricted according to the vocabulary of the corresponding date constraints as defined in [42]. Each fact type, e.g. "date1 of event" refers to time interval as the occurrence of date for the particular event. Note that event could be the same event or the different event.
The date constraints are classified into four categories. There are equality, inequality, in between, and begins before constraints.

1. Equality
   The notion of Allen [44] in the DTV [42], that is "time interval 1 equals time interval 2" is advocated. The general form of rule for specifying date constraints specifically for showing the equality of date is as follows:

   It is obligatory that exactly one date₁ of exactly one event equals exactly one date₂ of exactly one event.

   This notion defines both dates date₁ and date₂ are equal if and only if each is part of the other. It can be illustrated in one of the rules in the Acme Travel case study, It is obligatory that exactly one start date of exactly one reservation request equals exactly one check-in date of exactly one accommodation reservation. This rule constrains both dates of the beginning date of holiday (based on the itinerary as in the reservation request) and the check-in date for any accommodation requested by the customer must be the same date.

2. Inequality
   Inequality constraint for date is the opposite of equality constraint. Since the DTV does not provide the specification for the negation of equality, the negation of obligation, to be specific prohibition modality is employed. The identical notion of equality constraint on date, "time interval 1 equals time interval 2" is applied. The general form of rule for inequality date constraint is as follows:

   It is prohibited that exactly one date₁ of exactly one event equals exactly one date₂ of exactly one event.

   The prohibition modality restrict the specification. The rule prevents the occurrence of two dates at the same time. For instance, the rule from our Acme Travel case study, It is prohibited that exactly one check-in date of exactly one accommodation reservation equals exactly one check-out date of exactly one accommodation reservation restricts the dates for check-in and check-out for any accommodation that is requested by the customer. Both dates must not be on the same date.

3. In between
   The goal of in between notion for date constraint is to restrain the date must be occurred in between to other dates. The notion of "time interval 1 is between time interval 2 and time interval 3" accomplishes in between date constraint goal, which is advocated from the DTV. The general form of rule for this constraint is as follows:
It is prohibited that exactly one \textit{date} \textsubscript{1} of exactly one \textit{event} is between exactly one \textit{date} \textsubscript{2} of exactly one \textit{event} and exactly one \textit{date} \textsubscript{3} of exactly one \textit{event}.

This rule defines that \textit{date} \textsubscript{1} must occur after \textit{date} \textsubscript{2} and before \textit{date} \textsubscript{3}. The rule in the Acme Travel case study, It is obligatory that exactly one \textit{tour date} of exactly one \textit{tour reservation} is between exactly one \textit{start date} of exactly one \textit{reservation request} and exactly one \textit{end date} of exactly one \textit{reservation request} demonstrates the intention of \textit{in between date constraint}. This rule expresses the booking date of tour that is requested by the customer must follow the start date of holiday and precedes the end date of holiday that are decided by the customer.

4. \textbf{Begin before}

There is some situation where the start date of event occurs before or the same as start of another date of event. For instance a date of payment is effective until a date of a deadline. For specifying this kind of date constraints, the notion of "\textit{time interval 1 begins before time interval 2}" in the DTV [42] is advocated. The general form of rule for specifying this date constraint is as follows:

\begin{quote}
It is obligatory that exactly one \textit{date} \textsubscript{1} of exactly one \textit{event} begins before exactly one \textit{date} \textsubscript{2} of exactly one \textit{event}.
\end{quote}

This situation can be seen in our case study, the Tuition Fees. The rule It is obligatory that exactly one \textit{payment date} of exactly one \textit{September first payment instalment} begins before exactly one \textit{deadline date} of exactly one \textit{September first payment instalment} describes the date of instalment payment for September commencing course must be paid before or on the same date of the deadline date.

2.4 Verifying a choreography model using SBVR

Model checking is a method for formally verifying finite-state transition system. The specifications for the system is modelled as a finite state machine and consider all possible behaviours or executions of the system. The specifications are expressed as temporal logic formulas, which are then been checked if the specification holds or not by traversing the reachable states of model. The model checking does not check whether the specification of a system is correct or complete; it is only checking whether a system satisfies its specification or not. If the specifications fails, a counterexample is generated in the representation of states. A counterexample shows whether the states contradicts the specifications or the produced states do not contain and represent the desired specifications. In symbolic model checking, sets of states are represented implicitly using Boolean functions [45,46].
Satisfiability (SAT) solvers is an automated model checking based. SAT solvers is a multi-purpose tool, which has been applied in various different areas, such as in areas software and hardware verification, automatic test pattern generation, planning, scheduling, and even challenging problems from algebra. A SAT solvers also are successful dealing in wide range of problems. It solves the Boolean satisfiability problem. The formalism of specifications in SAT solvers is represented based on propositional logic formula to describe a finite set of states. If the formula is satisfiable, the satisfying assignment gives the desired state, otherwise, the desired state does not exist [46, 47].

Another automated solver based on model checking is Satisfiability Modulo Theories (SMT) solvers. SMT solvers enable applications such as extended static checking, predicate abstraction, test case generation, and bounded model checking over infinite domains. It could check a state machine of a billion states in seconds without any aid from the user. SMT solver is an extension of SAT solver that is used as the verification tool that can reason for the systems that are modelled at a higher level of abstraction than the Boolean level. The language of SMT solvers is first-order logic. The language includes the Boolean operations of Boolean logic, but instead of propositional variables, more complicated expressions involving constant, function, and predicate symbols are used. In SAT solvers, the truth of propositional logic formula is determined by a truth assignment over the propositional symbols, while SMT solvers uses a model (also called a structure) to determine the truth of a first-order logic formula [45].

There are many examples of model checker or SAT solver such as Z specification [48], OCL [49], B specification [50], etc. One of them is Alloy Analyzer [46], which is only one of several approaches to the modeling and analysis of software abstractions. The same features between them are in terms of the declarative specification of behaviours, the structures of states such as sets and relations, and their own syntax and notation are provided to capture software abstractions more succinctly and directly [46].

Alloy Analyzer is also model checking based. It relies on recent advances in SAT technology. The constraints specified in the Alloy Analyzer are solved by translating those constraints from Alloy into Boolean constraints (an off-the-shelf SAT solver). Alloy can scale larger problems, however it examines only a finite space of cases. Alloy language is a simple but expressive logic based on first-order logic and the notion of relations, and was inspired by the Z specification language and Tarskis relational calculus. The similarity for both languages, Alloy and Z is they describing structures. However, Alloys toolkit is smaller and simpler than Zs. Alloy was also strongly influenced by object modelling notations. Alloy Analyzer is an automatic analysis solver that specifies the constraints of a model, describes the constraints structure and finds structures that satisfy them. Structures are visualised graphically and gives immediate feedback.

Alloy is emphasised more on generating an automatic analysis and producing an instance structure of model. The analysis in Alloy is more powerful than the other
approaches’ analyses since Alloy Analyzer does not require the test cases which need a initial conditions and inputs by an user, also it does not restrict the language to an executable subset, and it is more eective at uncovering subtle bugs. Unlike the other approaches which are required to execute a model with a given test cases. However, Alloy is less expressive than the other languages because Alloys structures are strictly first order and relatively poor support for sequences and integers.

The notion of relation in Alloy is closely related to SBVR structure. In general, expressions in Alloy are built using set theoretic relational operators and constants. This means it is an appropriate target for SBVR models which are declarative in nature.

An Alloy model for specifying service choreography consists of a module containing a number of signatures. A signature defines a data domain, which is denoted as sig and represented as a set. A signature may extend another signature, as in the object-oriented paradigm, in which case the domain defined by the first is a subset of the domain of the extended signature. Extensions of a signature are mutually disjoint. A signature can be abstract in which case its domain only contains elements that belong to its extending signatures. sig or abstract sig in Alloy represents two main components in the SBVR model which play an important role in modelling the choreographies (as described in Chapter 2 (Section 2.3.2)). These main components are participants as the participating services involve in the collaboration and events as the exchanging messages which are performed amongst the participants. For instance, the participant participant\textsubscript{1} in the SBVR model is represented as "sig participant\textsubscript{1}". Both of them give the same illustration of each participant\textsubscript{1} who sending or receiving the event in the system. On the other hand, abstract sig presents as a superset either for the participants or the events. For example, the service provider participant\textsubscript{1} which acts as a superset encompasses participant\textsubscript{a} and participant\textsubscript{b}. This participant\textsubscript{1} will be characterised as "abstract sig participant\textsubscript{1}" in Alloy. This will discuss more details in Section 4.2.

The multiplicities one means exactly one, lone means zero or one, and some means one or more in Alloy have been employed to constrain each signature. For instance, the participant\textsubscript{1} in the SBVR rule of the SBVR model is described as "one sig participant\textsubscript{1}" in Alloy. In addition, the multiplicities are applied to illustrating the accurate meaning of the complex interactions as well as the ordering of messages exchanged in the choreography. This will discuss further in Section 4.4.

A signature introduces fields which are captured by relations. As a principle in Alloy, the declaration of field such as "sig participant\textsubscript{1} verb event\textsubscript{1}" describes a relation "verb" (field) whose domain is "participant\textsubscript{1}" and whose range is given by the expression "event\textsubscript{1}". It is worth to note that the "event\textsubscript{1}" is actually another signature in Alloy. Basically this notion of relation in Alloy illustrates the fact type in the SBVR model which is formed of the interconnection between terms and verb such as the fact type participant\textsubscript{1} verb event\textsubscript{1}. participant\textsubscript{1} is represented as "sig participant\textsubscript{1} verb event\textsubscript{1}"}, {"verb" is a field "verb", and "event\textsubscript{1}" is denoted as "event\textsubscript{1}"
which is the other signature in Alloy.

Axioms in Alloy are called facts and predicates. Both can be given a name, record various forms of constraints and expressions. The difference is a fact assumes the constraints always hold, while a predicate only holds when it is invoked. For specifying choreography, we exploit the fact to a certain case (e.g. alternative and concurrent interaction) for a particular signature. On the other hand, the predicate is employed to generate a choreography and verify the realisabilty.

The combination of signature, multiplicities, and field are the basis of the development of SBVR rule in Alloy. For example, the SBVR rule It is obligatory that the \( \text{participant}_1 \) \( \text{verb} \) exactly one \( \text{event}_1 \) is translated as "one sig participant_1: verb one event_1". This will be discussed further in Chapter 4.

2.5 Discussion

In this PhD research work, a declarative approach the OMG standard SBVR, in the natural language structured, is proposed to be advocated for modelling service choreographies and subsequently generating service choreographies. This approach is targeted to describe a set of global constraints capturing the ordering of service interactions and the complex interactions including alternative and concurrent interactions. The service interactions are between the heterogenous distinct participant services. This proposed model for service choreographies is aimed to be used by the end-users (business analysts, system analysts, stakeholders (non-experts)) to devise and set up the service choreographies. Thus, users can validate the requested service interactions by directly reading the structured natural language used by SBVR. The idea is that this can then be parsed and executed by a machine for generating the choreography and the choreography verification automatically.

There are some works on SBVR since it is gaining ground in systems specification and modelling due to its declarative nature. The work in [51] translates UML / OCL into SBVR to bridge the gap between developers and business users. The same purpose has been focused in the work on the transformation from BPMN2 into SBVR [52, 53]. In this work, SBVR is represented as the state (before and after, i.e. sends and receives by the same participant services), however the ordering of the overall collaborations are not considered. Similarly, the verification of the given SBVR from the given BPMN is not provided, hence, the business processes cannot be verified.

[54] translates natural language specification to SBVR. This was aimed to benefit a business rule analyst to automate the business rules translation into SBVR language. The business rules must respect to a target business domain which is provided by a UML class model. Nevertheless, in this work, the SBVR rules capture only the basic binary facts of rules that include modality, noun, verb concept, logical operation, and quantification.

On the other hand, since Object Constraint Language (OCL) [49] provides a complex syntax making the writing and the interpretation of OCL codes are difficult
to user, [55] translates SBVR to OCL [49] to improve OCL usability. The SBVR rules in this work map only those SBVR’s meaning metamodel that had equivalent elements in OCL metamodel. Similar to the previous work, SBVR rules do not concern on the ordering of interactions even no specification considering on the complex interactions.

[56] translates SBVR into a process BPMN and decision model which is used to validate SBVR rules, however the validation of SBVR rules limits to people having a knowledge about the notations of BPMN and DMN like business analysts, software engineers, or other people familiar with the processes and decisions in the company. Hence, the validating SBVR rules benefits to people who know BPMN and DMN only. The SBVR rules in this work consider a binary fact only. It uses if-condition to show service control-flow between participants or within the same participants. However, no verification on conformance and realisability, are provided.

In [57] SBVR is used to generate a relational database and transform SBVR business rules to SQL queries that can be executed against the data set. The purpose of this work is to validate the consistency of a given data set with the SBVR model.

[8] provides a first-order deontic-alethic logic (FODAL) to express business constraints defined in SBVR and perform a consistency check on the rule set, including alethic and deontic rules.

Most of the stated work on exploiting the OMG standard SBVR focuses on a transformation of SBVR from (to) the other well-known specification language for modelling service choreographies. The use of SBVR rules are only concerned on the basic specification such as a sending (receiving) of a message, the static constraint, some use SBVR rules for specifying conjunction and disjunction rule but some are not, and not concerned on the ordering of service interactions between autonomous participant services. In addition, in terms of verification, most stated work focuses the reasoning about the consistency of the rule set, which of course is an important aspect of verification, and less on explicitly capturing the orderings in terms of observable message exchanges.

In contrast, in this PhD research work, the OMG standard SBVR is advocated to develop a model for service choreographies. In the proposed approach the construction of rule restricted to deontic rules - that is, obligation - as the target is service choreographies and obligation is adequate. The deontic modality also covers prohibition which can be useful in working with choreographies. The semantic formulations introduced in SBVR standard are applied, which are aimed for specifying the complex interactions including a single or multiple participant services (the conjunction, the inclusive-disjunction, and the exclusive-disjunction on participant services) perform the interaction by sending (receiving) a single or multiple events (the conjunction, the inclusive-disjunction, and the exclusive-disjunction on events), the ordering of the interactions between the participant services, and the static constraints. This choreography model is targeted to be generated and verified in terms of conformance and realisability.
Since the SBVR standard is in the form of formal logic, the produced choreography model is aimed to be parsed to the constraint solver, Alloy Analyzer. This is purposely for generating and verifying choreography automatically. The idea is to automate the transformation process from the generated SBVR rules into Alloy codes by implementing a new tool.

The Alloy Analyzer has been used in a range of application domains. Of relevance here perhaps is the work [58,59] where Alloy is used for composing behavioural models from individual UML 2 sequence diagrams. The semantics of the composition applied labelled event structures to check the correctness of the composition process. Allow indicates the error if the conflicting statements are encountered. The tool SD2Alloy was introduce in these works. This tool captures the composition of sequence diagrams which the interactions are not described dynamically and the verification is not concerned on the conformance and realisability.

The other related work is [60]. This work generates OCL via SBVR from English text and then transform these constraints into Alloy. The generated Alloy code is used to check whether or not the original natural language constraints are consistent. The method suggested here is particularly applicable in early stages of software development during gather the informal requirements only. [61] also applies Alloy to analyse a pattern-based solution for the components of an e-commerce system which is modelled using UML use-case diagrams and activity diagrams. However, they focus only on the static design model.

The approach of implementation the new tool for automating the generating choreography model from the developed model using SBVR and verifying the correspond generated choreography model is closely related to only work in [21,62–64] which developed a tool for realising service choreographies. The coordinating service interactions is constructed using BPMN2 notation. However, the coordination could not capture the heterogenous interactions because it does not consider and does not provide a method on how to map the sent and the received messages with the different participant services. The automated realisation is only on the domain-specific interaction, not all possible collaborations can be automatically realised. This contrasts with the research proposed, where Alloy can generate the codes as similar as the specified SBVR rules in the choreography model which encapsulates the dynamic interaction between autonomous participant services. The verification is not only on the static constraints but it is aimed to facilitate the conformance and realisability verification.

The adoption of declarative approach, the OMG standard SBVR model poses some challenging questions:

Can an SBVR model be developed to specify the dynamic collaboration between heterogenous participant services? How is it possible to capture the specification involving the precedence, alternative, concurrent interactions and static constraints? Can the choreography model be generated using the proposed SBVR model without applying imperative approach, while still being declarative? How can the proposed
approach support the choreography verification for conformance and realizability? Can the choreography be generated automatically? Can the end-user apply the proposed model to devise and verify the choreography?

All answers are provided in the next chapters encompass Chapter 3 for developing the choreography model using the OMG standard SBVR, Chapter 4 the transformation the choreography model from SBVR into Alloy Analyzer and Chapter 5 the implementation of tool to automate the transformation between SBVR and Alloy models.
Chapter 3

Model for Service Choreography

The SBVR model is proposed to coordinate the collaboration between autonomous participant services (services choreography) by advocating the OMG standard SBVR [7] which expresses the business rules in a declarative manner and in natural language. This SBVR model is aimed to produce a choreography model that describes a set of global constraints that governs the ordering of messages exchanged between participant services.

In this chapter, a framework for the development of the SBVR model to specify a set of global constraints relating to generating a choreography is introduced. Its application in a representative case study is then demonstrated, and its style of coordinating in the light of online business applications across different participant services (business applications/organisations) is discussed.

3.1 A framework for developing choreography model

A framework for a development of the SBVR model is introduced in this section as shown in Table 3.1. The intended terms, fact types as well as the rules are discussed in the succeeding sections.

The designation of terms as described in Section 2.3.2, which are classified into participant, event, static constraint and time terms are essential to the construction of the fact types and subsequently for the rules. The framework is divided into three sections.

Section I a section to specify all the involving terms in the SBVR model. Participant terms demonstrate all participants who take part in the choreography, which are grouped as the participant set. The messages describing the interaction between distinct participants are designated as the event terms form the event set. All the involved static constraints characterising the domain-specific constraints for each participant (if any) and event (if any) are put
<table>
<thead>
<tr>
<th>TERM</th>
<th>FACT TYPE</th>
<th>RULE</th>
</tr>
</thead>
<tbody>
<tr>
<td>-Participant set</td>
<td>-Add your participant fact types here:</td>
<td>-Add your rules here:</td>
</tr>
<tr>
<td>-Add your participant terms here:</td>
<td>Fact Type:</td>
<td>Rule:</td>
</tr>
<tr>
<td>Term:</td>
<td>-Event set</td>
<td>-Add static constraints rules here (if any):</td>
</tr>
<tr>
<td>-Add your event terms here:</td>
<td>Fact Type:</td>
<td>Rule:</td>
</tr>
<tr>
<td>Term:</td>
<td>-Constraint</td>
<td>-A set of date:</td>
</tr>
<tr>
<td>-Add your static constraint terms here:</td>
<td>Fact Type:</td>
<td>Fact Type:</td>
</tr>
<tr>
<td>Term:</td>
<td>-Time</td>
<td>-Initialisation for date:</td>
</tr>
<tr>
<td>Term:</td>
<td>Term: T</td>
<td>Fact Type:</td>
</tr>
</tbody>
</table>

Table 3.1: A framework template for developing the SBVR model for service choreographies
together under static constraint terms. Note that time term $T$ represents any time interval, i.e $T_1, T_2,..., T_n$, which are used to show the ordering of messages exchanged of the same event.

**Section II** a section to specify the fact types, which are separated into three sections.

- Section I contains the fact types describing the participant and event set as well as their nesting of group (if any) as described in Section 2.3.2. The fact types for participant represent the local behaviour model of each participant including a set of messages exchanged containing the export and the import messages of the event, and the domain-specific constraints. The event fact types specify all events involve in the interaction between participants together with their own static constraints (if any). The fact type declares which event plays an important role as the initial event in the choreography is defined under initialisation part.

- Section II consists of the fact types specify the rules constraining the ordering of messages exchanged, relating to the time notion and the immediately precedes notion as discussed in Section 2.3.2.

- Section III refers to static constraints for events specifically on constraining dates of the particular events. In this section, there are fact types define a date set containing multiple distinct dates, an initialisation of dates assigning an initial date and a final date for the given date set, and date constraints which contribute to the construction of rules to constrain the dates in date set.

**Section III** a section to specify the rule (the global constraints) of the SBVR model which captures all complex interaction constraints include to specify concurrent and alternative interactions using the logical operations, the ordering of messages exchanged of the events, and the static constraints encapsulate the date constraints.

### 3.1.1 A generic SBVR model

In this section, a description of a step by step approach for developing the SBVR model is elaborated, which follows the proposed framework in Table 3.1. The general forms of the particular terms, fact types and rules which was discussed previously throughout this chapter are provided. These general forms summarise the requisite structure of terms, fact types, and rules in prescribing a set of global constraints that govern the allowed interactions and the ordering of the messages exchanged (interaction) amongst the participants. There are three steps are as follows:
Step 1: Designation of terms for participants, events, static constraints, and time

**Participant terms** The framework in Table 3.1 starts with specifying participant terms, which are all the participating services involved in the multiparty conversation. With reference to Table 3.2, note that subscript letters refer to participants that are grouped (or included) within other participant (a nesting of group). $participant_N$ refers to $N$ number of participants involved, while $participant_n$ indicates $n$ number of participants that are grouped in a nesting of group.

---

<table>
<thead>
<tr>
<th>Participant Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add your participant terms here:</td>
</tr>
<tr>
<td>Term: $participant$</td>
</tr>
<tr>
<td>Term: $participant_1$</td>
</tr>
<tr>
<td>Term: $participant_2$</td>
</tr>
<tr>
<td>.....</td>
</tr>
<tr>
<td>.....</td>
</tr>
<tr>
<td>Term: $participant_N$</td>
</tr>
<tr>
<td>Term: $participant_a$</td>
</tr>
<tr>
<td>Term: $participant_b$</td>
</tr>
<tr>
<td>.....</td>
</tr>
<tr>
<td>Term: $participant_n$</td>
</tr>
</tbody>
</table>

Table 3.2: Participant terms in the SBVR model

**Event terms** Once the participant terms are designated, it is then followed by the designation of event terms. Event terms denote all the events that capture messages sent and received by the participants in the multi-party conversation. Table 3.3 depicts $N$ number of events involved, i.e. $event_1$, ..., $event_N$. Note that subscript letters in this case refer to events that are grouped (or included) within other events (a nesting of group). $event_n$ refers to $n$ number of events that are grouped within other event (if any).

**Static constraint terms (if necessary)** These terms will be associated with any participants or any events in the fact types later such as $static_1$ up to $n$ number of static constraint $static_n$ in Table 3.4.

**Time term** It suffices specifying Term: $T$ only as it captures all time intervals $T_1$, $T_2$, ..., $T_n$ in our editor www.sbvr.co.
3.1. A FRAMEWORK FOR DEVELOPING CHOREOGRAPHY MODEL

---

Event Set
Add your event terms here:

Term: \textit{event}
Term: \textit{event}_1
Term: \textit{event}_2
....
....
Term: \textit{event}_N
Term: \textit{event}_a
Term: \textit{event}_b
....
....
Term: \textit{event}_n

---

Table 3.3: Event terms in the SBVR model

Constraint
Add your static constraint terms here:

Term: \textit{static}_1
Term: \textit{static}_2
....
....
Term: \textit{static}_n

---

Table 3.4: Static constraint terms in the SBVR model

Step 2: Designation of fact types for specifying participant set, event set, and local behaviour model for each participant

Fact types for specifying participant and event set Once all possible terms are defined, the fact types are specified subsequently. These fact types specify the participant set includes all the participant entities involved in the interactions, and any included participants. Also, the fact types specify the event set and included events (if any). The general forms in Table 3.5 shows the participant set contains $N$ number of participants, i.e. participant$_1$, ..., participant$_N$. Also, there are $n$ number of participants, which are group together in the participant$_3$ set.

event set encompasses event$_1$, event$_2$, ..., event$_N$. Also, there is a specifi-
cation of a group of nesting \textit{event}_3 comprises \( n \) of events \textit{event}_a, \textit{event}_b, ..., \textit{event}_n.

<table>
<thead>
<tr>
<th>Fact Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add your participant fact types here:</td>
</tr>
<tr>
<td>Fact Type: \textbf{participant} includes participant(_1)</td>
</tr>
<tr>
<td>Fact Type: \textbf{participant} includes participant(_2)</td>
</tr>
<tr>
<td>Fact Type: \textbf{participant} includes participant(_3)</td>
</tr>
<tr>
<td>Fact Type: participant(_3) includes participant(_a)</td>
</tr>
<tr>
<td>Fact Type: participant(_3) includes participant(_b)</td>
</tr>
<tr>
<td>Fact Type: participant(_3) includes participant(_n)</td>
</tr>
<tr>
<td>Fact Type: participant .....</td>
</tr>
<tr>
<td>Fact Type: \textbf{participant} includes participant(_N)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Fact Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add your event fact types here:</td>
</tr>
<tr>
<td>Fact Type: \textbf{event} includes event(_1)</td>
</tr>
<tr>
<td>Fact Type: \textbf{event} includes event(_2)</td>
</tr>
<tr>
<td>Fact Type: \textbf{event} includes event(_3)</td>
</tr>
<tr>
<td>Fact Type: event(_3) includes event(_a)</td>
</tr>
<tr>
<td>Fact Type: event(_3) includes event(_b)</td>
</tr>
<tr>
<td>Fact Type: event(_3) includes event(_n)</td>
</tr>
<tr>
<td>Fact Type: event .....</td>
</tr>
<tr>
<td>Fact Type: \textbf{event} includes event(_N)</td>
</tr>
</tbody>
</table>

Table 3.5: Participant set and event set fact types in the SBVR model

**Fact types for specifying messages exchanged and static constraints (if any)**

These fact types specify the local behaviour model for each participant. It consists of a set of messages exchanged, import and export messages (a sending or a receiving of a message) and the static (domain-specific constraints). The fact types describe each participant providing a service which is associated with event at a given time interval. Also, the fact types describe any static constraints that belongs to any event.

Table 3.6 shows local behaviour model for \textit{participant\(_1\)} and the constraint belongs to \textit{event\(_1\)}. Note that the local behaviour model of \textit{participant\(_1\)} which associates with \textit{event\(_1\)}, \text{T} refers to \textit{T}_1, or \textit{T}_2, or \textit{T}_3, \textit{static\(_1\)} and the specification of \textit{event\(_1\)} relating to \textit{static\(_1\)} exemplify the fact types that are used for specifying messages exchanged.

**Fact types for specifying the ordering of time and events** The ordering of
3.1. A FRAMEWORK FOR DEVELOPING CHOREOGRAPHY MODEL

Table 3.6: The messages exchanged fact types in the SBVR model

<table>
<thead>
<tr>
<th>Fact Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| **participant** includes **participant** | –Fact Types
| –Add your participant fact types here: | –Local behaviour model of participant 1: |
| **participant** includes **participant** | –Export messages: |
| **participant** verb **event** at **T** | –Import messages: |
| **participant** verb **event** at **T** | –Static constraints |
| **participant** verb **static** | Fact Type: **participant** includes **participant** |
| **participant** includes **participant** | –Local behaviour model of participant 1: |
| –Local behaviour model of participant N: | –Add your event fact types here: |
| –1. event 1 | –1. event 1 |
| **event** includes **event** | **event** includes **event** |
| **event** verb **static** | **event** verb **static** |
| –2. event 2 | –2. event 2 |
| **event** includes **event** | **event** includes **event** |
| ...... | ...... |
| –3. event N | –3. event N |
| **event** includes **event** | **event** includes **event** |

Table 3.6: The messages exchanged fact types in the SBVR model

time fact types describe the sequence of time interval associating with the same particular event. Table 3.7 represents \( n \) number of time intervals which are connected with \( \text{event}_1 \).

The ordering of events will involve the logical operations such as OR, XOR and AND to represent the different modes of interaction; sequential, alternative and concurrent interactions. For specifying fact types, the logical operations, however any constraints (rules) that require logical operations will have different fact types separately. \( \text{event}_1 \) immediately precedes \( \text{event}_2 \) or \( \text{event}_3 \) or ... or \( \text{event}_N \) will be specified in separate fact types as shown in the following Table 3.7. Note that the logical operation or could be replaced with and or or...but not all (but not both for binary events).

Nevertheless, if the ordering of event involves a nesting of event group, only
one fact type is specified regardless number of events inside the intended nesting group. $event_5$ immediately precedes $event_6$ that includes $event_n$ or ... or $event_m$ will be specified in the fact type as depicted in Table 3.7. Note that the logical operation or could be replaced with and or or...but not all (but not both for binary events).

| –Add your ordering of time fact types here: |
| Fact Type: $event_1$ at $T_1$ immediately precedes $event_1$ at $T_2$ |
| ...... |
| Fact Type: $event_1$ at $T_{n-1}$ immediately precedes $event_1$ at $T_n$ |

| –Add your ordering of events fact types here: |
| Fact Type: $event_1$ immediately precedes $event_2$ |
| Fact Type: $event_1$ immediately precedes $event_3$ |
| ...... |
| ...... |
| Fact Type: $event_1$ immediately precedes $event_N$ |
| ...... |
| Fact Type: $event_5$ immediately precedes $event_6$ |

Table 3.7: The ordering of time and event fact types in the SBVR model

**Fact Types for specifying static constraints particularly for dates** Fact types are specifically used to define all involving dates which associate with the particular event in the SBVR model. Note that each date must be named with a unique name. Table 3.8 demonstrates $n$ number of dates which are grouped in a date set. Only one date in date set is defined as an initial date and another one is a final date. Date constraint fact types will be employed for specifying date constraint rule later which relates to equality, inequality, in between, and begin before.

**Step 3: Developing SBVR rules**

Specifying the SBVR rules as specified in the informal requirements of the system. The rule is a combination of terms, fact types, modalities, quantification, and logical operation (if any). The construction of rules follow the principles structures of rules as discussed in Section 2.3.2. SBVR rules in the SBVR model are divided into four main rules.

**SBVR rules for specifying a sending or a receiving of event(s) by participant(s)** These rules are separated into two types of rules.

**Type 1: Participant is a single participant and event is a single or multiple events** These rules specify only one participant sending (receiving) the messages
### 3.1. A FRAMEWORK FOR DEVELOPING CHOREOGRAPHY MODEL

- Add your static constraints fact types here (if any):
  - A set of date:
    - Fact Type: *date*<sub>1</sub> is in *date*<sub>1</sub>
    - Fact Type: *date*<sub>2</sub> is in *date*<sub>2</sub>
    - ...
    - Fact Type: *date*<sub>n</sub> is in *date*<sub>n</sub>

- Initialisation for date:
  - Fact Type: *date*<sub>1</sub> is initial date
  - Fact Type: *date*<sub>n</sub> is final date

- Date constraints:
  - Fact Type: *date*<sub>1</sub> of event<sub>1</sub> equals *date*<sub>2</sub> of event<sub>2</sub>
  - ...
  - Fact Type: *date*<sub>1</sub> of event<sub>1</sub> is between *date*<sub>2</sub> of event<sub>2</sub> and *date*<sub>3</sub> of event<sub>3</sub>
  - ...
  - Fact Type: *date*<sub>1</sub> of event<sub>1</sub> begins before *date*<sub>2</sub> of event<sub>2</sub>

Table 3.8: The date constraint fact types in the SBVR model

of the event(s) at the given time interval *T*, where *T* refers to *T*<sub>1</sub>, or *T*<sub>2</sub>.. or *T*<sub>n</sub>. Event could be a single event or multiple events connecting with the logical operation. Table 3.9 shows the general forms of rules of Type 1.

The quantification the for participant could be replaced with exactly one. Note that in some circumstances, only part of events inside a nesting of event group is chosen in the interaction. According to the above specification, event<sub>1</sub> set contains event<sub>a</sub>, event<sub>b</sub> up to event<sub>n</sub>. However, only event<sub>a</sub> is selected to be performed by the participant. Hence the rule It is obligatory that the participant<sub>1</sub> verb exactly one event<sub>1</sub> that includes exactly one event<sub>a</sub> at exactly one *T* is specified.

In addition, it is worth to note that the logical operation or..but not both is employed to specify the messages exchanged of exclusive-choices of two (binary) events. On the other hand, the logical operation or..but not all expresses exclusive-choices of *n* number of events.

**Type 2: Participant is multiple participants and event is a single or multiple events**

Multiple participants are specified as described in participation constraints section in which the logical operation is designated between
- Add your rules here:
- Participant sending (receiving) a single event.
  Rule 1: It is obligatory that the participant verb exactly one event at exactly one T
  Rule 2: It is obligatory that the participant verb at least one event at exactly one T
  Rule 3: It is obligatory that the participant verb at most one event at exactly one T

- Participant sending (receiving) multiple events with the logical operation OR over event terms. Note that each T which associates with each event is identical.
  Rule 4: It is obligatory that the participant verb exactly one event at exactly one T or exactly one event at exactly one T or ... or exactly one event at exactly one T
  Rule 5: It is obligatory that the participant verb exactly one event that includes exactly one event or exactly one event or ... or exactly one event at exactly one T

- Participant sending (receiving) multiple events with the logical operation AND over event terms.
  Rule 6: It is obligatory that the participant verb exactly one event and exactly one event and ... and exactly one event at exactly one T
  Rule 7: It is obligatory that the participant verb exactly one event that includes exactly one event and exactly one event and ... and exactly one event at exactly one T

- Participant sending (receiving) multiple events with the logical operation XOR over event terms.
  Rule 8: It is obligatory that the participant verb exactly one event that includes exactly one event or exactly one event but not both at exactly one T
  Rule 9: It is obligatory that the participant verb exactly one event that includes exactly one event or exactly one event or ... or exactly one event but not all at exactly one T

Table 3.9: Rules for specifying a messages exchanged by a single participant in the SBVR model
3.1. A FRAMEWORK FOR DEVELOPING CHOREOGRAPHY MODEL

Events are specified as in Type 1. Type 2 of rules are illustrated in Table 3.10 for inclusive-choice (OR) of participants, Table 3.11 is for AND of participants, and Table 3.12 is for exclusive-choice (XOR) of participants.

Table 3.10: Rules for specifying a messages exchanged by multiple participants (OR of participants) in the SBVR model

SBVR rules for specifying the ordering of events In order to specify the ordering of messages exchanged of different events between participants, the following general form of rule is employed.

Rule: It is obligatory that exactly one \( term_1 \) immediately precedes exactly one \( term_2 \).

\( term_1 \) and \( term_2 \) refer to event terms. We assume that the left hand side statement ”exactly one \( term_1 \)” as \( P \), and the right hand side statement ”exactly one \( term_2 \)” as \( Q \). \( P \) and \( Q \) statements could be combined with any statements as specified in the Table 3.13. For instance, the rule It is
Add your rules here:
The logical operation AND over participant terms. Participants sending (receiving) a single event or multiple events.

Rule 1: It is obligatory that the participant\textsubscript{1} and the participant\textsubscript{2} and...and the participant\textsubscript{n}, \textit{verb} exactly one event\textsubscript{1} at exactly one T

**(Note: the quantification exactly one which associates with event\textsubscript{1} could be replaced with at least one or at most one).**

Rule 2: It is obligatory that the participant\textsubscript{1} \textit{that includes} the participant\textsubscript{a} and the participant\textsubscript{b} and...and the participant\textsubscript{n}, \textit{verb} exactly one event\textsubscript{1} at exactly one T

**(Note: the quantification exactly one which associates with event\textsubscript{1} could be replaced with at least one or at most one).**

-event\textsubscript{1} comprises \textit{n} number of events. If event\textsubscript{1} comprises binary events event\textsubscript{a} and event\textsubscript{b}, but not all must be replaced with but not both.

Rule 3: It is obligatory that the participant\textsubscript{1} and the participant\textsubscript{2} and...and the participant\textsubscript{n}, \textit{verb} exactly one event\textsubscript{1} \textit{that includes} exactly one event\textsubscript{a} or exactly one event\textsubscript{b} or... or exactly one event\textsubscript{n} but not all, \textit{at} exactly one T

Rule 4: It is obligatory that the participant\textsubscript{1} \textit{that includes} the participant\textsubscript{a} and the participant\textsubscript{b} and...and the participant\textsubscript{n}, \textit{verb} exactly one event\textsubscript{1} \textit{that includes} exactly one event\textsubscript{a} or exactly one event\textsubscript{b} or... or exactly one event\textsubscript{n} but not all, \textit{at} exactly one T

Table 3.11: Rules for specifying a messages exchanged by multiple participants (AND of participants) in the SBVR model

obligatory that exactly one event\textsubscript{1} \textit{immediately precedes} exactly one event\textsubscript{1} \textit{that includes} exactly one event\textsubscript{a} or exactly one event\textsubscript{b} but not both is a combination of Rule 1 and Rule 5 (binary events) in table.

SBVR rules for specifying the ordering of time intervals These rules specify the ordering of messages exchanged which is steered by the same event by the participant(s) as illustrated in Table 3.14.

SBVR rules for specifying static constraints specifically on dates Table 3.15 shows the static constraint rules constraining the equality and inequality of two different dates relating to the identical of different events, the date of the event that must be in between two other different dates associate with the event(s), and the date of the event must be begins before or at the same date of the other event. Note that, the prohibition modality is used to constrain the inequality of two dates.
3.1. A FRAMEWORK FOR DEVELOPING CHOREOGRAPHY MODEL

---

- Add your rules here:
- The logical operation XOR over participant terms. Participants sending (receiving) a single event or multiple events.
- \(\text{participant}_1\) comprises \(n\) number of participants. If \(\text{participant}_1\) comprises binary participants \(\text{participant}_a\) and \(\text{participant}_b\), but not all must be replaced with but not both.

**Rule 1:** It is obligatory that the \(\text{participant}_1\) that includes the \(\text{participant}_a\) or the \(\text{participant}_b\) or...or the \(\text{participant}_n\) but not all, \(\text{verb} \) exactly one \(\text{event}_1\) at exactly one \(T\)

**(Note: the quantification exactly one which associates with \(\text{event}_1\) could be replaced with at least one or at most one).**

- \(\text{event}_1\) comprises \(n\) number of events. If \(\text{event}_1\) comprises binary events \(\text{event}_a\) and \(\text{event}_b\), but not all must be replaced with but not both.

**Rule 2:** Rule: It is obligatory that the \(\text{participant}_1\) that includes the \(\text{participant}_a\) or the \(\text{participant}_b\) or...or the \(\text{participant}_n\) but not all, \(\text{verb} \) exactly one \(\text{event}_1\) that includes exactly one \(\text{event}_a\) or exactly one \(\text{event}_b\) or ... or exactly one \(\text{event}_n\) but not all, \(\text{at} \) exactly one \(T\)

**Rule 3:** It is obligatory that the \(\text{participant}_1\) that includes the \(\text{participant}_a\) or the \(\text{participant}_b\) or...or the \(\text{participant}_n\) but not all, \(\text{verb} \) exactly one \(\text{event}_1\) at exactly one \(T\) or exactly one \(\text{event}_2\) at exactly one \(T\) or ... or exactly one \(\text{event}_N\) at exactly one \(T\)

**Rule 4:** It is obligatory that the \(\text{participant}_1\) that includes the \(\text{participant}_a\) or the \(\text{participant}_b\) or...or the \(\text{participant}_n\) but not all, \(\text{verb} \) exactly one \(\text{event}_1\) that includes exactly one \(\text{event}_a\) or exactly one \(\text{event}_b\) or ... or exactly one \(\text{event}_n\), \(\text{at} \) exactly one \(T\)

**Rule 5:** It is obligatory that the \(\text{participant}_1\) that includes the \(\text{participant}_a\) or the \(\text{participant}_b\) or...or the \(\text{participant}_n\) but not all, \(\text{verb} \) exactly one \(\text{event}_1\) at exactly one \(T\) and exactly one \(\text{event}_2\) at exactly one \(T\) and ... and exactly one \(\text{event}_N\) at exactly one \(T\)

**Rule 6:** It is obligatory that the \(\text{participant}_1\) that includes the \(\text{participant}_a\) or the \(\text{participant}_b\) or...or the \(\text{participant}_n\) but not all, \(\text{verb} \) exactly one \(\text{event}_1\) that includes exactly one \(\text{event}_a\) and exactly one \(\text{event}_b\) and ... and exactly one \(\text{event}_n\), \(\text{at} \) exactly one \(T\)

---

Table 3.12: Rules for specifying a messages exchanged by multiple participants (XOR of participants) in the SBVR model
–Add your rules here:

**Rule 1:** It is obligatory that exactly one \( \text{event}_1 \) ....

**Rule:** It is obligatory that exactly one \( \text{event}_1 \) or exactly one \( \text{event}_2 \) or ... or exactly one \( \text{event}_N \) ....

**Rule 2:** It is obligatory that exactly one \( \text{event}_1 \) that includes exactly one \( \text{event}_a \) or exactly one \( \text{event}_b \) or ... or exactly one \( \text{event}_n \) ....

**Rule 3:** It is obligatory that exactly one \( \text{event}_1 \) and exactly one \( \text{event}_2 \) and ... and exactly one \( \text{event}_N \) ....

**Rule 4:** It is obligatory that exactly one \( \text{event}_1 \) that includes exactly one \( \text{event}_a \) and exactly one \( \text{event}_b \) and ... and exactly one \( \text{event}_n \) ....

\(-\text{event}_1\) comprises \( n \) number of events. If \( \text{event}_1\) comprises binary events \( \text{event}_a \) and \( \text{event}_b \), but not all must be replaced with but not both.

**Rule 5:** It is obligatory that exactly one \( \text{event}_1 \) that includes exactly one \( \text{event}_a \) or exactly one \( \text{event}_b \) or ... or exactly one \( \text{event}_n \) but not all

---

**Table 3.13:** Rules for specifying the ordering of events in the SBVR model

---

–Add your rules here:

**Rule 1:** It is obligatory that exactly one \( \text{event}_1 \) at \( T_1 \) immediately precedes exactly one \( \text{event}_1 \) at \( T_2 \) ....

....

**Rule 2:** It is obligatory that exactly one \( \text{event}_1 \) at \( T_{n-1} \) immediately precedes exactly one \( \text{event}_1 \) at \( T_n \).

---

**Table 3.14:** Rules for specifying the ordering of time intervals in the SBVR model
3.1. A FRAMEWORK FOR DEVELOPING CHOREOGRAPHY MODEL

---Add your rules here:

Rule 1: It is obligatory that exactly one date of exactly one event equals exactly one date of exactly one event.

Rule 2: It is prohibited that exactly one date of exactly one event equals exactly one date of exactly one event.

Rule 3: It is prohibited that exactly one date of exactly one event is between exactly one date of exactly one event and exactly one date.

Rule 4: It is prohibited that exactly one date of exactly one event begins before exactly one date of exactly one event.

---Table 3.15: Rules for specifying static constraints of dates in the SBVR model

3.1.2 A specific request on the generic SBVR model

The specific request refers to a request for the specific event(s) which are sent or received by the particular participant(s). The specified specific request rule is then can be verified whether it can be fulfilled by the developed SBVR model for service choreography. This will be discussed further in Chapter 4 for the realisability purpose. The specific request is expressed in a rule together with the related fact types. This specific request rule is specified in the developed generic SBVR model under a specific request section as shown in Table 3.16.

---Add your specific request here:

Fact Type: ...
....
....
Fact Type: ...

Rule: It is obligatory that ...

