VOML: Virtual Organization Modelling Language

Thesis submitted for the degree of
Doctor of Philosophy
at the University of Leicester

by
Noor Jehan Rajper
Department of Computer Science
University of Leicester

June 2012

Abstract

Virtual organizations (VOs) and their breeding environments are an emerging approach for developing systems as a consortium of autonomous entities formed to share costs and resources, better respond to opportunities, achieve shorter time-to-market and exploit fast changing market opportunities. VOs cater for those demands by incorporating reconfigurations making VOs highly resilient and agile by design. Reconfiguration of systems is an active research area. Many policy and specification languages have been dedicated for the purpose. However, all these approaches consider reconfiguration of a system as somewhat isolated from its business and operational model; it is usually assumed that the latter two remain unaffected through such reconfigurations and the reconfiguration is usually limited to dynamic binding of components the system consists of. However the demands of VO reconfiguration go beyond dynamic binding and reach the level where it becomes crucial to keep changing the organizational structure (process model) of the system as well, which leads to changes of the operational/-functional model. This continuous reconfiguration of the operational model emphasizes the need of a modelling language that allows specification and validation of such systems.

This thesis approaches the problem of formal specification of VOs through the Virtual Organization Modelling Language (VOML) framework. The core of this framework are three languages each capturing a specific aspect. The first language named *Virtual Organization Structural* modelling language (VO-S), focuses on structural aspects and many of the characteristics particular to VOs such as relationship between the members expressed in domain terminology. The second language named *VO Reconfiguration* (VO-R for short), permits different *reconfigurations* on the structure of the VO. This language is an extension of APPEL for the domain of VOs. The third language named *VO Operational* modelling language (VO-O) describes the operational model of a VO in more details. This language is an adaptation and extension of the *Sensoria Reference Modelling Language* for service oriented architecture (SRML).

Our framework models VOs using the VO-S and the VO-R which are at a high level of abstraction and independent of a specific computational model. Mapping rules provide guidelines to generate operational models, thus ensuring that the two models conform to each other.

The usability and applicability of of VOML is validated through two cases studies one of which offers travel itineraries as a VO service and is a running example. The other case study is an adaptation of a case study on developing a chemical plant from [14].

Declaration

The content of this submission was undertaken in the Department of Computer Science, University of Leicester, and supervised by Dr. Stephan Reiff-Marganiec during the period of registration. I hereby declare that the materials of this submission have not previously been published for a degree or diploma at any other university or institute. All the materials submitted is the result of my own research except as cited in the references.

Research work presented in some sections has been previously published, in particular:

- Some parts of the Chapter 3 (describing the overall VOML framework), basic concepts of VOs that shape the VO-Structural modelling language (Chapter 4) and overview of the VO-O modelling language (Chapter 5 and the Section 2.2.1) has been published in [12].
- The VO-Structural modelling language (Chapter 4) and the VO-R modelling language (Chapter 6) have been published in [58].

Noor Jehan Rajper
Leicester, June 2012

Acknowledgements

My list of acknowledgements can not begin other than the almighty ALLAH without Whose guidance and His given strength and will power I would not have been able to complete my PhD.

After ALLAH's support I have a long list of people who helped me and provided me with valuable support during the course of this PhD.

I am very fortunate to have been guided by many great mentors from my early schooling to PhD level studies. My mentors during early years of schooling at Agha Taj Muhammad High School, Hyderabad, Pakistan, gave me the strong foundations which helped me towards becoming an independent thinker and researcher. These foundations were brushed up and brought to forefront from dormancy by my esteemed supervisor, Dr. Stephan Reiff-Marganiec, for whom I fall short of words when it comes to express my deep gratitude. He not only provided me with the valuable input during the course of PhD but at many times believed in my work when I had lost hope myself.

I consider myself fortunate to have undergone PhD studies at University of Leicester, which provides a second supervisor to all the PhD students. Many thanks to my second supervisor José Fiadeiro, whose many years experience has a major share in shaping up my PhD work.

I have no words to express my gratitude for my parents who have always been with me through thick and thin and have always been my source of inspiration. Their pride and belief in me proved to be my source of strength and kept me going when I felt I could not do it any more.

My beloved family, husband Rizwan, and children (Hania and Arham) are probably the ones who have given the most sacrifices for me to complete my PhD. My husband was always there to support and encourage me when I was stuck in my PhD. My children made me forget the stress of my PhD and brought sheer joy on my face whenever I looked at them and strengthened my resolve to complete the PhD. It would not have been possible to complete my PhD had I not have my family's unconditional and endless love and understanding.

Last but not the least many thanks to friends and fellows at the University of Leicester for their help and valuable suggestions .

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This section introduces the context and research challenges of our thesis, presents aims, objectives and research questions, sets a thesis statement, highlights the contribution of the thesis and outlines its structure.

1.1 Context and Research Challenges

Ever-changing business demands and market turbulence demand of organizations to continuously and quickly adapt in order to remain competitive. This quite often requires new skills and resources which many organizations fail to have at their disposal. The most affected are small and medium organizations (SMEs). Hence it has become necessity in the existing highly dynamic market for the organizations to collaborate and coordinate with other autonomous (some times competing) organizations to make up for the skills and resources or capacity they lack to respond better to a business opportunity or even take the modest share from large organizations.
Virtual Organization (VO) is a concept that has emerged as the consequence; generally understood as a temporary alliance of autonomous entities (organizations, individuals, etc) that strategically share skills and resources supported by computer networks, to achieve some benefit not possible otherwise [28, 40, 19]. A VO is a dynamic organization that is formed according to the needs and opportunities of the market and

remains operational as long as these opportunities persist; once the opportunity ends, the organization dismantles itself.
Some of examples of virtual organizations are:

- To compete in business environment where opportunities are predominantly captured by large organizations, small and medium (SME) organizations come together to appear as a single large organization to increase profitability or to compensate for the missing skills.

- Incident management, disaster rescuing processes and emergency response teams, where it is necessary to immediately coordinate activities of a large number of entities such as fire brigades, army, volunteers, police, hospitals, local government, non-governmental organizations.

A VO achieves these attributes by allowing companies to seek complementaries that allow them to participate in competitive business opportunities and new markets, by creating partnerships to achieve its goals (tasks) or to achieve critical mass and appear in the market with a larger "apparent size". They can also achieve cost and effectiveness by sharing responsibilities between members if one member is either incapable to do it alone or will take longer.
All these attributes shape the composition of a VO in terms of who is assigned to what responsibility (task), how different tasks of a VO are shared between its members, which member is contributing how much of its resources, what the relationship between the members is, etc. To understand and describe this complex composition of VOs we need to model them first. Though there are many existing modelling languages which could be used to specify VOs, none of them completely and naturally satisfy the demands of VOs. The benefits of having a specification language specifically geared towards a particular domain are manifold and advocated in the literature. Key benefits are: first of all modelling languages abstract away unnecessary details of implementation; secondly it helps to work directly with notions and concepts of the domain at hand; thirdly having precise models allows for various analysis to be conducted on the blueprint of a VO (rather then developing a fully-fledged running VO).

The significance of having a modelling language for VOs has not gone unnoticed by the research community. For example at the more conceptual end of the spectrum there has been a dedicated effort by the European Commission Sixth Framework Programme Project (ECOLEAD) to understand the field in depth. The result of this effort is an abstract reference model called ARCON (Abstract Reference model for Collaborative Networks) [23], which is intended as a generic conceptual model that synthesizes and formalizes the base concepts of the domain. However, this reference model is generic and is not intended to be directly applicable to concrete cases.

Whereas, on the more concrete end of the spectrum, modelling of VOs for different sort of analysis, guarantees or properties is also gaining momentum. At this level VOs are modelled in already established generic formal modelling languages with tool support for analysis. One such example is [44] which has resulted from the GOLD project [3]. The limitation of such languages is that they require fluency in the chosen notation by the stakeholder (domain experts) in order to model the system.

Developing a modelling framework which on the one hand models VOs using terminology familiar to the stakeholders but on the other hand is formal and paves the way for different sorts of analysis is one research challenge. We provide a modelling language which uses the concepts developed in the ARCON but offers a more concrete realization thus being a step towards rigorous analysis.

While the composition of a VO is determined by the need to associate the most suitable set of capabilities contributed by the distinct organizations, a VO is expected to be flexible in its composition and structure so that it can reorganize itself by adding or expelling some members or by dynamically re-assigning tasks to its members; in particular by sharing a task across members or decomposing a complex task into smaller more specialized subtasks. These trends require organizations to be *agile*, that is for them to have "the ability to recognize, rapidly react and cope with the unpredictable changes in the environment, with the smooth adaptation of its structure to the new current reality" [21]. But, the prevailing system engineering approaches representing such organizations with "fixed organizational structures" obstruct such agility and resilience [1, 53, 54, 55]. A VO is derived by the business opportunity; hence any aspect of it might need to be changed in order to keep the VO goal aligned with the business opportunity. This not only affects its membership but may also be reflected in its structural organization which forms a challenge for modelling VOs. Our modelling framework does not only describes different aspects of the VO but also provides provisions to account for this flexibility.

While developing modelling languages for specific domains, two aspects of the system are mostly ignored or abstracted away: (a) the application aspect of the system i.e the actual goal or service (specification of business functionality) offered by the system, and (b) the coordination and communication model close to the underlying execution environment. The reason behind this abstraction (in our opinion) is that it is usually assumed that the business specification model (including coordination and communication model) are not affected by the domain level reorganization of the system. However any restructuring (reconfiguration) which adds something new, modifies or deletes something from the system does change the coordination and communication model representing the business aspect. Consider a scenario in which a VO demands a certain level of resource stock for some task; at the domain level (abstract level) of modelling it is just a matter of adding more than one member using a construct equivalent to an *Add* operation. What is usually left untouched is that at the operational level it is not just the matter of a simple Add operation. The implications are that now more than one member needs to be communicated with, which implies addition of some coordination, communication and possible computation operations and a possible increase in the number of components or other concrete entities representing the elements of the underlying execution paradigm.

Such complexities in the more concrete operational models are usually left out and provisions for such changes is specified at more abstract level. This approach, nonetheless plays a key role in understanding and describing systems in the early stages of system development. However, the picture remains incomplete without describing the affects of reconfigurations on the operational and communication model of the system

which is closet to the actual implementation of the system.
Many specification and policy languages exists that model systems in a specific domain and their reorganization (reconfiguration); all of these approaches to the best of our knowledge consider the modelling of a system some what isolated from its underlying coordination and communication model. Hence, looking at the effects of reconfiguration on business and coordination and communication model is another challenge. Our modelling framework addresses this by mapping concepts defined at the domain level to the concepts defined at operational level. This helps to identify what part of operational level gets affected by any reconfiguration done on its corresponding part at the domain level and how it gets affected.
In summary, we are putting forward a framework for modelling VOs at the structural level, catering for their reconfigurations and allowing to express affects on business and coordination models. In this thesis; we put forward a novel framework targeted towards VOs which addresses all of the above research challenges.

1.2 Criteria for Modelling Languages in the Domain of VOs

From the previous section, we can extract a number of criteria that a modelling language attempting to be useful for VBE and VO domain must posses. These criteria also help us in identifying the limitations and strengths of our research by comparing it with others' efforts based on the criteria laid out here. The selection and evaluation of the two case studies that are explored in this thesis are justified against these criteria, too.
A VO modelling language should be able to to satisfy the following criteria:

C1. Expressibility: Representing domain concepts as modelling language constructs.

C2. Ability of the language to evolve the model.

C3. Represent changing association of members with VBEs and VOs. Some members remain part of a VO from its inception to dissolution, while others keep changing quite frequently.

C4. Capture the concept of *collaboration* by:

1. Allowing more than one member to carry out a particular task.
2. Sub-dividing a task into more than one tasks and assigning each subtask to be performed by individual member.

C5. Define different kinds of relationships between members for example member cooperating or competing over a task.

C6. Adapt the goal of a VO to meet changing business circumstances.

C7. Resist VOs failing to operate or failing to provide the promised service as long as possible such as by proving provisions which allow VOs to operate with degraded performance .

C8. Preserve the autonomy of members. Members should be responsible for managing their resources while sharing them in VO with others.

C9. Free from fixed “organizational structure” i.e able to modify the operational model.

1.3 Overview of the Approach

To address all the research challenges presented in Section 1.1 and the criteria described in Section 1.2 we have developed a framework called *Virtual Organization Modelling Language* framework. The framework addresses the above research challenges by developing three modelling languages each focusing on one of the challenges. The framework is then evaluated against the above criteria in Chapters 6 and 8.

The first language called *Virtual Organization Structural (VO-S)* modelling language is concerned with describing the structural aspect of virtual organizations (Chapter 4). It describes how the VO is currently organized, mainly in terms of tasks, competencies, members and relationships between the members. The second research challenge which is about catering to the agility demands of VOs is approached through another modelling language called *Virtual Organization Reconfiguration (VO-R)* modelling language (Chapter 6), which incorporates reconfiguration aspect for VOs by a dedicated set of generic events, conditions and actions which apply to any VO irrespective of its application. The third aspect which focuses on the business functionality and coordination and communication model, is addressed through a third dedicated language (Chapter 5). The framework also links the three languages together as follows: The language used for reconfiguration uses constructs defined in the VO-S language which describes the structural aspects of VO using domain terminology; the events, conditions and actions of the reconfiguration related language is also defined in terms of constructs of the structure defining language. The link between the structure defining and operational aspect defining language is created through a set of mapping rules (Chapter 5). These mappings also help in checking the consistency between structural and operational models of the VO.

1.4 Aims and Objectives

The overall aim of this work is to provide a framework for modelling VBEs and VOs whose agility demands may also affect its operational model. It is useful to identify the particular problems that we need to overcome.