---Table 3.16: Specific request section in the SBVR model

In the previous section, Step 3 includes two types of rule structures for specifying the sending or the receiving of the event(s) by the participant(s) in the SBVR model. The particular rule as specified in the developed SBVR model which employs those types of rule structures, namely Type 1 or Type 2 being a reference for constructing the specific request rule. For instance, in the Acme Travel case study,
there is a rule It is obligatory that the *Acme Travel* requests for exactly one *airline reservation* or exactly one *accommodation reservation* or exactly one *transport reservation* or exactly one *tour reservation*. This rule uses Type 1 rule structure being a reference for specifying the specific request such as "the Acme travel requests for either the airline reservation only (no other reservations); or the airline reservation and the accommodation reservation, that is the hotel reservation", which is purposely to know whether the specific request can be fulfilled by the prescribed messages exchanged of collaboration of the participants. The following fact types and rule will be specified under the specific request section in the SBVR model.

---

**Fact Type:** Acme Travel requests for airline reservation at $T_i$

**Fact Type:** Acme Travel requests for accommodation reservation at $T_i$

**Fact Type:** accommodation reservation includes hotel reservation

**Rule:** It is obligatory that the *Acme Travel* requests for exactly one *airline reservation* or exactly one *airline reservation* and exactly one *accommodation reservation* that includes exactly one *hotel reservation* but not both

---

Table 3.17: An example of a specific request for the SBVR model for the Acme Travel case study

The fact types in Table 3.17 refer to the Acme Travel’s export message which relates to the specific request of the intended reservations. The specific request rule in the table, is specified based on the given example of rule in the Acme Travel case study previously. The given rule refers to the inclusive-choices of reservation which is requested by the Acme Travel. According to the definition of the inclusive-choices (OR) for the given rule, the request could be exactly one of the choices of reservation, or the combination of any reservation, or exclusive-choices (XOR) of reservation (as specified in Table 3.17).

It is important to note that the logical operation in the specific request is expressed based on the standard of logical operator precedence in logic, where the logical operator AND is stronger than the logical operator OR.

The general form of specific request rules are grouped into two types in relation to both types’ structure of rules as specified in Section 3.1.1 (Step 3).
3.1. A FRAMEWORK FOR DEVELOPING CHOREOGRAPHY MODEL

Type 1: Participant is a single participant and event is a single or multiple events

The specification of the specific request rule depends on the specified rule in the developed SBVR model. It can be any requests in connection with several rules according to the following rule structures:

1. **Rule 1:** It is obligatory that the \textit{participant}_1 \textit{verb} exactly one \textit{event}_1 or exactly one \textit{event}_2 or ... or exactly one \textit{event}_N.

The possible specific requests that are allowed to be specified from the above structure rule are as follows:

**A request when involving no logical operation over event terms.**

Rule: It is obligatory that the \textit{participant}_1 \textit{verb} exactly one \textit{event}_1.

This specific request rule describes only one event from the inclusive-choices of the events specified in Rule 1 is requested.

**A request when involving the logical operation XOR (or..but not all) logical operation over event terms.**

Rule: It is obligatory that the \textit{participant}_1 \textit{verb} exactly one \textit{event}_1 or exactly one \textit{event}_2 or ... or exactly one \textit{event}_N but not all.

This specific request rule specifies \( N \)-exclusive-choices of events from the specified Rule 1. It means that exactly one of the events must be chosen as a request. Note that, for specifying this specific request rule, the logical operation XOR is possible for binary events as well (or..but not both).

**A request when involving the logical operation AND over event terms.**

Rule: It is obligatory that the \textit{participant}_1 \textit{verb} exactly one \textit{event}_1 and exactly one \textit{event}_2 and ... and exactly one \textit{event}_N.

This specific request is possible when there are \( N \) intended events from Rule 1 take place in the activity of the sending or the receiving of the events by the \textit{participant}_1.

**A request when involving the exclusive-choices of the combination of the logical operations AND and OR (if any) over event terms.**
Rule: It is obligatory that the *participant* verb exactly one *event* ... and (or) exactly one *event* or exactly one *event* ... and (or) exactly one *event* but not both.

Note that the standard of the logical operations precedence begins with AND, followed by XOR, and finally OR. This rule is employed if there is a specific request of exclusive-choices of events from Rule 1 and a combination of the logical operations over event terms (but not both could be replaced with but not all for *N* exclusive-choices). For instance, with reference to the Rule 1 as well as the above general form of rule, we can construct a rule: It is obligatory that the *participant* verb exactly one *event* and exactly one *event* or exactly one *event* or exactly one *event* but not both. This rule describes a request of either (not both) i). the *event* and the *event* are performed by the *participant*; or ii). the other option is *event* or *event* (OR definition).

2. Rule 2: It is obligatory that the *participant* verb exactly one *event* that includes exactly one *event* or exactly one *event* or ... or exactly one *event* but not all.

The following rule is the possible specific request with reference to the above structure rule:

**A request when involving *n* exclusive-choices of events**

Rule: It is obligatory that the *participant* verb exactly one *event* that includes exactly one *event*.

When the specific request rule comes from the rule contains the exclusive-choices of events such as Rule 2, hence the specific request embraces exactly one of the event from a nesting of group for *n* exclusive-choices of events.

3. Rule 3: It is obligatory that the *participant* verb exactly one *event* that includes exactly one *event* or exactly one *event* or ... or exactly one *event*.

The elaboration for each specific request rule, which are constructed from Rule 3 structure of rule is similar to the description of each possible specific request rules, which are developed based on Rule 1. The different is only the possible specific request rules from Rule 3 consists of a nesting of event group and the request is towards the events inside the nesting of group *event*. The potential specific request rules that are constructed from Rule 3 are as follows:

**A request when involving no logical operation over event terms.**

Rule: It is obligatory that the *participant* verb exactly one *event*.
that \textit{includes} exactly one $\text{event}_a$.

A request when involving the logical operation XOR (or..but not all) logical operation over event terms.

Rule: It is obligatory that the $\text{participant}_1$ verb exactly one $\text{event}_1$ that \textit{includes} exactly one $\text{event}_a$ or exactly one $\text{event}_b$ or ... or exactly one $\text{event}_n$ but not all.

A request when involving the logical operation AND over event terms.

Rule: It is obligatory that the $\text{participant}_1$ verb exactly one $\text{event}_1$ that \textit{includes} exactly one $\text{event}_a$ and exactly one $\text{event}_b$ and ... and exactly one $\text{event}_n$.

A request when involving the exclusive-choices of the combination of the logical operations AND and OR (if any) over event terms.

Rule: It is obligatory that the $\text{participant}_1$ verb exactly one $\text{event}_1$ that \textit{includes} exactly one $\text{event}_a$ ... and (or) exactly one $\text{event}_b$ or exactly one $\text{event}_c$ ... and (or) exactly one $\text{event}_n$ but not both.

It is worth to note that the specific request relates to the rules containing the logical operation XOR and OR over event terms only. The events in the rule associating with the logical operation AND are impossible to be extracted as the specific request. This is because all specified events take part in the activity that is performed by the participant.

Type 2: Participant is multiple participants and event is a single or multiple events

The possible specific request rules in connection with the rule structures from Type 2, are referred to multiple participants perform the sending or the receiving of the event(s). The specification of multiple participants follow the participation constraints as described in Section 2.3.2. Below is the rules from Type 2 together with the potential specific request rules.

1. Rule 1: It is obligatory that the $\text{participant}_1$ or the $\text{participant}_2$ or ... or the $\text{participant}_N$, verb exactly one $\text{event}_1$.

The potential specific request rules, which are constructed based on the given Rule 1 in the corresponding SBVR model are as follows:
A request when involving no logical operation over participant terms.

Rule: It is obligatory that the participant verb exactly one event.

This rule specifies when there is probably a specific request rule describing exactly one participant is chosen from all specified participants in Rule 1. The participant is aimed to perform the messages exchanged of the event.

A request when involving the logical operation XOR (or...but not all) logical operation over participant terms.

Rule: It is obligatory that the participant or the participant or ... or the participant but not all, verb exactly one event.

This specific request rule shows the exclusive-choices of the selected participants from the specified participants in Rule 1 to send (receive) the event.

A request when involving the logical operation AND over participant terms.

Rule: It is obligatory that the participant and the participant and ... and the participant, verb exactly one event.

The above specific request rule is employed when each selected participant from Rule 1 take place in the activity of the sending or the receiving of the event.

A request when involving the exclusive-choices of the combination of the logical operations AND and OR (if any) over participant terms.

Rule: It is obligatory that the participant and (or) the participant ... or ... the participant and (or) ... the participant but not both, verb exactly one event.

This specific request rule is essential if there is the exclusive-choices of the participants participating the messages exchanged of the event. Each exclusive-choice of the participants can be a combination of participants whether each participant or either one of participants is requested to perform the messages exchanged.
2. Rule 2: It is obligatory that the \( \text{participant}_1 \) that includes the \( \text{participant}_a \) or the \( \text{participant}_b \) or ... or the \( \text{participant}_n \), \( \text{verb} \) exactly one \( \text{event}_1 \).

The potential specific request rules that are constructed based on Rule 2 are similar to the potential specific request rules for Rule 1 previously. However, note that the participants are selected from a set of nesting group \( \text{participant}_1 \). The possible specific rules are as follows:

A request when involving no logical operation over participant terms.

Rule: It is obligatory that the \( \text{participant}_1 \) that includes the \( \text{participant}_a \) \( \text{verb} \) exactly one \( \text{event}_1 \).

A request when involving the logical operation XOR (or..but not all) logical operation over participant terms.

Rule: It is obligatory that the \( \text{participant}_1 \) that includes the \( \text{participant}_a \) or the \( \text{participant}_b \) or ... or the \( \text{participant}_N \) but not all, \( \text{verb} \) exactly one \( \text{event}_1 \).

A request when involving the logical operation AND over participant terms.

Rule: It is obligatory that the \( \text{participant}_1 \) that includes the \( \text{participant}_a \) and the \( \text{participant}_b \) and ... and the \( \text{participant}_N \), \( \text{verb} \) exactly one \( \text{event}_1 \).

A request when involving the exclusive-choices of the combination of the logical operations AND and OR (if any) over participant terms.

Rule: It is obligatory that the \( \text{participant}_1 \) that includes the \( \text{participant}_a \) and (or) the \( \text{participant}_b \) or the \( \text{participant}_c \) ... and (or) the \( \text{participant}_n \) but not both, \( \text{verb} \) exactly one \( \text{event}_1 \).

3. Rule 3: It is obligatory that the \( \text{participant}_1 \) or the \( \text{participant}_2 \) or ... or the \( \text{participant}_N \), \( \text{verb} \) exactly one \( \text{event}_1 \) that includes exactly one \( \text{event}_a \) or exactly one \( \text{event}_b \) or ... or exactly one \( \text{event}_n \) but not all.

The following rules are the specific requests which refer to Rule 3:

A request when involving no logical operation over participant terms.

There are two kind of rules that can be constructed from Rule 3 when
exactly one of the given participants is requested to perform the messages exchanged. Rule i shows the participant$^1$ is selected to send (receive) the event which is chosen from a group of nesting events. This contrasts with Rule ii, which specifies the participant$^1$ sends (receives) exactly one event that has been chosen from a group of nesting events.

Rule i: It is obligatory that the participant$^1$ verb exactly one event$^1$ that includes exactly one event$_a$ or exactly one event$_b$ or ... or exactly one event$_n$ but not all.

Rule ii: It is obligatory that the participant$^1$ verb exactly one event$^1$ that includes exactly one event$_a$.

A request when involving the logical operation XOR (or...but not all) logical operation over participant terms.

Both rules Rule i and Rule ii as follows are specified when the exclusive-choices of participants are selected from the inclusive-choices of participants in Rule 3. However, Rule i demonstrates the potential participant sends (receives) the event from a set of exclusive-choices of events while Rule ii illustrates the participant sends (receives) exactly one event that is already been selected.

Rule i: It is obligatory that the participant$^1$ or the participant$^2$ or ... or the participant$^N$ but not all, verb exactly one event$^1$ that includes exactly one event$_a$ or exactly one event$_b$ or ... or exactly one event$_n$ but not all.

Rule ii: It is obligatory that the participant$^1$ or the participant$^2$ or ... or the participant$^N$ but not all, verb exactly one event$^1$ that includes exactly one event$_a$.

A request when involving the logical operation AND over participant terms.

Rule i and Rule ii represent the specific request rules that are constructed based on Rule 3, where each participant are considered to take part in the messages exchanged of the exclusive-choices of the events (Rule i) or the selected event from the exclusive-choices of the events, that is the event$_a$ (Rule ii).

Rule i: It is obligatory that the participant$^1$ and the participant$^2$ and ... and the participant$^N$, verb exactly one event$^1$ that includes exactly one event$_a$ or exactly one event$_b$ or ... or exactly one event$_n$ but not all.
3.1. A FRAMEWORK FOR DEVELOPING CHOREOGRAPHY MODEL

Rule ii: It is obligatory that the \( participant_1 \) and the \( participant_2 \) and
... and the \( participant_N \), verb exactly one \( event_1 \) that includes exactly one
\( event_a \).

A request when involving the exclusive-choices of the com-
bination of the logical operations AND and OR (if any) over
participant terms.

The following specific request rules demonstrate the exclusive-choices of
participants where each choice can be a combination of the logical operations
AND and OR (if any) over participant terms. The purpose of the each rule
is similar as the previous rules.

Rule i: It is obligatory that the \( participant_1 \) and (or) the \( participant_2 \)
... or ... the \( participant_3 \) and (or) ... the \( participant_N \) but not both, verb
exactly one \( event_1 \) that includes exactly one \( event_a \) or exactly one \( event_b \) or
... or exactly one \( event_n \) but not all.

Rule ii: It is obligatory that the \( participant_1 \) and (or) the \( participant_2 \) or
the \( participant_3 \) ... and (or) the \( participant_N \) but not both, verb exactly one
\( event_1 \) that includes exactly one \( event_a \).

4. Rule 4: It is obligatory that the \( participant_1 \) that includes the \( participant_a \)
or the \( participant_b \) or ... or the \( participant_n \), verb exactly one \( event_1 \) that
includes exactly one \( event_a \) or exactly one \( event_b \) or ... or exactly one \( event_n \)
but not all.

Rule 4 specifies the inclusive-choices of participants from a group of
nesting performing the sending or receiving of the event from the exclusive-
choices of nesting events. The purpose of the following specific request rules
are the same as the specific request rules constructing from Rule 3. The
different is only on the specification of the participants, where the following
rules specify a group of nesting of the \( participant_1 \).

A request when involving no logical operation over parti-
cipant terms.

Rule i: It is obligatory that the \( participant_1 \) that includes the \( participant_a \)
verb exactly one \( event_1 \) that includes exactly one \( event_a \) or exactly one
\( event_b \) or ... or exactly one \( event_n \) but not all.

Rule ii: It is obligatory that the \( participant_1 \) that includes the \( participant_a \)
verb exactly one \( event_1 \) that includes exactly one \( event_a \).
A request when involving the logical operation XOR (or..but not all) logical operation over participant terms.

Rule i: It is obligatory that the participant \( \text{participant}_1 \) that includes the participant \( \text{participant}_a \) or the participant \( \text{participant}_b \) or ... or the participant \( \text{participant}_n \) but not all, verb exactly one event \( \text{event}_1 \) that includes exactly one event \( \text{event}_a \) or exactly one event \( \text{event}_b \) or ... or exactly one event \( \text{event}_n \) but not all.

Rule ii: It is obligatory that the participant \( \text{participant}_1 \) that includes the participant \( \text{participant}_a \) or the participant \( \text{participant}_b \) or ... or the participant \( \text{participant}_n \) but not all, verb exactly one event \( \text{event}_1 \) that includes exactly one event \( \text{event}_a \).

A request when involving the logical operation AND over participant terms.

Rule i: It is obligatory that the participant \( \text{participant}_1 \) that includes the participant \( \text{participant}_a \) and the participant \( \text{participant}_b \) and ... and the participant \( \text{participant}_n \), verb exactly one event \( \text{event}_1 \) that includes exactly one event \( \text{event}_a \) or exactly one event \( \text{event}_b \) or ... or exactly one event \( \text{event}_n \) but not all.

Rule ii: It is obligatory that the participant \( \text{participant}_1 \) that includes the participant \( \text{participant}_a \) and the participant \( \text{participant}_b \) and ... and the participant \( \text{participant}_n \), verb exactly one event \( \text{event}_1 \) that includes exactly one event \( \text{event}_a \).

A request when involving the exclusive-choices of the combination of the logical operations AND and OR (if any) over participant terms.

Rule i: It is obligatory that the participant \( \text{participant}_1 \) that includes the participant \( \text{participant}_a \) and (or) the participant \( \text{participant}_b \) or the participant \( \text{participant}_c \) ... and (or) the participant \( \text{participant}_n \) but not both, verb exactly one event \( \text{event}_1 \) that includes exactly one event \( \text{event}_a \) or exactly one event \( \text{event}_b \) or ... or exactly one event \( \text{event}_n \) but not all.

Rule ii: It is obligatory that the participant \( \text{participant}_1 \) that includes the participant \( \text{participant}_a \) and (or) the participant \( \text{participant}_b \) or the participant \( \text{participant}_c \) ... and (or) the participant \( \text{participant}_n \) but not both, verb exactly one event \( \text{event}_1 \) that includes exactly one event \( \text{event}_a \).

5. Rule 5: It is obligatory that the participant \( \text{participant}_1 \) and the participant \( \text{participant}_2 \) and ... and the participant \( \text{participant}_N \), verb exactly one event \( \text{event}_1 \) that includes exactly one event \( \text{event}_a \) or exactly one event \( \text{event}_b \) or ... or exactly one event \( \text{event}_n \) but not all.

The following rule is the potential request from the given rule Rule 5.
A request when each participant takes part for the exchanging message of exactly one event of XOR on events.

Rule : It is obligatory that the participant\textsubscript{1} and the participant\textsubscript{2} and ...
... and the participant\textsubscript{N}, verb exactly one event\textsubscript{1} that includes exactly one event\textsubscript{a}.

6. Rule 6: It is obligatory that the participant\textsubscript{1} that includes the participant\textsubscript{a} and the participant\textsubscript{b} and ...
... and the participant\textsubscript{N}, verb exactly one event\textsubscript{1} that includes exactly one event\textsubscript{a} or exactly one event\textsubscript{b} or ... or exactly one event\textsubscript{N} but not all.

The following specific request rule has the same intention as the specific request rule of Rule 5. However, each participant in the following rule is from a nesting of participant\textsubscript{1}.

A request when each participant in a nesting group takes part for exchanging message of exactly one event of XOR on events.

Rule : It is obligatory that the participant\textsubscript{1} that includes the participant\textsubscript{a} and the participant\textsubscript{b} and ...
... and the participant\textsubscript{N}, verb exactly one event\textsubscript{1} that includes exactly one event\textsubscript{a}.

7. Rule 7: It is obligatory that the participant\textsubscript{1} that includes the participant\textsubscript{a} or the participant\textsubscript{b} or ...
... or the participant\textsubscript{N} but not all, verb exactly one event\textsubscript{1}.

The following specific request rule is constructed when the participant\textsubscript{a} is chosen from a set of nesting participants in Rule 7 to perform the messages exchanged of the event\textsubscript{1}.

A request when exactly one participant in a nesting group sends (receives) exactly one event.

Rule : It is obligatory that the participant\textsubscript{1} that includes the participant\textsubscript{a} verb exactly one event\textsubscript{1}.

8. Rule 8: It is obligatory that the participant\textsubscript{1} that includes the participant\textsubscript{a} or the participant\textsubscript{b} or ...
... or the participant\textsubscript{N} but not all, verb exactly one event\textsubscript{1} that includes exactly one event\textsubscript{a} or exactly one event\textsubscript{b} or ... or exactly one event\textsubscript{N} but not all.
The following specific request rules are constructed based on the above Rule 8. Rule i emphasises upon a request when exactly one of the participant is selected from a nesting of \textit{participant$_{1}$} in Rule 8 and the selected participant sends (receives) the exclusive-choices of events. Rule ii focuses on a request when the selected participant performs the messages exchanged of the selected event from the exclusive-choices of the events in Rule 8.

A request when exactly one participant in a nesting group sends (receives) either one of the events inside a nesting group.

Rule i: It is obligatory that the \textit{participant$_{1}$} that includes the \textit{participant$_{a}$} \texttt{verb} exactly one \textit{event$_{1}$} that includes exactly one \textit{event$_{a}$} or exactly one \textit{event$_{b}$} or ... or exactly one \textit{event$_{n}$} but not all.

A request when exactly one participant in a nesting group sends (receives) exactly one of events inside a nesting group.

Rule ii: It is obligatory that the \textit{participant$_{1}$} that includes the \textit{participant$_{a}$} \texttt{verb} exactly one \textit{event$_{1}$} that includes exactly one \textit{event$_{a}$}.

9. Rule 9: It is obligatory that the \textit{participant$_{1}$} that includes the \textit{participant$_{a}$} or the \textit{participant$_{b}$} or ... or the \textit{participant$_{n}$} but not all, \texttt{verb} exactly one \textit{event$_{1}$} or exactly one \textit{event$_{2}$} or ... or exactly one \textit{event$_{N}$}.

The potential specific request rules based on Rule 9 represent a request of exactly one participant is chosen from the exclusive-choices of participants to send (receive): Rule i: exactly one event; Rule ii: the selected inclusive-choices of events; Rule iii: the particular exclusive-choices of events; Rule iv: each selected event; Rule v: the exclusive-choices of events where each choice is the selected event(s) which takes part in the activity. All events are from the inclusive-choices of events in Rule 9. The specific request rules are as follows:

A request when exactly one participant in a nesting group sends (receives) exactly one event.

Rule i: It is obligatory that the \textit{participant$_{1}$} that includes the \textit{participant$_{a}$} \texttt{verb} exactly one \textit{event$_{1}$}.

A request when exactly one participant in a nesting group sends (receives) the exclusive-choices of the particular event(s).

Rule iii: It is obligatory that the \textit{participant$_{1}$} that includes the \textit{participant$_{a}$}
verb exactly one event\textsubscript{1} or exactly one event\textsubscript{2} or \ldots or exactly one event\textsubscript{N} but not all.

A request when exactly one participant in a nesting group sends (receives) each event.

Rule iv: It is obligatory that the participant\textsubscript{1} that includes the participant\textsubscript{a} verb exactly one event\textsubscript{1} and exactly one event\textsubscript{2} and \ldots and exactly one event\textsubscript{N}.

textbf A request when exactly one participant in a nesting group sends (receives) the exclusive-choices of the combination of the logical operations AND and OR (if any) over event terms.

Rule v: It is obligatory that the participant\textsubscript{1} that includes the participant\textsubscript{a} verb exactly one event\textsubscript{1} \ldots and (or) exactly one event\textsubscript{2} or exactly one event\textsubscript{3} \ldots and (or) exactly one event\textsubscript{N} but not both.

10. Rule 10: It is obligatory that the participant\textsubscript{1} that includes the participant\textsubscript{a} or the participant\textsubscript{b} or \ldots or the participant\textsubscript{n} but not all, verb exactly one event\textsubscript{1} that includes exactly one event\textsubscript{a} or exactly one event\textsubscript{b} or \ldots or exactly one event\textsubscript{n}.

The purpose of the possible specific request rules which are referred from Rule 10 are the same as the specific request rules that are based on the previous Rule 9. The different is all events in Rule 10 inside a group of event\textsubscript{1}.

A request when exactly one participant in a nesting group sends (receives) exactly one event in a nesting group.

Rule i: It is obligatory that the participant\textsubscript{1} that includes the participant\textsubscript{a} verb exactly one event\textsubscript{1} that includes exactly one event\textsubscript{a}.

A request when exactly one participant in a nesting group sends (receives) the exclusive-choices of the particular event(s).

Rule ii: It is obligatory that the participant\textsubscript{1} that includes the participant\textsubscript{a} verb exactly one event\textsubscript{1} that includes exactly one event\textsubscript{a} or exactly one event\textsubscript{b} or \ldots or exactly one event\textsubscript{n} but not all.

A request when exactly one participant in a nesting group sends (receives) each event.
Rule ii: It is obligatory that the participant$_1$ that includes the participant$_a$ verb exactly one event$_1$ that includes exactly one event$_a$ and exactly one event$_b$ and ... and exactly one event$_n$.

textbfA request when exactly one participant in a nesting group sends (receives) the exclusive-choices of the combination of the logical operations AND and OR (if any) over event terms.

Rule: It is obligatory that the participant$_1$ that includes the participant$_a$ verb exactly one event$_1$ that includes exactly one event$_a$ ... and (or) exactly one event$_b$ or exactly one event$_c$ ... and (or) exactly one event$_n$ but not both.

11. Rule 11: It is obligatory that the participant$_1$ that includes the participant$_a$ or the participant$_b$ or ... or the participant$_n$ but not all, verb exactly one event$_1$ and exactly one event$_2$ and ... and exactly one event$_N$.

The potential request from Rule 11 is exactly one selected participant is aimed to perform the activity of the sending or the receiving of each specified event in Rule 11. The specific request rule is as follows:

A request when exactly one participant in a nesting group sends (receives) each event.

Rule i: It is obligatory that the participant$_1$ that includes the participant$_a$ verb exactly one event$_1$ and exactly one event$_2$ and ... and exactly one event$_N$.

12. Rule 12: It is obligatory that the participant$_1$ that includes the participant$_a$ or the participant$_b$ or ... or the participant$_n$ but not all, verb exactly one event$_1$ that includes exactly one event$_a$ and exactly one event$_b$ and ... and exactly one event$_n$.

The specific request which is based on Rule 12 is similar to the specific request rule of Rule 11, nevertheless each event in the following rule inside a group of event$_1$.

A request when exactly one participant in a nesting group sends (receives) each event.

Rule i: It is obligatory that the participant$_1$ that includes the participant$_a$ verb exactly one event$_1$ that includes exactly one event$_a$ and exactly one event$_b$ and ... and exactly one event$_n$.
3.2 Case Studies

We illustrate our approach for developing the SBVR model for service choreography by considering three case studies: i) the Online Photoshop case study, ii) the Tuition Fees system case study: International Student, and iii) the ACME Travel (AT) system case study.

The Online Photo Shop case study is a simple yet significant case study is adopted from [6]. The case study involves three-party conversations instead of two-party in the original case study. The case study is concerned on the choreography online Photo Shop scenario. The photo shop provides a service for developing and printing their products, namely, photo, poster, and album. The providing service is used by the customers to place orders. Both entities has a connection with the bank for payment purposes. On the other hand, the Tuition Fees system case study is adapted from the student fee payment and enrolment regulations of University of Roehampton for session 2016/2017. However, we focus on the payment that need to pay by the international student. The choreography deals with the sequence of activity of events relating to the payment process by multi-part conversations such as the students and several distinct departments in the university. We demonstrate these two case studies in the Appendix A for the Online Photo Shop case study and in the Appendix B for the Tuition Fees system case study.

In this section, we demonstrate our approach with modelling a well known case from [65], also was originally studied in [27] in view of declarative specification of service choreographies, of a hypothetical ACME Travel scenario and the multi-party conversations involved in arranging travel. The case study has been amended to include more complex interactions, including concurrent and alternative interactions, which pose stringent requirements on service coordination. The amended specification considers the case where all the requests are executed successfully. The original case study in [27] considers the case of failure upon which compensating actions need to take place. The case where failure occurs mid-way through the execution of the choreography is beyond the scope of this research.

The business scenario involves several interacting participants: Customer, Acme Travel, Airline, Accommodation includes Hotel and Apartment, Transport includes Bus and Train, and Tour Agency. All services are offered by different providers which belong to multiple organisations and the conversation respects the policies underlying the business activities involved. Each participant has its own role in the choreography:

**Acme Travel** receives a reservation request (itinerary) from customer and sends a booking request to the corresponding providers, i.e. airline, accommodation (hotel or apartment), transport (bus or train), and tour agency. The potential response from the corresponding provider(s) is received by Acme Travel and the notification is sent to customer.
CHAPTER 3. MODEL FOR SERVICE CHOREOGRAPHY

Customer interests in making a reservations, e.g. airline reservation, hotel reservation, apartment reservation, bus reservation, train reservation, tour reservation and targets the notification of reservation from Acme Travel.

Accommodation which is either hotel or apartment receives a reservation and sends a potential response to Acme Travel.

Transport includes bus or train obtains a reservation and responds to Acme Travel.

Airline be notified of a reservation and gives a response to Acme Travel.

Tour Agency receives a reservation and sends a response to Acme Travel.

Note that an instance of the choreography centers around a single reservation that is made by a single customer. It starts with a customer sends a reservation request which may include one or more of airline, accommodation (hotel or apartment), transport (bus or train), and tour reservations. Once Acme Travel (AT) receives the itinerary request, it sends reservation requests to the different providers and awaits for responses. Once all (un)successful reservations are in, AT sends a notifications to the customer.

3.2.1 Informal constraints of AT system

Let us now informally describe the choreography constraints, using natural language statements, as they would be collected after a series of meetings with the domain experts. The informal constraints describe as in the following:

An instance of the choreography centers around a single reservation booked by a single customer. Each reservation is implicitly created as soon as an instance of the choreography begins.

The activity is begun with the customer makes the reservation request (an itinerary) to Acme Travel, for booking any combination or a single reservation of an airline, a hotel, an apartment, or a tour. The reservation request contain the customer’s name and all the potential dates of each reservation, a start and an end of holiday date (itinerary), outbound-date and inbound date of airline, check-in date and check-out date of accommodation (if any) for exactly one of hotel or apartment (no reservation for both accommodation), departure date and arrival date (if any) of transport for exactly one of bus or train (no reservation for both transport), and tour date. Note that the transport is used to travel from one destination to another destination, hence the departure date is from one destination and the arrival date is at another destination.

The check-in date, outbound-date, and start date of holiday must be on the same date. Similarly to the check-out date, the inbound-date, and the end date of
holiday must be on the same date. The start date of holiday is not allowed to be
the same date as the end date of holiday. The same thing applies to check-in and
check-out date, and the outbound-date and the inbound-date. The departure date
and arrival date of transport could be the same date, however, both dates must be
in between the start and the end date of holiday. Also, the tour date must be in
between the start date and the end date holiday.

The reservation request will be received by Acme Travel and will be executed
exactly once in one instance of the Acme Travel service (for each customer request).
Note that the request of reservations, i.e. airline reservation, accommodation reser-
vations which is either hotel reservation or apartment reservation, transport reserva-
tion includes bus reservation or train reservation, or tour reservation will not
execute until the activity of the receiving of reservation request from the customer
executes. After the activity the receiving of reservation request executes, all the po-
tential booking requests for reservation(s) will be sending simultaneously by Acme
Travel.

After every execution of sending and receiving of the potential booking requests
for reservation(s) from Acme Travel and to the potential reservation provider(s) (e.g.
airline, accommodation (hotel or apartment), transport (bus or train), tour agency)
respectively, each potential reservation provider will respond to Acme Travel.

The response of each reservation provider is either successful or unsuccessful.
Once each each reservation provider taking the decision for reservation response,
the potential response will be sending to Acme Travel in any order. Prevent that
the activity of receiving the response of reservation request executes before the
activity of booking request requests executes.

After Acme Travel receives the notification of either a successful or unsuccessful
booking reservation(s) in any order, the customer is notified about the successful
or unsuccessful booking reservation(s). Note that exactly one notification will be
received by the customer.

3.2.2 Model development for the AT system

The steps taken refer to the description in Section 3.1.1 and follow the framework
as shown in Table 3.1. The steps are as in the following:

Step 1: Designation of terms for participants, events, static constraints, and time.

The first step in modeling the choreography with the SBVR model is the
designation of terms including participant terms, event terms, and static constraint
terms. It is closely related to the identification of all participants involved in the
choreography, the potential of the events in the interaction, and the possible static
CHAPTER 3. MODEL FOR SERVICE CHOREOGRAPHY

constraints belongs to the participant or the event. The necessary information is obtained from the informal constraints described in the previous section. Towards the identification of the aforementioned terms, we therefore restructure the informal constraints description in Section 3.2.1 to obtain a contractual version of the choreography.

1. The choreography begins as soon as the customer sends exactly one reservation request (an itinerary) to Acme Travel. There is an information of the customer detail, e.g. name.

2. Acme travel receives the reservation request right after the customer sends the reservation request.

3. After Acme Travel receives the reservation request, Acme Travel makes a single reservation or any combination of airline reservation, accommodation reservation, transport reservation, or tour reservation simultaneously. Note that only one reservation can be booked for accommodation reservation, which is either hotel reservation or apartment reservation (not both), and for transport reservation, which is either bus reservation or train reservation (not both).

4. Right after Acme Travel sends the potential reservation(s) to the potential reservation provider(s) simultaneously, each correspond reservation provider(s) will receive the potential reservation(s) requested by Acme Travel.

5. The sending and the receiving of the reservation request must be before the sending and the receiving of the potential reservation(s).

6. The potential reservation provider includes accommodation, which is either exactly one hotel or exactly one apartment (both are not allowed to receive the accommodation reservation at the same time). Hotel will receive exactly one hotel reservation, while apartment will receive exactly one apartment reservation.

7. The potential reservation provider includes transport, which is either exactly one bus or exactly one train (both are not allowed to receive the transport reservation at the same time). Bus will receive exactly one bus reservation, while apartment will receive exactly one apartment reservation.

8. The potential reservation provider includes exactly one airline. Airline will receive exactly one airline reservation.

9. The potential reservation provider includes exactly one tour agency. Tour agency will receive exactly one tour reservation.
10. Once each reservation provider taking the decision for reservation response, the potential response will be sending to Acme Travel in any order. The response of each reservation provider is either successful or unsuccessful. Ensure that the sending of reservation response occurs right before the receiving the reservation response by Acme Travel.

11. The execution of sending and receiving the response of reservation are prevented to occur before the execution of sending and receiving the potential reservation(s).

12. Right after Acme Travel receives the notification of either a successful or unsuccessful reservation(s) in any order, the customer is notified about the successful or unsuccessful booking reservation(s) by Acme Travel. Note that exactly one notification will be received by the customer.

13. The reservation request has the start date of holiday and the end date of holiday. Both date could not be the same date.

14. Each reservation comes together with its own date(s). Accommodation reservation has check-in date and check-out date, transport reservation has departure date and arrival date, airline reservation has outbound-date and inbound-date, and tour reservation has tour date.

15. The check-in date, outbound-date, and start date of holiday must be on the same date.

16. The check-out date, the inbound-date, and the end date of holiday must be on the same date.

17. The departure date and arrival date of transport could be the same date. Both dates must be in between the start and the end date of holiday.

18. The tour date must be in between the start date and the end date holiday.

According to the contractual version of choreography, the participants and events involved in the choreography are designated as the terms in Figure 3.1 and Figure 3.2, respectively. The static constraints which associate with the events or the participant are designated as the static constraint terms in Figure 3.3. Note that time term is designated in the same figure.

**Step 2: Designation of fact types.**

The description of designation fact types in the SBVR model for the Acme Travel case study is as follows:
--Participant Set

--Add participant terms here:

Term: participant

Term: customer

Term: AT

Term: airline

Term: accommodation

Term: hotel

Term: apartment

Term: tour agency

Term: transport

Term: bus

Term: train

Figure 3.1: Participant terms in the SBVR model for the Acme Travel system

Fact Types for participant set, event set, and static constraints In the second step, we need to construct the fact types describing the participant set as discussed in Section 2.3.2 by applying Sets definition, such as participant includes customer. We are able to know the participants involved from the participant terms designation in Step 1. However, it is necessary to take note that some participant in the participant set has a group of nesting. We need to analyze the contractual choreography version described in Step 1 for identifying the participant which includes a group of nesting. No. 6 and No. 7 from the contract shows that accommodation is grouped into hotel or apartment, similarly, transport is grouped into bus or train. The instance of fact type demonstrating a group of nesting is accommodation includes hotel. The fact types illustrating the participant set and a group of nesting (if any) are shown in Figure 3.4, Figure 3.5, and Figure 3.6.

The fact types in these figures demonstrate the local behaviour model for each participant by describing the messages exchanged of the sending or the receiving of the events as specified in the choreography contract. Also, there is the fact type illustrating the domain-specific constraint (static constraint) which belongs to the particular participant. For the Acme Travel case study, the participant customer has a detail such as name.
Besides the designation fact types for participant set, we also need to construct the fact types for describing the event set by employing the sets definition. The list of events involves in the choreography could be seen in the designation of event terms in step 1. However, there are some events obtain a group of nesting. In order to identify if there exists any nesting group of events, the contractual choreography described in Step 1 needs to revisit. We notice that several groups of nesting exist such as accommodation reservation has its subsets: hotel reservation and apartment reservation, transport reservation is divided into bus reservation and train reservation, and the response for each reservation, i.e accommodation reservation, airline reservation, transport reservation, and tour reservation, is either successful or unsuccessful reservation. Figure 3.7, Figure 3.8, and Figure 3.9 illustrate the participant set and nesting groups as mentioned previously. It is essential to check whether there are domain-specific constraint belong to any events. For example, the event reservation request has start date and end date. This domain-specific constraints which belong reservation request is required to be specified as the fact type. This specification can be seen in Figure 3.7 under static
CHAPTER 3. MODEL FOR SERVICE CHOREOGRAPHY

--Constraint

--Add your static constraints terms here:

Term: name
Term: date
Term: start date
Term: end date
Term: outbound date
Term: inbound date
Term: destination
Term: check-in date
Term: check-out date
Term: departure date
Term: arrival date
Term: tour date

--Time

Term: t

Figure 3.3: Static constraint terms and Time term in the SBVR model for the Acme Travel system

constraints section of the event reservation request.

In addition, it is compulsory to identify which event becomes the first activity of the interaction in the choreography. As we read the choreography contract in Step 1, the event reservation request is the initial event of messages exchanged between the customer and Acme Travel. Hence, we define reservation request under initialisation section as the initial event as shown in Figure 3.7.

Fact Types for ordering time intervals The fact types illustrating the ordering of time interval, where each time interval associates with the same event is important to notice which participant(s) initiates the messages exchanged of the particular event, and which participant(s) performs next. In the participant fact types, there are fact types for each participant that describes the messages exchanged of the particular event connecting with time inter-
--Fact Types

Add your participant fact types here:

---Participant 1

**Fact Type:** participant includes customer

Local behaviour of customer

--export message

**Fact Type:** customer sends reservation request at t₁

--import message

**Fact Type:** customer receives notification at t₂

**Fact Type:** customer has name

---Participant 2

**Fact Type:** participant includes AT

Local behaviour of Acme Travel

--export message

**Fact Type:** AT requests for airline reservation at t₁

**Fact Type:** AT requests for accommodation reservation at t₁

**Fact Type:** AT requests for tour reservation at t₁

**Fact Type:** AT requests for transport reservation at t₁

**Fact Type:** AT sends notification at t₂

--import message

**Fact Type:** AT receives reservation request at t₂

**Fact Type:** AT receives airline reservation response at t₂

**Fact Type:** AT receives accommodation reservation response at t₂

**Fact Type:** AT receives tour reservation response at t₂

**Fact Type:** AT receives transport reservation response at t₂

---

Figure 3.4: Fact types for participant set in the SBVR model for the Acme Travel system
Figure 3.5: Fact types for participant set in the SBVR model for the Acme Travel system
3.2. CASE STUDIES

Figure 3.6: Fact types for participant set in the SBVR model for the Acme Travel system

--Participant 5

Fact Type: participant includes tour agency

--import message

Fact Type: tour agency receives tour reservation at t2

--export message

Fact Type: tour agency sends tour reservation response at t1

--Participant 6

Fact Type: participant includes transport

--import message

Fact Type: transport receives transport reservation at t2

--export message

Fact Type: transport sends transport reservation response at t1

--Type of transport

--transport 1

Fact Type: transport includes bus

--transport 2

Fact Type: transport includes train

val. For instance, Figure 3.4 encompasses the fact types customer sends reservation request at t1 and AT receives reservation request at t2 to illustrate the customer initiates the interaction and followed by Acme Travel. This constraint can be seen in the contractual choreography contract as No. 2. To constrain the ordering of time t1 and t2 associate with the event reservation request, the fact type reservation request at t1 immediately precedes reservation request at t2 is specified as shown in Figure 3.10. The same approach applies for ordering the other time intervals steered by the same event (see Section 1 for details). Each ordering of time interval relating to each event is illustrated in Figure 3.10.

Fact Types for ordering events The specification of these fact types are vital in the choreography. We need to revisit the choreography contract in
CHAPTER 3. MODEL FOR SERVICE CHOREOGRAPHY

---

Fact Types

--Add your event fact types here:

--Event 1

Fact Type: event includes reservation request

--static constraints

Fact Type: reservation request has start date

Fact Type: reservation request has end date

Fact Type: reservation request has destination

--initialisation

Fact Type: reservation request is initial event

--Event 2

Fact Type: event includes airline reservation

--static constraints

Fact Type: airline reservation has outbound date

Fact Type: airline reservation has inbound date

--Event 3

Fact Type: event includes accommodation reservation

--static constraints

Fact Type: accommodation reservation has check-in date

Fact Type: accommodation reservation has check-out date

--Type of accommodation reservation

Fact Type: accommodation reservation includes hotel reservation

Fact Type: accommodation reservation includes apartment reservation

---

Figure 3.7: Fact types for event set in the SBVR model for the Acme Travel system
3.2. CASE STUDIES

Figure 3.8: Fact types for event set in the SBVR model for the Acme Travel system

--Event 4

Fact Type: event includes transport reservation

--static constraints

Fact Type: transport reservation has departure date
Fact Type: transport reservation has arrival date

--Type of accommodation reservation

Fact Type: transport reservation includes bus reservation
Fact Type: transport reservation includes train reservation

--Event 5

Fact Type: event includes tour reservation

--static constraints

Fact Type: tour reservation has tour date

--Event 6

Fact Type: event includes airline reservation response

--Type of airline reservation response

Fact Type: airline reservation response includes successful airline reservation
Fact Type: airline reservation response includes unsuccessful airline reservation
--Event 7

Fact Type: event includes accommodation reservation response

--Type of airline reservation response

Fact Type: accommodation reservation response includes successful accommodation reservation
Fact Type: accommodation reservation response includes unsuccessful accommodation reservation

--Event 8

Fact Type: event includes tour reservation response

--Type of airline reservation response

Fact Type: tour reservation response includes successful tour reservation
Fact Type: tour reservation response includes unsuccessful tour reservation

--Event 9

Fact Type: event includes transport reservation response

--Type of airline reservation response

Fact Type: transport reservation response includes successful transport reservation
Fact Type: transport reservation response includes unsuccessful transport reservation

--Event 10

Fact Type: event includes notification

Figure 3.9: Fact types for event set in the SBVR model for the Acme Travel system
3.2. CASE STUDIES

Step 1 to identify the ordering of messages exchanged of the events such as No. 5 in the contract. No. 5 involves two interactions, firstly, the interaction is between the customer and Acme Travel in sending and receiving the event of reservation request. This interaction is illustrated in the fact types, customer sends reservation request and AT receives reservation request. Secondly, the interaction is between Acme Travel and any involving reservation provider accommodation (hotel or apartment), transport (bus or train), airline, and tour agency. This has been described in the fact types AT requests for accommodation reservation, AT requests for transport reservation, AT requests for airline reservation, and AT requests for tour reservation. Also the fact types demonstrating the receiving of each reservation event from each reservation provider such as accommodation receives accommodation reservation, transport receives transport reservation, airline receives airline reservation, and tour receives tour reservation. Thus, in order to show the ordering between these two interactions, the immediate precedes notion as described in Section 2.3.2 is applied as depicted in Figure 3.11. The aforementioned or-

---Add your ordering of time associates with same event here:

**Fact Type:** reservation request at $t_1$ immediately precedes reservation request at $t_2$

**Fact Type:** airline reservation at $t_1$ immediately precedes airline reservation at $t_2$

**Fact Type:** accommodation reservation at $t_1$ immediately precedes accommodation reservation at $t_2$

**Fact Type:** tour reservation at $t_1$ immediately precedes tour reservation at $t_2$

**Fact Type:** transport reservation at $t_1$ immediately precedes transport reservation at $t_2$

**Fact Type:** accommodation reservation response at $t_1$ immediately precedes accommodation reservation response at $t_2$

**Fact Type:** airline reservation response at $t_1$ immediately precedes airline reservation response at $t_2$

**Fact Type:** tour reservation response at $t_1$ immediately precedes tour reservation response at $t_2$

**Fact Type:** transport reservation response at $t_1$ immediately precedes transport reservation response at $t_2$

**Fact Type:** notification at $t_1$ immediately precedes notification at $t_2$

---

Figure 3.10: Fact types describe the ordering of times in the SBVR model for the Acme Travel system.
dering of interactions is shown in the first four fact types in the figure such as **reservation request immediately precedes airline reservation**. Similar approach is applied for the other interactions in the description of choreography contract in Step 1.