1.4.1 Aims

Define a modelling framework for VOs that:

1. Offer domain concepts as "first class" entities.

   This aim includes the provision of domain concepts the domain experts relate naturally and intuitively to and at the level of abstraction which is comprehensible to different stakeholders of domain without any technological expertise.

2. Allows to express the structure of VOs independent from its operational aspects. This aim includes provision of modelling constructs which are able to describe basic structural composition of VOs such as the members of VOs, the responsibilities assigned to them, the relationships between members, etc irrespective of the underlying execution environment.

3. Provides Return on Modelling Efforts (RoME).
   This aim includes provision of reconfiguration constructs for the VO models, impact of reconfigurations both at structural and operational level and lending models for formal analysis.

4. Ensures developing correct by construction operational models from structural (domain) model.
   This includes provision of rules which correctly transform a structural model into corresponding operational model.

1.4.2 Objectives

We have developed a framework targeting the above aims that address a number of concrete objectives. In particular the framework shall:

1. Be general and able to describe different kinds of VOs.

2. Be platform and technology neutral.

3. Be able to adapt VOs with changing circumstances.

4. Be simple by separating concerns and keeping the structural, dynamic and operational aspects separate.

5. Be able to generate the operational model from the structural model.

To address these objectives we need to answer following questions.

1.4.3 Questions

1. How general is the specification language in terms of the VO domain, i.e. does it cover all the domain where VOs occur (e.g business, social, governance)? What sort of VOs can be modelled with it?

2. How is the agility required by VOs offered through the VOML framework?

3. What is the effect of reconfiguration on the operational model, that is when does the operational model need to be changed?

4. Does the reconfigurations require abandoning the running instances?

5. How are reconfigurations checked to make sure that they do not put the VO into inconsistent state with respect to VOs' structural specification.

All the questions are revisited in the Section 8.4 to validate which of these questions the VOML framework is able to answer and how.

1.5 Thesis Statement

The features associated with VOs demand radical changes in the style we model, design and develop systems. Needs such as re-organizing VOs by adding/expelling some members, sharing or dividing a task between members or by dynamically reassigning tasks or roles to its members have profound implication on the systems' operational aspect as well. Current modelling languages hardly address these issues, especially considering the impact such changes can have on the operational model. In particular, there is no common modelling language for specifying VOs that allows to specify VOs and their reconfigurations at the abstraction level of domain experts, while also reflecting the effect of such reconfiguration on the operational model and at the same time being formal enough to allow analysis to be carried out.

It is our ultimate intent to provide a framework in which domain users are able to describe and understand the system specification in terms of their domain terminology and the IT community can derive a complete operational model from the former. Thus, it is our ultimate goal to build and impose a VO modelling language (VOML) generic and expressive enough for modelling a variety of VOs. With VOML we are aiming at the formal modelling, reconfiguration and generating operational model of VOs.

1.6 Main Contributions

This thesis introduces VOML, a novel framework for modelling VOs. The main contributions are:

1. Three different modelling languages are developed each capturing different aspect of VO.
    (a) *Virtual Organization-Structural* modelling language (VO-S) focuses on structural aspects and many of the characteristics particular to VOs such as different relationships that can exist between members of the VOs, their competencies, what tasks they are assigned etc.
    (b) *Virtual Organization-Reconfiguration* language permitting to reorganize the structure of VO through reconfigurations. This language is an extension of APPEL for the domain of VOs. In particular, it provides generic set of events, conditions and actions which can apply to every VO application.
    (c) *Virtual Organization-Operational* modelling language (VO-O) describes operational model of VO in more details. This language is an adaptation and extension of *Sensoria Reference Modelling Language* for service oriented architecture.
2. Mapping rules from VO-S to VO-O which generate a partial VO-O specification from VO-S specification. These rules also act as conformance verifier of VO-O model to their corresponding VO-S models.
3. Validate the generality of the VOML framework by specifying two characteristically different VOs:

(a) Travel booking VO.

(b) VO for developing chemical plants.

1.7 Thesis Organization and Summary

This chapter explained the purpose of the dissertation - *a framework for modelling virtual organization*. We explained the research challenges addressed in this dissertation which are: (a) provide a modelling language which offers constructs which different stakeholders of the VO easily relate to, (b) out of those constructs a concrete operational model of VO can be generated which cover the business aspect of the VO, and (c) offer provisions which allow to reorganize the VO structure to cater to the agility demands of the VOs. We listed research aims, objective and questions relating to them. Furthermore, we set of the criteria against which the developed modelling languages are evaluated in the subsequent chapters. We also provided clear thesis statement and research contributions.
The reminder of this dissertation is organized as follow:

- In Chapter 2, we give a detailed account of research background. Specifically we are going to discuss in detail the SRML language, which we have adapted and adjusted for the specifying operational aspects of VOs. The ARCON reference model is also introduces in this chapter which provides guideline on developing concrete models for different types of VOs and VBEs. We are also going to discuss related work there.

- In Chapter 3, we introduce VOML framework in detail.

- In Chapter 4, we discuss in detail the VO-S modelling language.

- In Chapter 5, We discuss the adjustments made to the SRML to adapt it as our VO-O modelling language. Furthermore, we specify mapping rules which generate VO-O specification from a given VO-S specification.

- In Chapter 6, we introduce VO-R modelling language and model and reconfigure a VO offering travel itineraries.
- In Chapter 7, we discuss the formal syntax of the three languages which is described using the xtext [4] language. Semantic integration of the languages is also discussed at the conceptual level.
- In Chapter 8, we evaluate our framework against another case study which develops chemical plants. We also evaluate VO-S language against ARCON reference model. Furthermore, we also evaluate our framework with respects to questions and the criteria described earlier in this chapter.
- In Chapter 9, we provide a concluding discussion, discuss future research directions.

Chapter 2

Background

In this chapter we introduce the background of the field of virtual organizations and their breeding environments, SRML and APPEL languages, aimed at helping in understanding our research context and the VOML framework.

2.1 Virtual Organizations and their Breeding Environments

Virtual organization (VO) is a new and emerging discipline which manifest itself in a large variety of forms, including collaborative networks, virtual enterprises, dynamic supply chains, professional virtual communities, collaborative virtual laboratories, extended enterprises etc. The main reason for these different manifestations is that so far there is no consolidated definition for this paradigm. Scores of different terms are used in the literature that either refer to the same concepts or its different perspectives [18]. Rapid development of information and communication technologies have opened new ways of doing business and socializing never possible before. The facilities offered by this technological development have raised new challenges for business, societal and and scientific worlds as well. These challenges are forcing entities (organizations or otherwise) to seek complementaries and join efforts that allow them to better participate in challenging and competitive environments, expand businesses and enter new markets, compete against business giants by collective size and capacity of the partners. The concept of rapidly finding a set of complementary partners and quickly configuring them into a virtual organization that best addresses the business opportunity and challenges faced in the turbulent markets has raised considerable expectations in both business and other non-business contexts [26, 18]. A large variety of VOs have emerged as a result covering a wide domain spectrum. Some examples of VOs in the business sector can be found in [67]. [16] is one example in the domain of professional communities [8, 53] are example of VOs offering services. Other examples include [57]. The Grid-based technologies have also gained momentum due to their incorporation of the VO concept as a basic service provision mechanism [40, 38, 39].

A VO is a timely creation of a temporary alliance of geographically and temporally dispersed organizations/individuals that collaborate and share their resources to achieve some benefit not possible otherwise, with the possibility of partners being unfamiliar [28, 40, 19]. In the optimal scenario, this alliance shall help reduce production time, lower production costs and provide a better exploitation of market opportunities, among other potential benefits.

2.1.1 Virtual Breeding Environment

Although VOs offer many potential and tempting advantages such as their capability to be created quickly and on time to capture the business opportunity or to change swiftly to react to environmental changes, VOs alone are not able to offer such advantages. Finding the right partners and establishing necessary conditions for starting the collaboration process are costly and time consuming activities and inhibit the desired agility. Besides, companies are reluctant to open up their resources to possibly unknown partners of VOs due to a lack of trust. Beside being reluctant to trust strangers there is a very good chance that even when different entities do decide to trust and collaborate, interoperability and integration of different organizations is difficult [26].

This issue is addressed by the ECOLEAD project in the form of Virtual Breeding Environments [2, 17].

All organizations or individuals interested in forming a virtual alliance create a permanent alliance called a *Virtual Breeding Environment* (VBE). The VBE serves the purpose of taking all the measures necessary for timely creation of temporary collaboration, such as resolving different infrastructure integration issues of the member organizations, standardizing data interoperability mechanisms, setting trust and security policies for member organizations, etc. The VBE forms the Breeding Environment for Virtual Organizations. When an opportunity arises, suitable partners are chosen quickly from among the VBE members to form a temporary alliance called *Virtual Organization* (VO) and when the objective is accomplished the VO is either dismantled or re-structured to provide new functionality. This way of VO creation within a breeding environment has the advantage of agility; finding partners, negotiating collaboration terms and conditions, and resolving heterogeneity issues from infrastructure to information are more efficient. Organizations serving a purpose similar to VBE can be seen in many real world scenarios with many different names such as chambers of commerces, VO clusters, enterprise networks [59] etc,.

2.1.2 The Life Cycle Model

The life cycle [23] of the VBEs consists of three main stages: *creation, operation* and *dissolution*. The meaning of these stages is obvious from their names. In the *creation* phase a VBE is formed, its members are selected, the basic infrastructure and other facilities are laid down for the support and management VBE itself. VBE members, competency profile and other working principles are set up. The *operation* phase of VBE predominantly consists of creation and dissolution of VOs as per business opportunities and environmental changes. The VBE also keeps evolving itself by joining and leaving of its members and updating its competency profile, hence some research considers another stage called *evolution* stage. The *dissolution* phase rarely occur in the VBE where a VBE terminates its existence. Usually instead of dissolution a VBE goes through a *Metamorphism* stage where its general form and or purpose can evolve. This stage may involve the transfer of collected knowledge/information, as well as the members to a third party.

A VO exists and dismantles during the operation phase of the VBE. Most of the literature also suggest similar life cycle stages for the VO i.e. VO *creation, operation, evolution and dissolution* stages with similar meaning as for VBEs. However, since a VO exists to capture a particular business opportunity and dismantles when the opportunity is served or no longer exists, a VO hardly under goes metamorphism phase. Rather, a different VO is created to capture new business opportunity. Hence, a VO usually does not undergo the metamorphism phase. During the *evolution* phase a VO undergoes some changes that slightly modify the goal or the approach to achieve the goal is changed in some way; be this change in membership, structure or the competencies of the VO. These changes could be triggered as a result of changes in the market dynamics. However, the initial goal which prompted the creation of VO still remains the same.

2.1.3 VO Classification

A VO can be classified in many different ways according to the its different characteristics. Just as there is no universally accepted terminology or definition of base concepts for the field of VOs and VBEs, similarly, there is no universally agreed upon classification of VOs and VBEs. However, the most common classification characteristics are [7, 18]:

- *Duration*

  VOs can be *short term* (if made for a single business opportunity and dissolve at the end of such process), or *long term* (lasting for an indefinite number of business processes or for a specific time span).

- *Coordination*

  VOs can have many different coordination structures, such as *flat/democratic al-*

liance (all cooperating nodes work on equal basis maintaining their autonomy, but joining in their core competencies), *star-like* (there is a dominant company surrounded by relatively fixed networks of suppliers) or *Tree/Federation* (having formed some kind of common coordination structure by realizing common management of resources and skills).

- *Configuration flexibility*

  VOs can be characterized by the flexibility of partners ability to dynamically join/leave the VO. Those VOs that do not offer such flexibility are called *fixed or static* VOs and those that do are termed *dynamic* VOs.

Beside these general classification, we can find a through classification of VOs and VBEs and their different manifestations in [23, 24]. This classification captures subtle differences and similarities between them. The main similarity between them is that they all are created to offer a product/service through collaboration of their members. Hence all these forms are put under the umbrella of *collaborative networks*.

2.1.3.1 ARCON Reference Model

ARCON (A Reference model for Collaborative Networks) is a generic abstract model for understanding the basic concepts and main elements of any organization(s) showing some features of VOs or VBEs collectively referred to as *Collaborative Network Organizations* (CNOs) [2, 25, 20, 56]. It is developed in the ECOLEAD project [2]. It is intended as a guide to facilitate the derivation of focused model for various manifestations of VOs or VBEs pertaining to specific execution environment [24]. The ARCON reference model has attempted to consolidate different concepts developed for VBEs and VOs in different domains, in particular *business-oriented* view, *technology-oriented* view and to some extent *style-oriented view* which focuses on the social concepts such as culture, values, norms, principals, trust, etc. The ARCON framework is divided into three perspectives: *Life cycle stages* perspective addresses diversity and evolution of different VOs and VBEs during their life cycle which is same as given in Section 2.1.2. *Environmental characteristics* perspective is divided into two parts: (a) the environment internal to the VOs and VBEs (*Endogenous characteristics*) and (b) the environment external to the VBEs and VOs (*Exogenous interactions*). Internal environment refers to elements such as VBE participants, VOs, resources used by VBE and VO, processes and operations such as member registration, competency management, governance rules, roles in VBE and VOs, etc. Endogenous elements are divided into four dimensions as follows:

- **Structural dimension:** This dimension refers to those entities which constitute the VBEs and VOs such as participants, relationships among the participants, the role and tasks performed by the participants, topological organization of VBEs and VOs, etcetera.
- **Componential dimension:** Consists of elements that are used and managed by the other elements of the VBEs and VOs such as different types of resources such as human, software, hardware, knowledge, etc.
- **Functional dimension:** Functional dimension refers to different processes and operations performed in CNOs to manage different aspects of CNOs.
- **Behavioural dimension:** Addresses different principles, policies, norms, culture, etc that drive or constrain the behaviour of VBEs and VOs members.