---

Add your ordering of events fact types here:

- **Fact Type:** reservation request immediately precedes airline reservation
- **Fact Type:** reservation request immediately precedes accommodation reservation
- **Fact Type:** reservation request immediately precedes tour reservation
- **Fact Type:** reservation request immediately precedes transport reservation
- **Fact Type:** accommodation reservation immediately precedes accommodation reservation response
- **Fact Type:** airline reservation immediately precedes airline reservation response
- **Fact Type:** tour reservation immediately precedes tour reservation response
- **Fact Type:** transport reservation immediately precedes transport reservation response
- **Fact Type:** airline reservation response immediately precedes notification
- **Fact Type:** accommodation reservation response immediately precedes notification
- **Fact Type:** tour reservation response immediately precedes notification
- **Fact Type:** transport reservation response immediately precedes notification

---

**Figure 3.11:** Fact types describing the ordering of events in the SBVR model for the Acme Travel system

**Fact Type for static constraint relating to dates** As we see in the choreography contract in Step 1, there are the static constraints particularly on date constraints. To constrain the date constraints which numbering No. 13 till No. 18 in the contract, a date set need to be specified using Sets definition as described in Section 2.3.2 and Section 3.1.1. Note that in the SBVR model each date must be named uniquely. It is then followed by the initialisation for date to identify the initial date and final date in date set.

We notice that the Acme Travel case study contains date constraints involving equality, inequality, in between, and begin before as described in Section 2.3.2. The fact types of date constraints as shown in Figure 3.12 follows the generic form of fact type for specifying date constraints as described in Section 3.8.
---Add your static constraints fact types here (if required):

---a set of date:

Fact Type: start date is in date
Fact Type: end date is in date
Fact Type: check-in date is in date
Fact Type: check-out date is in date
Fact Type: outbound date is in date
Fact Type: inbound date is in date
Fact Type: departure date is in date
Fact Type: arrival date is in date
Fact Type: tour date is in date

---initialisation for date

Fact Type: start date is initial date
Fact Type: end date is final date

---Date constraints:

Fact Type: start date of reservation request equals check-in date of accommodation reservation
Fact Type: end date of reservation request equals check-out date of accommodation reservation
Fact Type: start date of reservation request equals outbound date of airline reservation
Fact Type: end date of reservation request equals inbound date of airline reservation
Fact Type: tour date of tour reservation is between start date of reservation request
Fact Type: tour date of tour reservation is between end date of reservation request
Fact Type: departure date of transport reservation is between start date of reservation request
Fact Type: departure date of transport reservation is between end date of reservation request
Fact Type: arrival date of transport reservation is between start date of reservation request
Fact Type: arrival date of transport reservation is between end date of reservation request
Fact Type: start date of reservation request equals end date of reservation request
Fact Type: check-in date of accommodation reservation equals check-out date of accommodation reservation

Figure 3.12: Fact types for specifying the static constraints particularly on date constraints in the SBVR model for the Acme Travel system
Step 3: Developing SBVR rules.

SBVR rules for the SBVR model describe the constraints in the informal constraints or the choreography contract of the system. Terms and fact types are the basis components for the development of SBVR rules in the SBVR model which follow the standard semantic formulations of SBVR rules and the structure of SBVR rules for specifying the complex interactions in choreography. The semantic formulation is according to OMG standard as described in Section 2.3.2 and the principle of rule structure for specifying the complex interactions are based on the proposal of this SBVR model as described in Section 2.3.2.

As we see the contractual choreography version of the Acme Travel system in step 1, all four main kind of rules as elaborated in Section 3.1.1 are required. Figure 3.13, Figure 3.14, Figure 3.15, and Figure 3.16 demonstrate and capture the informal constraints of Acme Travel system choreography.

Rule 1, Rule 2, and Rule 29 in the figure demonstrate the customer initiate the messages exchanged of the event reservation request, then it is followed by Acme Travel. Rule 4 illustrates Acme Travel is able to make a reservation booking either for a single reservation or any combination of reservations simultaneously. The rule structure follows the structure of "SBVR rules for specifying a sending or a receiving of event(s) by participant(s)" as described in steps for generating the generic SBVR model (see Section 3.10 for details). Rule 3 reflects the constraint No. 5 in the choreography contract and describe the execution of the messages exchanged of the event reservation request must occurs immediate before the execution of the messages exchanged of the request of reservations, either accommodation reservation, or airline reservation, or transport reservation, or tour reservation (inclusive-choices (OR) over events). The construction of Rule 3 follows the rule structure described in Section 3.13 for "SBVR rules for specifying the ordering of events".

Rule 5 till Rule 10 in Figure 3.13 illustrate the receiving of each reservation by each corresponding potential reservation provider, i.e airline, accommodation (hotel or apartment), tour agency, and transport reservation (bus or train) once after the reservation booking is requested by Acme Travel. These six rules reflect the contract for No. 6 till No. 9. Rule 6 and Rule 7, Rule 9, and Rule 10 follow the rule structure, namely Type 2 for specifying the sending or the receiving of the event(s) by participant(s) as described in Section 3.1.1. Rule 6 demonstrates only one of subset accommodation, namely hotel will receive the hotel reservation. Both accommodation, hotel and apartment are not included in the Rule 6 as each reservation provider aims for different reservations. Similar approach is applied for Rule 7, Rule 9, and Rule 10. The ordering of the sending and the receiving of each reservation can be seen in the specification of Rule 30 up to Rule 33.
Rule 11 up to Rule 14 demonstrate the ordering of i). the messages exchanged of the activity for requesting the reservation(s) from Acme Travel to the reservation provider(s), and ii). the messages exchanged of the activity for responding the successful (unsuccessful) reservation(s) from the reservation provider(s) to Acme Travel. There are two distinct structure of rules in these rules. Rule 11 and Rule 13 in in Figure 3.13 represent the execution of exchanging the message of airline reservation (tour reservation) must occurs immediate before the execution of exchanging message of reservation response which is either exactly one of the response; successful or unsuccessful. On the other hand, Rule 12 and Rule 14 represent the exclusive-choices of messages exchanged of the accommodation reservation (transport reservation), where it must occurs immediate before the exclusive-choices of the response.

Rule 15 up to Rule 22 and Rule 34 up to Rule 37 demonstrate the constraints of the sending of the reservation response by the reservation provider(s) immediate before the receiving of the reservation response by Acme Travel, as described in the contract (No. 10) in Step 1. Rule 16 and Rule 18 describes the participation constraints, that is the exclusive-choices of accommodation (transport) sending the exclusive-choices of response.

The ordering of the interaction between the reservation provider(s) and Acme Travel regarding the reservation response, and the interaction between Acme Travel and the customer in relation to the notification are described in Rule 23 until Rule 26 in Figure 3.14. These rules follow the contract as described in No. 12. Rule 27, Rule 28 describe the sending and the receiving of the event notification from Acme Travel to the customer. The ordering of the sending and the receiving of the notification between them specified as in Rule 38. This rule reflects the choreography contract as described in No. 12.

Figure 3.16 describes the date constraints as elaborated in the contract of choreography in Step 1, numbering No. 13 up to No. 18. The construction of rules refer to the step described in Section 3.15 which is for specifying static constraints specifically on dates.

### 3.2.3 Specification of specific request

The specific request in the SBVR model for the Acme Travel case study is described in Figure 3.17. The construction of the specific request refers to the generic form of rule specified in Section 3.16, particularly Rule No. 1 in Type 1 section. With reference to the specific request rule for Acme Travel system, the request is based on Rule 4 in Figure 3.13 which expresses Acme Travel is able to make a single reservation or any combination of reservation simultaneously. In the specific request rule, Acme Travel makes a specific request of the exclusive-choices of reservation: i). accommodation reservation which is particularly for hotel reservation and transport reservation which is particularly for bus reservation. ii). airline reservation only.
--Add your rule here:

Rule 1: It is obligatory that the *customer sends* exactly one *reservation request at* exactly one $t_1$

Rule 2: It is obligatory that the *AT receives* exactly one *reservation request at* exactly one $t_2$

Rule 3: It is obligatory that exactly one *reservation request immediately precedes* exactly one *airline reservation or exactly one accommodation reservation or exactly one tour reservation or exactly one transport reservation*

Rule 4: It is obligatory that the *AT requests for* exactly one *airline reservation or exactly one accommodation reservation or exactly one tour reservation or exactly one transport reservation, at exactly one $t_1$*

Rule 5: It is obligatory that the *airline receives* exactly one *airline reservation at* exactly one $t_2$

Rule 6: It is obligatory that exactly one *accommodation that includes the hotel receives* exactly one *accommodation reservation that includes exactly one hotel reservation at* exactly one $t_2$

Rule 7: It is obligatory that exactly one *accommodation that includes the apartment receives* exactly one *accommodation reservation that includes exactly one apartment reservation at* exactly one $t_2$

Rule 8: It is obligatory that the *tour agency receives* exactly one *tour reservation at* exactly one $t_2$

Rule 9: It is obligatory that exactly one *transport that includes the bus receives* exactly one *transport reservation that includes exactly one bus reservation at* exactly one $t_2$

Rule 10: It is obligatory that exactly one *transport that includes the train receives* exactly one *transport reservation that includes exactly one train reservation at* exactly one $t_2$

Rule 11: It is obligatory that exactly one *airline reservation immediately precedes* exactly one *airline reservation response that includes exactly one successful airline reservation or exactly one unsuccessful airline reservation but not both*

Rule 12: It is obligatory that exactly one *accommodation reservation that includes exactly one hotel reservation or exactly one apartment reservation but not both immediately precedes exactly one accommodation reservation response that includes exactly one successful accommodation reservation or exactly one unsuccessful accommodation reservation but not both*

Rule 13: It is obligatory that exactly one *tour reservation immediately precedes exactly one tour reservation response that includes exactly one successful tour reservation or exactly one unsuccessful tour reservation but not both*

Rule 14: It is obligatory that exactly one *transport reservation that includes exactly one bus reservation or exactly one train reservation but not both immediately precedes exactly one transport reservation response that includes exactly one successful transport reservation or exactly one unsuccessful transport reservation but not both*

Rule 15: It is obligatory that the *airline sends* exactly one *airline reservation response that includes exactly one successful airline reservation or exactly one unsuccessful airline reservation but not both, at exactly one $t_1$*

Figure 3.13: SBVR rules in the SBVR model for the Acme Travel system
Rule 16: It is obligatory that exactly one accommodation that includes exactly one hotel or exactly one apartment but not both sends exactly one accommodation reservation response that includes exactly one successful accommodation reservation or exactly one unsuccessful accommodation reservation but not both, at exactly one t1.

Rule 17: It is obligatory that the tour agency sends exactly one tour reservation response that includes exactly one successful tour reservation or exactly one unsuccessful tour reservation but not both, at exactly one t1.

Rule 18: It is obligatory that exactly one transport that includes exactly one bus or exactly one train but not both sends exactly one transport reservation response that includes exactly one successful transport reservation or exactly one unsuccessful transport reservation but not both, at exactly one t1.

Rule 19: It is obligatory that the AT receives exactly one airline reservation response that includes exactly one successful airline reservation or exactly one unsuccessful airline reservation but not both, at exactly one t2.

Rule 20: It is obligatory that the AT receives exactly one accommodation reservation response that includes exactly one successful accommodation reservation or exactly one unsuccessful accommodation reservation but not both, at exactly one t2.

Rule 21: It is obligatory that the AT receives exactly one tour reservation response that includes exactly one successful tour reservation or exactly one unsuccessful tour reservation but not both, at exactly one t2.

Rule 22: It is obligatory that the AT receives exactly one transport reservation response that includes exactly one successful transport reservation or exactly one unsuccessful transport reservation but not both, at exactly one t2.

Rule 23: It is obligatory that exactly one accommodation reservation response that includes exactly one successful accommodation reservation or exactly one unsuccessful accommodation reservation but not both immediately precedes exactly one notification.

Rule 24: It is obligatory that exactly one airline reservation response that includes exactly one successful airline reservation or exactly one unsuccessful airline reservation but not both immediately precedes exactly one notification.

Rule 25: It is obligatory that exactly one tour reservation response that includes exactly one successful tour reservation or exactly one unsuccessful tour reservation but not both immediately precedes exactly one notification.

Rule 26: It is obligatory that exactly one transport reservation response that includes exactly one successful transport reservation or exactly one unsuccessful transport reservation but not both immediately precedes exactly one notification.

Rule 27: It is obligatory that the AT sends exactly one notification at exactly one t1.

Rule 28: It is obligatory that the customer receives exactly one notification at exactly one t2.

Rule 29: It is obligatory that exactly one reservation request at exactly one t1 immediately precedes exactly one reservation request at exactly one t2.
Chapter 3. Model for Service Choreography

**Figure 3.15:** SBVR rules in the SBVR model for the Acme Travel system

Rule 30: It is obligatory that exactly one airline reservation at exactly one t₁ immediately precedes exactly one airline reservation at exactly one t₂

Rule 31: It is obligatory that exactly one accommodation reservation at exactly one t₁ immediately precedes exactly one accommodation reservation at exactly one t₂

Rule 32: It is obligatory that exactly one tour reservation at exactly one t₁ immediately precedes exactly one tour reservation at exactly one t₂

Rule 33: It is obligatory that exactly one transport reservation at exactly one t₁ immediately precedes exactly one transport reservation at exactly one t₂

Rule 34: It is obligatory that exactly one accommodation reservation response at exactly one t₁ immediately precedes exactly one accommodation reservation response at exactly one t₂

Rule 35: It is obligatory that exactly one airline reservation response at exactly one t₁ immediately precedes exactly one airline reservation response at exactly one t₂

Rule 36: It is obligatory that exactly one tour reservation response at exactly one t₁ immediately precedes exactly one tour reservation response at exactly one t₂

Rule 37: It is obligatory that exactly one transport reservation response at exactly one t₁ immediately precedes exactly one transport reservation response at exactly one t₂

Rule 38: It is obligatory that exactly one notification at exactly one t₁ immediately precedes exactly one notification at exactly one t₂

The fact types in figure are necessary to focus which fact types are required for constructing the specific request rule.

3.3 Concluding remarks

In this chapter, we have focused on the specification of a service choreography in our approach, using SBVR.

The construction of the corresponding SBVR model involves three main parts, which are:

**Terms** designate two main components which are important in modelling choreographies. They are all participants involved and the messages involved which we called events. It is then followed by the designation of the static constraint terms which are known as the domain specific constraints for each participant and event (if any). These static constraint terms are central to building the local behaviour model for each participant.
3.3. CONCLUDING REMARKS

Figure 3.16: SBVR rules particularly for date constraints in the SBVR model for the Acme Travel system

--date static constraints

Rule 39: It is obligatory that exactly one start date of exactly one reservation request equals exactly one check-in date of exactly one accommodation reservation

Rule 40: It is obligatory that exactly one end date of exactly one reservation request equals exactly one check-out date of exactly one accommodation reservation

Rule 41: It is obligatory that exactly one start date of exactly one reservation request equals exactly one outbound date of exactly one airline reservation

Rule 42: It is obligatory that exactly one end date of exactly one reservation request equals exactly one inbound date of exactly one airline reservation

Rule 43: It is prohibited that exactly one start date of exactly one reservation request equals exactly one end date of exactly one reservation request

Rule 44: It is prohibited that exactly one check-in date of exactly one accommodation reservation equals exactly one check-out date of exactly one accommodation reservation

Rule 45: It is prohibited that exactly one outbound date of exactly one airline reservation equals exactly one inbound date of exactly one airline reservation

Rule 46: It is obligatory that exactly one tour date of exactly one tour reservation is between exactly one start date of exactly one reservation request and exactly one end date of exactly one reservation request

Rule 47: It is obligatory that exactly one departure date of exactly one transport reservation is between exactly one start date of exactly one reservation request and exactly one end date of exactly one reservation request

Rule 48: It is obligatory that exactly one arrival date of exactly one transport reservation is between exactly one start date of exactly one reservation request and exactly one end date of exactly one reservation request

Rule 49: It is obligatory that exactly one departure date of exactly one transport reservation begins before exactly one arrival date of exactly one transport reservation
Figure 3.17: Specific request in the SBVR model for the Acme Travel system

**Fact Types** specify the participant and event set including any nesting involved in the choreography. The fact types are also important for specifying the local behaviour model for each participant which describe the messages exchanged including the export and import messages as well as the domain-specific constraints. In some circumstances, events (if any) also has the domain-specific constraints. In the SBVR model, besides the aforementioned fact types, there are the fact types for specifying the ordering of messages exchanged and the static constraints particularly for date constraints.

**SBVR rules** are built on the specified term and fact types previously. The rules demonstrate the global constraints of the choreography. They include the rules for specifying i). the messages exchanged performed by each participant which contribute to constrain the ordering of messages exchanged of the same event executed by the participants involved. ii). the ordering of messages exchanged of (different events, iii). the static constraints, particularly on dates. Each rule can be specified for multi-party conversations in constraining the participant terms (participation constraints), by applying the logical operation OR (inclusive-choices of participants), XOR (exclusive-choices of participants), and AND (each participant involves in the interaction) over participant terms. Similarly, each rule can specify multiple events which are sent or received by the intended participants. We called this as the messages exchanged constraints where the logical operations OR (inclusive-choices of events), XOR (exclusive-choices of events), and AND (each event is executed
3.3. CONCLUDING REMARKS

by the intended participant(s) in the interaction) is applied over event terms.

The SBVR model is developed according to the introduced framework and the construction of terms, fact types, and rules, are based on the given general forms. The developed fact types and rules in the developed SBVR model are then be exploited to construct the specific request to verify the realisability in future (the details are described in Chapter 4). What is more, we have produced the description of steps taken to specify the choreography of the SBVR model.

By design SBVR does not include time. Its supplement DTV in [42] has proposed a notion of precedence as well as time interval whilst staying within the OMG specification. Hence, those two notions of the DTV are introduced in this chapter as the supplement for capturing the temporal operator to specify the ordering of messages exchanged in the choreography. We introduced them as the immediately precedes notion and the time notion (Section 2.3.2).

In addition, the formulation of exclusive disjunction (XOR) in SBVR limits to the binary logical operands only. In order to capture the specification for alternative interaction involving XOR on participants or (and) XOR on events in the proposed SBVR model, the definition and the formulation of n-ary XOR is adopted from [43]. XOR is then exploited to express exactly one sense of XOR which is relevant to apply in natural language.

Finally, we have illustrated our approach, by applying it to case studies: i) Acme Travel system (Section 3.2), ii). online photo shop (in Appendix A), and iii). tuition fee system (in Appendix B).

We have demonstrated how the SBVR model can be applied to model a service choreography at a purely declarative level using structured natural language. This SBVR model enables i). the flexibility in specification, i.e. replacing any services, changing includes adding or removing any local behaviour model for each participant as well as the static constraints for each event (if any), ii). the users (non-experts, e.g. stakeholder, business analyst) to validate the choreography model by directly reading the structured natural language. However, the questions are can this proposed SBVR model be translated into a constraint solver, the Alloy Analyzer? Can the Alloy Analyzer generate the choreography automatically and verify the realisability of the corresponding SBVR model effectively?. The next chapter of this thesis will provide an answer to these questions. The summary of the proposed SBVR model for service choreography can be found in [66].
In Chapter 2 and Chapter 3, there were detailed discussions how the SBVR model was developed to specify and generate a choreography, as advocated the OMG standard SBVR and the supplement, DTV. The SBVR model, which is a rule-based declarative language employs terms and fact types to develop rules according to SBVR standard semantics. These rules encapsulate the specification of complex interactions comprising the participation and messages exchanged constraints to specify alternative and concurrent interaction; the specification to govern the ordering of messages exchanged; and the specification to facilitate the static constraints.

This chapter presents how that SBVR model, which is an informal structured natural language that are represented in the form of formal logic can be transformed and compiled by the Alloy Analyzer, making it amenable to formal analysis. Alloy Analyzer enables the generation of choreography for the corresponding SBVR model and the verification on the generated choreography. The generating choreography involves a transformation taking the developed SBVR model and transforming the model into Alloy Analyzer [46] to automatically produce the exact solution that corresponds to the input SBVR model. Afterwards, the generated Alloy model, that is the choreography model corresponds to the SBVR model can readily to verify i). the realisability which refers to the verification whether a specific request can be realised by the given service choreography and ii). the domain-specific constraints which particularly on date constraints, i.e. static constraints since they do not depend on the ordering of the message exchange like realisability does.
4.1 An overview of Alloy model

As described in Chapter 2, Alloy Analyzer (Alloy) is based on a logic which provides the structures representing relations. Alloy model for specifying service choreography consists of a module containing a number of signatures and abstract signatures to represent the participants and events in the SBVR model. Each signature and abstract signature introduce fields which are captured by relations. These fields denote verb interconnecting with Terms in each Fact Type. The multiplicities in Alloy are applied to illustrating the accurate meaning of the complex interactions as well as the ordering of messages exchanged in the choreography. The combination of signature, multiplicities, and field are the basis of the development of SBVR rule in Alloy. This will discuss further in Section 4.4. In addition, facts and predicates in Alloy are deployed to constrain a certain case (e.g. alternative and concurrent interaction) for a particular signature and to generate a choreography and verify the realizability.

4.2 A transformation of Terms

In the beginning of the transformation from the SBVR model into the Alloy model, terms in the SBVR model which are designated to capture the participants, the events, the static constraints, as well as time involved in the choreography, are transformed into Alloy as the signatures.

The following terms show the generic terms for participants, events, static constraints, and time in the SBVR model, and the snippets show the signatures in Alloy which correspond to the given terms of participants, events, static constraints, and time respectively:

Participants in the SBVR model:
Term: participant
Term: participant_1
Term: participant_2
......
Term: participant_N

Events in the SBVR model:
Term: event
Term: event_1
Term: event_2
......
Term: event_N

Static constraints in the SBVR model:
Term: static_1
4.2. A TRANSFORMATION OF TERMS

Term: \( \textit{static}_2 \)
.....
Term: \( \textit{static}_n \)

Time in the SBVR model:
Term: \( T \)

Participants in the Alloy model:

1 sig participant{}
2 sig participant_1 {}
3 sig participant_2 {}
...
4 sig participant_N {}

Events in the Alloy model:

1 sig event{}
2 sig event_1 {}
3 sig event_2 {}
...
4 sig event_N {}

Static constraints in the Alloy model:

1 sig static_1 {}
2 sig static_2 {}
...
3 sig static_n {}

A part of time in the Alloy model:

1 sig time {}
4.3 A transformation of Fact Types

In this section, the transformation of fact types in the SBVR model into Alloy model encapsulates the fact types for designating i). the participant set and event set including a group of nesting; ii). the messages exchanged including the import and the export messages and the domain-specific constraints (if any) for each participant; iii) the static constraints associate with events. However, all fact types pertaining and contributing to the development of temporal operator rules and Date constraint rules will be discussed in Section 4.4.2 and Section 4.4.3, respectively.

Fact types for specifying participant and event set

As explained in Chapter 3, the SBVR model for service choreographies consists of multiple distinct participants and events which are grouped as the participant set and the event set, respectively. This means the participant set is classified into multiple different participants, such as participant\_1 until n number of participants participant\_n; and likewise the event set comprises event\_1 up to n number of events event\_n. These participant and event sets are specified in the fact types by employing sets definition as explained in the previous chapter (Section 2.3.2) such as participant includes participant\_1 and event includes event\_1.

On account of this approach, we employ abstract signature for specifying the participant set and event set. "abstract sig participant" and "abstract sig event" play a role as the superset of participants and events involved in the choreography as described in the SBVR model. This means the previous "sig participant" and "sig event" as described in Section 4.2 are necessarily to be changed as the abstract signature. On the other hand, all the subsets of the participant set and the event set are the extensions (subsignatures) of abstract sig participant and abstract sig event in Alloy, respectively. For instance, the participant\_1 in the SBVR model will be a subsignature in Alloy, which is defined as "sig participant\_1 extends participant". The same method applies for the subset of event set such as event\_1, this will be defined as "sig event\_1 extends event".

The transformation of the participant set and the event set can be seen as in the following.

Participant set fact types in the SBVR model:
Fact Type: participant includes participant\_1
.....
Fact Type: participant includes participant\_N

Participant set in the Alloy model:

1 abstract sig participant{ }
2 sig participant\_1 extends participant{ }
4.3. A TRANSFORMATION OF FACT TYPES

... 3 sig participant_3 extends participant {} 

Event set fact types in the SBVR model:
Fact Type: event includes event_1 
.....
Fact Type: event includes event_N 

Event set in the Alloy model:

4 abstract sig event{} 
5 sig event_1 extends event{} 
... 
6 sig event_N extends event{}

Note the use of abstract and concrete signatures in the above declarations. The declarations (lines 1-2) capture fact types that use includes such as participant includes participant_1. Similarly, the declarations in lines 4 and 5 above represent event includes event_1.

As can be seen in Chapter 3 (Section 2.3.2), there are the possibilities that several participants inside the participant set and several events inside the event set are defined using the fact types as a group of nesting. For example, participant_1 comprises participant_a up to participant_n, and event_1 encompasses event_a until event_n. The specification for nesting of participant_1 and event_1 in the SBVR model, are illustrated as in the following.

A nesting of participant_1 fact types in the SBVR model:
Fact Type: participant_1 includes participant_a 
.....
Fact Type: participant_1 includes participant_n 

A nesting of event_1 fact types in the SBVR model:
Fact Type: event_1 includes event_a 
.....
Fact Type: event_1 includes event_n 

The same approach that has been used for transforming the participant set and the event set is applied for the transformation of the group of nesting. It is depicted as in the following snippets with respect to the corresponding fact types.
A nesting of \textit{participant}$_1$ in the Alloy model:

7 abstract sig participant$_1$ extends participant{ }
8 sig participant$_a$ extends participant$_1${ }
... 
9 sig participant$_b$ extends participant$_1${ }

A nesting of \textit{event}$_1$ in the Alloy model:

10 abstract sig event$_1$ extends event{ }
11 sig event$_a$ extends event$_1${ }
... 
12 sig event$_n$ extends event$_1${ }

Note that all participants (inside the participant set) and events (inside the event set) involving in the multi-party conversation in the SBVR model are transformed as the signatures, which are the subsignatures of abstract signature participant and abstract signature event. However, if there are fact types appear defining the participant (event) as a group of nesting which use sets definition (verb \textit{includes} in between the participant (event) terms) such as \textit{participant}$_1$ \textit{includes} \textit{participant}$_a$, the following steps are applied. Firstly, the first term in the fact type (before verb \textit{includes}), namely the \textit{participant}$_1$ will be changed from the signature to the abstract signature in Alloy as shown in line 7. Subsequently, the second term after verb \textit{includes}, that is \textit{participant}$_a$ is defined as the subsignature in Alloy. The \textit{participant}$_a$, which is a subset of the \textit{participant}$_1$ is appeared as the subsignature of abstract sig \textit{participant}$_1$ (line 11). Similar method applies for fact type \textit{event}$_1$ \textit{includes} \textit{event}$_a$. \textit{event}$_1$ is defined as the abstract signature (line 10) in Alloy. Whereas the \textit{event}$_a$, which is the subset of the \textit{event}$_1$ is designated as the subsignature of abstract sig \textit{event}$_1$ (line 11).

\textbf{Fact types} for specifying messages exchanged and domain-specific constraints (if any) for each participant

The generic fact type for specifying the messages exchanged which is referred to a participant performing sending or receiving a message in the multi-party conversations, is expressed in the form of “\textit{participant} verb \textit{event} at \textit{T}”. participant and \textit{event} refer to any participants and events inside the participant set and the event set respectively, \textit{T} represents any times including $T_1$ up to $T_n$, and \textit{verb} refers to any verb involves sent events as export messages and those involving re-
ceived events as import messages. The generic fact type for specifying the domain-specific constraints which is belonged to each participant is specified in the form of "participant verb static". participant is any participants inside the participant set, verb refers to any verb that gives the meaning as the belonging, and static is any static constraint terms such as static1.

The transformation of such fact types into Alloy are illustrated as in the following:

Fact types for specifying a messages exchanged performing by the participant1 in the SBVR model:

Fact Type 1: participant1 verb event1 at T1
Fact Type 2: participant1 verb static1

The specification of a messages exchanged performing by the participant1 in the Alloy model:

13 sig participant.1 extends participant{verb: event.1, verb1: one static.1}
.....
14 sig event.1 extends event{at: t1_event.1}

The given Fact Type 1 illustrates the participant1 executes the messages exchanged of the event1. In order to transform this fact type, several steps are taken. Commencing with identifying the participant by looking at the first term in the Fact Type 1, the sig participant.1 (line 13) is considered for translating the participant1 into Alloy. Then, the verb in the Fact Type 1 is populated as a field of the sig participant.1 which is mapped to the range of the field verb is given by the expression event.1 (line 13), event1 after verb. Supposedly, the cardinality need to be assigned before event.1, however this will be captured in the transformation of SBVR rules in Section 4.4. Next, since the verb at is associated with the event1 (the term before at), at is translated as a field of the sig event.1 in Alloy. The field at (line 14) is mapped to another signature t1_event.1 (line 14) as the transformation of the verb at) associating with time T1. The definition and the description of the sig t1_event.1 will be discussed further in Section 4.4.2. This is closely related to the composition of the local behaviour model and the time notion.

In similar approach, the Fact Type 2 which describes the domain-specific constraint of the participant1, namely static1, is translated. Due to the first term in the Fact Type 2 is the participant1, the sig participant.1 in Alloy is taken into consideration. The verb after the term participant1 is translated as another field of the sig participant.1. Note that, two identical names of fields in the same
signature are not allowed in Alloy, which is why \textit{verb} is populated as \texttt{verb.1} (line 13) in Alloy. This field \texttt{verb.1} is mapped to its range \texttt{static.1}, which is another defined signature in Alloy. This illustrates the translation of the interconnection of \textit{verb} and \textit{static} in the Fact Type 2. In the SBVR model, each static is bound to exactly one quantification. Hence, \texttt{one} cardinality is assigned before \texttt{static.1} (line 13).

The above description captures the transformation of the local behaviour model of the \texttt{participant1}, which are composed of the basis of common events appearing in the respective sets of import and export messages of the \texttt{participant1}.

**Fact types for specifying static constraints (if any) for each event**

The general form of fact types showing the static constraints associate with the event, is specified in the form of "\textit{event verb static}". \texttt{event} represents any event inside the event set, \texttt{verb} indicates belonging, and \texttt{static} refers to the intended static constraint of the correspond event. This can be seen in the following fact type and its translation into Alloy in the snippet (line 15).

---

**Fact types for specifying static constraints associating with the event in the SBVR model:**

**Fact Type 3:** \texttt{event.1 verb static.1}

**Specifying static constraints associating with the event in the Alloy model:**

15 \texttt{sig event.1 extends event{verb: one static.1}}

---

The transformation implements the same method, which was used in translating the fact type relating to the domain-specific constraint associating with the participant, as described in the previous section. However, the \texttt{sig event.1} (line 15) is considered in Alloy since the first term in the Fact Type 3 refers to \texttt{event.1}.

### 4.4 A transformation of SBVR Rules

As discussed in Chapter 3, the SBVR rules in the SBVR model describe a set of global constraints prescribing the business constraints in the choreography. The SBVR rules capture the specification describing the complex interactions between distinct participants; the ordering of services interaction encapsulating i). the messages exchanged of the same event performing by different participants in multi-party conversations, ii). the messages exchanged of dissimilar events amongst different participant; and the static constraints particularly on date constraints. Note that all transformations of SBVR rules consider the obligation rule as our proposed
SBVR model focuses on the obligation statement. The transformation of prohibition rule which is specifically used for specifying 'inequality' for date constraints is defined as the fact in the intended signature. In this section, the essential steps involve in the transformation of such SBVR rules into Alloy are explained in details.

### 4.4.1 SBVR rules - complex interaction

The transformation of the SBVR rules in the SBVR model into Alloy concerns two main components of the complex interactions, namely the participation constraints and the messages exchanged (event) constraints.

**SBVR rules - participation constraints**

The participation constraints as discussed in Section 2.3.2 (Chapter 3) underlined the specification constraints on the participant terms. This constraint allows a single participant or multiple participants include a nesting of participants perform the messages exchanged of a single event or multiple events. Multiple participants refer to an inclusive disjunction (OR) on participants, an exclusive (XOR) disjunction on participants, and a conjunction (AND) on participants, perform the messages exchanged of the intended event(s).

In the SBVR model, the general form of SBVR rule "It is obligatory that the participant verb exactly one event", is employed. Note that the quantification the is bound to the participant term referring to the particular participating service involve in the multi-party conversations. participant represents any participants performing the intended messages exchanged. This participant could be a single participant or multiple participants (by exploiting the logical operations in the SBVR model). In addition, verb is any suitable verb reflecting the meaning of sending or receiving the event(s). Lastly, event denotes any events which are defined in the event set. In this section, the quantification of event is considered as exactly one, shows that the intended participant(s) executes the messages exchanged of exactly one event.

- A transformation of the SBVR rules describing a single participant performs the messages exchanged of exactly one particular event.

Note that participant1 is used as an illustration for the transformation. It can be any participants such as participant2 up to n number of participants participantN, which are defined as the participating service involved in the interaction (a subset of the participant set). event1 is used as an example of this transformation as well.

---

**SBVR rule in the SBVR model:**

**Rule 1:** It is obligatory that the participant1 verb exactly one event1 at exactly one T1
The signature which corresponds to the above SBVR rule in the Alloy model:

```
--A transformation of Rule 1
1 one sig participant_1 extends participant{verb: one event_1}
...
2 one sig event_1 extends event{at: one t1_event_1}
```

The transformation steps are described as follows. It begins with the identification of participant in Rule 1. This rule shows that the participant involves in the interaction. Hence, \texttt{sig participant\_1 extends participant} (line 1) is considered. Since the quantification \texttt{the} is connected to the \texttt{participant\_1}, the multiplicity (cardinality) \texttt{one} is populated before the signature for \texttt{participant\_1}.

Next, \texttt{verb} in Rule 1 is translated as a field \texttt{verb} whose domain is \texttt{participant\_1} and whose range is \texttt{event\_1}. As we assume the single event is performed by the participant, \texttt{exactly one} is translated as the cardinality \texttt{one} before \texttt{event\_1}. As described in Section 4.3, all signatures of events are defined as the signature without associating with cardinality. Since \texttt{exactly one} event is considered in Rule 1, therefore the cardinality \texttt{one} is designated for \texttt{sig event\_1 extends event} (line 2).

Lastly, as explained in Section 4.3, since \texttt{event\_1} is connected with \texttt{T_1} by the verb \texttt{at}, a field \texttt{at} in Alloy is populated in the signature for \texttt{event\_1}(line 2), which is mapped to \texttt{t1_event\_1} (showing \texttt{t1} associates with \texttt{event\_1}). The cardinality \texttt{one} is added before \texttt{t1_event\_1} which corresponds to \texttt{exactly one} \texttt{T_1}.

- A transformation of the SBVR rules describing an inclusive disjunction (OR) on participants perform the messages exchanged of exactly one particular event.

The following Rule 2 shows the general form of rule describing the inclusive disjunction (OR) on participants performing the sending (receiving) the \texttt{event\_1} in the choreography. The transformation of this rule into Alloy starts with the identification of participants involved in the interaction. All the involved participants as specified in Rule 2 are considered in Alloy as shown in the declaration of signatures in line 3 to line 5 below. Then, we need to identify the logical operation used over participant terms in Rule 2, either \texttt{OR}, \texttt{AND}, or \texttt{XOR}. Since Rule 2 expresses the inclusive disjunction (or) on participants, \texttt{lone} (a cardinality keyword in Alloy denoting at most one) for each signature of participant (\texttt{participant\_1, ..., participant\_N})
is declared. Finally, the translation of the remaining ”.... verb exactly one event at exactly one T1” refers to the same approach as described in the previous section, which applies to each intended signature for participants in lines 3-5.

SBVR rules in the SBVR model:

**Rule 2:** It is obligatory that the participant or the participant2 or... or the participantN, verb exactly one event at exactly one T1.

**Rule 3:** It is obligatory that the participant1 that includes the participanta or the participantb or ... or the participantn, verb exactly one event at exactly one T2.

The signatures which correspond to the above SBVR rules in the Alloy model:

```alloy
-- A transformation of Rule 2
3 lone sig participant_1 extends participant{verb: one event_1}
4 lone sig participant_2 extends participant{verb: one event_1}
...
5 lone sig participant_N extends participant{verb: one event_1}
...

-- A transformation of Rule 3
6 abstract sig participant_1 extends participant{verb: one event_1}
7 lone sig participant_a extends participant_1{}
8 lone sig participant_b extends participant_1{}
...
9 lone sig participant_n extends participant_1{}
...
...
10 one sig event_1 extends event{at1: one t1_event_1, at2: one t2_event_1}
```

Rule 3 expresses the inclusive disjunction (OR) on participants (participanta, ..., participantn), which are inside a nesting of participant1, execute the sending (receiving) of the event1. The transformation of Rule 3 into Alloy considers the following steps. This rule is translated into Alloy as illustrated in the given expression from line 6 to line 9.

First, the identification of participants in the interaction as specified in Rule 3 is required. As described in Rule 3, there is a nesting of participant1. Hence, participant1 is translated as an abstract signature for the participant_1 (line 6), while its subsets are defined as the subsignatures of participant_1 (lines 7-9).
Second, identifying the logical operation used over participant terms in Rule 3. Since or (to represent inclusive disjunction on participants) in Rule 3 is employed, the cardinality lone for each signature for participant_a, participant_b, until participant_n constrain the inclusive disjunction of the given participants. lone here defines either one of the participant_1’s subsignatures, or the combination of them, or each of them, execute the sending (receiving) of the event_1.

Third, identifying verb in Rule 3. verb connects with each participant inside the participant_1 set and the event_1. In view of each nested participant (participant_a, ..., participant_n) of participant_1 has similar responsibility to execute the messages exchanged of the event_1 and each of them is mutually disjoint, these are reason why the verb maps to event_1 in line 6 is populated in the abstract sig participant_1 extends participant (line 6).

Lastly, the expression "... at exactly one T_2" in Rule 3 is translated using the same method as for Rule 2. at2 which is mapped to t2_event_1 (line 10), is exploited to represent the verb at associating with T_2 for event_1 in Rule 3.

- **A transformation of the SBVR rules describing a conjunction (AND) on participants perform the messages exchanged of exactly one particular event.**

The general form of rules which captures the conjunction on participants sending (receiving) the messages exchanged of the particular event is illustrated as in Rule 4 and Rule 5. The dissimilarity between these rules are Rule 5 relates to a nesting of participant_1 involve in the messages exchanged. By contrast, Rule 4 concerns on the independent participants obligate in the interaction concurrently. The following Alloy codes snippet (lines 11-18) are produced by these SBVR rules capturing the aforementioned participation constraints.