External characteristics refers to the concepts such as how VBE is viewed by elements external to it and how those elements interact (operations) with the VBE. It consists of (a) *market dimension* which relates to interaction with both the customers and competitors of CNOs, (b) *support dimension* refers to services provided by third parties which help different aspects and elements of CNOs, (c) *societal dimension* refers to the impact that CNOs has on the society such as the impact on the environment, etc. Third perspective called *Modelling intent* explore different intents for modelling VBEs and VOs ranging from general representation models (called *general representation layer*) to specific models which are more detailed than general models (called *specific modelling* layer) and the models which are developed for specific execution environments called *implementation modelling layer*.

2.2 Business Process Modelling

Business process modelling languages describe the behaviour of a system with highest level of details such that the system can almost readily be executed on underlying execution environment. They describe the behaviour of a system into concrete operations and may also describe absolute orchestration between those operations. There are many process modelling languages but the most prevalent are activity diagrams of UML (Unified Modelling Languages) [32], BPMN (Business Process Modelling Notation) [13] and BPML (Business Process Modelling Languages). We, however have adapted Sensoria Reference Modelling Language (SRML) as our modelling language focusing on operational aspects of the VOs. The reason behind this decision is SRML's underlying mathematical basis and its support for different quantitative and qualitative analysis. The other reason for this decision is that components in SRML are business reflective i.e. they focus on the business functionality offered by the component and this is exactly what we want to represent at the operational level of the VOML models.

2.2.1 SRML: SENSORIA Reference Modelling Language

SRML [35] is defined under the umbrella of the SENSORIA project to operate at the higher level of abstraction of ”business” or ”domain” architectures.
Service and *Activity* are main modelling elements of the SRML; they are called *modules* in the SRML. A service that needs to be published and discovered when a request from external entities (other services) comes in, is modelled through service module of SRML. The activity modules are used to model applications that are developed to satisfy specific business requirements of a given business organizations. The activity modules are local to the given business organization.
In each case, the modules are composed of one or more *components* connected through *wires*. These components are classified as following:

- *Requires-interfaces:* Specify services, that are provided by external parties to the module themselves.

- *Uses-interfaces:* Specify resources used by the modules.
- *Provides-interface:* Possessed only by service modules; it describes the service that is offered by the service module. Each service module can at most have one provides interface.
- *Serves-interface:* Equivalent of *provides-interface* in activity modules.
- *Internal components:* Components other than requires, provides, uses, and serves are called internal components or simply components of the modules.

A Components is specified in terms of *Business role*; provides and requires interfaces are specified in terms of *Business protocols*; whereas, serves and uses interfaces specified in terms of *Layer protocols*. Wires on the other hand are specified in terms of *Interaction protocols*.

2.2.1.1 Interaction

Interactions are the basic means of communication between different components of the modules. SRML distinguishes between following types of interactions:

- *Asynchronous and Conversational interactions:* These interactions are called conversational as the party initiating the interaction expects a reply but does not blocks until the reply is received. In SRML, these interactions exchange many events between the communicating parties, but we only consider two events in our work (a) *interaction*🔔- the event of initiating *interaction*, and (b) *interaction*⊠- the reply-event of *interaction*. The following interaction types in this category are described from the point of view of the party in which they are defined:
    - **r&s :** This interaction is initiated by the co-party.
    - **s&r :** This interaction is initiated by the party.

- *Asynchronous and non-conversational interactions:* These interaction are called non-conversational as the party initiating the interaction does not expect any reply from the co-party and asynchronous because the party does not blocks until the reply. The following two interaction types fall under this category:
  - **rcv :** In this interaction the co-party initiates the interaction.
  - **snd :** In this interaction the party initiates the interaction.
- *Synchronous interactions :* Synchronous interactions are those interactions where the party synchronises with the co-party i.e. blocks until the co-party replies or completes. The following interaction types fall in this category:
  - **ask :** The party synchronises with the co-party to obtain data.
  - **rpl :** The party synchronises with the co-party to transmit data.
  - **tll :** The party requests the co-party to perform an operation.
  - **prf :** The party performs an operation.

2.2.1.2 The Business and Layer Protocols

In SRML, Business protocol types the requires- and provides- interfaces of a SRML module. It consists of a set of interactions called *signature* and a set of properties called *behaviour*. The signature lists the set of interaction the party, represented by the business protocol can engage in; while the behaviour, using these interactions specify either, what is required from the external services that need to be procured (in the case of requires-interface) or what is offered by the service orchestrated by the module (in the case of provides-interfaces. For instance, the service *TravelBK* specifies the following behaviour that it expects from a FlightAgent, given in figure 2.1 :

The first property specifies that the FlightAgent is ready to engage in *lockFlight* interaction (from its signature). The second property specifies that after receiving the commitment to the flight booking offer, the flight agent becomes ready to engage in the revoking of the offer from the customer until it is not already the day of departure.

```
BUSINESS PROTOCOL FlightAgent is
   INTERACTIONS
      r&s   lockFlight
            🔔 from,to:airport,
               out,in:date,
               traveller:usrdata
            ✉ fconf:fcode
               amount:moneyvalue
   SLA VARIABLES
      KD:[0..100],PERC:[0..100], MAX:[0..100]
   BEHAVIOUR
      initiallyEnabled lockFlight🔔?
      lockFlight✓? enables lockFlight✝?
      until today+KD ≥ bookTrip.out
```

Figure 2.1: A SRML Business Protocol

A property in SRML is defined using following patterns whose semantics have been defined in terms of formulas of the temporal logic UCTL [62]. The following are most commonly used patterns in SRML.

- ***initiallyEnabled e*****:** The behaviour of any protocol always starts with this sentence pattern. The property implies that the event *e* is enabled in the initial state and remains so until it is executed.

- ***a enables e*****:** This pattern implies that the event *e* can only be executed after the event *a* holds .

- ***a ensures e*****:** This patterns implies that the event *e* can only be published after the event *a* holds.

The Layer protocol on the other hand types the serves- and uses- interfaces of a SRML module. Like business protocol, a layer protocol consists of a signature and behaviour part. However, the interactions used in a layer protocol are usually synchronous and blocking as opposed to asynchronous interactions of business protocol.

2.2.1.3 Business Role

Business roles type the components (internal) of the SRML modules. Like protocols, the business role consists of set of interactions called signature; unlike protocols a business role describes how the interactions are orchestrated using states and transitions. The states model the control, defined in terms of a state chart as shown in the Figure 2.2.

The control flow is represented through a special variable (*s* in the Figure 2.2), which

```
ORCHESTRATION
   local s:[START, QUERIED, FLIGHT_OK, CONFIRMED, END_UNBOOKED, END_COMPENSATED],
         hconf:hcode
```

Figure 2.2: Local state of Orchestrator

basically describes the way the component (orchestrator) reacts to triggers (described below). The other state variables are used for storing data required during different stages of the orchestration.