---

**SBVR rules in the SBVR model:**

**Rule 4:** It is obligatory that the participant_1 and the participant_2 and... and the participant_N, verb exactly one event_1 at exactly one T_1

**Rule 5:** It is obligatory that the participant_1 that includes the participant_a and the participant_b and ... and the participant_n, verb exactly one event_1 at exactly one T_2

---

The signatures which correspond to the above SBVR rules in the Alloy model:

```alloy
-- A transformation of Rule 4
11 one sig participant_1 extends participant{verb: one event_1}
12 one sig participant_2 extends participant{verb: one event_1}
...```

---
The steps taken to translate the previous Rule 2 and Rule 3 into Alloy are employed for Rule 4 and Rule 5. However, the choice of one cardinality for each signature for participants (lines 11-13 and lines 15-17) is applied due to the logical conjunction (and) used over the participant terms in the SBVR rules. The cardinality one enables Alloy produces a choreography exhibits each declared participant performs the messages exchanged of exactly one event_1 concurrently. The exception is only for abstract sig participant_1 extends participant (line 14) as the participant_1 represents a superset (a group of nesting) as expressed in Rule 5.

A transformation of the SBVR rules describing an exclusive (XOR) disjunction on participants perform the messages exchanged of exactly one particular event.

Rule 6 illustrates the general form of rule expressing the exclusive disjunction (XOR) on participants, which is exactly one participant inside a group of nesting participant_1 is chosen to perform the activity of sending (receiving) the event_1. This rule is translated into Alloy as depicted in the following code snippets.

---

SBVR rules in the SBVR model:

**Rule 6**: It is obligatory that the participant_1 that includes the participant_1 or the participant_1 or ... or the participant_1 but not all, verb exactly one event_1 at exactly one T_1

---

The signatures which correspond to the above SBVR rules in the Alloy model:

---

```
13 one sig participant_n extends participant{verb: one event_1}
...
...
--A transformation of Rule 5
14 abstract sig participant_1 extends participant{verb: one event_1}
15 one sig participant_a extends participant_1{}
16 one sig participant_b extends participant_1{}
...
17 one sig participant_n extends participant_1{}
...
...
18 one sig event_1 extends event{at1: one t1_event_1, at2: one t2_event_1}
```
or
(participant_1 = participant_b and no participant_a and ... and no participant_n) or
... or (participant_1 = participant_n and no participant_a and ... and no participant_b)
21 lone sig participant_a extends participant_1{}
22 lone sig participant_b extends participant_1{}
...
23 lone sig participant_n extends participant_1{}
...
...
24 one sig event_1 extends event{at: one t1_event_1}

It can be seen that the abstract signature for participant_1 (line 19) represents the superset of the subsignatures for the participant_a, participant_b, up to participant_n. In this case, this participant_1, which in itself includes a XOR on its subsignatures of the aforementioned participants. The accompanying fact (line 20) and the lone cardinality (lines 21-23) on the associated each signature for participant_a, participant_b until participant_n, ensure that exactly one participants (subsignatures) executes the activity of sending (receiving) the event_1. Since all the subsignatures of participant_1 are mutually disjoint and they are belonged to the abstract signature of participant_1, the verb field which is mapped to one event_1 is defined in the abstract sig participant_1 extends participant (line 19). The similar approach that used in the previous rules is exploited to define at field which is related to signature event_1 and the time t1_event_1.

**SBVR rules - messages exchanged constraints**

Section 2.3.2 in Chapter 3 was described how the SBVR model captures the specification of messages exchanged relating to the concurrent and the alternative interactions. The concurrent interaction encapsulates when the participant sends (receives) each specified event (AND logical operation on events - see Rule 7 and Rule 8 below). This contrasts with the alternative interaction which concerns on the interaction when the participant(s) is able to choose either at least one of possible choices on events (OR logical operation on events - see the following Rule 9 and Rule 10) or exactly one of them (XOR logical operation on events - see Rule 11 below).

This section provides a description of transforming the specification constraining the events only, i.e. AND on events, OR on events, and XOR on events, in the SBVR rules into Alloy. The transformation on participation constraints part, i.e a single participant, AND on participants, OR on participants, XOR on participants, in the
4.4. A TRANSFORMATION OF SBVR RULES

SBVR rules could be referred to Section 4.4.1. This means the whole transformation of the specified rules (Rule 7 to Rule 11) in this section can be implemented by replacing verb field(s) and the accompany codes in each intended signature and abstract signature in Section 4.4.1 with any verb field(s) and the related codes define in this section. Ensure that each aforementioned intended signature and abstract signature corresponds with the participation constraints specify in the SBVR rules in this section.

- A transformation of SBVR rules pertaining to conjunction (AND) of events, which are performed by a single or multiple participants

There are two kind of structures of rules in the SBVR model. The first structure involves each distinct event inside the event set (Rule 7) are performed by the participant. On the other hand, the second structure relates to each different events that are grouped together, i.e. a group of nesting (Rule 8). Both rules employ the logical operation and over the event terms. It is important to notice that the participant in both rules refer to a single (exactly one) participants or the exclusive disjunction of participants (XOR on participants) only. The specification and its transformation into Alloy for these such participation constraints could be referred to Rule 1 for a single participant and Rule 6 for XOR on participants.

SBVR rules in the SBVR model:

Rule 7: It is obligatory that the ..... , verb exactly one event\textsubscript{1} at exactly one \(T_1\) and exactly one event\textsubscript{2} at exactly one \(T_1\) and ... and exactly one event\textsubscript{N} at exactly one \(T_1\)

Rule 8: It is obligatory that the ..... , verb exactly one event\textsubscript{1} that includes exactly one event\textsubscript{a} and exactly one event\textsubscript{b} and ... and exactly one event\textsubscript{n}, at exactly one \(T_1\)

The signatures which correspond to the above SBVR rules in the Alloy model:

```
--A transformation of Rule 7
25 ..... extends participant\{verb1: one event\_1, verb2: one event\_2, ..., verb3: one event\_N\}
26 one sig event\_1 extends event\{at: one t1\_event\_1\}
27 one sig event\_2 extends event\{at: one t1\_event\_2\}
...
28 one sig event\_N extends event\{at: one t1\_event\_N\}
```

```
--A transformation of Rule 8
29 ..... extends participant\{verb1: one event\_a, verb2: one event\_b, ..., verb3:
```
The transformation of the messages exchanged constraints part in Rule 7 into Alloy can be seen in the declaration of codes in lines 25-28. It begins with identifying the verb associating with the intended participant and the specified events in the rule. Populate the appropriate fields which represent verb in Rule 7 in the intended signature (abstract signature) of participant. However, it is worth to identify which event associates with the verb and involves in the execution of messages exchanged. As \( \text{event}_1, \text{event}_2 \) until \( \text{event}_N \) are specified in the rule, hence the different fields \( \text{verb}_1, \text{verb}_2, \) and \( \text{verb}_3 \) representing the same verb in Rule 7 are declared in the signature (abstract signature) of participant (line 25). With reference to those specified events, the signatures for \( \text{event}_1 \) (line 26), \( \text{event}_2 \) (line 27), up to \( \text{event}_N \) (line 28) are considered in Alloy and each of them is bound to one (exactly one) cardinality. It is then followed by the identification of the logical operation employed in the rule.

and logical operation is found in Rule 7 to indicate each event involves in the interaction. In conjunction with this purpose, one cardinality which is mapped with each declared verb and is connected to each aforementioned event is taken into consideration to be defined as shown in line 25. Finally, the specification in the rule showing the time associates with each event is identified, e.g. ... \( \text{event}_1 \) at exactly one \( T_1 \). Since each event connects with the same verb \( at \), at field is declared in each signature for event (lines 26-28). Although each verb \( at \) relates to the identical time \( T_1 \) in the rule, different expression for time is declared in Alloy. This is because each time \( T_1 \) associates with distinct events. As a result, \( t_1 \text{event}_1 \) (line 26) denotes the time \( T_1 \) for the \( \text{event}_1 \), \( t_1 \text{event}_2 \) (line 27) refers to \( T_1 \) for the \( \text{event}_2 \), and \( t_1 \text{event}_N \) (line 28) represents \( T_1 \) for the \( \text{event}_N \). one cardinality is considered for each time in Alloy to reflect the quantification exactly one for each time \( T_1 \).

The above code snippets from lines 29-33 capture the specification of Rule 8 where the specified unordered events inside the \( \text{event}_1 \) set, are performed by the intended participant. The steps taken to translate Rule 8 into Alloy is similar to the method is implemented to translate Rule 7 into Alloy. However, three different aspects between Rule 7 and Rule 8 are taken into account. Firstly, it relates to the mapping of each field \( \text{verb}_1, \text{verb}_2, \) and \( \text{verb}_3 \) in the signature (abstract signature) as depicted in line 29. With reference to Rule 8, the verb associates with each event acts as the subsets of the \( \text{event}_1 \).
Due to this reason, each field denoting verb in line 29 is connected with the subsignatures of the abstract signature for event\_1, namely event\_a, event\_b, until event\_n. Secondly, it pertains to the declaration of the abstract signature for event\_1 (line 30). This expresses the nesting of event\_1 in Rule 8. The subsets of the event\_1 encapsulates the extensions of event\_1, which are declared as the subsignatures one sig .... (lines 31-33). Thirdly, as discussed earlier in the previous sections, since the abstract signature for event\_1 has no elements except those belonging to its extensions characterising multiple disjoint, the at field is declared in that abstract signature (line 30).

- **A transformation of SBVR rules** pertaining to inclusive disjunction (OR) of events, which are performed by a single or multiple participants

The general forms of Rule 9 and Rule 10 capture the alternative interaction referring to either exactly one or some of the specified events (OR on events) are selected in the interaction. The choices of events apply to distinct events inside the event set (Rule 9) or the dissimilar events of a nesting group (Rule 10). The logical operation or is employed over the event terms in both rules. Similar to the previous Rule 7 and Rule 8, the participant in the rules refer to a single (exactly one) participant or the exclusive disjunction of participants (XOR on participants).

-- SBVR rules in the SBVR model:

**Rule 9:** It is obligatory that the ..... , verb exactly one event\_1 at exactly one T\_1 or exactly one event\_2 at exactly one T\_1 or ... or exactly one event\_N at exactly one T\_1

**Rule 10:** It is obligatory that the ..... , verb exactly one event\_1 that includes exactly one event\_a or exactly one event\_b or ... or exactly one event\_p, at exactly one T\_1

The signatures which correspond to the above SBVR rules in the Alloy model:

```
34 ...... extends participant{verb1: lone event_1, verb2: lone event_2, ..., verb3: lone event_N}
35 {#verb1 = 1 or #verb2 = 1 or ... or #verb3 = 1}
36 one sig event_1 extends event{at: one t1_event_1}
37 one sig event_2 extends event{at: one t1_event_2}
... 38 one sig event_N extends event{at: one t1_event_N}
...```
A transformation of Rule 10

39 \ldots \text{ extends participant\{verb: some event\_1 \}}
40 \{\#verb \geq 1\}
41 \text{abstract sig event\_1 extends event\{at: one t1\_event\_1\}}
42 \text{lone sig event\_a extends event\_1\{\}}
43 \text{lone sig event\_b extends event\_1\{\}}
44 \ldots
45 \text{lone sig event\_n extends event\_1\{\}}

The translation of Rule 9 and Rule 10 into Alloy follow the same translation steps as described for Rule 7 and Rule 8. However, the contrasting aspect between those rules is the purpose of rule specification that characterises from the use of the logical operation in the SBVR rules, i.e. and for the execution of conjunction of events and or for the execution of inclusive disjunction of events. This affects the declaration in Alloy in terms of the use of a fact and the different cardinality.

The codes in Alloy for Rule 7 uses one cardinality to connect with each specified event which is mapped by each verb field (line 25). This reflects the SBVR rule (Rule 7) which specifies the intended participant executes the messages exchanged of a sending (receiving) of each unordered event (AND on events). On the other hand, the codes in Alloy for Rule 9 employs lone cardinality (line 34) associating with each event and mapping from each corresponding verb that connects with the signature (abstract signature) for the intended participant. This declaration is to ensure the verb is mapped to at most one corresponding event to capture OR on events. Also, the fact in line 35 is declared in order to constrain more on the sending (receiving) at least one of the selected events by the intended participant, as specified in Rule 9.

As can be seen the codes in Alloy for Rule 8 and Rule 9, we notice there are some discrepancy. First, since the intended participant aims to send (receive) each event (the subsets of event\_1) in the interaction as described in Rule 8, this is necessarily each defined verb field (line 29) maps to the intended event as specified in the rule, i.e. \ldots \text{verb\_1: one event\_1, \ldots, verb\_3: one event\_n}. This contrasts with the declaration code (line 39) representing the intended participant targeting to send (receive) at least one of the events inside the nesting of event\_1 as described in Rule 10. This is why verb field maps to abstract signature for event\_1, which is bound to some cardinality. The declaration of some cardinality associating with the event\_1 is the second difference. Third, the fact (line 40), which is defined for the signature (abstract signature) for the intended participant, is purposely to guarantee the sending (receiving) of inclusive disjunction of the events by the intended participant is at least one. Finally, lone cardinality associating with each subsignature for
4.4. A TRANSFORMATION OF SBVR RULES

A transformation of SBVR rules pertaining to exclusive disjunction (XOR) of events, which are performed by a single or multiple participants

Rule 11 represents the general form of rule capturing the alternative interaction concerns on the sending (receiving) of exactly one event inside the nesting of given event (XOR on events). The logical operation or ... but not all (XOR on events) is exploited over the event terms. Dissimilar to the rules which specify the messages exchanged of OR on events and AND on events, this rule allows the specification of a single (exactly one) participant, or the inclusive disjunction (OR) of participants, or the conjunction (AND) of participants to execute the messages exchanged of the exclusive disjunction of events. Rule 11 is translated into Alloy by using the codes as declared in the following snippet.

SBVR rules in the SBVR model:

Rule 11: It is obligatory that the ..... , verb exactly one event1 that includes exactly one eventa or exactly one eventb or ... or exactly one eventn but not all, at exactly one Ti

The signatures which correspond to the above SBVR rules in the Alloy model:

```alloy
--A transformation of Rule 11
45 ..... extends participant{verb: one event_1 } ...
46 abstract sig event_1 extends event{at: one t1_event_1}
47 { (event_1 = event_a and no event_b and ... and no event_n) or (event_1 = event_b and no event_a and ... and no event_n) or ... or (event_1 = event_n and no event_a and ... and no event_b) }
48 lone sig event_a extends event_1{}
49 lone sig event_b extends event_1{}
50 lone sig event_n extends event_1{}
```

Note that the translation on the specification for participants in the rule into Alloy could be referred to the transformation of rules pertaining to the participation constraints in Section 4.4.1. It can be replaced into the code snippet in line 45. The next step relates to allocate the verb field representing the verb in Rule 11. The verb field that maps to event_1 is declared in the signature (abstract signature) for the intended participant. The approach for translating Rule 10 is applied for translating Rule 11 as well. However, one
cardinality instead of some cardinality associated on event\textsubscript{1} is populated (line 45) to capture Rule 11. This confirms the intended participant will be sending (receiving) exactly one event of event\textsubscript{1}’s subsets. It is essential to define the fact for event\textsubscript{1} as depicted in line 47 to guarantee only one event (e.g. event\textsubscript{a}, or event\textsubscript{b}, ..., or event\textsubscript{n} and not the combination of them) will be true once event\textsubscript{1} is invoked. To be more precise, lone cardinality is declared for each subsignature of event\textsubscript{1}.

\textbf{SBVR rules - special concerns on messages exchanged constraints}

The rules which concern on the particular specification of messages exchanged are referred to the SBVR rules discussed in Section 2.3.2. Note that the produced code snippets here are translated from the given rules in Section 2.3.2. There are two types of rules as in the following.

- Type I : The rules which specify exactly one participant executing the messages exchanged by selecting the particular event(s) (not all of them) in a nesting of event set.

By assuming the event\textsubscript{3} comprises event\textsubscript{a}, event\textsubscript{b}, and event\textsubscript{c}; and event\textsubscript{4} encompasses event\textsubscript{d} and event\textsubscript{e}, the following rules are taken into consideration.

\textbf{SBVR rules in the SBVR model:}

\textbf{Rule 1}: It is obligatory that the \textit{participant\textsubscript{1}} verb exactly one \textit{event\textsubscript{3}} that includes exactly one event\textsubscript{a}

\textbf{Rule 2}: It is obligatory that the \textit{participant\textsubscript{1}} verb exactly one \textit{event\textsubscript{3}} that includes exactly one event\textsubscript{a} or exactly one event\textsubscript{b}

\textbf{Rule 3}: It is obligatory that the \textit{participant\textsubscript{1}} verb exactly one \textit{event\textsubscript{3}} that includes exactly one event\textsubscript{b} and exactly one event\textsubscript{c}

\textbf{Rule 4}: It is obligatory that the \textit{participant\textsubscript{1}} verb exactly one \textit{event\textsubscript{4}} that includes exactly one event\textsubscript{d} or exactly one event\textsubscript{e} but not both

\textbf{Rule 5}: It is obligatory that the \textit{participant\textsubscript{4}} verb exactly one \textit{event\textsubscript{4}} that includes exactly one event\textsubscript{d}

The signatures which correspond to the above SBVR rules in the Alloy model:

---A transformation of Rule 1
4.4. A TRANSFORMATION OF SBVR RULES

1 one sig participant_1 extends participant {verb: one event_a }
2 abstract sig event_3 extends event {} 
3 one sig event_a extends event_3 {} 
4 one sig event_b extends event_3 {} 
5 one sig event_c extends event_3 {} 
... 
--A transformation of Rule 2
6 one sig participant_1 extends participant {verb_1: lone event_a, verb_2: lone event_b} 
7 {#verb_1 = 1 or #verb_2 = 1} 
8 abstract sig event_3 extends event {} 
9 lone sig event_a extends event_3 {} 
10 lone sig event_b extends event_3 {} 
11 lone sig event_c extends event_3 {} 
... 
--A transformation of Rule 3
12 one sig participant_1 extends participant {verb_1: one event_b, verb_2: one event_c} 
13 abstract sig event_3 extends event {} 
14 one sig event_a extends event_3 {} 
15 one sig event_b extends event_3 {} 
16 one sig event_c extends event_3 {} 
... 
--A transformation of Rule 4 and Rule 5
17 one sig participant_1 extends participant {verb_1: one event_4} 
18 one sig participant_4 extends participant {verb_1: one event_d} 
19 abstract sig event_4 extends event {} 
20 { (event_4 = event_d and no event_e) or 
    (event_4 = event_e and no event_d) } 
21 lone sig event_d extends event_4 {} 
22 lone sig event_e extends event_4 {} 

Rule 1 elaborates when the participant\textsubscript{1} aims to execute event\textsubscript{a} only, although event\textsubscript{3} has the other subsets event\textsubscript{b} and event\textsubscript{c}. The translation of this rule into Alloy is illustrated as the above code snippet in line 1. Since only event\textsubscript{a} is chosen, verb field is mapped to event\textsubscript{a} only, which is connected to one cardinality.

The specification for Rule 2 is slightly different with Rule 1. The participant\textsubscript{1} selects event\textsubscript{a} or event\textsubscript{b} in the nest of event\textsubscript{3} (OR on events), where event\textsubscript{c} is not selected. To transform this rule into Alloy, two fields for one sig participant\textsubscript{1} . . . (line 6) is required. verb\textsubscript{1} field is designated for event\textsubscript{a} and verb\textsubscript{2} field is defined for event\textsubscript{b}. Both events interconnected
with logical operation or (OR on events), which is why the following three declarations are considered: i). lone cardinality is declared for each event \texttt{event.a} and \texttt{event.b} in the signature for \texttt{participant.1} (line 6), ii). lone cardinality is also be defined for each subsignature of \texttt{event.3} (lines 9-11) even though \texttt{event.c} is not be chosen, and iii). the fact for one sig \texttt{participant.1} .... is added to ensure Alloy generates the specification as specified in Rule 2.

Rule 3 expresses that each event inside the \texttt{event.3} set excepts \texttt{event.a} is executed in the interaction by the \texttt{participant.1}. Rule 2 and Rule 3 have the similarity in a way on how to transform the rules into Alloy. The different is only on the use of one cardinality for each specified event in the signature for \texttt{participant.1} (line 12) and for each subsignature as declared in lines 14-16, as well as no fact for signature participant.1 is needed.

The concerns on Rule 4 and Rule 5 is when there are two distinct choices of events in the \texttt{event.4} set (XOR on events). However, one participant decides to choose exactly one of the listed events, namely \texttt{event.c} or \texttt{event.d} but not both, and the other participant opts to execute one event only, that is \texttt{event.d} \texttt{(event.c} in not be selected). The transformation of Rule 4 and Rule 5 into Alloy is defined in the above code snippets in lines 17-22. Line 17 captures Rule 4 to ensure the \texttt{participant.1} executes the exclusive disjunction (XOR) on \texttt{event.4}'s subsets. This can be seen on the declaration of \texttt{verb.1} field in the signature \texttt{participant.1}, which is mapped to \texttt{event.4} which encapsulates both events \texttt{event.c} or \texttt{event.d}. On the other hand, line 18 represents Rule 5, where the defined \texttt{verb} field is mapped to one \texttt{event.d} only.

- Type II: The rules which specify some participants who are specified as a nesting of group require to send (receive) different event(s).

Rule 6 to Rule 8 concern on the participation constraints whenever there is a nesting of \texttt{participant.2} which includes \texttt{participant.a}, \texttt{participant.b}, and \texttt{participant.c}. Nevertheless, not all participants inside the \texttt{participant.2} set have been considered.

Rule 6 and Rule 7 have the similarity since only two participants inside the \texttt{participant.2} set have been specified to execute the messages exchanged of the \texttt{event.1}. A contrary logical operation over participant terms in both rules causes lone cardinality is defined for each subsignature of \texttt{participant.2} for Rule 6 while one cardinality is declared for each subsignature of \texttt{participant.2} for Rule 7. \texttt{verb} field is populated only in the intended signatures, that are signature for \texttt{participant.a} and \texttt{participant.b} (lines 24-25 and lines 29-30).

Rule 8 shows the \texttt{participant.c} is only be chosen in the interaction for sending (receiving) the \texttt{event.2}. In this situation, \texttt{verb} field is declared in the subsignature for \texttt{participant.c} only (line 36).
SBVR rules in the SBVR model:

Rule 6: It is obligatory that the \texttt{participant}_2 that includes the \texttt{participant}_a or the \texttt{participant}_b, \texttt{verb} exactly one \texttt{event}_1.

Rule 7: It is obligatory that the \texttt{participant}_2 that includes the \texttt{participant}_a and the \texttt{participant}_b, \texttt{verb} exactly one \texttt{event}_1.

Rule 8: It is obligatory that the \texttt{participant}_2 that includes the \texttt{participant}_c, \texttt{verb} exactly one \texttt{event}_2.

Rule 9: It is obligatory that the \texttt{participant}_2 that includes the \texttt{participant}_c, \texttt{verb} exactly one \texttt{event}_2 that includes exactly one \texttt{event}_a or exactly one \texttt{event}_b.

Rule 10: It is obligatory that the \texttt{participant}_2 that includes the \texttt{participant}_c, \texttt{verb} exactly one \texttt{event}_3 that includes exactly one \texttt{event}_a and exactly one \texttt{event}_b.

Rule 11: It is obligatory that the \texttt{participant}_2 that includes the \texttt{participant}_c, \texttt{verb} exactly one \texttt{event}_3 that includes exactly one \texttt{event}_a or exactly one \texttt{event}_b, but not both.

The signatures which correspond to the above SBVR rules in the Alloy model:

```alloy
-> A transformation of Rule 6
23 abstract sig participant_2 extends participant{}
24 lone sig participant_a extends participant_2{verb: one event_1}
25 lone sig participant_b extends participant_2{verb: one event_1}
26 lone sig participant_c extends participant_2{}
...
27 one sig event_1 extends event
...

-> A transformation of Rule 7
28 abstract sig participant_2 extends participant{}
29 one sig participant_a extends participant_2{verb: one event_1}
30 one sig participant_b extends participant_2{verb: one event_1}
31 one sig participant_c extends participant_2{}
...
32 one sig event_1 extends event
...

-> A transformation of Rule 8
33 abstract sig participant_2 extends participant{}
34 one sig participant_a extends participant_2{}
```
Rule 9 to Rule 11 concern on exactly one participant, that is \textit{participant}_c.
from a nesting of \textit{participant}_2 is opted to perform the messages exchanged of either at least one of the specified events (OR on events) (Rule 9), or each specified event (AND on events) (Rule 10), or exactly one of the specified events (XOR on events) (Rule 11). However, note that not all events inside the \textit{events}_3 set are chosen.

The translation of Rule 9, Rule 10, and Rule 11 into Alloy can be seen in the above code snippets from lines 38-46, lines 47-54, and lines 55-62, respectively. Each intended field for representing \textit{verb} in the rule is populated in the extension of \textit{participant}_2, that is the subsignature for \textit{participant}_c since only this participant is chosen in the interaction. Each field is mapped to the intended events but with different cardinality.

\section{SBVR rules - precedence (temporal operator)}

The precedence that is concerned in the development of the SBVR model relates to two notions of temporal operators, namely \textit{time notion} and \textit{immediate precedes notion} as described in details in Chapter 3 (Section 2.3.2). The following sections discuss on how to transform the temporal operators introduced in the SBVR model into Alloy.

\section*{SBVR rules - time notion}

The purpose of \textit{time notion} is to capture the ordering of messages exchanged between the different participants performing the same event, which is incorporated in composing the local behaviour model (Section 1). The composition of local behaviour models involves organising the export and import messages from the distinct local behaviour models (participants).

The \textit{time notion} in Alloy starts with a declaration of abstract signature for time (line 7). As discussed earlier in Section 4.2, Time term (\textbf{T}) in the SBVR model is transformed as the \textbf{sig time} \{} \textbf{\} in Alloy, initially. This signature for time must be replaced with the abstract signature (line 7) due to its role as a superset of its subsets, that are a set of time for each event, e.g. \textit{time_event}_1 reflects a set of time for \textit{event}_1 and \textit{time_event}_2 represents a set of time for \textit{event}_2.

Next, since each each extension of \textbf{abstract sig time}.. contains \textit{n} number of time, it is necessarily to be declared as the abstract signature (line 8 and line 11). Each time \textit{t}1.., \textit{t}2... , ..., \textit{t}n (lines 9-10 and line 12) connecting with the intended event is defined as the subsignature of the corresponding abstract signature for the time set for each event.
CHAPTER 4. GENERATION AND VERIFICATION CHOREOGRAPHY MODEL

Rule 1: It is obligatory that the \textit{participant} \textsubscript{1} \textit{verb} exactly one \textit{event} \textsubscript{1} at exactly one \textit{T} \textsubscript{1}

The local behaviour model of the \textit{participant} \textsubscript{2} in the SBVR model:
Fact Type 2: \textit{participant} \textsubscript{2} \textit{verb} \textit{event} \textsubscript{1} at \textit{T} \textsubscript{2}

Rule 2: It is obligatory that the \textit{participant} \textsubscript{2} \textit{verb} exactly one \textit{event} \textsubscript{1} at exactly one \textit{T} \textsubscript{2}

The local behaviour model of the \textit{participant} \textsubscript{3} in the SBVR model:
Fact Type 3: \textit{participant} \textsubscript{3} \textit{verb} \textit{event} \textsubscript{2} at \textit{T} \textsubscript{1}

Rule 3: It is obligatory that the \textit{participant} \textsubscript{3} \textit{verb} exactly one \textit{event} \textsubscript{2} at exactly one \textit{T} \textsubscript{1}

The local behaviour model of the \textit{participant} \textsubscript{4} in the SBVR model:
Fact Type 4: \textit{participant} \textsubscript{4} \textit{verb} \textit{event} \textsubscript{2} at \textit{T} \textsubscript{1}

Rule 4: It is obligatory that the \textit{participant} \textsubscript{4} \textit{verb} exactly one \textit{event} \textsubscript{2} at exactly one \textit{T} \textsubscript{1}

The precedence rule in the SBVR model:
Rule 5: It is obligatory that the \textit{event} \textsubscript{1} at exactly one \textit{T} \textsubscript{1} immediately precedes \textit{event} \textsubscript{1} at exactly one \textit{T} \textsubscript{2}

The signatures which correspond to each local behaviour model:

1 one sig participant\textsubscript{1} extends participant\{verb: one \textit{event} \textsubscript{1}\}

...  

2 one sig participant\textsubscript{2} extends participant\{verb: one \textit{event} \textsubscript{1}\}

...  

3 one sig participant\textsubscript{3} extends participant\{verb: one \textit{event} \textsubscript{2}\}

...  

4 one sig participant\textsubscript{4} extends participant\{verb: one \textit{event} \textsubscript{2}\}

...  

5 one sig \textit{event} \textsubscript{1} extends \textit{event}\{at1: one \textit{t1_event} \textsubscript{1}, at2: one \textit{t2_event} \textsubscript{1}\}

...  

6 one sig \textit{event} \textsubscript{2} extends \textit{event}\{at: one \textit{t1_event} \textsubscript{2}\}

--Time notion in Alloy
7 abstract sig time{}  

--Time associating with \textit{event} \textsubscript{1}
4.4. A TRANSFORMATION OF SBVR RULES

Lastly, it is followed with the declaration of each participant and each event signature which correspond to each specified rule (lines 1-6). Previously, the method for transforming the fact type and the SBVR rule concerning the time (a part of time notion) has been described as similar as in the following fact types and rules and its transformation in lines 1-6. Each fact type which makes up the corresponding rule illustrates the export (import) message in each local behaviour model. This encapsulates the composition of local behaviour models such as Fact Type 1 and Fact Type 2 are composed to represent the messages exchanged of the event (one of participants exports (import) the message) and to describe the ordering of messages exchanged of the event (the participant initiates the interaction at , then precedes by the participant at ).

In order to compose the local behaviour model completely in Alloy, each signature such as one sig t1_event_1..., one sig t2_event_1... (lines 9-10) has a field by: one participant (line 9) and by: one participant (line 10), respectively, mapping to the intended participant. This illustrates which participant performs the intended event at the specified time to reflects the corresponding fact type as well as the SBVR rule. For instance, the composition of local behaviour models of participant (Fact Type 1 and Rule 1) and participant (Fact Type 2 and Rule 2) are represented in the declarations of signature for participant (line 1), participant (line 2), event (line 5), time_event_1 (line 8) and its subsignatures in lines 9-10.

Rule 3 and Rule 4 show the messages exchanged of the same event, nevertheless both participants execute the interaction at the same time interval . This describes the occurrence of event by the participant and the participant is concurrent. In Alloy, it is transformed by the declaration of two fields of by1 and by2 associate with the participant and the participant, respectively in the signature for t1_event_2 (line 9).

SBVR rules - immediate precedes notion

The immediate precedes notion captures two intentions of developing the SBVR model for choreography (detailed see Section 2 in Chapter 3). The first aim is to order the time interval associating with the same event which is performed by the different participants as described in Section 4.4.2. The other aim is to manage the
ordering of the messages exchanged of distinct events.

The time notion is advocated to indicate the ordering of messages exchanged between different participants and to encapsulate the composition of local behaviour models. However, there is no clear clue which time interval initiates the interaction and takes place afterwards. Hence, to constrain it more the immediate precedes notion is exploited. Rule 5 in Section 4.4.2 employs the verb *immediately precedes* in between \( T_1 \) and \( T_2 \), which are engaged by the same \( \text{event}_1 \). This is vital to ensure the ordering between the specified time interval. Therefore, the additional field *immediately precedes* in the signature \( t_1\_\text{event}_1 \) which is mapped to \( t_2\_\text{event}_1 \) implies Rule 5.

The next intention of the immediate precedes notion is used in the general form of rule: It is obligatory that exactly one \( \text{term}_1 \) *immediately precedes* exactly one \( \text{term}_2 \) as prescribed in Section 3.1.1, where \( \text{term}_1 \) and \( \text{term}_2 \) refer to event terms. Assume that the statement of "exactly one \( \text{term}_1 \)" as P statement, and the statement of "exactly one \( \text{term}_2 \)" as Q statement. The actual rule describing the ordering of different events is a combination of P and Q statements as described in the following rules.

---

**SBVR rules relating to the P statements in the SBVR model:**

**Rule 1:** It is obligatory that exactly one \( \text{event}_1 \) *immediately precedes* ...

**Rule 2:** It is obligatory that exactly one \( \text{event}_1 \) or ... or exactly one \( \text{event}_N \) *immediately precedes* ...

**Rule 3:** It is obligatory that exactly one \( \text{event}_1 \) that *includes* exactly one \( \text{event}_a \) or exactly one \( \text{event}_b \) or ... or exactly one \( \text{event}_n \) *immediately precedes* ...

**Rule 4:** It is obligatory that exactly one \( \text{event}_1 \) and ... and exactly one \( \text{event}_N \) *immediately precedes* ...

**Rule 5:** It is obligatory that exactly one \( \text{event}_1 \) that *includes* exactly one \( \text{event}_a \) and exactly one \( \text{event}_b \) and ... and exactly one \( \text{event}_n \) *immediately precedes* ...

**Rule 6:** It is obligatory that exactly one \( \text{event}_1 \) that *includes* exactly one \( \text{event}_a \) or exactly one \( \text{event}_b \) or ... or exactly one \( \text{event}_n \) but not all *immediately precedes* ...

**Rule 7:** It is obligatory that exactly one \( \text{event}_1 \) that *includes* exactly one \( \text{event}_a \) *immediately precedes* ...

---

The signatures which correspond to the above SBVR rules in the Alloy model:
4.4. A TRANSFORMATION OF SBVR RULES

--- A transformation of Rule 1
1 one sig event_1 extends event{immediatelyprecedes: ... }

--- A transformation of Rule 2
2 lone sig event_1 extends event{immediatelyprecedes: ... }

... 3 lone sig event_N extends event{immediatelyprecedes: ... }

--- A transformation of Rule 3
4 abstract sig event_1 extends event{immediatelyprecedes: ... }
5 lone sig event_a extends event_1{}

... 6 lone sig event_n extends event_1{}

--- A transformation of Rule 4
7 one sig event_1 extends event{immediatelyprecedes: ... }

... 8 one sig event_N extends event{immediatelyprecedes: ... }

--- A transformation of Rule 5
9 abstract sig event_1 extends event{immediatelyprecedes2: ... }
10 one sig event_a extends event_1{}

... 11 one sig event_n extends event_1{}

--- A transformation of Rule 6
12 abstract sig event_1 extends event{immediatelyprecedes: ... }

13 {(event_1 = event_a and ... and no event_n) or (event_1 = event_n and ... and no event_a)}

14 lone sig event_a extends event_1{}

... 15 lone sig event_n extends event_1{}

--- A transformation of Rule 7
16 abstract sig event_1 extends event{}

... 17 ... sig event_a extends event_1{immediatelyprecedes: ... }

... 18 ... sig event_n extends event_1{}

__________________________________________________________________________

The above rules Rule 1 to Rule 6 prescribe the possible P statements that are able to be applied in the aforementioned general form of rule. Each rule has its own purpose. Rule 1 describes when the messages exchanged of exactly one event occurs immediate before the messages exchanged of the other event(s). Then, Rule 2 and Rule 3 express the interaction involving the inclusive disjunction (OR) of different events inside the event set (Rule 2) and the inclusive disjunction (OR) of events from a nesting of the event_1 (Rule 3). These kind of events occur right
before the next interaction. Next, the expression of rules to describe the messages exchanged of unordered events (AND) inside the event set or inside the nesting of event occur right before the messages exchanged of the other event(s), are prescribed in Rule 4 and Rule 5, respectively. Lastly, Rule 6 expresses the interaction of multi-party conversations involving the exclusive disjunction (XOR) of events, which is executed immediate before the next interaction. All rules are transformed into Alloy as declared in the above codes snippet (lines 1-15).

The transformation of the above rules take the following steps. The first thing that must be considered is to identify the first event term in the P statement as illustrated in Rule 1 to Rule 6 above. Since the first event term in each rule is the $event_1$, the next step is finding the next word after $event_1$. The word is either the logical operations (OR/AND/XOR) (if any) or the verb immediately precedes.

Firstly, consider the next word after the $event_1$ in Rule 1, that is the verb immediately precedes. To transform this rule into Alloy, find the signature one event_1 (line 1) to capture ”exactly one $event_1$ ... ”. Then, populate immediatelyprecedes as a field of the signature for event_1 reproduces the verb immediatelyprecedes associating with the term $event_1$ as specified in Rule 1.

Secondly, the consideration is on whenever the next word after the $event_1$ is the logical operation ”or” (OR) and that logical operation is designated in between the specified events as illustrated in Rule 2. The transformation of Rule 2 into Alloy begins with finding all signatures for the specified events such as $event_1$ (line 2) and $event_N$ (line 3). Ensure that the cardinality for each signature is lone to represent at least one of the specified events (OR) are selected in the interaction. Next, the immediatelyprecedes field is populated in every signature for event_1 and $event_N$.

The dissimilar approach is taken to transform Rule 3 which employs the same logical operation ”or” (OR) in the rule. The verb includes after the term $event_1$ in Rule 3 shows the nesting of $event_1$ is specified. Once this kind of rule is noticed, find the nested events of the $event_1$ after the words includes exactly one and after each logical operation ”or” (in between the nested event terms). Transform this rule into Alloy by first finding the signature for event_1 and ensure it is the abstract signature since it has the subsets (the identified nested events) as shown in the above snippet (line 4). Then, each nested event ($event_a$ and $event_n$) is declared as the extension of $event_1$ (lines 5-6) and is associated on lone cardinality because at least one of the specified events is considered in the interaction. Afterwards, the immediatelyprecedes field is declared in the abstract signature for event_1 as each subsignature is the elements which are belonged to event_1.

Thirdly, the concern pertains to the rules whenever each unordered event is prescribed in the rule such as Rule 4 and Rule 5. Both rules employ the logical operation ”and” (AND) to illustrate its aim, however different word(s) is specified after the term $event_1$. Rule 4 specifies ”and” after the $event_1$ and in between the other specified event(s) such as $event_N$. Therefore, the signatures illustrating the specified events must be identified. The code snippets in lines 7 to 8 are defined
to illustrate these signatures, where one cardinality is bound to each signature for representing each event will be executed in the interaction. This contrasts with the translation of Rule 5 into Alloy since the verb includes is specified after the \textit{event}_1. It is aimed to show the selection of OR on events inside the nesting of \textit{event}_1 is executed in the interaction. This specification is transformed by identifying the abstract signature for \textit{event}_1 as defined in the snippet of line 9, together with its subsignatures \textit{event}_a (line 10) and \textit{event}_n (line 11). Ensure that one cardinality is declared for each subsignature. Then, the next step is applied for both rules. The \textit{immediately precedes} field is allocated in each signature for \textit{event}_1 (line 7) and \textit{event}_N (line 8) to capture Rule 4 as well as in the abstract signature for \textit{event}_1 (line 9) to reflect Rule 5.

Finally, Rule 6 contemplates the verb \textit{includes} after the term \textit{event}_1 and the logical operation "or...but not all" in between the specified event terms. To transform this rule into Alloy, the abstract signature for \textit{event}_1 (line 12) is identified and its extensions associating with lone cardinality as well as the fact for \textit{event}_1 signature are declared as shown in the snippets (lines 13-15). This essential to guarantee only one of the nested events of \textit{event}_1 is executed in the interaction immediate before the other event(s). Lastly, the \textit{immediately precedes} field is populated in the \textit{event}_1’s abstract signature.

It is important to note that in some circumstances in the choreography, there is a possibility when the verb \textit{immediately precedes} is specified after the nested event as prescribed in Rule 7. In order to capture this kind of rule, the identification of which event associates with the verb \textit{immediately precedes} in the rule, is required. In this case, the \textit{immediately precedes} field is defined in the signature for \textit{event}_a (line 17). This method is applicable for any kind of rules employing OR/AND/XOR on nested events which occur immediate before the other event(s).

Note that, in the above description, the declaration of the \textit{immediately precedes} field in each intended signature refers to the verb \textit{immediately precedes} as specified in Rule 1 to Rule 6. Nevertheless, in the real situation of choreography, the declaration of the \textit{immediately precedes} field in Alloy is supposedly based on the specification of Q statement in the SBVR rule. This will be explained in the next description of rule transformation.

<table>
<thead>
<tr>
<th>SBVR rules relating to the Q statements in the SBVR model:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rule 8: It is obligatory that ... \textit{immediately precedes} exactly one \textit{event}_2</td>
</tr>
<tr>
<td>Rule 9: It is obligatory that ... \textit{immediately precedes} exactly one \textit{event}_2 or ... or exactly one \textit{event}_N</td>
</tr>
<tr>
<td>Rule 10: It is obligatory that ... \textit{immediately precedes} exactly one \textit{event}_2 that \textit{includes} exactly one \textit{event}_a or exactly one \textit{event}_b or ... or exactly one \textit{event}_n</td>
</tr>
<tr>
<td>Rule 11: It is obligatory that ... \textit{immediately precedes} exactly one \textit{event}_2</td>
</tr>
</tbody>
</table>
and ... and exactly one $event_N$

**Rule 12:** It is obligatory that ... immediately precedes exactly one $event_2$ that includes exactly one $event_a$ and exactly one $event_b$ and ... and exactly one $event_n$

**Rule 13:** It is obligatory that exactly one $event_1$It is obligatory that ... immediately precedes exactly one $event_2$ that includes exactly one $event_a$ or exactly one $event_b$ or ... or exactly one $event_n$ but not all

The signatures which correspond to the above SBVR rules in the Alloy model:

```
--A transformation of Rule 8
19 ...... extends event{immediatelyprecedes: one event_2 }
20 one sig event_2 extends event{}
--A transformation of Rule 9
21 ...... extends event{immediatelyprecedes1: lone event_2, ..., immediatelyprecedes2: lone event_N }
22 {#immediatelyprecedes1 = 1 or ... or #immediatelyprecedes2 = 1
23 and immediatelyprecedes1 = participant_1.verb1 and immediatelyprecedes2 = participant_1.verb2}
24 one sig event_2 extends event{}
...
25 one sig event_N extends event{}
...
26 one sig participant_1 extends participant{verb1: lone event_2, verb2: lone event_N}
{.....}
--A transformation of Rule 10
27 ...... extends event{immediatelyprecedes: some event_2}
28 {#immediatelyprecedes1 >= 1
29 and immediatelyprecedes = participant_1.verb}
30 abstract sig event_2 extends event{}
31 lone sig event_a extends event_2{}
...
32 lone sig event_n extends event_2{}
...
33 one sig participant_1 extends participant{verb: some event_2}
{.....}
--A transformation of Rule 11
34 ...... extends event{immediatelyprecedes1: one event_2, ..., immediatelyprecedes2: one event_N}
35 one sig event_2 extends event{}
```
The specification as shown in Rule 8 to Rule 13 represent the possible of Q statements specify in the aforementioned general form of rule. Rule 8 describes when exactly one event in the interaction is performed. Rule 9 and Rule 10 capture the inclusive disjunction (OR) of events, on the other hand Rule 11 and Rule 12 capture the conjunction (AND) of events. Then, Rule 13 involves the exclusive disjunction (XOR) of events. All interactions are occurred immediate after the other event(s).

The transformation of the previous rules into Alloy are followed by the transformation of the required rules as given in Rule 8 to Rule 13.

After considering all the previous steps, the next step is to find which event associates after the verb *immediately precedes* in the specified SBVR rule.

In Rule 8, the *event2* is connected after the verb *immediately precedes*. Hence,
the *immediately precedes* field of the intended signature as shown in the snippet line 19 is populated. This field is mapped to *event_2* associates with one cardinality captures exactly one *event_2*.

There are distinct events (*event_2*, ..., *event_N*) with the logical operation "or" over these event terms are specified after the verb *immediately precedes* in Rule 9. To transform this rule into Alloy, firstly, find the signature (abstract signature) for the intended event. Since each distinct event is specified after the verb *immediately precedes* in the given rule, the *immediately precedes1* and the *immediately precedes2* fields are allocated in the signature (abstract signature) for the intended event (line 21). Each field maps to each intended distinct event, that is *event_2* for *immediately precedes1* field and *event_N* for *immediately precedes2* field. These events associate with lone cardinality to guarantee the specification of OR on event terms in Rule 9. This is followed by the declaration of the fact (lines 22 only) underneath the intended event signature (abstract signature). Assume that there is a rule in the SBVR model where some other participant sends (receives) (represented by *verb*) the same events as specified in Rule 9. Then the fact as shown in the above snippet (line 23) will need to be added for the intended signature (abstract signature). This fact specifies the *immediately precedes1* and the *immediately precedes2* fields are equal to that intended participant’s signature associates with its fields *verb1* and *verb2* that map to the corresponding events (line 26). Thus, *immediately precedes1* is equal to *participant_1.verb1* while *immediately precedes2* is equal to *participant_1.verb2*. The purpose of this declaration is to make sure each "*immediately precedes*" field is mapped to the same event as that is sent (received) by the specified participant. In other words, it is to ensure Alloy produces the choreography which exhibits the participant sends (receives) the inclusive disjunction of events as similar as OR of events that must be occurred immediate after the other event(s).

Similar approach applies for transforming Rule 10 into Alloy. However, only *immediately precedes* field associates with some cardinality (line 27) in the intended signature (abstract signature) of event is allocated. Also, the fact (lines 28) underneath the same signature is added to encapsulate at least one event (OR) on events inside the nesting of *event_2* is selected. The additional fact (line 29) is declared if there is some participant executes a sending (receiving) of the same event (line 33) as specified as the occurrence of event immediate after the other event(s).

The subsequent rules concern on whenever each distinct unordered event inside the event set (Rule 11) or inside a group of nesting (Rule 12) is occurred right after the other event(s). To transform these rules into Alloy, the specified events after the verb *immediately precedes* and in between the logical operation "and" for Rule 11 are identified. Since *event_2* up to *event_N* are specified in the rule, *immediately precedes1* field maps to *event_2* and *immediately precedes2* field maps to *event_N* are declared in the signature (abstract signature) for the intended event (line 34). Both fields are bound with one cardinality. The declaration for each event *event_2* and *event_N* must be defined as the signatures depicted in
the snippets (lines 35-36). Similar to Rule 12, the transformation begins with the identification of the event terms after the verb immediately precedes. The \( \text{event}_2 \) including its subsets \( \text{event}_{a} \) until \( \text{event}_{n} \) are specified with the logical operation ”and” in between the event terms (subsets of the \( \text{event}_2 \)). Since the intended event(s) occurs immediate before each subsets of the \( \text{event}_2 \), each field in the signature (line 37), that are immediatelyprecedes\(_1\) associating with one \( \text{event}_a \) and immediatelyprecedes\(_2\) associating with one \( \text{event}_n \) is declared. Furthermore, \( \text{event}_2 \) is defined as the abstract signature (line 38), which is a superset of its subsignatures (lines 39-40).