The actions performed by component are represented through transitions as shown in Figure 2.3. A transition has an optional name and a number of possible features as follow:

- The `triggeredBy` feature represents the trigger which enables the transition to be executed.
- The *guardedBy* feature denotes the guard which is a condition that identifies the states in which the transition can occur. The transition is not executed even if the triggering event of the transition holds, unless the guard holds as well.
- The *effects* feature describes one or more sentences that specify the effects of the transition on the local state and other local variables i.e. a given state or local variable *var*, changes its state/value to *var'* to denote the state that the

```
transition Request
   triggeredBy bookTrip🔔
   guardedBy s=START
   effects
      hconf'=findHotel(bookTrip.in,bookTrip.out)
           ∧ hconf'≠NIL ⊃ s'=QUERIED
           ∧ hconf'=NIL ⊃ s'=END_UNBOOKED
   sends hconf'≠NIL ⊃ bookFlight🔔
           ∧ bookFlight.from=bookTrip.from
           ∧ bookFlight.to=bookTrip.to
           ∧ bookFlight.out=bookTrip.out
           ∧ bookFlight.in=bookTrip.in
           ∧ bookFlight.traveller=bookTrip.traveller
      ∧ hconf'=NIL ⊃ bookTrip✉ ∧ bookTrip.Reply=False
```

Figure 2.3: A Transition in SRML

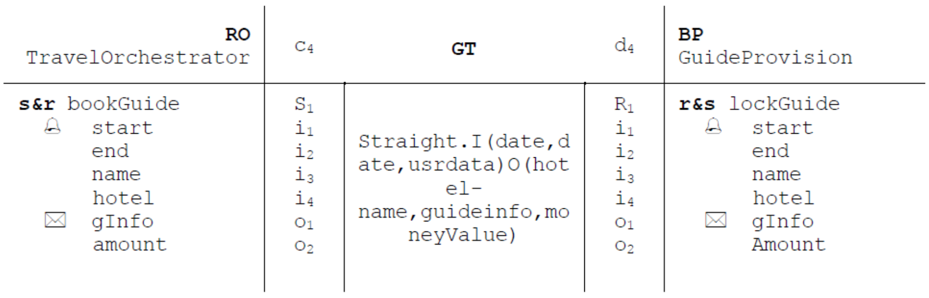
component moves to after the transition or the changed value the local variable gets as a result of some computation.

- The *sends* specifies the events that are published during the execution of the transition.

2.2.1.4 Wires

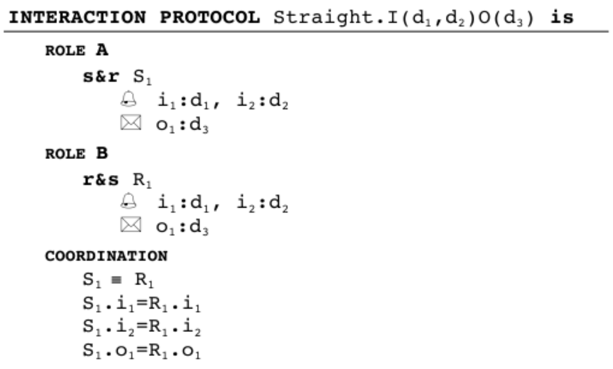
The specification of wires in SRML consists of rows as shown in the Figure 5.5. Each row is typed with what is called a *connector*. A connector coordinates an interaction (and its parameters) between two parties joined through the wire. The main ingredients of connector are two roles (A and B) and an interaction protocol. The interaction protocol describes the coordination behaviour between the roles, which in turn are mapped to the actual parties' interactions. The semantics of the coordination behaviour (i.e. how the interactions are coordinated) of the interaction protocol is provided through a collection of sentences called *interaction glue*.

As an example, consider the following protocol used in the wire that connects two parties. This is a `straight` protocol that connects two entities over two conversational interactions that have two 🔔-parameters and one ✉-parameter. The property S1 $\equiv$ R1



RO TravelOrchestrator	c4	GT	d4	BP GuideProvision
s&r bookGuide 🔔 start end name hotel ✉ gInfo amount	S1 i1 i2 i3 i4 o1 o2	Straight.I(date,date,usrdata)O(hotel-name,guideinfo,moneyValue)	R1 i1 i2 i3 i4 o1 o2	**r&s** lockGuide 🔔 start end name hotel ✉ gInfo Amount

Figure 2.4: A SRML wire



```
INTERACTION PROTOCOL Straight.I(d1,d2)O(d3) is
  ROLE A
     s&r S1
        🔔 i1:d1, i2:d2
        ✉ o1:d3
  ROLE B
     r&s R1
        🔔 i1:d1, i2:d2
        ✉ o1:d3
  COORDINATION
     S1 ≡ R1
     S1.i1=R1.i1
     S1.i2=R1.i2
     S1.o1=R1.o1
```

Figure 2.5: A Interaction Protocol

establishes that the events associated with each interaction are the same, for example that S_1🔔 is the same as R_1🔔.

2.2.1.5 Configuration Policies

SRML deals with the issues such as processes of discovery, selection and instantiation of services with what are called Configuration policies. Furthermore, the policies which address issues relate to the service instantiation such as the initialisation of service components and the triggering of the discovery of required services by policies called *internal configuration* policies. These policies are denoted by the symbol 🕛 in diagrammatic representation of the modules. The discovery, negotiation and selection of external service that need to be procured by the module are the issues that are addressed by the *external configuration* policies. The external configuration policy is represented by the symbol in the diagrammatic representation of the module and it lists a set of variables that establish a service level agreement (SLA) during negotiation, and a set of constraints that are taken into account during the discovery and selection of external parties. The approach taken by SRML for service level agreement is based on constraint sanctification and optimization framework of [10].

2.3 Reconfiguration Mechanisms

Policy languages are used for defining policies for different purposes from access control, distributed systems, to security and management of distributed systems. They are used to modify the behaviour of the system [49]. They describe information specifying the user requirements, preferences and constraints [41]. Some examples of policy languages are Ponder [29], KAoS [63], Rei [45], XACML, and WSPL. Each of these languages addresses a specific aspect(s) of a specific domain. Though any of these languages could be used for those aspects of virtual organizations which are common between virtual and other traditional business domains; there are certain aspect which are particular to (or at least become more paramount in) the domain of VO. One such example can be policies for evolution. This need is convincing enough to look for policy description which directly addresses VO needs. Having a policy description which directly addresses the main notions of VO, complements and enhances the expressiveness of the modelling language for Virtual Organization.
While policies provide a feasible approach towards reconfiguration, a special policy language is required which provides explicit constructs suitable for the domain of VO reconfigurations. Typically a policy language can be completely domain specific or a more generic ECA (event-condition-action) based language can be adapted to new domains by providing appropriate actions, triggers and conditions. It is the latter approach that we follow by adapting the APPEL (Adaptable Programmable Policy Environment Language) as our reconfiguration language [48].