Finally, Rule 13 capturing the messages exchanged of XOR on events immediate after the other event(s) is translated into Alloy as defined in the codes snippet (lines 41-47). The transformation starts with finding the event (\( \text{event}_2 \)) and its nested events together with the logical operation ”or...but not all” after the verb immediately precedes. Therefore, the immediatelyprecedes field associates with one \( \text{event}_2 \) is defined in the signature for the intended event (line 41). This is followed with the declaration of \( \text{event}_2 \) as the abstract signature (line 44), its subsignatures (lines 46-47), and its fact (line 45). The fact (line 42) underneath the signature for the intended event (line 41) is defined if there is some participant(s) in the SBVR model sends (receives) the same exclusive disjunction of \( \text{event}_2 \). This can be seen as the declarations define in the snippets lines 48-51. The snippet in line 48 defines the \( \text{participant}_1 \) sends (receives) (illustrating by the verb field) the XOR of \( \text{event}_2 \), which is why the fact in line 42 contains immediatelyprecedes equals to \( \text{participant}_1.\text{verb} \). On the other hand, the snippet in line 49-51 defines each subsignature of \( \text{participant}_2 \), namely \( \text{participant}_a \) or \( \text{participant}_b \) sends (receives) the XOR of \( \text{event}_2 \). As a result, immediatelyprecedes in the fact (line 43) equals to \( \text{participant}_a.\text{verb} \) or \( \text{participant}_b.\text{verb} \).

### 4.4.3 SBVR rules - static constraints

In this section, the static constraints refer to the date constraints involving a Date set in the SBVR model. In order to capture the specification of date constraints in the SBVR model, three main components are essential to be defined in Alloy, namely the declaration of a signature for Date and its subset signatures to encapsulate a Date set in the SBVR model; the declaration of the ordering on dates using the library module ordering utility; and finally the declaration of date constraints from the corresponding SBVR rules in the particular predicate in Alloy.

**A Date set transformation**

As discussed in Chapter 3 (Section 2.3.2), each distinct date associating with the particular event is defined in a Date set to demonstrate \( n \) number of dates involved in the messages exchanged of multi-party conversations. This can be seen as in the following Fact Type 2 to Fact Type 4.
Each date in the *Date* set is belonged to the particular event. For instance, the following Fact Type 1 illustrates the `date_1`, which is one of specified dates in the *Date* set, is belonged to the `event_1`. Note that `date_1` (it could be any date in the *Date* set) in the previous discussion refers to any static term to illustrate static constraint for the particular event. This fact type is translated into Alloy as declared in the following snippet (line 1).

The transformation of the *Date* set in the SBVR model into Alloy employs the declaration of subset signatures notion [46]. The *Date* set requires a signature that contains all the specified dates. To achieve this, `sig Date` (line 2) is introduced in Alloy. Each specified dates in the *Date* set is translated to be a subset signature (lines 3-5) of the `sig Date`. The purpose of using this subset signatures notion is due to its feature unlike the extension signatures introduced previously. This notion is not necessarily mutually disjoint, hence two subset signatures for the `sig Date` may intersect unless there is a constraint does not allowed to. For instance, since there are a declarations `one sig date_1 in Date` and `one sig date_2 in Date`, so both dates are allowed to be the same (intersect) date.

---

Fact types for specifying static constraints associating with the event in the SBVR model:

**Fact Type 1:** `event_1 verb date_1`

Fact types for specifying a *Date* set in the SBVR model:

**Fact Type 2:** `date_1 is in date`

**Fact Type 3:** `date_2 is in date`

...  

**Fact Type 4:** `date_n is in date`

---

--Specifying static constraints (date) associating with the event in the Alloy model:

1 `one sig event_1 extends event{verb: one date_1}`

...  

--Specifying a Date set in the Alloy model:

2 `sig Date[]`

3 `one sig date_1 in Date[]`

4 `one sig date_2 in Date[]`

...  

5 `one sig date_n in Date[]`
The ordering on dates in Alloy

The ordering on dates pertain to the specification of the Date set in the SBVR model to be an ordered set in Alloy. It is needed for checking the date constraints later. To accomplish this aim, the ordering utility of Alloy, namely the library module `util/ordering` is employed (line 6). This module performs the ordering on a set of any type. In this choreography model, since `[Date]` (line 6) (refer to the signature for Date) is defined, so when the module is opened the elements in a type Date will be considered for arranging them in order. This ordering on Date in Alloy is referred as `DateOrder`.

The next step is related to the declaration of `pred init` (line 7) in Alloy. This is closely connected with the specification of Fact Type 5 to define the initial condition. Commencing with the declaration of `d` as an initial date in Date’s set `[d:Date]` (line 7), the identification of the initial date in Fact Type 5 is required. Since `date1` is appeared as the initial date, the next step is aimed to find which event associates with the `date1`. With reference to Fact Type 1, `event1` is appeared as the event connects with `date1`. Thus, the signature for `event1` must be identified and it is then followed with finding its field `verb` which maps to `date1` (line 1). In order to set `date1` as the initial date in the Date set in Alloy, `d` must be equal to `event1.verb` (this join represents `date1`) as defined in line 7.

The fact traces in line 9 specifies the constraints that make the ordering a trace. This trace for Date arranges the elements of Date in order (lines 10-15), as `d`, `d1`, `d2`, and `d"`. It starts with considering the initial condition `init` (refers to the initial condition in `pred init`) for the first date in the trace (line 10 and line 12). Then, defining the last date as `d"` (line 11) in the trace, `d1` as the successor of `d`, and `d2` comes after `d1` (line 13). This ordering on dates `d`, `d1`, `d2`, and `d"` are related by the constraints of `orderdate[...](line 15)` operation which characterises the `pred orderdate[...](line 8)` operation. Line 14 represents all events involve in the operation specifies in line 15 (which comes from line 8). Likewise, `d`, `d1`, `d2`, and `d"` in line 15 inherits the declaration for these elements in the `pred orderdate[...]. These elements are defined as `disjoint` to ensure no overlapping of these elements.

---

Fact types for initialising dates in the SBVR model:
- **Fact Type 5**: `date1` is initial date
- **Fact Type 6**: `date_n` is final date

---

--Specifying the ordering of dates in the Alloy model:
--Declaration of the utility for ordering in the Alloy model:
6 open util/ordering [Date] as DateOrder
...
--Declaration of the initialisation of date in the Alloy model:
7 pred init[d:Date]{d = event1.verb}
--Declaration of predicate relating to date constraints in the Alloy model:
SBVR rules - date constraints

As discussed in Chapter 3 (Section 2.3.2), there are four types of date constraints: i). equality - describing the equality of two dates of the event(s) (it could be the same (different) events); ii). inequality - emphasising on the inequality of two dates of the event(s) (similar to the equality, events is allowed to be an identical (distinct) events); iii). in between - ensuring that one date of the particular event is between two given dates of the same (different) events; iv). begin before - expressing one date starts before (it might be the same date) another date of similar (dissimilar) events. The following rules are applied from the general forms of rules described in Section 2.3.2 for each constraint rule.

SBVR rules pertaining to date constraint (equality) in the SBVR model:

Rule 1: It is obligatory that exactly one date1 of exactly one event1 equals exactly one date2 of exactly one event2

SBVR rules pertaining to date constraint (inequality) in the SBVR model:

Rule 2: It is prohibited that exactly one date1 of exactly one event1 equals exactly one date2 of exactly one event2

SBVR rules pertaining to date constraint (in between) in the SBVR model:

Rule 3: It is obligatory that exactly one date1 of exactly one event1 is between exactly one date2 of exactly one event2 and exactly one date3 of exactly one event2

SBVR rules pertaining to date constraint (begin before) in the SBVR model:

Rule 4: It is obligatory that exactly one date1 of exactly one event1 begins before exactly one date2 of exactly one event2
The transformation of Rule 1 into Alloy is defined in the above code snippet in lines 16-18. Since both dates associate with different events, i.e. \(date_1\) is belonged to \(event_1\) while \(date_2\) be the property of \(event_2\), two signatures for each event is considered (line 16 and line 18). Next, the fact for the signature \(event_1\) is declared to constrain the equality of \(date_1\) and \(date_2\). By referring to \(date_1\) which is mapped by the field of \(sig\ event_1\ \ldots,\ verb\) is assigned to be equal with \(event_2.verb\ (date_2\ connecting\ with\ verb\ for\ event_2’s\ signature)\) as defined in line 17.

The transformation of Rule 2 (lines 19-21) follows the same method as applied for translating Rule 1 into Alloy. However, because of two specified dates are prohibited
to be equal, \( \text{verb} \neq \text{event}_2.\text{verb} \) (line 20) is defined as the fact for signature \text{event}_1.

Rule 3 which concerns on \textit{in between} constraint is translated into \texttt{pred orderdate} (lines 22-24). \texttt{date}_1 \texttt{of event}_1 is constrained to be occurred in between \texttt{date}_2 and \texttt{date}_3 in which both are the the dates of \texttt{event}_2. Starting with the declaration for each signature \texttt{event}_1 (line 25) and \texttt{event}_2 (line 26) together with their own potential field \texttt{verb} connects with their own dates, the operation of \texttt{pred orderdate} (lines 23-24) is subsequently defined. In this definition, both dates \texttt{date}_2 and \texttt{date}_3 are assumed to be declared as the initial date \texttt{d} and the final date \texttt{d}" in Date set in Alloy. Thus, \texttt{e2.verb}_1 = \texttt{d} and \texttt{e2.verb}_2 = \texttt{d}" are defined in line 23, where \texttt{e2} represents \texttt{event}_2 as initialised in the code snippet appears in line 22. To capture \textit{in between} constraint of date, the codes define in line 24 is essentially considered. These codes show \texttt{e1.verb} (which points to \texttt{date}_1) is allowed to be defined as the initial date \texttt{d} (\texttt{date}_2), or \texttt{d}_1 (\texttt{d}'s successor) or \texttt{d}_2 (the date after \texttt{d}_1), or the final date \texttt{d}" (\texttt{date}_2). Note that only the involved dates and events are initialised in line 22. In the case of Rule 3, all dates \texttt{d} to \texttt{d}" and \texttt{event}_1 and \texttt{event}_2 are involved.

There is slightly different in specifying the codes in Alloy relating to \textit{begin before} constraint in Rule 4. Since Rule 4 specifies both dates \texttt{date}_1 and \texttt{date}_2 associate with the same event \texttt{event}_1, so the signature for \texttt{event}_1 (line 35) declares two fields \texttt{verb}_1 and \texttt{verb}_2 which both map to \texttt{date}_1 and \texttt{date}_2, respectively. To capture the \textit{begin before} constraint, the potential codes relating to that constraints specify in the \texttt{pred orderdate} (lines 27-34). All dates \texttt{d} to \texttt{d}" and only \texttt{event}_1 are instantiated in line 27 as all of them are considered in the following codes (lines 28-34). The negation \texttt{not} constraint (line 28) is employed to guarantee \texttt{date}_1 is always before \texttt{date}_2. Hence, \texttt{date}_1 (refers to \texttt{e1.verb}1) is not allowed to be the final date when \texttt{date}_2 (refers to \texttt{e1.verb}2) is the initial date (line 29), or \texttt{date}_2 is \texttt{d}_1, \texttt{date}_2 is \texttt{d}_2. The same approach applies for the codes specify in lines 32-34. Nevertheless, both dates are possible to be the same date.

4.5 Generating the choreography in Alloy

In the previous sections, there were a fully description how the SBVR model was transformed into Alloy. As soon as a complete Alloy model representing the SBVR model was developed, the Alloy Analyzer can be used to automatically generate all possible executions of the required service interactions, in other words the service choreography.

As soon as Alloy Analyzer takes the global constraints of Alloy model which is transformed from the SBVR model, Alloy Analyzer will find structures that satisfy them. These structures that we call choreography is used to explore the model by generating an instance graphically and verifying them subsequently. An instance is a situation in which both the facts in the model and the predicates hold. However,
4.5. GENERATING THE CHOREOGRAPHY IN ALLOY

if Alloy does not find any possible execution that can conform this choreography; it returns no instance found.

In analysing and finding the solution of the choreography model particularly, or to be more precise, in finding an instance showing the global constraints, a predicate is employed in Alloy. We name that particular predicate as \texttt{pred initialevent} as shown in the following code snippet.

Note that while Alloy does generate all possible executions it does not output them all in one graph, instead it includes a ”Next” feature on its interface that allows the user to go through the possible executions one at a time. In this section, the description is emphasised on how to generate the \texttt{pred initialevent} for the given choreography model. The instance showing the possible choreography will be discussed further in Section 4.7 relating to generate the choreography for the conducting case study.

The development of \texttt{pred initialevent} for generating the choreography in Alloy depends on two important components. Firstly, it is strongly depends on the fact type in the SBVR model that initialises an initial event of all possible executions of the required service interactions. This can be seen in the following fact type.

\begin{verbatim}
Fact types for initialising an initial event:
Fact type: event is initial event
\end{verbatim}

The above fact type is the general form of defining an event initiating the choreography. \texttt{event} refers to any event \texttt{event\textsubscript{1}}, \texttt{event\textsubscript{2}}, ..., \texttt{event\textsubscript{N}} in the event set, while \texttt{initial event} is a fixed term that is used in that fact type.

Secondly, the development of \texttt{pred initialevent} depends on the SBVR rule which initiate the initial event. Concisely, that SBVR rule pertaining to immediate precedes notion rule to describe the ordering of messages exchanged for different events, e.g, the general form of rule is ”It is obligatory that ... \texttt{immediately precedes} ...”. These rules encompass a diverse structure of rules.

The development of the \texttt{pred initialevent} is described in the following details step by step and is referred to the following structure of rules and the corresponding codes in Alloy.

**Step 1:** Create a predicate \texttt{pred initialevent[...]{...}} in Alloy as shown in the the following code snippet (line 2).

**Step 2:** Identify an initial event in the fact type of the SBVR model which is specifically initialising the initial event such as \texttt{event\textsubscript{1} is initial event}. Then, identify the intended event. In this case, we know that \texttt{event\textsubscript{1}} is an initial event. Allocate this event in the \texttt{pred initialevent[e1:event\textsubscript{1}...]{...}} (line 1) and initiate it with any variable.
CHAPTER 4. GENERATION AND VERIFICATION CHOREOGRAPHY MODEL

Step 3: Identify a rule which specifies the first event term after the expression

\textit{It is obligatory that exactly one} .. as the initial event, e.g. \texttt{event}_1. Then, it is followed by identifying the specified event(s) that occur immediate after the initial event, that is after the verb \texttt{immediately precedes} and the logical operations for OR/XOR/AND on events involved (if any), in the identified rule. The following conditions give a clue the possible rules that initiate the initial event:

- No logical operations and exactly one event occurs after the initial event

The possible rule that follows this condition is Rule 1. The \texttt{pred initialevent} from lines 2-5 is produced if the initial event is specified as the first event term in the \texttt{Rule 1}. The development of \texttt{pred initialevent} begins with identifying the event that occurs immediately after \texttt{event}_1 as described in Rule 1. The event appears is the \texttt{event}_2.

Hence add the \texttt{event}_2 in the square bracket of the \texttt{pred initialevent} and initiate \texttt{event}_2 with any variable such as \texttt{e2} (line 2). Next, find the signature for \texttt{event}_1 and identify its field associating with \texttt{event}_2 (line 1). Since \texttt{event}_1 is initiated as \texttt{e1}, its field is \texttt{immediatelyprecedes}, and \texttt{event}_2 is initiated as \texttt{e2}, populate \texttt{e1.immediatelyprecedes} = \texttt{e2} (line 3) in the predicate \texttt{pred initialevent}. This describes the next event after the occurrence of \texttt{event}_1 (\texttt{e1.immediatelyprecedes}) is \texttt{event}_2.

- Inclusive disjunction (OR) on events occur after the initial event

Rule 2 and Rule 3 capture this kind of rule. It is found that \texttt{event}_2 and \texttt{event}_3 are specified after the verb \texttt{immediately precedes} in Rule 2. On the other hand, \texttt{event}_2 and \texttt{event}_4 which are included in \texttt{event}_2 are specified after the verb \texttt{immediately precedes} in Rule 3. For developing the \texttt{pred initialevent} corresponding to both rules, the identified events \texttt{event}_2 and \texttt{event}_3 (for Rule 2) are allocated in the square bracket of the \texttt{pred initialevent} (line 6). However, no events are required to be assigned in the \texttt{pred initialevent} in line 10 for capturing Rule 3. This is due to \texttt{event}_2 and its subsets are not be defined in that predicate. Then, find the signatures for the initial event \texttt{event}_1 as indicated in line 5 for Rule 2 and in line 9 for Rule 3. Identifying the fields for each signature which map to the respective events. It can be seen that \texttt{immediatelyprecedes1} maps to \texttt{event}_2 while \texttt{immediatelyprecedes2} maps to \texttt{event}_3 in line 5. Whereas \texttt{immediatelyprecedes} maps to \texttt{event}_2 only in line 9. In each \texttt{pred initialevent}, assign \texttt{e1.immediatelyprecedes1} = \texttt{e2} or \texttt{e1.immediatelyprecedes2} = \texttt{e3} (line 7) to encapsulate Rule 2 and \texttt{#e1.immediatelyprecedes} >= 1 to capture Rule 3.

- Conjunction (AND) on events occur after the initial event
The rules under this condition could be seen as described in Rule 4 and Rule 5. The codes in Alloy for generating choreography in which the initial event is specified in these rules are defined in lines 14-15 (Rule 4) and in lines 18-19 (Rule 5). The events that occur immediate after the \textit{event}\textsubscript{1} are expressed in Rule 4 as \textit{event}\textsubscript{2} and \textit{event}\textsubscript{3}, while in Rule 5 as \textit{event}\textsubscript{a} and \textit{event}\textsubscript{b} (the nested events of \textit{event}\textsubscript{2}). Then, each identified event is allocated in each \texttt{pred initialevent}. \textit{event}\textsubscript{2} and \textit{event}\textsubscript{3} are declared in line 14. By contrast, \textit{event}\textsubscript{a} and \textit{event}\textsubscript{b} are defined in line 18. Next, by looking at the signatures for \textit{event}\textsubscript{1} (line 13 and line 17) for each rule, identify the fields which connect to the respective events. The fields \texttt{immediatelyprecedes1} and \texttt{immediatelyprecedes2} are appeared in both signatures where each of them maps to the respective event. Populate \texttt{e1.immediatelyprecedes1 = e2} and \texttt{e1.immediatelyprecedes2 = e3} for both \texttt{pred initialevent} in line 15 and line 19. However, these codes represent different meaning since \texttt{e2} in line 15 denotes \textit{event}\textsubscript{2} while \texttt{e2} in line 19 designates \textit{event}\textsubscript{a}. Similarly, \texttt{e3 in line 15} means \textit{event}\textsubscript{3} whereas \texttt{e3 in line 19} defines \textit{event}\textsubscript{b}. 

- **Exclusive disjunction (XOR) on events occur after the initial event**

Rule 6 describes this condition. The remaining codes as shown in the following codes snippet (lines 21-24) represent the required codes for generating the choreography as if the initial event is specified as in Rule 6. Since \texttt{event}\textsubscript{a} and \texttt{event}\textsubscript{b} (a nesting of \texttt{event}\textsubscript{2}) are expressed as the potential events which occur immediate after the initial event \textit{event}\textsubscript{1}, both events together their initiation are defined for the \texttt{pred initialevent} (line 22). Line 21 shows the declaration of \textit{event}\textsubscript{1}’s signature. Identifying its field which connects to the \textit{event}\textsubscript{2}. Then, assigning (\texttt{e1.immediatelyprecedes = e2} and \texttt{e1.immediatelyprecedes != e3}) or (\texttt{e1.immediatelyprecedes = e3} and \texttt{e1.immediatelyprecedes != e2}) in the \texttt{pred initialevent} (line 23). Note that \texttt{e1}, \texttt{e2}, and \texttt{e3} refer to \textit{event}\textsubscript{1}, \textit{event}\textsubscript{a}, and \textit{event}\textsubscript{b} respectively, while \texttt{immediatelyprecedes} is the field of the signature for \textit{event}\textsubscript{1}. These codes are essential to capture Rule 6 for ensuring only one of the given events will be selected as the first execution in the choreography.

**Step 3: Assign code for generating the choreography** in each \texttt{pred initialevent} as defined in the following snippets. The declaration of \texttt{run initialevent} is required to execute the Alloy model and generate an instance showing the given choreography.

The possible SBVR rules in the SBVR model:

**Rule 1:** It is obligatory that exactly one \texttt{event1 immediately precedes} exactly one
**Event 2**

**Rule 2:** It is obligatory that exactly one $event_1$ immediately precedes exactly one $event_2$ or exactly one $event_3$

**Rule 3:** It is obligatory that exactly one $event_1$ immediately precedes exactly one $event_2$ that includes exactly one $event_a$ or exactly one $event_b$

**Rule 4:** It is obligatory that exactly one $event_1$ immediately precedes exactly one $event_2$ and exactly one $event_3$

**Rule 5:** It is obligatory that exactly one $event_1$ immediately precedes exactly one $event_2$ that includes exactly one $event_a$ and exactly one $event_b$

**Rule 6:** It is obligatory that exactly one $event_1$ immediately precedes exactly one $event_2$ that includes exactly one $event_a$ or exactly one $event_b$ but not both

The predicates which correspond to the above SBVR rules in the Alloy model:

```
--pred initialevent for Rule 1:
1 one sig event_1 extends event{immediatelyprecedes: one event_2}
...
2 pred initialevent[e1:event_1, e2:event_2]{
3 e1.immediatelyprecedes = e2 }
4 run initialevent
--pred initialevent for Rule 2:
5 one sig event_1 extends event{immediatelyprecedes1: lone event_2, immediatelyprecedes2: lone event_2}
{...}
...
6 pred initialevent[e1:event_1, e2:event_2, e3:event_3]{
7 e1.immediatelyprecedes1 = e2 or e1.immediatelyprecedes2 = e3 }
8 run initialevent
--pred initialevent for Rule 3:
9 one sig event_1 extends event{immediatelyprecedes: some event_2}
{...}
...
10 pred initialevent[e1:event_1]{
11 #e1.immediatelyprecedes >= 1}
12 run initialevent
--pred initialevent for Rule 4:
13 one sig event_1 extends event{immediatelyprecedes1: one event_2, immediatelyprecedes2: one event_3}
{...}
...
14 pred initialevent[e1:event_1, e2:event_2, e3: event_3]{
15 e1.immediatelyprecedes1 = e2 and e1.immediatelyprecedes2 = e3}
```
4.6. VERIFYING THE CHOREOGRAPHY IN ALLOY

The Alloy Analyzer is also be used to perform the verification on the generated choreography pertaining to realisability as well as the static constraints including date constraints.

4.6.1 Verifying - realisability

The Alloy Analyzer can be used to perform verification on the generated choreography. In particular, it can readily verify whether a specific request can be realised by the given service choreography. This specific request refers to a request on the particular participant(s) executing the messages of the specific event(s). This is to guarantee each participant service should be able to execute its part of the choreography as freely as possible, preserving its local autonomy and replaceability of services. We refer this purpose as realisability.

The specific request in Alloy refers to the produced specific request rule in the SBVR model. As described in Chapter 3 (Section 3.1.2), the construction of the specific request rule as specified in the listed general forms of rules in Section 3.1.2, is based on kind of constraints which are represented in the SBVR rule that was constructed in the the developed SBVR model (the generated choreography). Such constraints are emphasised on a single participant or multiple participants involved in executing the messages exchanged of a single event or multiple events. These
rules were grouped into two types (detailed see Section 3.1.2): Type 1 refers to all rules describing a single (exactly one) participant performs the interaction; Type 2 refers to all rules describing multiple participants (OR/XOR/AND on participants) perform the interaction. The interaction is the messages exchanged of a single event or multiple events.

This specific request can be formulated as a predicate in Alloy. This is shown in the following code snippet. We name this predicate as `pred concreterequest`.

When this predicate is executed, the request is checked against the generated choreography. If there is at least one instance (one possible execution) of the choreography which meets the request, then we say the request can be realised. Otherwise, if Alloy does not find any possible execution that can realise this request, it returns no instance found. This is reflected by the Alloy realisability check.

This section focuses on how to develop the potential specific request in Alloy and at the same time to verify it for realisability purpose. The development of the `pred concreterequest` in Alloy is described in the following details step by step. Each `pred concreterequest` is based on the SBVR rule (Type I and Type II) (as the basis for the construction of each specific request rule) and the respective general form of specific request rule in Section 3.1.2.

**Step 1**: Create a predicate `pred concreterequest[...]{...}` in Alloy as shown in the following codes snippet.

**Step 2**: Identify a specific request rule in the SBVR model where this rule is constructed based on the rules developed in the generated choreography model. It is grouped in two types (Type I and Type II).

- **Type 1**: The rules describing participant is a single participant and event is a single or multiple events.

  Three possible rules under Type 1 together their specific request are constructed in Alloy as described as in the following. Note that the specific request rules that are constructed based on the given SBVR rules in Type 1 could be referred to Section 3.1.2.

  1. **Rule 1**: It is obligatory that the `participant_1` `verb` exactly one `event_1` or exactly one `event_2` or ... or exactly one `event_N`.

     ```alloy
     Rule 1 (type 1 in Section 3.1.2) captures the global constraint whenever one participant is required to send (receive) an inclusive disjunction of the given event set in the multi-party conservations. This rule is translated in Alloy as in the code snippet line 1. Each field of the signature for `participant_1`, namely `verb1` for `event_1`, `verb2` for `event_2`, up to `verb3` for `event_N`, is defined to associate with each specified event. There are four possible specific requests that could be constructed from Rule 1.
     ```
Lines 2-3 define the specific request in which the `participant_1` performs only one event, that is the `event_1` from the inclusive disjunction of the given event set. Thus, `participant_1` and `event_1` designate as `p` and `e1`, respectively, are declared in the snippet line 2. Next, `p.verb1 = e1` must be defined in line 3 to ensure the specific request pertains to the `participant_1` performs the messages exchanged of the `event_1` only. The additional statement, `no p.verb2` until `no p.verb3` together the logical operator `and` in between them are specified to guarantee the other events are not considered in the given instance in Alloy.

Lines 5-8 reflect the specific request rule when exactly one event of the given event set is selected to be executed by the `participant_1`. In this, each event is designated for a declaration in line 5, where `e_1`, `e_2`, and `e_3` denote `event_1`, `event_2`, and `event_N`, respectively. In order to encapsulate the given specific request, the declaration in line 6 is specified to guarantee only `event_1`, or line 7 to ensure only `event_2` is performed, or line 8 to show the other else selected event, is performed at a time by the `participant_1`.

The declaration in lines 10-11 is employed to verify whether the specific request on each unordered given event that is executed by the `participant_1` is realised in the given choreography, particularly in the given Rule 1. Hence, each event that is declared in line 10 and each statement showing `participant_1` sends (receives) each declared event, is defined in line 11 associates with the logical operator `and` over them.

The last specific request that is possible to be verified for the respective Rule 1 is defined in lines 13-15. This specific request refers to the exclusive disjunction of request, whenever only selected events are required in the verification. In this case, the involved events are declared in line 13. Then, two statements are specified connecting with the logical operator `or` (represents XOR) in between them (lines 14-15). Each statement defines which the execution of events is needed in the specific request. Line 14 means the execution of the `event_1` and (or) `event_2` instead of `event_N` is needed. On the other hand, line 15 means the only required execution is on the `event_N`.

The predicates of specific request which correspond to each specific request rule for the respective SBVR rules (refer to Section 3.1.2):

---Type 1:

---A transformation of Rule 1:
1 one sig participant_1 extends participant{verb1: lone event_1, verb2: lone event_2, ..., verb3: lone event_N} {
2 ...} 
3 --pred concreterequest for Rule 1: 
4 --A request when involving no logical operation over event terms. 
5 pred concreterequest[p:participant_1, e1: event_1] { 
6 p.verb1 = e1 and no p.verb2 and ... and no p.verb3 } 
7 run concreterequest 
8 --A request when involving the logical operation XOR (or..but not all) logical operation over event terms. 
9 pred concreterequest[p:participant_1, e1: event_1, e2: event_2, ..., e3: event_N] { 
10 (p.verb1 = e1 and no p.verb2 and ... and no p.verb3) 
11 or (p.verb2 = e2 and no p.verb1 and ... and no p.verb3) 
12 ... or (p.verb3 = e3 and no p.verb1 and ... and no p.verb2) 
13 run concreterequest 
14 --A request when involving the logical operation AND over event terms. 
15 pred concreterequest[p:participant_1, e1: event_1, e2: event_2, ..., e3: event_N] { 
16 p.verb1 = e1 and p.verb2 = e2 and ... and p.verb3 = e3} 
17 run concreterequest 
18 --A request when involving the exclusive disjunction of the combination of the logical operations AND and OR (if any) over event terms. 
19 pred concreterequest[p:participant_1, e1: event_1, e2: event_2, ..., e3: event_N] { 
20 (p.verb1 = e1 and (or) p.verb2 = e2 and ... and no p.verb3) 
21 or (... and (or) p.verb3 = e3 and no p.verb1 and p.verb2 and ... ) } 
22 run concreterequest 

2. Rule 2: It is obligatory that the participant verb exactly one event that includes exactly one event or exactly one event or ... or exactly one event but not all.

Rule 2 (type 1 in Section 3.1.2) represents the rule specifying one participant performs the messages exchanged of the exclusive disjunction of the event. In Alloy, this rule is transformed as in the code snippet line 17. The signature for participant has a field verb which is mapped to event that includes event up to event. Only one possible specific request that is possible to be constructed from Rule 2.
4.6. VERIFYING THE CHOREOGRAPHY IN ALLOY

The `pred concreterequest` in lines 18-19 defines the `participant_1` performs exactly one event from the given choices (XOR) of `event_1`. Since `event_a` is only be selected, it is only is declared in line 18. Next, `p.verb = a` is specified in line 19 for ensuring only `event_a` is chosen to be verified.

The predicates of specific request which correspond to each specific request rule for the respective SBVR rules (refer to Section 3.1.2):

```
--Type 1:
--A transformation of Rule 2:
17 one sig participant_1 extends participant{verb: one event_1}
{...}
--A request when involving n exclusive disjunction of events.
18 pred concreterequest[p:participant_1, e:event_a] {
19 p.verb = e
}
20 run concreterequest
```

3. Rule 3: It is obligatory that the `participant_1` `verb` exactly one `event_1` that includes exactly one `event_a` or exactly one `event_b` or ...
or exactly one `event_n`.

Rule 3 (type 1 in Section 3.1.2) shows at least one event inside the `event_1` is selected by one participant in the given choreography. This rule is represented in Alloy as in the code snippet line 21. The `verb` field associates to `event_1` is defined in the signature for `participant_1`.

The possible specific requests that could be constructed from this rule are either i). exactly one event is chosen inside the `event_1` set such as `event_a` is specified in lines 22-23, which is why only `event_a` is defined in line 22 as well as only the statement `p.verb = a` is specified in line 23; ii). either one of the event inside the `event_1` set but not all of them is chosen. This is declared in lines 25-28.

Each possible event, which is specified as the subsets of the `event_1`, is designated in line 25. Then, line 26 specifies only `event_a` is selected, similarly, in line 27 and line 28 declare only `event_b` or `event_n` is chosen, respectively. This declaration is defined to highlight only one of them is chosen at a time; iii). each event inside the `event_1` set is chosen as the specific request for verification. Lines
30-31 reflect this specific request. Thus each subset of the event_1
set is declared in line 30 as well as each statement characterises the
participant must be executed the messages exchanged of each event
concurrently is defined in line 31. Note that the logical operator and
is employed for constraining this specific request; iv). two separate
(XOR) requests are expressed showing the participant sends (re-
ceives) the possible combination of events. All the involved events
for this specific request are declared in line 33. Also, two state-
ments correspond to two specific requests are specified in lines 34-35.

The predicates of specific request which correspond to each specific
request rule for the respective SBVR rules (refer to Section 3.1.2):

```
--Type 1:
--A transformation of Rule 3:
21 one sig participant_1 extends participant{verb: some event_1}
{...}
--A request when involving no logical operation over event terms.
22 pred concreterequest[p:participant_1, a:event_a] {
23 p.verb = a}
24 run concreterequest
--A request when involving the logical operation XOR (or .. but not all)
logical operation over event terms.
25 pred concreterequest[p:participant_1, a:event_a, b:event_b, .....,
n:event_n] {
26 (p.verb = a and p.verb != b and ... and p.verb != n)
27 or (p.verb = b and p.verb != a and ... and p.verb != n)
28 or ... or (p.verb = n and p.verb != a and ... and p.verb != b)}
29 run concreterequest
--A request when involving the logical operation AND over event terms.
30 pred concreterequest[p:participant_1, a:event_a, b:event_b, .....,
n:event_n] {
31 (p.verb = a and p.verb = b and ... and p.verb = n)}
32 run concreterequest
--A request when involving the exclusive disjunction of the combination of
the logical operations AND and OR (if any) over event terms.
33 pred concreterequest[p:participant_1, a:event_a, b:event_b, .....,
n:event_n] {
34 (p.verb1 = a and (or) p.verb2 = b and ... and p.verb != n)
35 or (.... and (or) p.verb = n and p.verb != a and p.verb != b and ...) }
36 run concreterequest
```
• Type 2: The rules describing participant is multiple participants and event is a single or multiple events.

There are several possible rules in the SBVR model that are grouped in this type associate with the corresponding specific request rules. Each possible specific request rule is declared in Alloy as in the given code snippet. Note that the corresponding specific request rule could be referred to Section 3.1.2.

1. Rule 1: It is obligatory that the \texttt{participant}_1 or the \texttt{participant}_2 or ... or the \texttt{participant}_N, verb exactly one \texttt{event}_1.

Rule 2: It is obligatory that the \texttt{participant}_1 that includes the \texttt{participant}_a or the \texttt{participant}_b or ... or the \texttt{participant}_n, verb exactly one \texttt{event}_1.

Rule 1 and Rule 2 (type 2 in Section 3.1.2) have the identical purpose in describing the interaction. Both of them specify the inclusive disjunction of participants execute exactly one event in the multi-party conversation. However, that particular event is either executed by the inclusive disjunction of the participant set (e.g. \texttt{participant}_1, \texttt{participant}_2, ..., \texttt{participant}_N) as described in Rule 1 or by the inclusive disjunction of the \texttt{participant}_1 set (the subsets of of the \texttt{participant}_1. e.g. \texttt{participant}_a, \texttt{participant}_b, ..., \texttt{participant}_n) as specified in Rule 2. Therefore, the statements define these rules in Alloy are slightly different as indicated in the following snippets lines 37-39 for Rule 1, while lines 56-58 for Rule 2.

The first potential specific request that could be constructed and verified from both rules concerns on exactly one participant from the inclusive disjunction of the given participant set, is obligated for sending (receiving) the specified event. The \texttt{event}_1 is aimed to be performed, so it is designated as \texttt{e1} in line 40 and line 59. Similarly, since the \texttt{participant}_1 (\texttt{participant}_a), \texttt{participant}_2 (\texttt{participant}_b), up to \texttt{participant}_N (\texttt{participant}_n) are the inclusive disjunction of participants involved in the interaction, as specified in Rule 1 (Rule 2), each of them is declared with their own initialisation. In order to capture the request, only the specified participant \texttt{participant}_1 (line 41) and \texttt{participant}_a (line 60) associate with its defined field must be equal to \texttt{e1}. The other participants are not considered, which is why each of them is defined as the negation.
The second potential specific request relates to the exclusive disjunction of participants inside the participant set (Rule 1) and inside the participant set (Rule 2), are chosen to perform the interaction of sending (receiving) the event. The similar approach as applied for the first potential request in defining the participants and the event, is specified in line 43 and line 62. Each statement is defined to ensure exactly one participant is chosen for each interaction. This is illustrated as a declaration in lines 44-47 and lines 62-66.

Each specified participant in both rules, which is targeted to send (receive) the event, is illustrated as the third potential specific request for verification. Note that, the declaration of the participants and the event are definitely the same as the previous declarations (line 49 and line 68). In order to encapsulate the specific request, each participant connecting with its own field which is equal to the event is defined in Alloy (lines 49-50 and lines 68-69).

Finally, the potential specific request relating to both rules Rule 1 and Rule 2, is concerned on the separate (XOR) request describing either one of them is targeted to be true. In each request, the participants could be the combination of participants (involving OR (AND) on participants) execute the interaction. For this case, only the involved participant(s) is required to be defined (line 52 and line 71). Subsequently, each statement capturing the request is declared in lines 53-54 and lines 72-73.

The predicates of specific request which correspond to each specific request rule for the respective SBVR rules (refer to Section 3.1.2):

```
--Type 2:
--A transformation of Rule 1:
37 lone sig participant_1 extends participant{verb: one event_1}  
38 lone sig participant_2 extends participant{verb: one event_1}  
...  
39 lone sig participant_N extends participant{verb: one event_1}  

--A request when involving no logical operation over participant terms.
40 pred concreterequest[p1:participant_1, p2:participant_2, ..., p3:participant_N, e1:event_1] {  
41 p1.verb = e1 and no p2.verb ... and no p3.verb}  
42 run concreterequest

--A request when involving the logical operation XOR (or..but not all) logical operation over participant terms.
43 pred concreterequest[p1:participant_1, p2:participant_2, ..., p3:participant_N, e1:event_1] {
```
4.6. VERIFYING THE CHOREOGRAPHY IN ALLOY

\[
\begin{align*}
44 \ (p.1.verb = e1 \text{ and no } p.2.verb \ldots \text{ and no } p.3.verb) \\
45 \ \text{or } (p.2.verb = e1 \text{ and no } p.1.verb \ldots \text{ and no } p.3.verb) \\
46 \ \ldots \\
47 \ \text{or } (p.3.verb = e1 \text{ and no } p.1.verb \ldots \text{ and no } p.2.verb) \\
48 \ \text{run concreterequest} \\
\quad \text{--A request when involving the logical operation AND over participant terms.} \\
49 \ \text{pred concreterequest}[p1:participant_1, p2:participant_2, \ldots, p3:participant_N, e1:event_1] \{ \\
50 \ \ p1.verb = e1 \text{ and } p2.verb = e1 \ldots \text{ and } p3.verb = e1 \\
51 \ \text{run concreterequest} \\
\quad \text{--A request when involving the exclusive disjunction of the combination of} \\
\quad \text{the logical operations AND and OR (if any) over participant terms.} \\
52 \ \text{pred concreterequest}[p1:participant_1, p2:participant_2, \ldots, p3:participant_N, e1:event_1] \{ \\
53 \ \ (p1.verb = e1 \text{ and (or) } p2.verb = e1 \text{ and } \ldots \text{ and no } p3.verb) \\
54 \ \text{or } (\ldots \text{ and (or) } p3.verb = e1 \text{ and no } p1.verb \text{ and no } p2.verb \text{ and } \ldots)) \\
55 \ \text{run concreterequest} \\
\quad \text{--A transformation of Rule 2:} \\
56 \ \text{abstract sig participant_1 extends participant{verb: one event_1}} \\
57 \ \text{lone sig participant_a extends participant_1{}} \\
\quad \ldots \\
58 \ \text{lone sig participant_n extends participant_1{}} \\
\quad \text{--A request when involving no logical operation over participant terms.} \\
59 \ \text{pred concreterequest}[p_a:participant_a, p_b:participant_b, \ldots, p_n:participant_n, e1:event_1] \{ \\
60 \ \ p_a.verb = e1 \text{ and no } p_b.verb \ldots \text{ and no } p_n.verb \\
61 \ \text{run concreterequest} \\
\quad \text{--A request when involving the logical operation XOR (or..but not all) } \\
\quad \text{logical operation over participant terms.} \\
62 \ \text{pred concreterequest}[p_a:participant_a, p_b:participant_b, \ldots, p_n:participant_n, e1:event_1] \{ \\
63 \ \ (p_a.verb = e1 \text{ and no } p_b.verb \ldots \text{ and no } p_n.verb) \\
64 \ \text{or } (p_b.verb = e1 \text{ and no } p_a.verb \ldots \text{ and no } p_n.verb) \\
65 \ \ldots \\
66 \ \text{or } (p_n.verb = e1 \text{ and no } p_a.verb \ldots \text{ and no } p_b.verb) \\
67 \ \text{run concreterequest} \\
\quad \text{--A request when involving the logical operation AND over participant terms.} \\
68 \ \text{pred concreterequest}[p_a:participant_a, p_b:participant_b, \ldots, p_n:participant_n, e1:event_1] \{ \\
69 \ \ p_a.verb = e1 \text{ and } p_b.verb = e1 \ldots \text{ and } p_n.verb = e1 \\
70 \ \text{run concreterequest} \\
\quad \text{--A request when involving the exclusive disjunction of the combination of} \\
\quad \text{the logical operations AND and OR (if any) over participant terms.}
\end{align*}
\]
2. **Rule 3**: It is obligatory that the `participant_1` or the `participant_2` or ... or the `participant_N`, `verb` exactly one `event_1` that includes exactly one `event_a` or exactly one `event_b` or ... or exactly one `event_n` but not all.

**Rule 4**: It is obligatory that the `participant_1` that includes the `participant_a` or the `participant_b` or ... or the `participant_n`, `verb` exactly one `event_1` that includes exactly one `event_a` or exactly one `event_b` or ... or exactly one `event_n` but not all.

**Rule 3** and **Rule 4** (type 2 in Section 3.1.2) are intentionally to describe the inclusive disjunction of participants execute the exclusive disjunction of the subsets for the `event_1`, i.e. `event_a` or `event_b`, up to `event_n`, but not all of them, in the multi-party conversation. The following codes snippet in lines 75-78 illustrate the specification of **Rule 3**, whereas lines 107-110 represent the specification of **Rule 4** in Alloy. Each respective signature defines its own field to be mapped to the abstract signature for the `event_1`.

There are two possible specific requests that could be verified from the developed **Rule 3** and **Rule 4**. The first potential specific request refers to the particular participant that is aimed to send (receive) the exclusive disjunction of the `event_1` set. This is specified in lines 79-80 (**Rule 3**) and lines 111-113 (**Rule 4**). As previously, each participant and the `event_1` is declared in each predicate. It is then followed by the declaration of the statements in Alloy for capturing the particular participant involves in the interaction. The statement `(p1.verb = e1 and no p2.verb ... and no p3.verb)` in line 80 represents only `participant_1` is chosen to execute the interaction. Similarly, `p_a.verb = e1 and no p_b.verb ... and no p_n.verb` in line 112 means only `participant_a` is selected to perform the interaction instead of the other participants. On the other hand, the second potential specific request refers to the particular participant that is targeted to send (receive) exactly one event from the given choices (XOR). In this case, each event is required to be
defined in the predicate for constraining the unwanted event as well (line 82 and line 114). Thus, the statements (p1.verb = a and ... and p1.verb != n no p2.verb ... and p_a !n no p_b.verb ... and no p_n.verb) in line 83 as well as (p_a.verb = a and ... and p_a != n no p_b.verb ... and no p_n.verb) in line 115 is provided to verify against the given rules, which is only event_a is considered to be performed by the aforementioned participant.

Also, there are two possible specific requests pertain to the exclusive disjunction of participants inside the participant set (Rule 3) and inside the participant_1 set (Rule 4) is targeted to perform the interaction. The interaction refers to the sending (receiving) the exclusive disjunction on the event set or exactly one event_a of the given subsets of the event_1. The declaration for each predicate is applied as the same method as the previous specific request (line 85 and line 117). Lines 86-88 and lines 118-120 describe either one (XOR) of the specified participants from the given participant set is chosen to execute the exclusive disjunction on events. By contrast, lines 91-93 and lines 123-125 express either one (XOR) of participants inside the respective participant set is aimed to execute exactly one event_a from the given event_1 set.

The next possible specific requests refer to each specified participant is targeted to send (receive): i). the exclusive disjunction of the event_1 that consists of event_a, ..., event_n (as specified in line 96 and line 128); and ii). exactly one event such as event_a which is a subset of the event_1 (as specified in lines 99-101 and lines 131-133).

Then, the potential specific requests relating to the distinct (XOR) requests encompass the combination of the given participants execute the exclusive disjunction of the event_1 (lines 104-105 and lines 136-137) or exactly one event, that is event_a inside the exclusive disjunction of the event_1 (lines 108-109 and lines 140-145).

The predicates of specific request which correspond to each specific request rule for the respective SBVR rules (refer to Section 3.1.2):

```
--Type 2:
--A transformation of Rule 3:
75 lone sig participant_1 extends participant{verb: one event_1}
76 lone sig participant_2 extends participant{verb: one event_1}
...
77 lone sig participant_N extends participant{verb: one event_1}
```
abstract sig event_1 extends event{}
{
  ...
}
--A request when involving no logical operation over participant terms.
--A transformation of Rule i:
79 pred concreterequest[p1:participant_1, p2:participant_2, ..., p3:participant_N, e1:event_1] {
  80 (p1.verb = e1 and no p2.verb ... and no p3.verb)}
81 run concreterequest
--A transformation of Rule ii:
82 pred concreterequest[p1:participant_1, p2:participant_2, ..., p3:participant_N, a:event_a, ..., n:event_n] {
  83 (p1.verb = a and ... and p1.verb != n no p2.verb ... and no p3.verb)}
84 run concreterequest
--A request when involving the logical operation XOR (or...but not all)
logical operation over participant terms.
--A transformation of Rule i:
85 pred concreterequest[p1:participant_1, p2:participant_2, ..., p3:participant_N, e1:event_1] {
  86 (p1.verb = e1 and no p2.verb ... and no p3.verb)
  87 or (p2.verb = e1 and no p1.verb ... and no p3.verb)
  ...
  88 or (p3.verb = e1 and no p1.verb ... and no p2.verb)}
89 run concreterequest
--A transformation of Rule ii:
90 pred concreterequest[p1:participant_1, p2:participant_2, ..., p3:participant_N, a:event_a, ..., n:event_n] {
  91 (p1.verb = a and ... and p1.verb != n no p2.verb ... and no p3.verb)
  92 or (p2.verb = a and ... and p2.verb != n no p1.verb ... and no p3.verb)
  ...
  93 or (p3.verb = a and ... and p3.verb != n no p1.verb ... and no p2.verb)}
94 run concreterequest
--A request when involving the logical operation AND over participant terms.
--A transformation of Rule i:
95 pred concreterequest[p1:participant_1, p2:participant_2, ..., p3:participant_N, e1:event_1] {
  96 (p1.verb = e1 and p2.verb = e1 ... and p3.verb = e1)}
97 run concreterequest
--A transformation of Rule ii:
98 pred concreterequest[p1:participant_1, p2:participant_2, ..., p3:participant_N, a:event_a, ..., n:event_n] {
  99 (p1.verb = a and ... and p1.verb != n)
  100 and (p2.verb = a and ... and p2.verb != n)
4.6. VERIFYING THE CHOREOGRAPHY IN ALLOY

...  
101 and (p3.verb = a and ... and p3.verb != n)  
102 run concreterequest  

"A request when involving the exclusive disjunction of the combination of the logical operations AND and OR (if any) over participant terms.

103 pred concreterequest[p1:participant_1, p2:participant_2, ..., p3:participant_N, e1:event_1] {  
104 (p1.verb = e1 and (or) p2.verb = e1 ... and no p3.verb)  
105 or (... and (or) p3.verb = e1 and no p1.verb ... and no p2.verb)  
106 run concreterequest  

------------------------------------------  

"A transformation of Rule 4:

107 abstract sig participant_1 extends participant{verb: one event_1}  
108 lone sig participant_a extends participant_1{}  
...  
109 lone sig participant_n extends participant_1{}  
...  
110 abstract sig event_1 extends event{}  
{...}  

"A request when involving no logical operation over participant terms.

"A transformation of Rule i:

111 pred concreterequest[p_a:participant_a, p_b:participant_b, ..., p_n:participant_n, e1:event_1] {  
112 (p_a.verb = e1 and no p_b.verb ... and no p_n.verb)  
113 run concreterequest  

------------------------------------------  

"A transformation of Rule ii:

114 pred concreterequest[p_a:participant_a, p_b:participant_b, ..., p_n:participant_n, a:event_a, ..., n:event_n] {  
115 (p_a.verb = a and ... and p_a != n no p_b.verb ... and no p_n.verb)  
116 run concreterequest  

"A request when involving the logical operation XOR (or but not all)
logical operation over participant terms.

"A transformation of Rule i:

117 pred concreterequest[p_a:participant_a, p_b:participant_b, ..., p_n:participant_n, e1:event_1] {  
118 (p_a.verb = e1 and no p_b.verb ... and no p_n.verb)  
119 or (p_b.verb = e1 and no p_a.verb ... and no p_n.verb)  
...  
120 or (p_n.verb = e1 and no p_b.verb ... and no p_n.verb)  
121 run concreterequest  

"A transformation of Rule ii:

122 pred concreterequest[p_a:participant_a, p_b:participant_b, ..., p_n:participant_n, a:event_a, ..., n:event_n] {
CHAPTER 4. GENERATION AND VERIFICATION CHOREOGRAPHY MODEL

123 \( (p_a.\text{verb} = a \text{ and } \ldots \text{ and } p_a.\text{verb} \neq n \text{ and } p_b.\text{verb} \ldots \text{ and no } p_n.\text{verb} ) \)
124 or \( (p_b.\text{verb} = a \text{ and } \ldots \text{ and } p_b.\text{verb} \neq n \text{ and } p_a.\text{verb} \ldots \text{ and no } p_n.\text{verb} ) \)
125 or \( (p_n.\text{verb} = a \text{ and } \ldots \text{ and } p_n.\text{verb} \neq n \text{ and } p_a.\text{verb} \ldots \text{ and no } p_b.\text{verb} ) \)
126 run concreterequest

--A request when involving the logical operation AND over participant terms.

--A transformation of Rule i:
127 pred concreterequest\[p_a:\text{participant}_a, p_b:\text{participant}_b, \ldots, p_n:\text{participant}_n, e_1:\text{event}_1]\{  
128 \( (p_a.\text{verb} = e_1 \text{ and } p_b.\text{verb} = e_1 \text{ and } \ldots \text{ and } p_n.\text{verb} = e_1 ) \}
129 run concreterequest

--A transformation of Rule ii:
130 pred concreterequest\[p_a:\text{participant}_a, p_b:\text{participant}_b, \ldots, p_n:\text{participant}_n, a:\text{event}_a, \ldots, n:\text{event}_n]\{  
131 \( (p_a.\text{verb} = a \text{ and } \ldots \text{ and } p_a.\text{verb} \neq n ) \)
132 and \( (p_b.\text{verb} = a \text{ and } \ldots \text{ and } p_b.\text{verb} \neq n ) \)
133 and (\ldots and \( p_n.\text{verb} = a \text{ and } \ldots \text{ and } p_n.\text{verb} \neq n \)\})
134 run concreterequest

--A request when involving the exclusive disjunction of the combination of the logical operations AND and OR (if any) over participant terms.

--A transformation of Rule i:
135 pred concreterequest\[p_a:\text{participant}_a, p_b:\text{participant}_b, \ldots, p_n:\text{participant}_n, e_1:\text{event}_1]\{  
136 \( (p_a.\text{verb} = e_1 \text{ and } (\text{or}\, p_b.\text{verb} = e_1 \text{ and } \ldots \text{ and no } p_n.\text{verb} ) \)
137 or (\ldots and (\text{or}\, p_n.\text{verb} = e_1 \text{ and no } p_a.\text{verb} \text{ and } \ldots \text{ and no } p_b.\text{verb} )\})
138 run concreterequest

--A transformation of Rule ii:
139 pred concreterequest\[p_a:\text{participant}_a, p_b:\text{participant}_b, \ldots, p_n:\text{participant}_n, a:\text{event}_a, \ldots, n:\text{event}_n]\{  
140 \( ((p_a.\text{verb} = a \text{ and } \ldots \text{ and } p_a.\text{verb} \neq n ) \)
141 and (\text{or}\, p_b.\text{verb} = a \text{ and } \ldots \text{ and } p_b.\text{verb} \neq n )
142 and (\ldots and no p_n.\text{verb} )
143 or (\ldots and (\text{or}\, p_n.\text{verb} = a \text{ and } \ldots \text{ and } p_n.\text{verb} \neq n )
144 and no p_a.\text{verb} \text{ and } no p_b.\text{verb} )\}
145 run concreterequest

---
4.6. VERIFYING THE CHOREOGRAPHY IN ALLOY

3. Rule 5: It is obligatory that the \textit{participant}_1 and the \textit{participant}_2 and ... and the \textit{participant}_N, \textit{verb} exactly one \textit{event}_1 that \textit{includes} exactly one \textit{event}_a or exactly one \textit{event}_b or ... or exactly one \textit{event}_n but not all.

Rule 6: It is obligatory that the \textit{participant}_1 that \textit{includes} the \textit{participant}_a and the \textit{participant}_b and ... and the \textit{participant}_n, \textit{verb} exactly one \textit{event}_1 that \textit{includes} exactly one \textit{event}_a or exactly one \textit{event}_b or ... or exactly one \textit{event}_n but not all.

Next, the rules expressing each unordered participant execute the messages exchanged of the exclusive disjunction of the \textit{event}_1 are illustrated in both Rule 5 and Rule 6 (type 2 in Section 3.1.2). Each rule has only one potential specific request, which specifies each participant performing the particular event, e.g \textit{event}_a in the interaction. These declaration are shown in the following snippets, lines 152-153 and lines 159-160.

The predicates of specific request which correspond to each specific request rule for the respective SBVR rules (refer to Section 3.1.2):

```
--Type 2:
--A transformation of Rule 5:
147 one sig participant_1 extends participant{verb: one event_1}
148 one sig participant_2 extends participant{verb: one event_1}
...
149 one sig participant_N extends participant{verb: one event_1}
...
150 abstract sig event_1 extends event{}
{...}
--A request when each participant takes part for the exchanging message of exactly one event of XOR on events.
151 pred concreterequest[p1:participant_1, p2:participant_2, ..., p3:participant_N, a:event_a, ..., n:event_n] {
152 (p1.verb = a and ... and p1.verb != n)
152 and (p2.verb = a and ... and p2.verb != n)
...
153 and (p3.verb = a and ... and p3.verb != n)}
154 run concreterequest
--A transformation of Rule 6:
155 abstract sig participant_1 extends participant{verb: one event_1}
```
156 one sig participant_a extends participant_1{}
...
157 one sig participant_n extends participant_1{}
...
158 abstract sig event_1 extends event{}
{
...
}
--A request when each participant in a nesting group takes part for
exchanging message of exactly one event of XOR on events.
159 pred concreterequest[p1:participant_1, a:event_a, ..., n:event_n] { 
160 (p1.verb = a and ... and p1.verb != n)}
161 run concreterequest

4. Rule 7: It is obligatory that the participant_1 that includes the
participant_a or the participant_b or ... or the participant_n but not
all, verb exactly one event_1.

The following rules concern on the exclusive disjunction of par-  
ticipants involve in the given choreography. Rule 7 describes the  
specified XOR on participants executes the messages exchanged of the  
event_1. This is translated in the signature for participant_1 in  
lines 162-164. The specific request that could be constructed from  
this rule defines in the following codes snippet in lines 165-166. The  
subset of the participant_1, that is the participant_a (as declared  
in line 165) is targeted to send (receive) the event_1. The other par-  
ticipants are not considered.

The predicates of specific request which correspond to each specific  
request rule for the respective SBVR rules (refer to Section 3.1.2):

--Type 2:
--A transformation of Rule 7:
162 abstract sig participant_1 extends participant{verb: one event_1}
{
...
} 
163 lone sig participant_a extends participant_1{}
...
164 lone sig participant_n extends participant_1{}
--A request when exactly one participant in a nesting group sends (receives)
exactly one event.
165 pred concreterequest[p_a:participant_a, ..., p_n:participant_n,  
e1:event_1] { 

5. Rule 8: It is obligatory that the participant that includes the participant or the participant or ... or the participant but not all, verb exactly one event that includes exactly one event or exactly one event or ... or exactly one event but not all.

Rule 8 that is transformed into Alloy as in the snippet lines 168-171 specifies the specified XOR on the participant set performs the messages exchanged of XOR on the event set. Two potential specific requests that could be possible to be verified from this rule are: i) when exactly one participant such as the participant (a subset of the participant) is chosen to execute either one (XOR) of the given events in the event set (line 173); ii) when exactly one participant from the participant set is selected to execute exactly one event such as event, which is chosen from the given choices (XOR) of events inside the event set (line 176).

The predicates of specific request which correspond to each specific request rule for the respective SBVR rules (refer to Section 3.1.2):

--Type 2:
--A transformation of Rule 8:
168 abstract sig participant_1 extends participant{verb: one event}
{...}
169 lone sig participant_a extends participant_1{}
...
170 lone sig participant_n extends participant_1{}
...
171 abstract sig event_1 extends event{}
{...}
--A transformation of Rule i:
--A request when exactly one participant in a nesting group sends (receives) either one of the events inside a nesting group.
172 pred concreterequest[p_a:participant_a, p_n:participant_n, e1:event_1] {
173 (p_a.verb = e1 and ... and no p_n.verb)
174 run concreterequest
--A transformation of Rule ii:
A request when exactly one participant in a nesting group sends (receives) exactly one of events inside a nesting group.

\[
\text{pred concreterequest}\{p_a: \text{participant}_a, \ldots, p_n: \text{participant}_n, a: \text{event}_a, \ldots, n: \text{event}_n\} \{
   (p_a.\text{verb} = a \text{ and } p_a.\text{verb} \neq n \text{ and no } p_b.\text{verb} \ldots \text{ and no } p_n.\text{verb})
\}
\]

6. Rule 9: It is obligatory that the participant\(_1\) that includes the participant\(_a\) or the participant\(_b\) or ... or the participant\(_n\) but not all, verb exactly one event\(_1\) or exactly one event\(_2\) or ... or exactly one event\(_N\).

Rule 10: It is obligatory that the participant\(_1\) that includes the participant\(_a\) or the participant\(_b\) or ... or the participant\(_n\) but not all, verb exactly one event\(_1\) that includes exactly one event\(_a\) or exactly one event\(_b\) or ... or exactly one event\(_n\).

Lines 178-180 and lines 193-196 are translated from the specification of Rule 9 and Rule 10, respectively. Both rules concern on inclusive disjunction on events (either the events inside the event set or inside the event\(_1\) set) are executed by the exclusive disjunction of the participant\(_1\). There are four specific requests relate to both rules. Firstly, lines 181-183 define the pred concreterequest capturing only one participant, namely participant\(_a\) is chosen to send (receive) exactly one event\(_1\) which is chosen from the given event set. Lines 197-198 declares the predicate representing the participant\(_a\) is selected to send (receive) the event\(_a\) (the subset of the event\(_1\)). Secondly, the specific request concerns on exactly one participant, the participant\(_a\) is chosen for executing the messages exchanged of the exclusive disjunction of the given events (event\(_1\), ..., event\(_N\), but not all of them) (lines 185-187). This slightly contrasts with the specific request which is defined in lines 200-202. The declaration of that statement describes the participant\(_a\) is aimed to execute the messages exchanged of the exclusive disjunction of the event\(_1\) set (event\(_a\), ..., event\(_n\), but not all of them).

Thirdly, it is referred to the specific request on the particular participant inside the nesting participant\(_1\) to send (receive) each specified event. This can be seen in the snippet as defined in lines 189-191 and lines 204-205, the participant\(_a\) is obligated for sending (receiving) all specified events.
Lastly, the specific request is considered when there are two exclusive disjunction of requests, where each of them specifies the selected participant \( a \) is targeted to perform the combination of the given events. This is given in the defined predicate as shown in lines 189-191 and lines 207-209.

The predicates of specific request which correspond to each specific request rule for the respective SBVR rules (refer to Section 3.1.2):

```
--Type 2:
--A transformation of Rule 9:
178 abstract sig participant_1 extends participant {verb1: lone event_1, verb2: lone event_2, ..., verb3: lone event_N}
{...}
179 lone sig participant_a extends participant_1{}
...
180 lone sig participant_n extends participant_1{}
...

--A request when exactly one participant in a nesting group sends (receives) exactly one event.
181 pred concreterequest[p_a:participant_a, ..., p_n:participant_n, e1:event_1] {
182 (p_a.verb1 = e1 and no p_a.verb2 and ... and no p_a.verb3)
183 and ... and (no p_n.verb1 and no p_n.verb2 and ... and no p_n.verb3)}
184 run concreterequest

--A request when exactly one participant in a nesting group sends (receives) the exclusive disjunction of the particular event(s).
185 pred concreterequest[p_a:participant_a, ..., p_n:participant_n, e1:event_1, ..., e3:event_N] {
186 (p_a.verb1 = e1 and no p_a.verb2 and ... and no p_a.verb3 and ... and no p_n.verb1 and no p_n.verb2 and ... and no p_n.verb3)
187 or ... or (p_a.verb3 = e3 and no p_a.verb1 and ... and no p_a.verb2 and ... and no p_n.verb1 and no p_n.verb2 and ... and no p_n.verb3)}
188 run concreterequest

--A request when exactly one participant in a nesting group sends (receives) each event.
186 pred concreterequest[p_a:participant_a, ..., p_n:participant_n, e1:event_1, ..., e3:event_N] {
187 (p_a.verb1 = e1 and ... and p_a.verb3 = e3 and no p_n.verb1 ... and no p_n.verb3)}
188 run concreterequest

--A request when exactly one participant in a nesting group sends (receives)
the exclusive disjunction of the combination of the logical operations AND and OR (if any) over event terms.

188\[\text{pred concreterequest}\{p_a:participant_a, ..., p_n:participant_n, e1:event_1, e2:event_2, ..., e3:event_N\} \{
189 \text{\{p.a.verb1 = e1 and (or) p.a.verb2 = e2 and ... and no p.a.verb3 and ... and no p.n.verb1 ... and no p.n.verb3\}}
190 \text{or (... and(or) p.a.verb3 = e3 and no p.a.verb1 and and no p.a.verb2 and ... and no p.n.verb1 ... and no p.n.verb3)}\}
191 \text{run concreterequest}
192 \text{---------------------------------------------------------}
193 \text{--A transformation of Rule 10:}
194 \text{abstract sig participant_1 extends participant{verb: some event_1}}\{
195 \text{...}
196 \text{abstract sig event_1 extends event{}  }
197 \text{---A request when exactly one participant in a nesting group sends (receives) exactly one event in a nesting group}
198 \text{pred concreterequest}\{p_a:participant_a, ..., p_n:participant_n, a:event_a, ..., n:event_n\} \{
199 \text{\{p.a.verb = a and ... and p.a.verb != n and ... and no p.n.verb\}}
200 \text{run concreterequest}
201 \text{---A request when exactly one participant in a nesting group sends (receives) the exclusive disjunction of the particular event(a).}
202 \text{pred concreterequest}\{p_a:participant_a, ..., p_n:participant_n, a:event_a, ..., n:event_n\} \{
203 \text{\{p.a.verb = a and ... and p.a.verb != n and ... and no p.n.verb\}}
204 \text{or ... or \{p.a.verb = n and ... and p.a.verb != a and ... and no p.n.verb\}}
205 \text{run concreterequest}
206 \text{---A request when exactly one participant in a nesting group sends (receives) each event.}
207 \text{pred concreterequest}\{p_a:participant_a, ..., p_n:participant_n, a:event_a, ..., n:event_n\} \{
208 \text{\{p.a.verb = a and ... and p.a.verb = n and ... and no p.n.verb\}}
209 \text{run concreterequest}
210 \text{---request when exactly one participant in a nesting group sends (receives) the exclusive disjunction of the combination of the logical operations AND and OR (if any) over event terms.}
211 \text{pred concreterequest}\{p_a:participant_a, ..., p_n:participant_n,}
4.6. VERIFYING THE CHOREOGRAPHY IN ALLOY

7. Rule 11: It is obligatory that the \textit{participant$_1$} that includes the \textit{participant$_a$} or the \textit{participant$_b$} or ... or the \textit{participant$_n$} but not all, \textit{verb} exactly one \textit{event$_1$} and exactly one \textit{event$_2$} and ... and exactly one \textit{event$_N$}.

Rule 12: It is obligatory that the \textit{participant$_1$} that includes the \textit{participant$_a$} or the \textit{participant$_b$} or ... or the \textit{participant$_n$} but not all, \textit{verb} exactly one \textit{event$_1$} that includes exactly one \textit{event$_a$} and exactly one \textit{event$_b$} and ... and exactly one \textit{event$_n$}.

Rule 11 and Rule 12 are the possible constraints in the developed SBVR model describing the possible participants that could be chosen (XOR) from the \textit{participant$_1$} set to execute each unordered event. These events are referred to either each event in the event set (Rule 11) which is translated in lines 211-213, or each event which is the subsets of the \textit{event$_1$} (Rule 12). This is translated into Alloy as declared in lines 217-220.

The potential specific request states that exactly one of the participant is chosen to execute the interaction involving each unordered event. Lines 214-215 defines the predicate that captures the \textit{participant$_a$} sends (receives) each specified event, that are \textit{event$_1$}, \textit{event$_2$}, ..., \textit{event$_N$}. Unlike lines 221-222, the declaration is employed when the \textit{participant$_a$} sends (receives) each specified event inside the \textit{event$_1$} set, namely \textit{event$_a$}, ..., \textit{event$_n$}.

The predicates of specific request which correspond to each specific request rule for the respective SBVR rules (refer to Section 3.1.2):

\begin{verbatim}
  "Type 2:
  "A transformation of Rule 11:
  211 abstract sig participant$_1$ extends participant{verb1: one event$_1$,
  verb2: one event$_2$, ..., verb3: one event$_N$}
\end{verbatim}
Step 3: Assign code for verifying the specific request in each `pred concreterequest` as defined at the end of each predicate, namely `run concreterequest`.

4.6.2 Verifying - static constraints

Apart from issuing specific requests to explore whether they can be realised, the user might want to check static constraints which do not pertain to the orderings of the messages, but relate instead to properties of the application domain itself. Global constraints pertaining to static constraints specifically for date constraints in the Alloy model are verified by applying assertions. Assertion in the Alloy model for the choreography is employed to detect flaws in the model, particularly on
date constraints only. Note that, if an assertion is not valid, Alloy will produce a counterexample.

The assertions necessary for verifying the date constraints that were declared in the SBVR rules. The following snippet shows each assertion corresponding and referring to each given rule which represents each date constraints as described in Section 4.4.3. The date constraints namely equality, inequality, in between, and begin before is transformed into Alloy as defined in the snippet in Section 4.4.3. We name each assertion with distinct name to characterise the purpose of each of them.

```
--assertion for Rule 1:
1 assert equalitydate {all e1:event_1, e2:event_2 | 
2 (e1.verb = e2.verb)}
3 check equalitydate
--assertion for Rule 2:
4 assert inequalitydate {all e1:event_1, e2:event_2 | 
5 (e1.verb != e2.verb)}
6 check inequalitydate
--assertion for Rule 3:
7 assert inbetweendate {all e1:event_1, e2:event_2 | 
8 orderdate[d,d1,d2,d",e1,e2] implies 9 (e1.verb = d or e1.verb = d1 or e1.verb = d2 or 
9 e1.verb = d")}
10 check inbetweendate
--assertion for Rule 4:
11 assert beginbeforedate {all e1:event_1 | 
12 orderdate[d,d1,d2,d",e1] implies 
13 ((e1.verb1 = e1.verb2) or ( e1.verb1 = d and e1.verb2 = d1) 
14 or (e1.verb1 = d and e1.verb2 = d2) or (e1.verb1 = d and e1.verb2 = d") 
15 or (e1.verb1 = d1 and e1.verb2 = d2) or (e1.verb1 = d1 and e1.verb2 = d") 
16 or (e1.verb1 = d2 and e1.verb1 = d"))}
17 check beginbeforedate
```

In further explanation, the statements in the above snippet as defined in lines 1-3 check the equality of two dates date_1 for event_1 and date_2 for event_1. Lines 4-6 is the assertion to validate that the inequality of date_1 and date_2. The assertion that is defined in lines 7-10 verifies whether the specified date_1 for event_1 are within date_2 and date_3 of the event_2. In line 12, the predicate pred orderdate which was declared in Section 4.4.3 for Rule 3 is invoked for validating the declared constraints in that predicate. Similar to the assertion for checking the constraint of begin before, the predicate pred orderdate that is
defined for Rule 4 is invoked. This is shown in the statements in lines 11-17. It is checked whether date\_3 starts before date\_2 or on the same date.

### 4.7 Case Studies

As discussed in Chapter 3 (Section 3.2), there is a whole details on an illustration for developing the SBVR model for service choreography by considering the case study of ACME Travel scenario which is adopted from [6]. The demonstration for other two case studies, namely the Online Photoshop case study as well as the Tuition Fees system case study: International Student, are included in Appendix A and Appendix B, respectively.

In this section, we demonstrate how to transform the SBVR model for ACME Travel (AT) case study and to develop the corresponding Alloy model that enables to generate the choreography that could be verified for realizability. The Alloy model representing the other two case studies are illustrated in Appendix C and Appendix D. Note that the terms, the fact types, as well as the SBVR rules involved in the SBVR model for AT case study will be referred to the developed model in Section 3.2.

The description commences with the transformation that takes the terms and fact types and all relating SBVR rules, i.e. complex interactions, precedence, and static constraints, to produce the corresponding Alloy model for the AT case study. In parallel, there are the elaborations for developing the predicates pertaining to generate choreography, realizability verification, and static constraint verification. They are described as in the following.

**Transformation of terms and fact types from the SBVR model into Alloy**

As a brief, the business scenario in the AT case study pertains to the arranging travel scenario involving several interacting participants; Customer, Acme Travel, Airline, Accommodation includes Hotel and Apartment, Transport includes Bus and Train, and Tour Agency. The collaboration between them requires several events that need to be performed such as reservation request, airline reservation, etc, so that the collaboration goal could be achieved. In the SBVR model, this strongly relates to the development of the terms which act as the basis for developing the fact types and the rules subsequently.

Terms in figure 3.1 represents all participants involved in the choreography for the AT system, figure 3.2 shows terms for all potential events involved in the interaction between the participants in the AT system, and figure 3.3 indicates terms for the possible static constraints belongs to the involved participant or the event (if any).

All terms are transformed into Alloy as the signatures basically. However, it is important to note that all the participants and the events involved in the AT system are grouped accordingly in the given participant set and the event
4.7. CASE STUDIES

set as described in the corresponding fact types (see figure 3.4, figure 3.5, and figure 3.6) for the participant set and figure 3.7, figure 3.8, and figure 3.7 for the event set). As depicted in all figures, there are some participants as well as events obtain a group of nesting such as the participant in the AT system, accommodation is divided into two, namely hotel and apartment, while the event in the AT system, accommodation reservation has its subsets: hotel reservation and apartment reservation. In Alloy, the abstract signature and its extension play an essential role to represent the participant set, the event set, as well as a group of nesting. Figure 4.1 illustrates the abstract signatures and the signatures involve for the participant set.

Since the participant set encompasses customer, AT, airline, accommodation, tour agency, and transport as described in the fact types (e.g. participant includes customer), they are declared as the extension of the abstract sig participant{}. There are two other abstract signatures, namely accommodation and transport, demonstrating them as the superset for hotel’s signature and apartment’s signature, as well as for bus’s signature and train’s signature, respectively. Also, this reflects the fact type defining a group of nesting (e.g. accommodation includes hotel). Note that all signatures for participants (except abstract signatures) are bound to one cardinality since each participant represent individual participant service.

Figure 4.2 depicts all events involving in the messages exchanged between the participants in the AT system. It can be seen that all signatures for event are the extension of the abstract sig event{}. This represents the fact types presenting the event set. Abstract signatures for accommodation and transreservation illustrate a nesting of accommodation reservation and a nesting of transport reservation in the SBVR model. Similar to all response events to represent their subset events, either successful or unsuc-
Figure 4.2: Abstract signatures and signatures for the event set
cessful. All of them are translated to be the abstract signatures in Alloy. Figure 4.2 shows the abstract signatures, namely accommodation reservation response set, airlineresponse designates the airline reservation response set, tourresresponse denotes the tour reservation response set, and transresresponse corresponds to the transport reservation response the, in the SBVR model. As mentioned for the participant’s signatures (the participant set), all event signatures (except abstract signatures) are defined using one cardinality since each event in for the event set is the single event.

All figures illustrating the fact types for participants demonstrate the local behaviour model for each participant by describing the messages exchanged of the sending or the receiving of the events. Also, there is the fact type illustrating the domain-specific constraint (static constraint) which belongs to the particular participant. These are captured by defining fields for each signature representing the participants. For instance, sends, receives, and has fields connecting to reservationrequest, notification, and name, respectively are defined in the signature for customer in figure 4.1. These to characterise the local behaviour model for the customer which includes the export and the import message as well as the domain specific constraint as described in the fact types illustrated in Figure 3.4. Each export and import message fact types specify the association time interval as well. As explained in Section 4.3, the verb at associating with the particular event and the time interval are declared in that particular event signature. The fact type, customer sends reservation request at t1 in Figure 3.4 shows the verb at associates with the event reservation request and t1. Hence, the field at is defined in the one sig reservationrequest extends event {...} and is mapped to the signature for T1 RR representing the initial time for reservation request event (Figure 4.2). The transformation of fact types for other local behaviour models in the AT system into Alloy follow the identical method as the aforementioned description.

It is essential to note that there are domain-specific constraint belong to any events. For example, the fact type in figure 3.7 describes the event reservation request has two static constraints, namely start date and end date. This domain-specific constraints is required to be define in the reservationrequest’s signature such as has field inside this signature represents the startdate static constraint. All domain-specific constraints that are belonged to any events are defined in the intended event’s signature applying the same approach.

Transformation of SBVR rules from the SBVR model into Alloy

The SBVR rules in the SBVR model for the AT system contains the global constraints relating to the complex interaction represents the concurrent and alternative interactions, the precedence shows the ordering of events, and the
static constraints describes the date constraints involved in the AT system. The transformation of all involved rules in figure 3.13, figure 3.14, figure 3.15, and figure 3.16 are described as in the following:

1. **Complex interaction**

As explained in Chapter 3 (section 2.3.2), the SBVR rules pertaining to complex interactions describe a single or multiple participants execute the messages exchanged of a single or multiple events. In the SBVR model for the AT system, diverse of rules are developed to capture the global constraints involving multi-party conversations amongst distinct participants in the AT system.

**Rule 1, Rule 2, Rule 8, Rule 27, and Rule 28** in the figures illustrate a single participant executes the messages exchanged of a single event at a given time interval. For instance, **Rule 1** and **Rule 2** give a description of interaction between the customer and the AT in executing the same event that is reservation request. However, through reading both rules, we know that the interaction is initiated by the customer then it is followed by AT. This rules are translated into Alloy as depicted in figure 4.1 and figure 4.2. **Rule 1** shows the participant involves in executing the `reservation request` is the `customer`. Thus, the signature `one sig customer extends participant{...}` in figure 4.1 is considered. The cardinality is bound to that signature as the quantification `the` is employed for describing `customer` in the rule. Then, the `sends` verb after term `customer` in the rule is translated in Alloy as a field for the `customer`'s signature whose range is `reservation request` associating with `one` cardinality. This encapsulates the statement ".... sends exactly one `reservation request`" in the rule. Afterwards, the time interval `t1` connecting with the event `reservation request` in which the verb `at` is specified in between them, are defined in the signature for `reservation request` with a field `at` is mapped to the time interval 1 for reservation request, that is `T1_RR` (figure 4.2). Since this time interval refers to one time, `one` cardinality is declared before `T1_RR`. The similar mechanism is applies for transforming the other identical rules as mentioned above, into Alloy.

**Rule 4** in the SBVR model for the AT system describes an inclusive disjunction (OR) on events (i.e. `airline reservation` or `accommodation reservation` or `tour reservation` or `transport reservation`) which are executed by a single participant **AT**. This is captured by a declaration of the fields `requestfor`, `requestfor1`, `requestfor2`, and `requestfor3` in the **AT**'s signature, which each field is mapped to the respective events. Note also the `lone` cardinality on the associated `requestsfor` fields in the signature of the **AT** participant, which denote that some or all might happen in a given
execution and the commented as a fact underneath the signature for AT, is a condition that must hold at all times. Since, the time interval \( t_1 \) in the rule associates with each specified event, hence, the time interval 1 for each event is populated in each corresponding event’s signature. For example, in the signature of the airlinereservation event, there is a at field connecting with one T1,AR showing the time interval 1 for airline reservation event. The same approach is exploited to define the time interval for the other events accomreservation, tourreservation, and transreservation.

Rule 15 and Rule 17 as well as Rule 16 and Rule 18 are intentionally describing a sending of either one of responses, successful or unsuccessful response (exclusive disjunction (XOR) of events) by a single participant (Rule 15 and Rule 17) or by exclusive disjunction (XOR) of participants (Rule 16 and Rule 18).

The declaration for Rule 15 and Rule 17 in Alloy can be seen in the one sig airline extends participant\{...\} and in the one sig touragency extends participant\{...\} (figure 4.1). Both signatures with one cardinality represents single participant, have the sends field which is connected to each response one airlineresponse and one tourresponse. Each response is the abstract signatures for the airlineresponse event and the abstract signatures for the tourresponse event. The abstract signature of the airlineresponse event includes at least one of succairlres and unsuccairlres events, on the other hand, The abstract signature of the tourresponse event includes at least one of succtourres and unsucctourres events (figure 4.2). The choice of lone cardinality for each subsignature and the fact statement underneath each abstract signature are due to the exclusive disjunction (XOR) applied to the events. The similar method for translating the other rules such as Rule 19, Rule 20, Rule 21, and Rule 22, into Alloy is applied. These rules describe the receiving of the exclusive disjunction of each response event, either successful or unsuccessful, by the AT participant.

A slight different approach is employed for defining Rule 16 and Rule 18 in Alloy. Since both rules represent the exclusive disjunction (XOR) on the accommodation and the transport participants, the abstract signatures for each participant are considered as exhibited in figure 4.1. The choices of lone cardinality for each subsignatures and the declaration of fact for each abstract signature are designated to capture the exclusive disjunction (XOR) applied to the participants. Then, the declaration for the exclusive disjunction (XOR) on each response event accomresponse and transresresponse that is sending by the accommodation and the transport participants apply the same method described previously.

Rule 6 and Rule 7 show that the subset of the accommodation
participant, that is the hotel and the apartment participants have their own responsibility in receiving the hotel reservation and the apartment reservation events, respectively. Note that both events are the subsets of the accommodation reservation event. In this case, the abstract signature for accommodation is considered. Nevertheless, since both apartment and hotel participants (the extensions of accommodation) perform the distinct event although those events are the subsets of the same event, each event hotelreservation and apartmentreservation is assigned in each signature for hotel and apartment, respectively, instead of assigning the accomreservation event in the abstract signature of the accommodation participant. The receives field is declared in each corresponding signature participant associating with the intended event, hotelreservation and apartmentreservation. The identical approach is used to translate Rule 9 and Rule 10.

2. Precedence

There are two types of precedence are employed for expressing the rules describing the ordering of time intervals associating with the same event and the ordering of messages exchanged of different events in the AT system.

a) Transformation of SBVR rules pertaining to time notion

All rules describing complex interactions in the SBVR model employs the time notion as an indication for describing the ordering of messages exchanged of the common event, in parallel with the specification of Rule 29 to Rule 38. For instance, Rule 27, Rule 28, and Rule 38 in the figures demonstrate the declaration of time for the notification event explicitly, where \( t_1 \) immediately precedes \( t_2 \). In this case it declares the fact that AT sending the notification happens before the customer receiving it.

The declaration of these time interval associating with the notification event are captured in the at and at1 fields of the event signature notification in Alloy. For the sake of completeness we note that time is also declared as a signature, hence the above SBVR rules would require the declarations as described in figure 4.3 in Alloy.

Since both \( t_1 \) and \( t_2 \) represent the time interval for the notification event, thus both are translated as the extensions of the abstract sig notification.time extends event {...} (the subsignature of the abstract signature for time). The subsignatures \( T1_N \) and \( T2_N \) denote \( t_1 \) and \( t_1 \) for the notification event, respectively. Each signature populates which participant is connected with. Since, AT initiates the messages exchanged of the notification event, AT is declared in the signature of \( T1_N \) time
4.7. CASE STUDIES

Figure 4.3: Time declarations in Alloy for the SBVR model of the AT system

interval. The same approach applies for the customer. In order to show the precedence between the corresponding time intervals, immediatelyprecedes field is defined in the T1_N’s signature, which is mapped to the next time interval T2_N.

The declaration of time interval here shows how the constraints in the local behaviour are combined and lifted to the global behaviour across participants. We will have more to say on this once we have addressed precedence between events.

b) Transformation of SBVR rules pertaining to immediately precedes notion

As discussed previously, the rules describing the ordering of events could capture the messages exchanged of a single event or multi-
ple events immediately precedes the messages exchanged of another single event or the other multiple events. This is closely related to the previous time notion. Continuing from the specifications of Rule 27 and Rule 28, Rule 23 defines precedence between the exclusive disjunction of the accommodation reservation response and the notification events. This is captured using the immediatelyprecedes field of the preceding events signature. In this case, this is accomresponse, which in itself includes a XOR on successful accommodation reservation and unsuccessful accommodation reservation. The accompanying fact ensures that only one or the other of these events occurs at any one execution. The last statement of the fact ensures that the fields of every participant in the SBVR model who sends/receives the event (notification) that this event (accomresponse) precedes is assigned that event (e.g., immediatelyprecedes = customer.receives). It might be instructive here to recall the declaration of the customer’s signature. The declaration of rules having the same structure as Rule 23 exploits the same approach such as for Rule 25 to Rule 26.

Rule 11 and Rule 13 illustrate the conversion of Rule 23. For instance, Rule 11 describes precedence between a single event airline reservation and the exclusive disjunction (XOR) of the airline reservation response event. This is defined in Alloy in the signature of airlinereservation event, where the immediatelyprecedes field is employed to represent the precedence between airlinereservation and airlineresponse which includes the XOR on successful and unsuccessful reservation for airline. In addition, as mentioned previously, there is a statement in the fact for the airlinereservation’s signature to confirm the airline reservation immediately precedes the event succairlres or unsuccairlres, that is sent and received by the intended participant. In this case, since airlineresponse event is targeted to be sent and received by the airline and AT participants, respectively, immediatelyprecedes = airline.sends and immediatelyprecedes = AT.receives is assigned in the fact.

Rule 12 and Rule 14 specify the messages exchanged of XOR of one event immediately precedes the messages exchanged of the XOR of another event. The specification of Rule 12 into Alloy is captured when the immediatelyprecedes field is declared in the abstract signature of the accomreservation event which is associated with accomresponse to illustrate the precedence between these two event’s signatures. The similar approach is used to define the fact for the accomreservation signature. There are a statement to guarantee only one of its extensions (hotelreservation
4.7. CASE STUDIES

or apartmentreservation) occurs at any one execution, and another statement to ensure the intended participant sends (receives) the same event (succaccomres or unsuccaccomres) of the immediate after the accomreservation event. Since the accommodation that includes the XOR on the hotel or the apartment participants are aimed to send the XOR of the accomresponse event, the statement immediatelyprecedes = hotel.sends or immediatelyprecedes = apartment.sends are defined, with or logical operator between them. The same mechanism applies for translating Rule 14 into Alloy. It can be seen in the abstract signature for transreservation.

Rule 3 in the SBVR model for the AT system describes the execution of the reservationrequest event occurs immediate before the execution of OR on airlinereservation, accomreservation, tourreservation, and transreservation. This is transformed into Alloy with the declaration of immediatelyprecedes fields for each OR on event in the reservationrequest’s signature. lone cardinality is used to ensure at least one of the events is occurred immediate after the reservationrequest event. It is more confirm when the fact statement, e.g. #immediatelyprecedes = 1 or #immediatelyprecedes1 = 1 or ... is designated underneath the signature for reservationrequest. Similar approach as the previous rules, any participant who sends (receives) any events which are immediately after the reservationrequest event must be equal to the defined field that event (airlinereservation, accomreservation, tourreservation, and transreservation) associates with.

3. Static constraints

This static constraints are related to the specification for the date constraints. The development of rules for date constraints pertains to the declaration of a date set in fact types as shown in figure 3.12. This date set is translated into Alloy by defining the signature for Date and its subset signatures as depicted in figure 4.4.

The SBVR rules for date constraints for AT system are illustrated in figure 3.16. Figure 4.5 shows how these are transformed into the Alloy model. The ordering utility of Alloy is employed (open util/ordering [Date] as DateOrder) to specify Date to be an ordered set. The ordering on dates is needed for the checking the constraints later. The predicate pred init sets the start date as the first date in the Date set. The set is arranged in order in the fact traces, as d, d1, d2, and d". The predicate pred traveldate is used to constrain the various dates as specified in the corresponding SBVR model. The remaining code defines the necessary constraints on the dates for the example con-
Figure 4.4: A signature for *date*'s declarations in Alloy for the SBVR model of the AT system

```alloy
sig Date()
one sig startdate in Date{}
one sig enddate in Date{}
one sig outbounddate in Date{}
one sig inbounddate in Date{}
one sig tourdate in Date{}
one sig checkindate in Date{}
one sig checkoutdate in Date{}
one sig departuredate in Date{}
one sig arivaldate in Date{}
```

Figure 4.5: Predicates for constraining the rules for date constraints in Alloy for the SBVR model of the AT system

```alloy
open util/ordering [Date] as DateOrder

pred traveldate [disjoint d,d1,d2,d' : one Date , r:reservationrequest, t:tourreservation, p:transreservation] {
  r.has = d and r.has1 = d'
  t.has = d or t.has = d1 or t.has = d2 or t.has = d'
  p.has = d or p.has = d1 or p.has = d2 or p.has = d'
  p.has1 = d or p.has1 = d1 or p.has1 = d2 or p.has1 = d'
  not ((p.has = d' and p.has1 = d) or (p.has = d' and p.has1 = d1) or (p.has = d' and p.has1 = d2)
   or (p.has = d2 and p.has1 = d) or (p.has = d2 and p.has1 = d1) or (p.has = d1 and p.has1 = d))
}

pred init [d:Date] {d = reservationrequest.has}

fact traces {
  init [first]
  let d'' = last |
  some d : Date - last |
  let d1 = d.next, d2 = d1.next |
  some t: tourreservation, r: reservationrequest, p:transreservation
  traveldate[d,d1,d2,d'',r,t,p]
}
```
4.7. CASE STUDIES

Constraints expressed in Rule 46 the tour date can be any date in the \textit{Date} set, while the departure and arrival dates of the transport reservation are in between the start and end date of the reservation request (Rule 47 and Rule 48), with the additional condition that the departure date must be on or after the arrival date (Rule 49).

Rule 39 to Rule 42 shows the quality of dates. This can be seen in the fact statement for the signature for \textit{reservationrequest} event. The statement \texttt{has = accomreservation.has and has = airlinereservation.has} is used to ensure the start date of the reservation request equals to the check-in date of the accommodation reservation and the outbound date of the airline reservation. Similarly, the statement \texttt{has1 = accomreservation.has1 and has = airlinereservation.has1} is defined to make sure the end date of the reservation request equals to both the check-out date and the inbound date of the accommodation reservation and the airline reservation, respectively.

On the other hand, Rule 43 to Rule 45 represent the inequality of dates. The statement \texttt{has != has1} in the fact of the \textit{reservationrequest}’s signature is defined to capture Rule 43. The same statement applies for each signature for the \textit{accomreservation} and the \textit{airlinereservation}.

**Generating and verifying the choreography** Generating the choreography for \textit{in} Alloy employs the predicate \texttt{pred initialevent}. Figure 4.6 illustrates the predicate \texttt{pred initialevent} that is used for generating the choreography for the AT system.

\texttt{pred initialevent[reservationrequest, a:airlinereservation, c:accomreservation, t:tourreservation, s:transportreservation]
\{(r:immediatelyprecedes = a or r:immediatelyprecedes1 = c or r:immediatelyprecedes2 = t or r:immediatelyprecedes3 = s)\} run initialevent}

Figure 4.6: Predicates for generating the choreography for the SBVR model of the AT system

Two important components are required for developing the \texttt{pred initialevent} and subsequently for generating the choreography. Firstly, identifying an initial event of all possible executions of the required service interactions and secondly, identifying the SBVR rule which specifies the precedence between that initial event and the other required event(s) in the SBVR model. In figure 3.7, there is a fact type declaring \textit{reservation request} as the initial event for executing all possible executions of the interaction in the AT system. Then, Rule 3 in figure 3.13 illustrates the \textit{reservation request} initiates the execution immediate before the inclusive disjunction (OR) on \textit{airline reservation}, \textit{accommodation reservation}, \textit{tour reservation}, and \textit{transport reservation} events. Thus, all the specified events in the rule are declared in the \texttt{pred initialevent} as shown in figure 4.6. As can be
seen in the signature for reservationrequest event (figure 4.2), its fields immediatelyprecedes, immediatelyprecedes1, immediatelyprecedes2, and immediatelyprecedes3 associate with each intended event. Therefore, the statement $r$.immediatelyprecedes = a or $r$.immediatelyprecedes1 = c or $r$.immediatelyprecedes2 = t or $r$.immediatelyprecedes3 = s are defined in the pred initialevent to ensure the execution of the messages exchanged of the reservationrequest occurs right before the execution of the messages exchanged of the inclusive disjunction (OR) on the aforementioned events. Finally, run initialevent is specified for executing the pred initialevent and generating the required services interactions (instances) simultaneously.

As the instance in Alloy does not output all possible executions in one graph, hence it is not possible to show them all in one figure. For presentation purposes, the most cases possible in one request in the predicate are included (figure 4.6). Hence, all four reservation requests are considered - airline, accommodation, transport, tour - and one type of accommodation (apartment) and transport (train) are chosen, and the case of successful responses from all are considered. Figure 4.7 illustrates the corresponding instance of the choreography (global constraints) generated by Alloy.

Figure 4.7: Graphical representation of the generated choreography in Alloy for the SBVR model of the AT system

The predicate pred initialevent applied above declares the reservation request as the initial event. Thus, the diagram should be read start-
The incoming arrows from participants customer and AT, annotated by sends and receives respectively, capture the interaction. Note time is not included here, to not clutter the diagram but it can be included. The outgoing arrows of reservationrequest depict what happens next: airlinereservation, apartmentreservation, tourreservation and trainreservation occur immediately after reservationrequest in no particular order, i.e., they are unordered.

The incoming arrows of these four events also depict the fact that AT sends the requests to each provider, in no particular order. The outgoing arrows from the these four events capture the precedence of the request over the response from the respective provider (train, touragency, airline, apartment). Finally, the incoming / outgoing arrows on the four successful response events depicts the precedence of the four responses over the notification which is subsequently sent by AT to the customer.

In order to verify realisability, the Alloy model can readily verify for a specific request whether it can be realised by the given service choreography. This request can be formulated as a predicate in Alloy. This is shown in figure 4.8 with respect to the specific request rule as illustrated in figure 3.17.