2.3.1 APPEL

The APPEL (Adaptable and Programmable Policy Environment and Language) had been developed with a clear separation of domain and core language. APPEL has a style closer to natural language as it was aimed at use by non-technical end users, rather than for developers or system administrators, something which should benefit the users of VO modelling languages. APPEL was originally aimed at call control, but has since been specialised for domains such as sensor networks, elderly care, and now VOs. The *Policy rules* are the basic building block of APPEL. To group more than one policy rule in a single policy a number of operators (sequential, parallel, guarded and unguarded choice) are offered. A policy rule is a variant of an ECA rule, consisting of an optional trigger, an optional condition, and an action. The applicability of a rule depends on whether its trigger has occurred (if one is defined) and whether its conditions are satisfied. A condition expresses properties of the state and the trigger parameters (in some domains it is natural to have triggers with parameters, such as the to and from in a call request). **and, or** and **not** operators are provided to combine more than one condition, with the operators having the expected meaning. Actions have an effect on the system to which the policies are applied. Several operators are available to compose actions: **and** leads to the execution of both actions in either order, **andthen** specifies that the first action precedes the second in any execution, **or** specifies that either one of the actions should be taken, and **orelse** that is like or but prescribes that the first option is preferred. Triggers and actions are domain specific atoms. Conditions are either domain specific or a more generic (e.g. time) predicates. The core of APPEL has been given a formal semantics in [50].

A typical APPEL policy looks as follows:

```
policy policy-name appliesTo task-id/member-id/VO-id
when optional trigger(s)
if   optional condition(s)
do   action(s)
```

The specialization of APPEL to Virtual organisations forms the Virtual Organisations Reconfiguration language VO-R. For VO-R we have added a number of triggers, conditions and actions specific to VOs. We also adapted the meaning of the localisation: `appliesTo` allows to 'locate' the elements of the VO, on which the given policy applies. The values for `appliesTo` can be a VO task or its membership, in our case. In distributed settings this would allow to specify the location of a policy, for VOs this allows to express which task (or tasks) a policy applies to.

2.4 Related Work

We have argued in the thesis statement (Section 1.5), there is no general modelling language for specifying VOs using domain terminology. Although some research focus on formalizing VO model in order to carry out different sort of analysis or to analysis different properties of VOs using already existing formal languages and tools. Whereas, some researches have focused on providing technology oriented solutions [55] and some research efforts have focused on providing general reference models for the VBEs and VOs. However, the approach taken to address the research challenges set up in the Section 1.1 helps to justify our research in terms of above mentioned and some other work which we are going to present in this section.

2.4.1 Modelling Approaches

Number of ad-hoc attempts for modelling VOs and VBEs exists in the literature. Some are limited to preliminary modelling of VO concepts [52, 51], while some have put dedicated amount of effort in this direction [23, 24, 14, 44, 30].

2.4.1.1 Dynamic Coalitions

This work has been motivated by work within the GOLD [3] project, which seeks to build an architecture to facilitate the creation and maintenance of Virtual Organisations within the Chemical Engineering sector [27]. In this work a VO is modelled as dynamic coalition using the Vienna Development Method (VDM) specification language [37, 36]. It defines a VO consisting of choices made in five orthogonal dimensions including *membership, information representation, provenance, time* and *trust* [44]. This work is mainly interested in the flow of information around models of coalitions (VOs) [14]. For example, identifying states of formal models in which information has reached the wrong actor (one who is not supposed to get the concerned information), or where information has not reached the right actor (one who is supposed to get the concerned information). These models are analysed to identify these states. In each dimension, information is distinguish between the one which is related to the the material traded between agents in a coalition (which we refer to as business functionality) and the one about the agents, coalitions or information itself so called *meta-information*, such as the age of a piece of information, or the identity of a coalition. Each dimension corresponds to a form of meta-information and the models make that meta-information explicit. In each model, consideration of the invariants, preconditions etc. leads to alternative models representing design choices. The result of this analysis is a suite of models that deal with individual dimensions and present the coalition architect with a range of design alternatives, allowing a particular architecture to be placed within the space of coalitions.

Being a formal model different kinds of analysis and verification are possible. Our work differs in the sense that it tries to develop a modelling language with the level of abstraction raised to a point where it is possible to directly support notions and concepts that are paramount in the domain of VO such as a VO consisting of different permanent and transient members and resources being utilized, the relationship between the members.

The focus of the dynamic coalition is the meta-information and the information related to the business functionality is abstracted away to support the architect in designing coalition structures while the focus of the VOML is the information related to the business functionality and developing business applications using VO terminology. Dynamic coalitions model VOs using languages well know for the analysis purpose and supported by tools for such analysis. The trade-off made for such analysis is that domain concepts are either ignored or defined using the terminology of the language being used for analysis. Mapping is used to make the analogy between domain concepts and the constructs available in the underlying language used. The VOML framework complements such efforts by making domain concepts available as language constructs.

2.4.1.2 Research Supporting Replication or Composition of Entities

The provision of constructs in a language that allow sharing a task between more than one entity or decomposing a task into sub tasks is, to the best of our knowledge a novel construct in the VOML framework. However, the concept itself can be traced in other work weather implicitly or otherwise. In particular, we are going to discuss two research efforts which justify the VOML's such constructs in the area of VO.

- **MetaSelf Architecture:** The MetaSelf attempts to design distributed systems that can dynamically adapt in a predictable way to react to unexpected events by self-organizing and self-adapting [31, 60]. The Metaself attempts to add resilience to systems that reside in dynamically changing environment, where the components required by the system arrive, depart or get modified during the course of systems' existence. To ensure that such systems would respect key properties during the dynamic evolution, MetaSelf provides the solution in the form of system architecture with architectural patterns called *dynamic resilience mechanisms (DRMs)*. The DRMs use run-time information to maintain resilience through adaptation, e.g. by dynamically composing a satisfactory service from lower-specification components. This allows the systems to evolve dynamically to offer a continued, if necessary (predictably) degraded, service using the resources available at the time the negative event is detected.
  DRMs rely on the availability at run-time of resilience metadata information about system components, sufficient to govern decision-making about dynamic reconfiguration. The metadata is used to guide reconfiguration in accordance

with resilience policies (e.g. to increase the number of alternate services if availability starts to decline). A run-time environment acquires, maintains and publishes metadata. Hence, this approach allows making dependability-maintaining adaptations at run-time.We can relate to this work in two ways: first it validates our concept of sharing a task between more than one member or dividing a task into sub tasks such that their emergent behaviour conforms to the original description of the task. Second, rather than proving a fixed predefined solution the Meta-self system can react differently to the same problem using resilience policies; we are able to do the same using the VO-R language.
However, the VOML framework goes a step further by providing a modelling language with built-in constructs for such task distribution. Furthermore, it also reflects the affect of such evolution on the underlying models of the VO (VO-S and VO-O).