```
pred concreterequest [t:AT, h:hotelreservation, a:airlinereservation, t:tourreservation]
{ (t.requestsfor1 = h and no t.requestsfor and t.requestsfor2 = r and no t.requestsfor3) 
or (t.requestsfor = a and no t.requestsfor1 and no t.requestsfor2 and no t.requestsfor3) } 
run concreterequest
```

Figure 4.8: Predicates for verifying the specific request on the generated choreography of the AT system

Assume that a user wishes to make either an airline reservation only (t.requestsfor = a), or an accommodation reservation but restricted to hotel (t.requestsfor1 = h) and a transport reservation restricted to a bus (t.requestsfor3 = b), and no tour reservation. This request is formulated in the pred concreterequest as depicted in figure 4.8. When this pred concreterequest is executed, the request is checked against the generated choreography. If there is at least one instance (one possible execution) of the choreography which meets the request, then we say the request can be realised.

Figure 4.9 depicts one possible execution for the request in question. It shows that AT requests for a bus reservation and hotel reservation, and airline reservation have been reserved successfully. The successful responses for each reservation have been sent by each provider (bus provider and hotel provider, and airline provider in the other case). These are immediately preceded by the reservation requests from AT. They themselves immediately precede the noti-
CHAPTER 4. GENERATION AND VERIFICATION CHOREOGRAPHY MODEL

Figure 4.9: One possible realisation of the user request concreterequest on the generated choreography of the AT system

The subsequent figure 4.10 of pred concreterequest represents a user request concerning a hotel reservation and an apartment reservation. When this is executed, Alloy does not find any possible execution that can realise this request; it returns no instance found. This is because the SBVR model includes a constraint (exclusive disjunction (XOR) on the hotel reservation and the apartment reservation) that prohibits these two events occurring in the same run. This is reflected by the Alloy realisability check.

For verifying the static constraints specifically for date constraints as specified in the SBVR rules in figure 3.16, the following assertions (figure 4.11) are applied.

The statements in assert flightaccomvalidate check whether the start (end) date of the reservation request is equal to the outbound (inbound) date


Figure 4.11: Assertions for verifying the static constraints in the SBVR model for the AT system

of the airline reservation and the check-in (check-out) date of the accommodation reservation \( (\text{r.has} = \text{a.has} \text{ and } \text{r.has1} = \text{c.has1} \text{ and } \text{a.has} = \text{c.has}) \) as well as check whether the start and the end date of the reservation request is not equal \( (\text{r.has} \neq \text{r.has1}) \). \texttt{assert transvalidate} is the assertion to validate that the specified dates for the transport reservation are within the start and end date of the reservation request.

If an assertion is not valid, Alloy will produce a counterexample which can be shown in a graph like before (e.g., recall Figure 4.7). In this case, both assertions are valid.

4.8 Concluding remarks

We have seen in this chapter how the developed SBVR model was taken as the input and then transformed into Alloy Analyzer to automatically produce the exact solution that corresponds to the input SBVR model (generating the choreography) and to verify the realisability and static constraints (if any).

The detailed steps and methodology has been produced commencing with taking terms and fact types in the SBVR model for specifying the participant set, the event set, the domain-specific constraints, and the messages exchanged into Alloy. These fact types are then translated as the signatures or abstract signatures in Alloy and then populate the intended constraints as the fields. It is then followed by transforming the SBVR rules capturing the specification of the complex interactions (encapsulate the alternative and concurrent interactions), the precedence between events (the time notion and the immediate precedes notion), and the static constraints (date constraints). All of these are defined into suitable signatures or abstract signatures, by employing the appropriate cardinality associating with the intended signatures and fields, as well as the fact construct to ensure and guarantee Alloy produces the exact solution that corresponds to the input SBVR model. Finally, the predicates and the facts are applied for generating the choreography model together with verifying the realisability on the generated choreography, and the domain-specific constraints.
The formalisation and the transformation of the proposed SBVR model into Alloy have some challenges and limitations.

First, Alloy only includes the cardinality all, some (at least one), lone (at most one), one (exactly one), and no. There is an absence of the quantification types at-least-n and at-most-n in Alloy, yet not in SBVR. However, the types of cardinality introduced in Alloy as well as the participation and the messages exchanged constraints addressed in the model transformation, were enough to capture the message exchange in the choreography. This can be seen when the transformation encapsulates in three case studies that were conducted. Furthermore, there was no necessity to define rules that refer to the multiplicity of messages involving at-least-n and at-most-n. For the purposes of generating service choreographies, since our aim as one of capturing the interaction described in a sequence diagram or a message sequence chart (to resemble the multiplicity constraints of the Unified Modeling Language (UML)), there is no multiplicity on a specific message being sent from a given participant (object, or component, or service), messages are sent and received by each participant during the multi-party conversation.

Second, as described in Chapter 2 and this chapter, the expression of exclusive disjunction (XOR) on participants or on events in the SBVR rule must include a level of nesting such as "It is obligatory that ... exactly one \( \text{event}_1 \) that includes exactly one \( \text{event}_a \) or exactly one \( \text{event}_b \) but not both". This is because, as far as we are concerned, only the extensions of a signature (mutually disjoint)’s notion in Alloy are able to capture the declaration of the participant set and the event set in the SBVR model and subsequently to produce the required alternative interaction in the choreography. This were left out a flexibility part in SBVR in specifying the rules involving the logical operation.

Third, the transformation of time notion from the SBVR model into Alloy is not a straightforward implementation. In the SBVR model, the time notion is used in the rule such as "It is obligatory that the participant\(_1\) verb exactly one \( \text{event}_1 \) at exactly one \( t_1 \)" and the immediately precedes notion is used to specify the precedence between the time intervals. In order to guarantee the transformation of the time notion into Alloy as what we aimed, the abstract signature for time is introduced. The intention is to ensure the specified time interval associates with the correspond event and participant, and to represent the ordering of time interval on the same event.

Fourth, Date in Alloy was introduced in a Date set to enable the library module util/ordering is employed. This is necessary to capture the date constraints since Alloy is not representable to express the comparability as well as to produce the precedence between dates. Consequently the unique name for each date must be considered and the initialisation for the initial and the final date must be identified.

Lastly, although Alloy is poor in support the sequence for producing the dynamic interaction, its advantage in modelling the connections as a relation between endpoints, or as a set of connection atoms (each mapped to its endpoints), Alloy makes it possible to model the dynamic coordination.
4.8. CONCLUDING REMARKS

The compatibility of Alloy structures with the SBVR structures and its advantageous in automatic analysis as mentioned in Chapter 2, the transformation and the formalisation of the SBVR model into Alloy is finally been introduced. It can be seen when the proposed transformation methodology is able to be applied in three conducted case studies i) Acme Travel system (in Section 4.7), ii). online photo shop (Appendix C), and iii). tuition fee system (Appendix D). These case studies have been used to develop and to produce the SBVR model in Chapter 3. They are then transformed into Alloy to generate the choreography model and afterwards verify the realisability. The summary of the transformation from the SBVR model into Alloy model which enables the automatic translation between them using the SBVR2Alloy (Chapter 5) tool was described in [67].
Chapter 5

**SBVR2Alloy: an SBVR to Alloy Compiler**

In Chapter 3, the detailed methodology of developing the SBVR model for service choreographies was described. In Chapter 4, we presented the proposed approach on formulating and transforming the SBVR model into the Alloy model for generating the service choreographies from the corresponding SBVR model and for verifying the generated choreography model (realisability) and static constraints particularly on date constraints.

This chapter emphasises on describing the implementation of an automated tool, namely SBVR2Alloy that translates a declarative specifications, the SBVR model into the Alloy model automatically. The SBVR2Alloy compilation tool builds on the model transformation described in Chapter 4. The mechanism is the SBVR2Alloy compiler will take an SBVR model, which is written following the template as described in Chapter 3 and will produce an Alloy file (.als) which can be compiled into Alloy syntax and effectively be verified. In the following sections, there are a description of an architecture of the proposed SBVR2Alloy tool, a details implementation of the SBVR2Alloy tool together with its illustration, and a challenging in implementing the SBVR2Alloy tool.

### 5.1 Architecture of SBVR2Alloy

The development of the SBVR2Alloy tool is based on the UML diagram as depicted in Figure 5.1. As shown in the figure, a Model class which associates with a ReadTextFile class as well as a Rules class play an important role in the tool development. The ReadTextFile class enables to extract the SBVR file input and to be undertaken and executed by the Rules class. There are several main
functions in the \textit{Rules} class to foster the development of the tool. The following Table 5.1 describes each responsibility:

![UML diagram of SBVR2Alloy tool](image)

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{uml_diagram.png}
\caption{UML diagram of SBVR2Alloy tool}
\end{figure}

\subsection{Implementation of SBVR2Alloy}

The implementation of \textit{SBVR2Alloy} tool was realised using Java and a simple GUI was created using the Java Swing classes. This enabled us to provide a platform independent tool for translating SBVR to Alloy.

The development of tool follows a state machine paradigm. State machines are thoroughly used in the literature to define models that can behave in different ways depending on their state \cite{68}. This approach is applicable as generating Alloy code for participant signatures differs from that of event signatures. On account of this, we defined the framework (template) in Table 3.1 which contains the prefixed comments enable the tool to parse the template (SBVR file) and changes its state depending on prefixed comments in the code. For example as shown in the template (Table 3.1), the first comment is the ‘–Participants Set’ so when the tool reaches that comment it changes its state to participant (‘p’) and handles all participant related processes for translation. The same applies for events, constraints and rules.

A total of 12 states were identified to encompass the two stages of translation. The provided template also enables the user to develop the SBVR model by providing the required information about the specific terms, fact types and rules. This essentials for implementing the \textit{SBVR2Alloy} tool to transform the developed SBVR model into Alloy model.
5.2. IMPLEMENTATION OF SBVR2ALLOY

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>checkRules</td>
<td>Checking the quantification specifically after the <em>verb</em> or the verb <em>immediately precedes</em> in the specified rules for assigning the cardinality in Alloy.</td>
</tr>
<tr>
<td>chekUtils</td>
<td>Identifying the static constraints pertaining to a <em>date</em> set for generating the library module <em>ordering</em> utility.</td>
</tr>
<tr>
<td>checkSets</td>
<td>Checking the existence of the superset of any participant or event set for transforming as the abstract signature.</td>
</tr>
<tr>
<td>checkOrdering</td>
<td>Determining the ordering between events rule structure such as for OR on events to defining the <em>immediately precedes</em> field and identifying the appropriate associating cardinality.</td>
</tr>
<tr>
<td>addEventTimedSignature</td>
<td>Identifying and storing the specified event associating with the specified time to be defined in Alloy.</td>
</tr>
<tr>
<td>generateFactParticipant</td>
<td>Determining the structure and the logical operation used in the specified rule involving the participation constraints, i.e. OR or XOR on participants for defining the <em>fact</em> for the respective signature for the participant.</td>
</tr>
<tr>
<td>generateFactParticipant</td>
<td>Determining the structure and the logical operation used in the specified rule involving the messages exchanged (event) constraints, i.e. OR or XOR on events for defining the <em>fact</em> for the respective signature for the event.</td>
</tr>
</tbody>
</table>

Table 5.1: Participant services and associated sets of events

The implementation of the tool encompasses several exact steps for the transformation between the SBVR and Alloy models. The translation is a two stage process, with each stage involving multiple steps which need to be combined for generating Alloy code.

5.2.1 SBVR2Alloy - the transformation between the SBVR and the Alloy models

The stages that involve for executing the transformation from the SBVR model into the Alloy model comprise two stages. As shown in Figure 5.2, the first stage involves a transformation of terms and fact types in the SBVR model capturing the specifications on the participant set, the event set, the messages and the domain-specific constraints (as described in Section 4.2 and Section 4.3). These specifications are translated as the abstract signatures and signatures. It is then followed by populating the respective export and import messages (the local behaviour model) including the time association (and a time declaration) and the domain-specific constraints (static constraints) associating with each signature for participant and event.

The second stage comprises a process for transforming all related constraints which are specified using the SBVR rules (Section 4.4) and for generating and verifying the generated choreography (Section 4.5). This transformation involves the generation of possible facts, predicates, as well as fields associating with the
CHAPTER 5. SBVR2ALLOY: AN SBVR TO ALLOY COMPILER

Figure 5.2: Stages process for the transformation between the SBVR and Alloy model in implementing the SBVR2Alloy tool

generated signatures.

**Stage 1**

The first stage encompasses a generation of Alloy codes by reading in the input SBVR file. The first step of this stage involves parsing the SBVR ‘Term’ and ‘Fact Type’ keywords and generating the corresponding Alloy structures, namely the participant and event signatures. Let us say, the following SBVR model is a part of specification using the defined template.

---

–Participant set
–Add your participant terms here:
  Term: participant$_1$
  Term: participant$_a$
  ...
–Constraint
–Add your static constraint terms here:
  Term: date$_1$
  ...
–Add your participant fact types here:
  Fact Type: participant$_1$ includes participant$_a$
  Fact Type: participant$_1$ verb event$_1$ at T$_1$
  ....
–Add your ordering of time fact types here:
  Fact Type: event$_1$ at T$_1$ immediately precedes event$_1$ at T$_2$
  ....
–Add your static constraints fact types here (if any):
  –A set of date:
  Fact Type: date$_1$ is in date$_1$
As mentioned previously, the prefixed comments in the template is a basis of implementing the tool to parse the template and to change its state. After taking the SBVR input as specified above, when the tool reaches the ’–Participant set’ comment it changes its state to participant (i.e., 'p'). The tool then arrives at the specification which starts with the ‘Term:’ keyword. At this stage, it stores the defined participant term into the array list, namely the participant list. When the tool finds the ’–Add your participant fact types here:' comment it changes to another state as the fact type for participants. All the stored participant terms previously are added into Alloy once they are encountered after the 'Add your participant fact types here:' comment. If there is a fact type contains a verb includes, the participant term that assigned before includes will be translated as the abstract signature and the extension of the participant’s signature. This can be seen in the following code snippet (line 1). Furthermore, the participant term after includes is declared as the subsignature for the participant_1 (line 2). Note that the function of checkSets is employed in this stage to check whether the particular participant is part of a superset and modify the alloy translation accordingly. The same methodology applies for generating the abstract signatures and signatures for events.

```
1 abstract sig participant_1 extends participant{verb: event_1}
2 sig participant_a extends participant_1{}
3 one sig ...... extends participant{} ....
4 one sig event_1 extends event{at: one t1_event_1} ....
5 abstract sig event_1_time extends time{}
6 one sig t1_event_1 extends event_1_time {by: one participant_1, immediately_precedes: one t2_event_1}
7 one sig t2_event_1 extends event_1_time {by: one ......}
.....
8 one sig date_1 in date{}
```

The messages exchange (the export and the import message) and the timing information regarding participants and associated events is also important; throughout this stage when a participant 'Fact Type' includes any verb declaration associating with the particular event such as event_1 (as described in the given template), that verb is allocated in the corresponding participant’s signature (line 1). At this stage, that verb (the inheritance information) is also stored in memory. However, the assignation of cardinality is hold for a while. The appropriate cardinality is either 'lone' or 'one' is chosen (according to the related specification in the rule part)
as checked by the checkRules function. This task is done by the checkRules function. This approach is also applied for translating the static constraints (domain-specific constraints) belonged to participants and events as well as the ordering of events fact types.

Additionally, when a participant 'Fact Type' includes at keyword (as shown in the same fact type used in the previous example), the time and the event association is saved in memory to generate the corresponding time declarations in Alloy during stage 2. For instance, when the tool reaches at keyword, the association term event1 and the time $T_1$ will be stored in memory. After the event1’s signature is generated, the keyword at is assigned as its field (line 4) associating with the corresponding time and event, i.e. $t_1.event1$ is declared. This functionality is supported by the function of addEventTimedSignature. In parallel, the .time postfix is used to define the time declarations that associate with events. That event1 term is then be kept track and the .time is appended to. This is used for the time declaration in Alloy as defined in lines 5-7.

Note that in order to comply with Alloy, an event that takes place at time $t_1$ must always precede an event that takes place at $t_2$. In addition, we have followed the same approach to dealing with the word verb, e.g., immediately precedes in the SBVR model (as defined in the given fact type) is interpreted as immediately_precedes (line 2 in the above code snippet) in the Alloy model.

The checkUtils is responsible for this implementation.

Stage 2

This stage completes the translation process and involves more complex steps that rely on the Alloy code we have generated in the first stage.

The translation concerns on the rules specifying the ordering between events. It is related to the function of checkOrdering. In this stage, the identification of the quantification (with or without the includes verb) and the logical operation specified over the event terms are essential. For instance, assume the following rule specifies OR on events immediate after the execution of the other event(s). Since both immediately_precedes verb and or logical operation in the rule are employed, the field immediately_precedes together with lone cardinality associating with each specified event are defined as shown in line 9. Similarly, if but not both or but not all include in the rule, one cardinality is populated in the signature for the respective event.
Rule: It is obligatory that ... *immediately precedes* exactly one \( \text{event}_2 \) or ... or exactly one \( \text{event}_N \)

\[
9 \ldots \quad \text{sig} \ldots \quad \text{extends} \quad \text{event}\{ \text{immediately\_precedes}_1: \quad \text{lone} \quad \text{event}_2, \ldots, \\
\text{immediately\_precedes}_2: \quad \text{lone} \quad \text{event}_N \}
\]
\[
10 \{ \#\text{immediately\_precedes}_1 = 1 \quad \text{or} \quad \#\text{immediately\_precedes}_2 = 1 \}
\]

Subsequently, we append the Alloy facts for each participant and event by consulting both the SBVR rules and the verbs in the signatures we have generated. The inheritance information, captured during the first stage, also needs to be utilised to generate the facts. For this case, both the *generateFactParticipant* and the *generateFactEvent* functions play an important role to ensure the correct cardinality associating with the signature is defined and the appropriate fact is declared, in Alloy. The tool will parse the rule for checking whether or not the rule contains the *immediately precedes* verb, if it is found the *generateFactEvent* generates the fact for event’s signature. Otherwise, the *generateFactParticipant* function generates the fact for participant’s signature. In terms of cardinality, if the rule contains both *or* and ...*but not both* (*all*) logical operations, the fact is generated. Besides, the verb *includes* is also checked to ensure the cardinality for signature.

The following rule defines XOR on participants because ..*but not all* is included in the rule. In Alloy, the signature for \( \text{participant}_1 \) is ensured to be defined as the abstract (line 11). Then, the accompanying fact (lines 12-13) is specified to ensure only one or the other of the specified participant sends (receives) the messages exchange of the particular event(s) at any one execution.

Rule: It is obligatory that the \( \text{participant}_1 \) that *includes* the \( \text{participant}_a \) or the \( \text{participant}_b \) or ... or the \( \text{participant}_n \) but not all, *verb* ....

\[
11 \quad \text{abstract} \quad \text{sig} \quad \text{participant}_1 \quad \text{extends} \quad \text{participant}\{ \text{immediately\_precedes}: \quad \ldots \}
\]
\[
12 \quad \{ (\text{participant}_1 = \text{participant}_a \quad \text{and} \quad \text{no} \quad \text{participant}_b \quad \ldots \quad \text{and} \quad \text{no} \quad \text{participant}_n) \\
13 \quad \ldots \quad \text{or} \quad \{ (\text{participant}_1 = \text{participant}_n \quad \text{and} \quad \text{no} \quad \text{participant}_a \quad \ldots \quad \text{and} \quad \text{no} \quad \text{participant}_b) \} \}
\]

Next, when the tool parse ‘*Add static constraints rules here (if any):’* it changes to another state for generating the predicate and the fact for date constraints. For this purpose, the specific verb that used for specifying the date constraints are identified. On one hand, the tool will examine whether the rule contains *equals* verb associating with the *obligatory* or the *prohibited* modalities. The association be-
between obligatory and equals (as specified in the following rule) indicates the equality date constraint. Hence the fact as in line 15 is assigned. Otherwise, the inequality date constraint is specified and the symbol ! = instead of = (line 15) is defined. On the other hand, the tool will parse the rule and check whether begins before verb or is between verb are employed in the rule. It is then followed by identifying the specified events in the rule as well as the initialisation for date to ensure the corresponding pred orderdate and the fact traces (for ordering date purposes) are generated.

- Add static constraints rules here (if any): Rule: It is obligatory that exactly one date₁ of exactly one event₁ equals exactly one date₂ of exactly one event₂

```
14 one sig event₁ extends event{verb: one date₁}
15 {verb = event₂.verb}
16 one sig event₂ extends event{verb: one date₂}
```

After appending all the information in the Alloy signatures, the next step is to finalise the alloy code by removing excess characters (', ', '{', '}'). The alloy code is then used to generate the service choreography by executing the output Alloy model (the .als file) in the Alloy Analyzer tool. Again this step includes the Alloy signature of the initial event and defines the interactions between associated events.

Finally, concrete requests are identified and translated to Alloy using a combination of the generated Alloy signatures, the SBVR rules and request-specific rules, which can be also defined in the SBVR model.

### 5.2.2 SBVR2Alloy: illustration by example

#### Alloy code generation

The input to SBVR2Alloy is a plain text version of the SBVR model. The output is the Alloy file (.als) that can be parsed by the Alloy Analyzer tool [46]. The input plain text file is uploaded from the local drive and is then shown in the pane on the left of the SBVR2Alloy graphical user interface, shown in Figure 5.3.

The input can be transformed to Alloy code now by pressing the ‘Convert’ button on the top right. The result is shown in Figure 5.4. The Alloy code can be exported locally as .als file which can then be opened with the Alloy Analyzer.

In this way the declarative specification of the multi-party conversation, including constraints on the message ordering, which was given as an SBVR model is now manifested in the resulting Alloy code. This means the Alloy Analyzer can be readily used to find structures that satisfy these constraints, effectively generating the underlying service choreography.
5.2. **IMPLEMENTATION OF SBVR2ALLOY**

Figure 5.3: SBVR model input uploaded as a text file

Figure 5.4: Alloy code (right pane) produced for the SBVR model (left)
5.3 Failure Model and Testing

Previously, the development of the SBVR model focuses on the successful model without having any error in the rules structure. In this section, we consider the SBVR model causing failure.

5.3.1 Syntax checking

The SBVR2Alloy tool does not provide a syntax error checking. Although the template is created for building the SBVR models, the SBVR2Alloy is unable to let the user know whether the developed rules follow the correct structure of the rules as described in Chapter 3. For example, in Chapter 3, we have a rule to describe OR on participants sending (receiving) the particular event, e.g. It is obligatory that the participant\textsubscript{1} that includes the participant\textsubscript{a} or the participant\textsubscript{b} verb exactly one event\textsubscript{1}. In case the user forgets to specify the logical operator "or", the SBVR2Alloy will fail to transform the SBVR model into Alloy. In order to capture the syntax error, as discussed in Chapter 6, this will be a future work where the SBVR2Alloy is aimed to be integrated with the SBVR editor from rulemotion found at "sbvr.co" as this editor is able to detect a syntax error by highlighting the font of the specification.

5.3.2 Failure model

There is a number of ways in which the user can provide incorrect input. We cannot address all kinds of failures that might appear in ad-hoc user request. An illustration is given by an example cases that might cause a failure in the model. This should give an idea of how a wider range of cases can be addressed.

Case 1

The following rule shows two verbs sends receives that misleads the rule in the model.

Rule:

It is obligatory that the participant\textsubscript{1} sends receives exactly one event\textsubscript{1}

Technically, the transformation of this rule into Alloy can be done and the Alloy Analyzer can execute to produce an instance. This is due to the use of parsing the SBVR Term and Fact Type keywords during the transformation and the generation of the corresponding Alloy structures. For the case of the above rule, once the tool reaches the cardinality exactly one, it will recognise the term before that cardinality term (sends receives) as the verb even the term consists of two or more verbs. The transformation of the above rule into Alloy can be seen as follows:

\begin{verbatim}
1 abstract sig participant {}
2 one sig participant_1 extends participant{sends_receives: one event_1}
\end{verbatim}
One possible way to avoid this type of failure is to create a list of stop words which draws from the field of information retrieval. In short, the tool will accept verb with a white space unless there are included in the stop list, e.g. bring-on, come-on, etc, otherwise the tool will give a message to the user the specified verb in that rule is specified wrongly.

Case 2

The similar case as in Case 1 is shown in the following rule. However, the unintended verb is specified at the end of the rule.

Rule:

It is obligatory that the \textit{participant}_1 \textit{verb} exactly one \textit{event}_1 \textit{verb} 1.

For this case, the transformation into Alloy will throw exception. According to the SBVR model structure, after another verb must be another term. The compiler need alerts the user about the correct structure of the developed rule by giving an error message, for example, the message e.g. another term must followed by the \textit{verb} 1 is given for the above rule. The transformation of the above rule into Alloy is shown as follows:

\begin{verbatim}
1 abstract sig participant ()
2 one sig participant_1 extends participant{verb: one event_1}
3 abstract sig event()
4 one sig event_1 extends event{verb_1: ??}
\end{verbatim}

The Alloy Analyzer fails to generate an instance for the above codes since the field "verb_1" in line 4 must be mapped to another signature that is another term in SBVR.

Case 3

In this case, there is a term after \textit{verb} 1, however this structure does not exist in developing the SBVR model.

Rule:

It is obligatory that the \textit{participant}_1 \textit{verb} exactly one \textit{event}_1 \textit{verb} 1 exactly one \textit{event}_1.

The transformation into Alloy will throw exception. As described in Chapter 3, the SBVR model allows only the event term to be associated with the time interval
term. The compiler need alerts the user by giving the correct structure of that rule. The transformation of the given rule into Alloy is shown as in the following:

```
1 abstract sig participant {} 
2 one sig participant_1 extends participant{verb: one event_1} 
3 abstract sig event{} 
4 one sig event_1 extends event{verb: one event_2} 
```

### Case 4

There is a case when no rule is specified showing which participant sending or receiving the particular event. Assume that a rule representing the participant receiving the `event_1` is not provided in the SBVR model. The following rule describes the `event_1` is sent by the `participant_1`.

**Rule:**

It is obligatory that the `participant_1` sends exactly one `event_1`.

The transformation of the given rule into Alloy is shown as in the following:

```
1 abstract sig participant {} 
2 one sig participant_1 extends participant{sends: one event_1} 
```

In this case, the transformation into Alloy model can be implemented as well as the instance can be produced in Alloy Analyzer despite the fact that there is no rule indicates which participant receiving the event 1. One of the possible solution for this case is the development of the assertion in Alloy to ensure the `event_1` must be sent and received by any different participants. This enables the `SBVR2Alloy` compiler identifies which participants are sent and received the `event_1`.

The following snippets illustrate the required assertion for checking the event must be sent and received by the intended participants.

```
3 assert event_1{all p1:participant_1, e:event_1, p2:?? | 
4 p1.sends = e and p2.receives= e} 
5 check event_1 
```

The code in line 4 defines the `event_1` should be sent and received by `p1` and `p2`, respectively. Since a rule expressing the `event_1` is received by the particular
participant is not specified in the model, the Alloy Analyzer will give a message, ”p2 (which is another participant) cannot be found”.

**Case 5**

Oppositely, this case represents both rules which express a sending and a receiving of an event. This is illustrated as in the following.

**Rule 1:**
It is obligatory that the \textit{participant}_1 \textit{sends} exactly one \textit{event}_1.

**Rule 2:**
It is obligatory that the \textit{participant}_1 \textit{receives} exactly one \textit{event}_1.

The transformation of the given rules into Alloy are depicted as follows:

```alloy
1 abstract sig participant {}
2 one sig participant_1 extends participant {sends: one event_1, receives: one event_1}
3 abstract sig event {}
4 one sig event_1 extends event{}
```

However, **Rule 2** expresses the same participant (\textit{participant}_1) as specified in **Rule 1** receives the same event \textit{event}_1. This is impossible in the choreography which leads to produce an inaccurate choreography model.

The possible way to refrain from this type of failure, the assertion in Alloy is created to guarantee the intended event is sent and received by distinct participants. The assertion for this kind of purpose is shown as follows.

```alloy
5 assert event_1 {all p1:participant_1, e:event_1, p2:participant_1 | p1 != p2}
6 p1.sends = e and p2.receives= e}
7 check event_1
```

For this case, Alloy must be able to ensure both participants p1 and p2 are from different signatures of participant that associate with the same event \textit{e}. Since the rule expresses and the code snippet in line 2 defines the \textit{event}_1 is received by the \textit{participant}_1, the message to alert the user where both participants are the same participant will be given.

**Case 6**

The other possible case is related to the rule expressing the ordering of the events where one event comes before the same event. The example of rule for this case can
be seen as in the following.

Rule:
It is obligatory that the \( \text{event}_1 \) immediately precedes exactly one \( \text{event}_1 \).

The above rule expresses the same event \( \text{event}_1 \) occurs one after another which is not allowed in the structure for developing the SBVR model. Hence, the transformation for this kind of rule is impossible to be executed.

### 5.3.3 Testability Matrix

Table 5.2: Testability Matrix

<table>
<thead>
<tr>
<th>Test Number</th>
<th>Requirement No</th>
<th>Preconditions/Dependencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>F1</td>
<td>Display the template upon loading.</td>
</tr>
<tr>
<td>2</td>
<td>F1</td>
<td>Upload an SBVR model.</td>
</tr>
<tr>
<td>3</td>
<td>F2</td>
<td>Convert the SBVR model into Alloy model.</td>
</tr>
<tr>
<td>4</td>
<td>F3</td>
<td>Save the Alloy model as .als file.</td>
</tr>
<tr>
<td>5</td>
<td>F4</td>
<td>Upload .als file in Alloy Analyzer.</td>
</tr>
<tr>
<td>6</td>
<td>F5</td>
<td>Execute the Alloy model.</td>
</tr>
<tr>
<td>7</td>
<td>F6</td>
<td>Show the model.</td>
</tr>
<tr>
<td>8</td>
<td>F7</td>
<td>Verify the model.</td>
</tr>
</tbody>
</table>

Table 5.3: Testability Matrix

<table>
<thead>
<tr>
<th>Test Number</th>
<th>Expected Inputs</th>
<th>Expected Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
<td>The SBVR2Alloy template.</td>
</tr>
<tr>
<td>2</td>
<td>An SBVR model file (.txt file). Click the button on the left side.</td>
<td>The SBVR model on the left pane.</td>
</tr>
<tr>
<td>3</td>
<td>Click the button on the right side.</td>
<td>The Alloy model on the right pane.</td>
</tr>
<tr>
<td>4</td>
<td>File is saved as .als file.</td>
<td>File can be opened in Alloy Analyzer.</td>
</tr>
<tr>
<td>5</td>
<td>Open .als file.</td>
<td>The Alloy model on the left pane.</td>
</tr>
<tr>
<td>6</td>
<td>Predicate ”initialevent” is executed. Click the ”Execute” button on the top.</td>
<td>The predicate is consistent and the instance found.</td>
</tr>
<tr>
<td>7</td>
<td>Click the instance to see the model.</td>
<td>The Alloy model which corresponds to the SBVR model.</td>
</tr>
<tr>
<td>8</td>
<td>Predicate ”specificrequest” is executed. Click the ”Execute” menu and run ”specificrequest”.</td>
<td>The Alloy model satisfying the specific request.</td>
</tr>
</tbody>
</table>
### Table 5.4: Testability Matrix

<table>
<thead>
<tr>
<th>Test Number</th>
<th>Actual Output</th>
<th>Test Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Refer to Figure 5.5</td>
<td>Passed/Failed.</td>
</tr>
<tr>
<td>2.</td>
<td>Refer to Figure 5.6 and Figure 5.7</td>
<td>Passed/Failed.</td>
</tr>
<tr>
<td>3.</td>
<td>Refer to Figure 5.8</td>
<td>Passed/Failed.</td>
</tr>
<tr>
<td>4.</td>
<td>Refer to Figure 5.9</td>
<td>Passed/Failed.</td>
</tr>
<tr>
<td>5.</td>
<td>Refer to Figure 5.10 and Figure 5.11</td>
<td>Passed/Failed.</td>
</tr>
<tr>
<td>6.</td>
<td>Refer to Figure 5.12</td>
<td>Passed/Failed.</td>
</tr>
<tr>
<td>7.</td>
<td>Refer to Figure 5.13</td>
<td>Passed/Failed.</td>
</tr>
<tr>
<td>8.</td>
<td>Refer to Figure 5.14 and Figure 5.15</td>
<td>Passed/Failed.</td>
</tr>
</tbody>
</table>

![SBVR2Alloy Converter](image)

**Figure 5.5:** The SBVR2Alloy template
CHAPTER 5. SBVR2ALLOY: AN SBVR TO ALLOY COMPILER

Figure 5.6: Uploading a .txt file

Figure 5.7: Display SBVR model
5.3. FAILURE MODEL AND TESTING

Figure 5.8: Convert the SBVR model

Figure 5.9: Save as .als file
CHAPTER 5. SBVR2ALLOY: AN SBVR TO ALLOY COMPILER

Figure 5.10: Open .als file in Alloy Analyzer

Figure 5.11: The Alloy model in Alloy Analyzer

Figure 5.12: Execute the Alloy model
5.3. FAILURE MODEL AND TESTING

Figure 5.13: An instance of Alloy model

Figure 5.14: Verify the Alloy model

Figure 5.15: An instance of the verified Alloy model
5.4 Challenges

The difficulty of translating semantic languages became apparent in the early stages of our analysis and design. A concrete example is the usage of the 'Term' keyword in SBVR, which it is used to define participants, events and constraints (both static such as those on dates discussed earlier, and dynamic such as those relating to message ordering). These concepts carry a distinct meaning in a multiparty conversation and consequently their handling in Alloy is not uniform.

Common use of SBVR models involves using comments to add structure and increase usability as well as readability. Creating a universal SBVR to Alloy compiler would involve taking into account an infinite number of comments. Instead, we have created a template for building SBVR models that should be followed when using our **SBVR2Alloy** tool. This template serves to identify the different sections in the SBVR model for a more structured approach to translation.

We have found that Alloy is more restrictive than SBVR in a number of ways. For example, SBVR allows for terms comprising multiple words whereas Alloy only allows for single word terms. To overcome this we replaced the white spaces between words in multi-word terms with underscores, e.g., the term 'event 1' in the SBVR model is translated to 'event_1' in Alloy.

In terms of the user input which is the SBVR model developing by the user, we cannot expect the full range of mistake that the user can make. For this case, we have addressed certain failures in Section 5.3 or a few type of mistakes that might be done by the users.

Another challenge we faced in the translation process had to do with the use of the same verb in different relationships involving the same term. SBVR can reuse the same verb for a relationship between two terms, whereas Alloy allows for a verb to be used only once for each term. For example, the definition for the transport reservation in our ACME Travel case study uses the verb 'has' twice as seen below:

**Fact Type:** event includes transport reservation
**Fact Type:** transport reservation has departure date
**Fact Type:** transport reservation has arrival date

In addition, the synonyms or the connectives such as that, at, of - these may not be used consistently by the end-user, hence are a bit more cumbersome to handle in the automated transformation.

For the corresponding Alloy signature to be valid, we have added numbers to the end of each field capturing the 'has' verb, resulting in the following signature:

17 abstract sig transport_reservation extends event{ has: one departure_date, has1: one arrival_date }
increased the complexity of the code as it resulted in an additional dimension in the data structure and one more parsing of related constructs in the SBVR model. This was illustrated in our ACME Travel case study, whenever the precedence between two main events are related with the nested participants as shown in the following rules in the SBVR model.

**Rule 1:** It is obligatory that exactly one *accommodation reservation* that includes exactly one *hotel reservation* or exactly one *apartment reservation* but not both immediately precedes exactly one *accommodation response* that includes exactly one *successful accommodation reservation* or exactly one *unsuccessful accommodation reservation* but not both.

**Rule 2:** It is obligatory that exactly one *accommodation* that includes exactly one *hotel* or exactly one *apartment* but not both sends exactly one *accommodation response* that includes exactly one *successful accommodation reservation* or exactly one *unsuccessful accommodation reservation* but not both, at exactly one $T_1$.

Rule 1 describes the *accommodation reservation* event (XOR of apartment reservation and hotel reservation) must occurs immediately before the *accommodation response* event (which is either successful or unsuccessful reservation response). And note that the *accommodation response* event must be sent by either one of the nested accommodation participant (hotel or apartment) (Rule 2). In order to guarantee the Alloy model generates the corresponding SBVR model, the codes snippet in lines 18-24 are defined. More specifically, line 22 plays an important task to ensure the precedence between *accomreservation* and *accomresponse* inherits what is sent by the nested of accommodation (either hotel or apartment sends the response).

```alloy
18 abstract sig accommodation extends participant {sends: one accomresponse}
{...}
19 lone sig hotel extends accommodation{...}
20 lone sig apartment extends accommodation{...}
...
21 abstract sig accomreservation extends event {immediatelyprecedes: one accomresponse}
22 {... and (immediatelyprecedes = hotel.sends or immediatelyprecedes = apartment.sends)}
23 lone sig hotelreservation extends accomreservation{}
24 lone sig apartmentreservation extends accomreservation{}
```
However, the Alloy language requires this type of indirection (or inheritance in SBVR) in order to distinguish between inclusive (OR) and exclusive disjunction.

5.5 Concluding remarks

The SBVR2Alloy tool is implemented to enable the transformation between the SBVR and the Alloy models by taking the SBVR model which is written following the defined template in Chapter 3, and then producing the Alloy file (.als file) which is readily executed in Alloy Analyzer. The developed tool covers all different types of rules in, specifically, the combinations of logical operations for specifying the complex interactions and the precedence between the messages exchanged of events in multi-party conversations and in other SBVR rule constructs such as the specification involving the static constraints.

Our SBVR2Alloy tool does not implement the full breadth of the SBVR standard specification [7] but rather a large and usable subset which can be used to express complex rules, with a focus on capturing constraints on the orderings of service invocations. Also, the SBVR2Alloy tool is unable to predict all the possible specific requests towards producing a predicate that enables the verification such as realisability. This is because the generating of such predicate depends on the specific rule specified by the user. Hence, the enhancement of the tool to enable the automatic transformation of the specific request SBVR rule into a predicate concreterequest (Section 4.6.1) in Alloy, is one of the future works.

As mentioned before, in order to use the SBVR2Alloy tool, the template for building SBVR models must be followed. It means that the tool compiler was assigned certain keywords to structure the translation. For example, in the template (Table 3.1), there is a comment \(--Add your static constraint terms here:\), where the keyword static was assigned to identify the static constraints specified by the user. For this reason, if the Term static_1 is specified by the user in this section, the tool will produce an error in which the automatic transformation between the SBVR model and Alloy model could not be done.
6.1 Main Achievements

The main contents in this thesis present the achievement that have been done according to an addressed objectives in Chapter 1. The details of the achievement are described as follows:

- **Specification** - A new model for modelling Service Choreographies using a declarative approach, namely an *Semantics of Business Vocabulary and Rules* (SBVR) model for specifying service choreographies, is introduced. This new choreography model advocates the OMG standard SBVR and its supplement DTV. The main purpose of proposing such a model is to develop the choreography model in a declarative manner which provides a way to express complex business requirements in natural language. The complex business requirements are the global constraints describing the service interactions and include allowing the ordering of messages exchange which happens either in concurrent or alternative interactions. At the same time, the model allows the specification of local behaviour of each participant service, which includes the domain-specific constraints. Since the SBVR language is represented in natural language, users can validate the requested service composition by directly reading the structured natural language used by SBVR. This can then be parsed and executed by a machine. In addition, the SBVR model allows the flexibility at the specification level for specifying the collaboration, e.g., replace of a service; changing the local behaviour for a participant service.

- **The formulation of the SBVR model for generating and verifying the service choreography** - A transformation from SBVR model into Alloy model using Alloy Analyzer is introduced. This transformation encapsulates the specification of the SBVR model which is aimed to generate the choreography model from the developed SBVR model automatically that conforms to
the global constraints and allows the verification of specific request against the generated choreography as well as checking certain type of static constraints. This permits the realisability verification.

- **SBVR2Alloy tool** - A fully automated tool is implemented for transforming from the developed SBVR model into the Alloy model. This tool takes the SBVR model input before translating it into the Alloy codes automatically by generating the Alloy (.als) file. Subsequently this file is used in Alloy Analyzer for analysis purposes.

These three work achievements have been represented into the manuscript which was submitted to a Journal of IEEE Transactions on Services Computing [69].

### 6.2 Summary

The proposed SBVR model is introduced to coordinate service composition. This is done by describing the conversation between them using the proposed structure of SBVR rules from a global perspective. The proposed SBVR model does not only handle the dependencies on interactions (e.g., a given service invocation must occur before another one) well, but it is able to support alternative and concurrent interactions.

The specification of proposed SBVR model uses OMG standard SBVR which is a declarative language approach. The specification focuses on *what* the properties are that a solution must have rather than *how* the steps followed to achieve the solution. This enables flexibility at the specification level, where any changes on the local behaviour for a participant service (their own business processes) or replacement of a participant service, will not affect the whole choreography model.

The proposed SBVR model provides notions for controlling and defining the flow dependencies in the collaboration between different participant services. Two notions were introduced, *time* and *immediately precedes* notions to capture the global behaviour, that is a set of SBVR rules that govern the ordering of exchanged messages as agreed between the interacting participants.

The main objective of this PhD research was to produce a model that can generate the choreography model corresponding to the developed SBVR model. This is done by formulation and transformation the SBVR model into Alloy model. In addition to verifying conformance to message ordering constraints, the generated choreography model can be verified and checked in terms of realisability as well as the static constraints particularly on date constraints. The transformation between the SBVR and Alloy model can be done automatically using the developed tool, *SBVR2Alloy*.

Finally, the significance of this research is targeted at the end-users such as business analyst and system analyst (non-experts), where they are able to devise a composition of business models as a composition of services using the proposed
In terms of the specification for describing service choreographies using the SBVR model, focusing is on a deontic rule expressing obligation. However, we have seen in date constraints, a deontic rule specifically for asserting the inequality of two dates can also be expressed in the form of prohibition. For example, Rule: "It is prohibited that exactly one check-in date of exactly one accommodation reservation equals exactly one check-out date of exactly one accommodation reservation". Incorporating such prohibited behaviours in the specification by the SBVR model and subsequently in the verification afforded by Alloy is one possible future extension of this research.

As far as we are concerned, the SBVR model could capture the quantification for at least $n$ and at most $n$, this probably will benefit for the additional features in describing the choreography model. However, as mentioned, Alloy could not capture this kind of quantifications, therefore another future works to do on exploring the Alloy model or another constraints solver for encapsulating this kind of feature.

Our SBVR2Alloy tool does not implement the full breadth of the SBVR standard specification but rather a large and usable subset which can be used to express complex rules, with a focus on capturing constraints on the orderings of service invocations. The tool can be extended to include less common features of SBVR and indeed this is part of the future work planned.

Immediate future work includes integration with the SBVR editor found at http://www.sbvr.co so the SBVR model is written in the browser and then integration at the other end with Alloy to try and execute the model within the same webpage. This paves the way for playing with the output of Alloy, the graphical output which can be tailored further for exploring the behaviours in a service choreography.

Although the SBVR2Alloy tool was developed, another future work concerns on the formal semantics for the transformation (useful in order to prove correctness) from the SBVR model into Alloy model. The possible mechanism for mapping correctness is adopted global graphs and communicating finite-state machines (CF-SMs) [70]. The global graph is applied to formalise global views of choreographies, by generating a graphical model which captures the distributed work-flow. On the other hand, CFSMs are used to formalise local views of choreographies and well-known as an established model for communication protocol design. Hence, the investigation for both global graph and CFSMs will be studied. The idea for mapping correctness is to compare the generated instance of choreography model from
the corresponding SBVR model with the generated global views which is make of CFSMs translating from the SBVR model.

Explore and investigate the correlations between the proposed method for service choreography, Multi-Agent Systems (MAS), and Clinical Guidelines (CGs), and subsequently to apply MAS and CGs into the proposed SBVR model are another future work. Basically, service choreographies and MAS have similarities in terms of the interacting entities, the communication via messages exchange, as well as the local and global view interaction [19]. The noticed different is the interacting entities in MAS, namely an agent can find a solution that is required to do for itself independently for achieving its design objectives, without having to be told explicitly what is required to do at a given time [19]. Another possible realm is CGs. For applying the SBVR model into MAS and CGs, extra factors need to be taken into consideration, for example by considering all deontic operators, obligation, permission, and prohibition in the model. However, as the SBVR model could describe a set of rules that govern the ordering of messages exchange as agreed between the interacting participants, it is possible to the SBVR model for systematically developed statements or guidelines to assist practitioner and patient decisions about appropriate health care for specific clinical circumstances.
Appendices
XX
s
Appendix A

The SBVR Model for the
Online Photo Shop case study

This case study is the first case study that we conducted, which is adopted from [6]. The case study concerns on an online photo shop which enables a customer (individual or other shops) to place orders through an online application, which is provided by the Online Photo Shop. Our online photo shop case study is amended where a new participant is added, which is a bank to show multi-party conversations (more than two participants). The interaction respects the policies underlining the business activities involved, as described in [6]. However, we notice that there are certain problems with the specification of the Online Photo Shop as given in [6], such as ambiguities in defining an activity (e.g., open order and register) while some prescribed orderings on activities are questionable (e.g., customer may pay before or after Photo Shop performs charge). We have amended the specification slightly to steer away from such issues. This will allow us to focus on using the SBVR model to specify the choreography rather than elaborating the specification itself. Below is the description of responsibilities by the participants (web services) Customer, Photo-Shop, and Bank.

**Photo Shop** records the customer’s account for keeping the customer’s personal details. It allows the customer to make a request for ordering at least one of its products, namely photo, poster, and album. All the ordered products is printed by the photo shop. All paid products will be delivered to the customer.

**Customer** register an account at photo shop by entering data, such as name, address, and credit card number. The customer is allowed to make an order for at least one product and is received the ordered product once the payment has been made.
Bank receives the product payment by the customer and notifies the payment to both, the customer and the photo shop.

The following description elaborates the informal constraints of choreography for the online photo shop.

1. The choreography begins as soon as the customer registers an account at the online photo shop with giving some personal details. Once the registration is made, the shop will update its data via recording the registration.

2. The customer proceeds the activity with making an order. The customer is allowed to make a request for ordering at least one of product’s types: album, poster, and photo, simultaneously. Right after the customer orders the product(s), the photo shop will receive the ordered product by the respective customer.

3. It is not permitted that the execution of the requesting of product(s) by the customer and the receiving of the requested product by the photo shop occurs before the execution of the registration and the update of an account by the customer and the photo shop, respectively.

4. The customer must make a payment once the product(s) is ordered. The bank will receive the respective payment once the payment has been made by the customer.

5. The bank sends the notification of the payment, which is either successful or unsuccessful to the customer as well as the photo shop. Both, the customer and the photo shop must receive the respective notification.

6. The activity of the payment is made by the customer and is received by the bank must be occurred immediate before the notification of the payment is sent by the bank and is received by both, the customer and the photo shop.

7. Each ordered product has to be delivered by the photo shop. The customer receives the ordered product(s) only after the photo shop delivered them.

8. Once the successful notification of the payment has been sent by the bank and been received by the customer and the photo shop, the ordered must be delivered to the respective customer.

The SBVR model for the online photo shop case study has been developed according to the amended choreography contract. The details of the SBVR model which describes the global constraints of the online photo shop case study can be referred as in the following figures.

Note that Figure A.9 shows the specific request to verify whether the request of exactly one photo request or exactly one album request (but not both) can be realised by the given choreography of the SBVR model.
--Participant Set

--Add participant terms here:

Term: participant
Term: customer
Term: photoshop
Term: bank

--Event Set

--Add event terms here:

Term: event
Term: account registration
Term: product request
Term: product delivery
Term: product payment
Term: payment notification
Term: successful notification
Term: unsuccessful notification
Term: photo request
Term: poster request

--Constraint

--Add your static constraints terms here:

Term: name
Term: address
Term: credit card number
Term: product

--Time

Term: t

Figure A.1: Terms in the SBVR model for the online photo shop
APPENDIX A. THE SBVR MODEL FOR THE ONLINE PHOTO SHOP CASE STUDY

--Fact Types

--Add your participant fact types here:

--Participant 1
Fact Type: participant includes customer
--Local behaviour of customer
--export message
Fact Type: customer makes account registration at t1
Fact Type: customer makes product request at t1
Fact Type: customer makes product payment at t1
--import message
Fact Type: customer receives product delivery at t2
Fact Type: customer receives payment notification at t2
--static constraints
Fact Type: customer has name
Fact Type: customer has address
Fact Type: customer has credit card number

Figure A.2: Fact types for specifying the participant set (the local behaviour for each participant) in the SBVR model for the online photo shop
--Participant 2

**Fact Type:** participant includes photoshop

--Local behaviour of photoshop

--import message

**Fact Type:** photoshop records account registration at t1

**Fact Type:** photoshop makes product delivery at t1

--export message

**Fact Type:** photoshop receives product request at t2

**Fact Type:** photoshop receives payment notification at t2

--static constraints

**Fact Type:** photoshop prints product

--Participant 3

**Fact Type:** participant includes bank

--Local behaviour of bank

--import message

**Fact Type:** bank receives product payment at t2

--export message

**Fact Type:** bank sends payment notification at t1

Figure A.3: Fact types for specifying the participant set (the local behaviour for each participant) in the SBVR model for the online photo shop
--Fact Types

--Add your event fact types here:

--Event 1

Fact Type: event includes account registration

--Initialisation

Fact Type: account registration is initial event

--Event 2

Fact Type: event includes product request

--Type of product request

Fact Type: product request includes photo request
Fact Type: product request includes poster request
Fact Type: product request includes album request

--Event 3

Fact Type: event includes product delivery

--Event 4

Fact Type: event includes product payment

--Event 5

Fact Type: event includes payment notification

--Type of payment notification

Fact Type: payment notification includes successful notification
Fact Type: payment notification includes unsuccessful notification

Figure A.4: Fact types for specifying the event set in the SBVR model for the online photo shop
--Add your ordering of time associates with same event here:

Fact Type: account registration at $t_1$ immediately precedes account registration at $t_2$

Fact Type: product request at $t_1$ immediately precedes product request at $t_2$

Fact Type: product payment at $t_1$ immediately precedes product payment at $t_2$

Fact Type: payment notification at $t_1$ immediately precedes payment notification at $t_2$

Fact Type: product delivery at $t_1$ immediately precedes product delivery at $t_2$

Figure A.5: Fact types for specifying the ordering of time associates with the same event in the SBVR model for the online photo shop

--Add your ordering of events fact types here:

Fact Type: account registration immediately precedes product request

Fact Type: product request immediately precedes product payment

Fact Type: product payment immediately precedes payment notification

Fact Type: payment notification immediately precedes product delivery

Figure A.6: Fact types for specifying the ordering of events in the SBVR model for the online photo shop
--Add you rule here:

**Rule:** It is obligatory that the customer makes exactly one account registration at exactly one \( t_1 \)

**Rule:** It is obligatory that the photoshop records exactly one account registration at exactly one \( t_2 \)

**Rule:** It is obligatory that exactly one account registration immediately precedes exactly one product request that includes exactly one photo request or exactly one poster request or exactly one album request

**Rule:** It is obligatory that the customer makes exactly one product request that includes exactly one photo request or exactly one poster request or exactly one album request, at exactly one \( t_1 \)

**Rule:** It is obligatory that the photoshop receives exactly one product request that includes exactly one photo request or exactly one poster request or exactly one album request, at exactly one \( t_1 \)

**Rule:** It is obligatory that exactly one product request that includes exactly one photo request or exactly one poster request or exactly one album request, immediately precedes exactly one product payment

**Rule:** It is obligatory that the customer makes exactly one product payment at exactly one \( t_1 \)

**Rule:** It is obligatory that the bank receives exactly one product payment at exactly one \( t_2 \)

**Rule:** It is obligatory that exactly one product payment immediately precedes exactly one payment notification that includes exactly one successful notification or exactly one unsuccessful notification but not both

Figure A.7: Rules for specifying the global constraints in the SBVR model for the online photo shop
Rule: It is obligatory that the bank sends exactly one payment notification that includes exactly one successful notification or exactly one unsuccessful notification but not both, at exactly one $t_1$.

Rule: It is obligatory that the customer receives exactly one payment notification that includes exactly one successful notification or exactly one unsuccessful notification but not both, at exactly one $t_2$.

Rule: It is obligatory that the photoshop receives exactly one payment notification that includes exactly one successful notification or exactly one unsuccessful notification but not both, at exactly one $t_2$.

Rule: It is obligatory that exactly one payment notification that includes exactly one successful notification immediately precedes exactly one product delivery.

Rule: It is obligatory that the photoshop makes exactly one product delivery at exactly one $t_1$.

Rule: It is obligatory that the customer receives exactly one product delivery at exactly one $t_2$.

Rule: It is obligatory that exactly one account registration at exactly one $t_1$ immediately precedes exactly one account registration at exactly one $t_2$.

Rule: It is obligatory that exactly one product payment at exactly one $t_1$ immediately precedes exactly one product payment at exactly one $t_2$.

Rule: It is obligatory that exactly one product request at exactly one $t_1$ immediately precedes exactly one product request at exactly one $t_2$.

Rule: It is obligatory that exactly one payment notification at exactly one $t_1$ immediately precedes exactly one payment notification at exactly one $t_2$.

Rule: It is obligatory that exactly one product delivery at exactly one $t_1$ immediately precedes exactly one product delivery at exactly one $t_2$.

Figure A.8: Rules for specifying the global constraints in the SBVR model for the online photo shop.
--Specific request:

Fact Type: customer makes product request at t1
Fact Type: product request includes photo request
Fact Type: product request includes album request

Rule: it is obligatory that the customer makes exactly one product request that includes exactly one photo request or exactly one album request but not both, at exactly one t1

----------------------------------------------------------------------------------------------------------------------------------

Figure A.9: Rules for specifying the specific request in the SBVR model for the online photo shop
Appendix B

The SBVR model for the
Tuition Fee System case study

The SBVR model for the tuition fee case study is implemented to consider the other kind of application describing the choreography. The regulation of the tuition fee of this case study is adapted from the regulation that has been used by University of Roehampton. The tuition fee regulation in our case study has a smaller scope which describes the tuition fee payment applies for the international students - a self-funding student. The interactions involve multi-party conversations comprises a self-funding student which separated into a student who requires a visa and a student who does not required a visa, an international office, and a finance department. Each of them is assumed to have a different web service. This case study is more complicated than our other case studies, as it has few nesting of group that need to be considered regardless a group of nesting for participant or event. The responsibility of each participant is illustrated as in the following:

**Self-funding student** is divided into two kind of groups: a student who requires a visa for study and a student who does not require visa for study. A self-funding student looking to undertake an undergraduate or taught postgraduate programme at the University will be required to pay a deposit to secure a place on the course and to obtain a Confirmation of Acceptance letter. The student will receive the confirmation acceptance letter from the international office. It is then followed by the receiving of the notification from the international office as well. If the application for undertaking the programme is failed, the respective student is allowed to request a deposit refund. Otherwise, if the application is succeed, the respective student is required to make a tuition fee payment.
**International office** sends the confirmation acceptance and the notification to the intended student.

**Finance department** receives the deposit payment that has been made by the customer. If the application of the intended student is failed, the finance department will receive a deposit refund request by the respective student. Otherwise, the finance department will receive the fee payment.

The informal constraints describing the choreography of the tuition fee payment system is illustrated as in the following a contractual version of the choreography.

1. The choreography begins as soon as the self-funding student regardless from which kind of groups is started makes a deposit payment. The finance department must receive the payment once the deposit has been paid by the respective student.

2. The self-funding student will receive the confirmation of acceptance of the deposit payment once after the confirmation of acceptance is sent by the international office.

3. The occurrence of deposit payment that has been made and received by the self-funding student and the finance department, respectively, must be occurred immediate before the occurrence of the sending and the receiving of the acceptance confirmation of the deposit payment by the international office and the student, respectively.

4. The international office sends the notification of application status, which is either the application is successful or unsuccessful. Prevent that both application status are sent. The respective student must receive the corresponds notification which is sent by the international office.

5. If the notification is unsuccessful, the intended student will send a request for the deposit refund to the finance department. Ensure that the sending and the receiving of the unsuccessful offer by the international office to the respective student, occurs before, the sending and the receiving of the requesting of the deposit refund by the respective student to the finance department.

6. Otherwise, if the notification is successful, the intended student is required to pay the tuition fee effectives on the date of receiving the successful offer until the given tuition fee payment deadline. The tuition fee can be paid in two methods: either paying the whole tuition fee at once or paying the tuition fee as the instalment. Ensure that both methods are not allowed to be applied at the same time.

7. The occurrence of the sending and the receiving of the notification of the successful application by the international office to the respective student must be occurred immediate before the occurrence of paying the tuition fee (a full
payment of an instalment payment, but not two kind of methods at the same time) by the respective student to the finance department.

8. If the self-funding student requires a visa to study, they are expected to be able to fund the whole of their programme of study themselves.

9. Otherwise, if the self-funding student does not required a visa to study, they are able to pay the tuition fee with two aforementioned methods before.

10. The respective payment must be paid by the student to the finance department.

11. The instalment is grouped into two types: either the instalment payment is for the student who takes September commencing course or January commencing course. The instalment for each course is expected to be paid two times. Ensure that the first instalment must be occurred immediate before the second instalment. This rule applies for both kind of instalments.

12. The full payment of the tuition fees has a date when the payment is paid and a deadline for paying the tuition fee. The payment date must be before or at the same date as the deadline. Note that the deadline must be equal with the tuition fee payment deadline.

13. The instalment payment for September commencing course has a payment date and a deadline for both first and second instalment. The payment date must be before or on the same date as the deadline. Also, the deadline for the first instalment must be before or on the same date for the payment date of second instalment. Note that the second instalment deadline must be equal with the tuition fee payment deadline.

14. The instalment payment for January commencing course has a payment date and a deadline for both first and second instalment. The payment date must be before or on the same date as the deadline. Also, the deadline for the first instalment must be before or on the same date for the payment date of second instalment. Note that the second instalment deadline must be equal with the tuition fee payment deadline.

The SBVR model for the tuition fee system case study has been developed according to the above choreography contract, where the scope of the case study is smaller that the actual tuition fee regulation. We focus on the tuition fee payment which applies to the international student only, who is either the self-funding or the non self-funding student. The details of the SBVR model which describes the global constraints of the SBVR model for the tuition fee system case study can be referred as in the following figures.

Note that Figure shows the specific request to see if the notification of successful application status, which is sent by the international office can be realised by the given choreography of the SBVR model.
APPENDIX B. THE SBVR MODEL FOR THE TUITION FEE SYSTEM CASE

254

STUDY

--Participant Set

--Add participant terms here:

Term: participant
Term: international student
Term: self-funding student
Term: sponsor student
Term: visa prerequisite student
Term: non-visa prerequisite student
Term: finance department
Term: international office

--Event Set

--Add event terms here:

Term: event
Term: deposit payment
Term: confirmation acceptance
Term: offer notification
Term: successful offer
Term: unsuccessful offer
Term: deposit refund request
Term: tuition fee payment
Term: full payment
Term: instalment payment
Term: sept commencing course first instalment
Term: sept commencing course second instalment
Term: jan commencing course first instalment
Term: jan commencing course second instalment

--Constraint

--Add your static constraints terms here:

Term: sept first instalment payment date
Term: sept first instalment payment deadline
Term: sept second instalment payment date
Term: sept second instalment payment deadline date
Term: jan first instalment payment date
Term: jan first instalment payment deadline date
Term: jan second instalment payment date
Term: jan second instalment payment deadline date
Term: unsuccessful reason
Term: visa reason
Term: conditional reason
Term: study programme reason
Term: refund request letter
Term: full payment date
Term: full payment deadline
Term: successful offer date
Term: tuition fee payment deadline

--Time

Term: †

Figure B.1: Terms in the SBVR model for the tuition fee system
Figure B.2: Fact types for specifying the participant set (the local behaviour for each participant) in the SBVR model for the tuition fee system

--Fact Types

--Add your participant fact types here:

--Participant 1

Fact Type: participant includes self-funding student
Fact Type: self-funding student includes visa prerequisite student
Fact Type: self-funding student includes non-visa prerequisite student

--Local behaviour of self-funding student

--export message

Fact Type: self-funding student makes deposit payment at t1
Fact Type: self-funding student requests for deposit refund request at t1
Fact Type: self-funding student makes tuition fee payment at t1

--Import message

Fact Type: self-funding student receives confirmation acceptance at t2
Fact Type: self-funding student receives offer notification at t2

--Participant 2

Fact Type: participant includes international office

--Local behaviour of international office

--export message

Fact Type: international office sends confirmation acceptance at t1
Fact Type: international office sends offer notification at t1

--Participant 3

Fact Type: participant includes finance department

--Local behaviour of finance department

--import message

Fact Type: finance department receives deposit payment at t2
Fact Type: finance department receives deposit refund request at t2
Fact Type: finance department receives tuition fee payment at t2
APPENDIX B. THE SBVR MODEL FOR THE TUITION FEE SYSTEM CASE

256

--Fact Types

--Add your event fact types here:

--Event 1

Fact Type: event includes deposit payment

--initialisation

Fact Type: deposit payment is initial event

--Event 2

Fact Type: event includes confirmation acceptance

--Event 3

Fact Type: event includes offer notification

Fact Type: offer notification includes successful offer

Fact Type: offer notification includes unsuccessful offer

--static constraints

Fact Type: successful offer has successful offer date

Fact Type: unsuccessful offer has unsuccessful reason

Fact Type: unsuccessful reason is visa reason

Fact Type: unsuccessful reason is conditional reason

Fact Type: unsuccessful reason is study programme reason

--Event 4

Fact Type: event includes deposit refund request

--static constraints

Fact Type: deposit refund request has refund request letter

Figure B.3: Fact types for specifying the event set in the SBVR model for the tuition fee system
--Event 5

Fact Type: event includes tuition fee payment

Fact Type: tuition fee payment includes full payment

Fact Type: tuition fee payment includes instalment payment

Fact Type: instalment payment includes sept commencing course first instalment

Fact Type: instalment payment includes sept commencing course second instalment

Fact Type: instalment payment includes jan commencing course first instalment

Fact Type: instalment payment includes jan commencing course second instalment

--static constraints

Fact Type: tuition fee payment has tuition fee payment deadline

Fact Type: full payment has full payment date

Fact Type: full payment has full payment deadline

Fact Type: sept commencing course first instalment has sept first instalment payment date

Fact Type: sept commencing course first instalment has sept first instalment payment deadline date

Fact Type: sept commencing course second instalment has sept second instalment payment date

Fact Type: sept commencing course second instalment has sept second instalment payment deadline date

Fact Type: jan commencing course first instalment has jan first instalment payment date

Fact Type: jan commencing course first instalment has jan first instalment payment deadline date

Fact Type: jan commencing course second instalment has jan second instalment payment date

Fact Type: jan commencing course second instalment has jan second instalment payment deadline date

Figure B.4: Fact types for specifying the event set in the SBVR model for the tuition fee system
APPENDIX B. THE SBVR MODEL FOR THE TUITION FEE SYSTEM CASE

258

--Add your ordering of time associates with same event here:

Fact Type: deposit payment at t1 immediately precedes deposit payment at t2

Fact Type: deposit refund request at t1 immediately precedes deposit refund request at t2

Fact Type: confirmation acceptance at t1 immediately precedes confirmation acceptance at t2

Fact Type: offer notification at t1 immediately precedes offer notification at t2

Fact Type: tuition fee payment at t1 immediately precedes tuition fee payment at t2

Figure B.5: Fact types for specifying the ordering of time associates with the same event in the SBVR model for the tuition fee system

--Add your ordering of events fact types here:

Fact Type: deposit payment immediately precedes confirmation acceptance

Fact Type: confirmation acceptance immediately precedes offer notification

Fact Type: offer notification immediately precedes deposit refund request

Fact Type: offer notification immediately precedes tuition fee payment

Fact Type: sept commencing course first instalment immediately precedes sept commencing course second instalment

Fact Type: jan commencing course first instalment immediately precedes jan commencing course second instalment

Figure B.6: Fact types for specifying the ordering of events in the SBVR model for the tuition fee system

STUDY
--Add your static constraints fact types here (if required):

--a set of date:

Fact Type: successful offer date is in date
Fact Type: tuition fee payment deadline is in date
Fact Type: sept first instalment payment date is in date
Fact Type: jan first instalment payment deadline date is in date
Fact Type: jan first instalment payment date is in date
Fact Type: sept second instalment payment deadline date is in date
Fact Type: sept second instalment payment date is in date
Fact Type: jan second instalment payment date is in date
Fact Type: jan second instalment payment deadline date is in date
Fact Type: full payment date is in date
Fact Type: full payment deadline is in date

--initialisation for date

Fact Type: successful offer date is initial date
Fact Type: tuition fee payment deadline is final date

Figure B.7: Fact types for specifying the static constraints particularly on date constraints in the SBVR model for the tuition fee system
--Date constraints:

Fact Type: full payment date of full payment equals successful offer date of successful offer

Fact Type: full payment deadline of full payment equals tuition fee payment deadline of tuition fee payment

Fact Type: full payment date of full payment begins before full payment deadline of full payment

Fact Type: sept first instalment payment date of sept commencing course first instalment begins before sept first instalment payment deadline date of sept commencing course first instalment

Fact Type: jan first instalment payment date of jan commencing course first instalment begins before jan first instalment payment deadline date of jan commencing course first instalment

Fact Type: sept first instalment payment deadline date of sept commencing course first instalment begins before sept second instalment payment date of sept commencing course second instalment

Fact Type: sept second instalment payment date of sept commencing course second instalment begins before sept second instalment payment deadline date of sept commencing course second instalment

Fact Type: jan first instalment payment deadline date of jan commencing course first instalment begins before jan second instalment payment date of jan commencing course second instalment

Fact Type: jan second instalment payment date of jan commencing course second instalment begins before jan second instalment payment deadline date of jan commencing course second instalment

Fact Type: sept first instalment payment date of sept commencing course first instalment equals successful offer date of successful offer

Fact Type: jan first instalment payment date of jan commencing course first instalment equals successful offer date of successful offer

Fact Type: sept second instalment payment deadline date of sept commencing course second instalment equals tuition fee payment deadline of tuition fee payment

Fact Type: jan second instalment payment deadline date of jan commencing course second instalment equals tuition fee payment deadline of tuition fee payment

Figure B.8: Fact types for specifying the static constraints particularly on date constraints in the SBVR model for the tuition fee system
Rule: It is obligatory that the **self-funding student** makes exactly one **deposit payment** at exactly one **t1**

Rule: It is obligatory that the **finance department** receives exactly one **deposit payment** at exactly one **t2**

Rule: It is obligatory that exactly one **deposit payment** immediately precedes exactly one **confirmation acceptance**

Rule: It is obligatory that the **international office** sends exactly one **confirmation acceptance** at exactly one **t1**

Rule: It is obligatory that the **self-funding student** receives exactly one **confirmation acceptance** at exactly one **t2**

Rule: It is obligatory that exactly one **confirmation acceptance** immediately precedes exactly one **offer notification** that includes exactly one **successful offer** or exactly one **unsuccessful offer** but not both

Rule: It is obligatory that the **international office** sends exactly one **offer notification** that includes exactly one **successful offer** or exactly one **unsuccessful offer** but not both, at exactly one **t1**

Rule: It is obligatory that the **self-funding student** receives exactly one **offer notification** that includes exactly one **successful offer** or exactly one **unsuccessful offer** but not both, at exactly one **t2**

Figure B.9: SBVR rules in the SBVR model for the tuition fee system
Rule: it is obligatory that exactly one \textit{offer notification} that \textit{includes} exactly one \textit{unsuccessful offer immediately precedes} exactly one \textit{deposit refund request}

Rule: It is obligatory that the \textit{self-funding student requests for} exactly one \textit{deposit refund request at} exactly one \textit{t1}

Rule: It is obligatory that the \textit{finance department receives} exactly one \textit{deposit refund request at} exactly one \textit{t2}

Rule: It is obligatory that exactly one \textit{offer notification} that \textit{includes} exactly one \textit{successful offer immediately precedes} exactly one \textit{tuition fee payment} that \textit{includes} exactly one \textit{full payment} or exactly one \textit{instalment payment} but not both

Rule: It is obligatory that the \textit{self-funding student that includes the visa prerequisite student makes} exactly one \textit{tuition fee payment} that \textit{includes} exactly one \textit{full payment at} exactly one \textit{t1}

Rule: It is obligatory that the \textit{self-funding student that includes the non-visa prerequisite student makes} exactly one \textit{tuition fee payment} that \textit{includes} exactly one \textit{full payment} or exactly one \textit{instalment payment} but not both, \textit{at} exactly one \textit{t1}

Rule: It is obligatory that the \textit{finance department receives} exactly one \textit{tuition fee payment that includes exactly one full payment or exactly one instalment payment but not both, at exactly one t2}

Rule: It is obligatory that exactly one \textit{instalment payment that includes exactly one sept commencing course first instalment immediately precedes exactly one instalment payment that includes exactly one sept commencing course second instalment}

Rule: it is obligatory that exactly one \textit{instalment payment that includes exactly one jan commencing course first instalment immediately precedes exactly one instalment payment that includes exactly one jan commencing course second instalment}

Figure B.10: SBVR rules in the SBVR model for the tuition fee system
Figure B.11: SBVR rules particularly for date constraints in the SBVR model for the tuition fee system
APPENDIX B. THE SBVR MODEL FOR THE TUITION FEE SYSTEM CASE

264

STUDY

--Specific request:

**Fact Type:** international office sends offer notification at t₁

**Fact Type:** offer notification includes successful offer

**Rule:** it is obligatory that the international office sends exactly one offer notification that includes exactly one successful offer, at exactly one t₁

Figure B.12: Specific request in the SBVR model for the tuition fee system
Appendix C

The Alloy Model for the Online Photo Shop case study

The development of the Alloy model for the online photo shop is based on the developed SBVR model which describes the global constraints of the online photo shop case study as described in Appendix A.

The transformation from the SBVR model into Alloy is started form translating the terms and the fact types relating to the participant set, the event set, and the domain-specific constraints (if any). Figure A.2 and figure ?? shows the fact types for the participant set as well as the local behaviour for each participant describing the export, import messages, and the domain-specific constraints. These are captured in the Alloy model by defining the abstract signature for participant and its extensions involved in the online photo shop. This can be seen in Figure ??.

```
abstract sig participant{}

one sig customer extends participant{makes: one accountregistration, makes1: some productrequest, makes2: one productpayment, receives1: one productdelivery, receives2: one paymentnotification, has1: one name, has2: one address, has3: one creditcardnumber} { #makes1 >= 1 }

one sig photoshop extends participant{records: one accountregistration, receives1: some productrequest, receives2: one paymentnotification, makes: one productdelivery} { #receives1 >= 1 }

one sig bank extends participant{receives: one productpayment, sends: one paymentnotification}
```

Figure C.1: Abstract signatures and signatures for the participant set
On the other hand, the fact types relating to the specification for the event set in figure A.4 is transformed into Alloy as illustrated in figure C.2.

```alloy
-- a set of events
abstract sig event{}

one sig accountregistration extends event{immediatelyprecedes: some productrequest, at1: one T1_AR, at2: one T2_AR}
{ (#immediatelyprecedes = 1 and immediatelyprecedes = customer.makes1 and immediatelyprecedes = photoshop.receives1 )}

abstract sig productrequest extends event{immediatelyprecedes: one productpayment, at1: one T1_PR, at2: one T2_PR}
{ immediatelyprecedes = customer.makes2 and immediatelyprecedes = bank.receives }

lose sig photorequest extends productrequest{}
lose sig albumrequest extends productrequest()
lose sig posterrequest extends productrequest{}

one sig productdelivery extends event{at1: one T1_PD, at2: one T2_PD}

one sig productpayment extends event{immediatelyprecedes: one paymentnotification, at1: one T1_PP, at2: one T2_PP}
{ immediatelyprecedes = customer.receives2 and immediatelyprecedes = photoshop.receives2 and immediatelyprecedes = bank.sends }

abstract sig paymentnotification extends event{at1: one T1_PN, at2: one T2_PN}
{ (paymentnotification = successfulnotification and no unsuccessfulnotification) or (paymentnotification = unsuccessfulnotification and no successfulnotification) }

lose sig successfulnotification extends paymentnotification{immediatelyprecedes: one productdelivery}
{ immediatelyprecedes = customer.receives1 and immediatelyprecedes = photoshop.makes }

lose sig unsuccessfulnotification extends paymentnotification{}
```

Figure C.2: Abstract signatures and signatures for the event set

All global constraints as described in the SBVR rules for the SBVR model of the online photo shop, regardless the purpose of the specification whether to specify the complex interactions or the precedence between events as described in figure A.7 and figure A.7 are defined in the respective abstract signatures or signatures (figure C.1, figure C.2 and figure C.3) which follow the transformation method discussed in Chapter 4.

For generating the choreography for the online photo shop is shown in figure C.4, while the specific request rule that was described in figure A.9 is translated in Alloy using the pred concretereguest as depicted in figure C.5.
Figure C.3: Abstract signature for Time and its extensions

```plaintext
--Time declaration
abstract sig time{}

abstract sig accountregistration_time extends time{}
one sig T1_AR extends accountregistration_time {by: one customer, immediatelyprecedes: one T2_AR}
one sig T2_AR extends accountregistration_time {by: one photoshop}

abstract sig productrequest_time extends time{}
one sig T1_PR extends productrequest_time {by: one customer, immediatelyprecedes: one T2_PR}
one sig T2_PR extends productrequest_time {by: one photoshop}

abstract sig productpayment_time extends time{}
one sig T1_PP extends productpayment_time {by: one customer, immediatelyprecedes: one T2_PP}
one sig T2_PP extends productpayment_time {by: one bank}

abstract sig productdelivery_time extends time{}
one sig T1_PD extends productdelivery_time {by: one photoshop, immediatelyprecedes: one T2_PD}
one sig T2_PD extends productdelivery_time {by: one customer}

abstract sig paymentnotification_time extends time{}
one sig T1_PN extends paymentnotification_time {by: one bank, immediatelyprecedes: one T2_PN}
one sig T2_PN extends paymentnotification_time {by: one customer, by2: one bank}
```

--Static Constraints
one sig name{}
one sig address{}
one sig creditcardnumber{}

```
pred initialevent[a:accountregistration]
{(#a.immediatelyprecedes >= 1 )}
run initialevent
```

Figure C.4: Predicates for generating the choreography for the SBVR model of the online photo shop

```
pred concreterequest [c:customer, h:photorequest, b:albumrequest, s:posterrequest]
{(c.makes1 = h and c.makes1 != b and c.makes1 != s) or (c.makes1 = b and c.makes1 != h and c.makes1 != s) }
run concreterequest
```

Figure C.5: Predicates for verifying the specific request on the generated choreography of the online photo shop
Appendix D

The Alloy Model for the Tuition Fee System case study

The development of the Alloy model for the Tuition Fee System is based on the developed SBVR model which describes the global constraints of the Tuition Fee System case study as described in Appendix B.

The transformation from the SBVR model into Alloy is started form translating the terms and the fact types relating to the participant set, the event set, and the domain-specific constraints (if any). Figure B.2 shows the fact types for the participant set as well as the local behaviour for each participant describing the export, import messages, and the domain-specific constraints. These are captured in the Alloy model by defining the abstract signature for participant and its extensions involved in the Tuition Fee System. This can be seen in Figure ??.

On the other hand, the fact types relating to the specification for the event set in figure ?? and figure B.4 are transformed into Alloy as illustrated in figure D.2.

All global constraints as described in the SBVR rules for the SBVR model of the Tuition Fee System, regardless the purpose of the specification whether to specify the complex interactions or the precedence between events as described in figure B.9 and figure B.10, are defined in the respective abstract signatures or signatures

Figure D.1: Abstract signatures and signatures for the participant set

abstract sig participant()
abstract sig selffundingstudent extends participant(makes: one deposit/payment, receives: one confirmation/acceptance, receives1: one offer/confirmation, requests for: one deposit/refund request)
{selffundingstudent = visaprepaymentstudent and not nonvisaprepaymentstudent} or
{selffundingstudent = nonvisaprepaymentstudent and not visaprepaymentstudent}

lone sig visaprepaymentstudent extends selffundingstudent(makes: one tuition/payment)

lone sig nonvisaprepaymentstudent extends selffundingstudent(makes: one tuition/payment)

one sig financedepartment extends participant(receives: one deposit/payment, receives1: one deposit/refund request, receives2: one tuition/payment)
one sig internationaloffice extends participant(sends: one confirmation/acceptance, sends1: one offer/confirmation)

269
Figure D.2: Abstract signatures and signatures for the event set

```alloy
abstract sig event()

one sig depositpayment extends event (immediateprecedes: one confirmationacceptance; at: one T1_DP; all: one T2_DP)
  (immediateprecedes = selfrefundingsstudentreceives and immediateprecedes = selfinternationalofficeid)

one sig confirmationacceptance extends event (immediateprecedes: selfrefundingsstudentaccepts; at: one T1_CS; all: one T2_CS)
  (immediateprecedes = selfrefundingsstudentreceives and immediateprecedes = selfinternationalofficeid)

one sig depositrefundrequest extends event (has: one refundauthorization; at: one T1_DR; all: one T2_DR)
  (immediateprecedes = selfrefundingsstudentreceives and immediateprecedes = selfinternationalofficeid)

abstract sig offernotification extends event (at: one T1_OH)
  (offernote = successfuloffer and not unsuccessfuloffer) or
  (offernote = unsuccessfuloffer and not successfuloffer)
  (baseofoffernotification)

these sig successfuloffer extends offernotification (immediateprecedes: one tuitionrefundpayment; has: one successfulofferdata)
  (immediateprecedes = nonsurpriserequeststudentmakes; or immediateprecedes = vicepresidentstudentdesk)
  and immediateprecedes = financedepartmentreceives)

these sig unsuccessfuloffer extends offernotification (has: one unsuccessfuloffer; immediateprecedes: one depositrefundrequest)
  (immediateprecedes = selfrefundingsstudentrequestfor and immediateprecedes = financedepartmentreceives)

one sig unsuccessfulrequest extends event (to: one vicepresident; si1: one conditionrequest; si2: one studyprogramrequest)

abstract sig tuitionrefundpayment extends event (at: one T1_TR; all: one T2_TR; has2: one tuitionrefundpaymentdeadline)
  (tuitionrefundpayment = fullpayment and no installmentpayment) or
  (tuitionrefundpayment = installmentpayment and no fullpayment)
  (has = successfuloffer and has2 = has2)

abstract sig installmentpayment extends tuitionrefundpayment
  (installpayment = sepinstallpayment; is: one sepinstallpaymentpayment; immediateprecedes: one sepinstallpaymentpayment)

these sig sepinstallpaymentpayment extends installmentpayment (has: one sepinstallpaymentpayment; has1: one sepinstallpaymentpayment)
  (has = successfuloffer; has1 and immediateprecedes = nonsurpriserequeststudentmakes; and immediateprecedes = financedepartmentreceives)

these sig jepinstallpaymentpayment extends installmentpayment (has: one jepinstallpaymentpayment; has1: one jepinstallpaymentpayment)
  (has = successfuloffer; has1 and immediateprecedes = nonsurpriserequeststudentmakes; and immediateprecedes = financedepartmentreceives)

--Time declaration
abstract sig time{}

abstract sig accountregistration_time extends time{}

one sig T1_AR extends accountregistration_time (br: one customer, immediatelyprecedes: one T2_AR)
one sig T2_AR extends accountregistration_time (br: one photoshop)

abstract sig productrequest_time extends time{}

one sig T1_PR extends productrequest_time (br: one customer, immediatelyprecedes: one T2_PR)
one sig T2_PR extends productrequest_time (br: one photoshop)

abstract sig productpayment_time extends time{}

one sig T1_PP extends productpayment_time (br: one customer, immediatelyprecedes: one T2_PP)
one sig T2_PP extends productpayment_time (br: one bank)

abstract sig productdelivery_time extends time{}

one sig T1_PD extends productdelivery_time (br: one photoshop, immediatelyprecedes: one T2_PD)
one sig T2_PD extends productdelivery_time (br: one bank)

abstract sig paymentnotification_time extends time{}

one sig T1_PN extends paymentnotification_time (br: one bank, immediatelyprecedes: one T2_PN)
one sig T2_PN extends paymentnotification_time (br: one customer, by2: one bank)

--Static Constraints
one sig name()
one sig address()
one sig creditcardnumber()
```

Figure D.3: Abstract signature for Time and its extensions
(figure C.1, C.2, and figure D.3) which follow the transformation method discussed in Chapter 4.

The declaration of fact types specifying a **Date** set and its initialisation in Figure B.7 and the specification of date constraints as shown in Figure B.11 are translated into Alloy as defined in Figure D.4.

```alloy
sig Date()
  one sig successfulDate in Date()
  one sig tuitionPaymentDeadline in Date()
  one sig fullPaymentDeadline in Date()
  one sig receiptDeadline in Date()
  one sig paymentDeadline in Date()
  one sig nextPaymentDeadline in Date()
  one sig nextReceiptDeadline in Date()
  one sig nextNextReceiptDeadline in Date()
  one sig nextFullPaymentDeadline in Date()
  one sig nextFullPaymentDeadline in Date()
  one sig nextNextFullPaymentDeadline in Date()

pred orderedDate (d1,d2,d3; one Date, successfulDate, tuitionPayment, fullPayment, s1: seconcourseInstalment, s2:seconcourseInstalment, s1:jancourseInstalment, s2:jancourseInstalment) {
  d1 <= d2 and d2 <= d3
}

pred allDates (d1,d2,d3; one Date, successfulDate, tuitionPayment, fullPayment, s1: seconcourseInstalment, s2:seconcourseInstalment, s1:jancourseInstalment, s2:jancourseInstalment) {
  d1 <= d2 and d2 <= d3
}

pred initInitialisation {
  init traces {
    init true |
    let d1 = init |
    some d2 : Date in d1 <= d2 and d2 <= init |
    some successfulDate, tuitionPayment, fullPayment, s1: seconcourseInstalment, s2:seconcourseInstalment, s1:jancourseInstalment, s2:jancourseInstalment |
    orderedDate(d1,d2,d3) |
  }
}
```

**Figure D.4:** The declaration of date constraints

For generating the choreography for the Tuition Fee System is shown in figure D.5, while the specific request rule that was described in figure B.12 is translated in Alloy using the **pred concreterequest** as depicted in figure D.6.

```alloy
pred concreterequest {
  (lHas = s1 and lHas = s2) or (lHas = s1 and lHas = d1)
}
```

**Figure D.5:** Predicates for generating the choreography for the SBVR model of the Tuition Fee System
Figure D.6: Predicates for verifying the specific request on the generated choreography of the Tuition Fee System

```
pred concreterequest [c:customer, h:photorequest, b:albumrequest, s:posterrequest]
{(c.makes1 = h and c.makes1 != b and c.makes1 != s)
 or (c.makes1 = b and c.makes1 != h and c.makes1 != s) }
run concreterequest
```
Bibliography


273