- **Conoise-G:** The CONOISE-G (Constraint Oriented Negotiation in an Open Information Services Environment for the Grid) project focuses on providing a technology-oriented solution for formation of reliable and scalable virtual organizations in a dynamic, open and competitive environment [1]. It provides agent-based infrastructure to support robust and resilient VO creation, operation, evolution and dissolution [55, 53, 54, 61]. In the Conoise-G architecture a VO consists of set of autonomous agents, each with some capabilities and resources, who have come together for obtaining potential benefits. When the business context changes, the VO either disbands or rearrange itself to better fit the new circumstances. It is this rearrangement that we are interested in. For example a service which demands 50 minutes per month of *Entertainment* (video clips) than this service demands can be provided by two service providers if a single provider can not satisfy the number of minutes of video clips [55, 53, 54, 61]. This corresponds to replication of task based on capacity (minutes) in our work. Our work aims at developing a reconfigurable modelling language rather then providing a platform and technology specific solution. Another difference is that their reconfiguration is limited to dynamically replacing one member with another having similar behaviour and capabilities whereas we also allow sharing a task between members with each contributing either part of the capacity required or different capabilities required for the task.

2.4.1.3 ASSL: Autonomic System Specification Language

The ASSL is dedicated to the development of self-managing autonomic computing [64, 65]. It offers a formal specification framework for specification and code generation of autonomic system [46, 47]. The ASSL specification model consists of multi-tier specification of autonomic systems. These tiers decompose an AS into two levels: first into levels of functional abstraction and second into functionally related sub-tiers [46, 47]. The first level decomposes the system into further three tiers:

- **Autonomic System (AS):** The AS tier defines the global perspective of an autonomic system in terms of system rules, architecture, actions, events and metrics which must be achieved by the constituent elements (AEs) of the AS.
- **AS Interaction Protocol (ASIP):** The ASIP defines the global communication protocol between AEs (autonomic elements).
- **Autonomic Element (AE):** The AEs are individual computing elements which form the overall AS. These interacting set of individuals are defined with their own behaviour, which must be synchronized with rules from the global AS perspective.

The second level further decomposes the above three tiers consisting of elements relevant at each tier, for example the AS tier consists of service level objective, self-managing properties, AS architecture, actions, events and metrics. From ASSL specification the JAVA code is generated.
The VOML framework follows same style of specifying systems i.e first a VO is defined in terms of domain defined abstract languages (VO-S and VO-R) and from that specification an implementation oriented code is generated. However, unlike ASSL's

approach of providing a specific language code, at the implementation level, we define our VO in another modelling languages which is neutral to any technology, but detailed enough to be readily used by the underlying execution environment. The language used for implementation level description is adaptation of SRML, which having an underlying mathematical background opens ways for performing analysis and verifying different properties.
The other deviation that we see in ASSL and our framework is that ASSL specification style abstract away from the functionality offered by resources they are managing. This in our understanding, implies that an assumption is made in ASSL framework that the behaviour of the managed resources does not change and neither does a resource get replaced by two or more resources whose collective behaviour corresponds to the behaviour of replaced resources; ASSL also does not account for the behaviour that gets affected due the structural changes made through policies by the ASSL. The agility and flexibility demands of VOs and VBEs on the other hand compels us to consider the behaviour of the entities (resources or other wise) as first class entity. When any entity is unable to provide the behaviour required for some reason than VO should be flexible to replace it with two or more entities whose collective behaviour satisfies the behaviour of the replaced entities. This replacement consequently changes the structure of the VO and eventually the operational structure of the VO (at the VO-O level) as well. Hence, the VOML framework considers both the VO specification in terms of VO concepts, its reconfiguration and the reconfiguration affects on the underlying entities.

2.5 Summary

This Chapter described in detail the background literature of VOs. It also explained the SRML language which has been adapted and extended by the VOML framework for VO-O language. The APPEL policy language is also introduced in this chapter which has been extended for the VO-R language.

In addition the VOML framework was compared with a few literature that could be related to the approach taken in the VOML framework.

Chapter 3

The VOML Framework

This chapter attempts to help the reader understand the VOML framework in detail. The VOML framework defines different levels of representation of VBEs, VOs and their activities. The focus of this thesis however, has been virtual organizations, but, since a VBE represents the organizational context in which the creation and operation of VOs takes place different aspects of VOs are described in terms of different levels of VBE representation. Hence, this chapter first introduces relevant aspects of the VBE level and then building on that, it describes different aspects of VOs.

In synopsis, VOML supports the definition of a structural and behavioural model of a fixed VBE based on three different levels of representation: the *persistent, business configuration* and *state configuration* levels. A VO is described at the business and state configuration levels. Each level focuses on specific aspects of a VO using different modelling languages namely the *VO-Structural, VO-Reconfiguration* and *VO-Operational* modelling language.

Moreover, we also introduce our two case studies here, one of which is going to be used through out the remaining chapters to elaborate on different segments of our research work and the other one will be used for the evaluation of the research work.

Parts of this chapter have been published in [12].

3.1 Representation Levels of VOML

The VOML framework consists of a number of languages addressing different levels of representations. Over these levels VOML, captures different aspects of VOs and VBEs using modelling languages specifically developed to capture the desired level of details at that level. These levels are not 'architectural layers': they do not contain entities that interact with entities in other layers. Rather, they represent a hierarchy of representations at a fixed time.

- At Level (1) *persistent* functionalities of the VBE are described such as its current partners, assets (resources and supporting tasks), competencies and governance policies. This level is invariant, i.e. it provides a representation of those aspects of a VBE that will not change. Focus of our research has been on modelling different aspects of VOs therefore, we have not modelled persistent level using any modelling language, it seems plausible that the VO-S language can be used to model persistent level.
- At the level (2) the *transient* functionalities of the VOs that are offered by the VBE at a specific moment in time are defined, what we call a *business configuration* of the VBE; this level captures the way a VBE is logically organised at that time in terms of VOs and VBEtasks. It is at this level that we define the structural aspects of the VOs using the VO-S modelling language and any changes (reconfigurations) induced in the VO structural model through the VO-R modelling language. The model describing structural aspects of VO is called *Structural model.*
- At level (3), the ensemble of 'components' (instances) connected through 'wires' that, at that time, deliver the services offered by the VOs present in the business configuration, what we call a *state configuration* are defined. The state configuration represents the actual 'physical' instances of the VOs that are currently operational, i.e. which specific services are currently being provided within the

VBE. ‘Real’ entities are only represented in state configurations: the other levels deal only with types of entities.
The components and wires representing VOs focus on the coordination and communication aspect and how their particular organization realizes the service promised by the VO. The language used for the description of components and wires is called VO-O modelling language and the model describing VOs in terms of components and wires is called *operational model*.

3.1.1 Formal Model of the VOML Framework

More specifically, the three levels of representation are modelled as follows:

– **Persistent Level**
  A VBE consists of (1) a collection of *resources*; (2) a consortium of (persistent) *partners*; (3) a number of *policies* constraining the way resources can be shared and the partners agree to do business together, including rules for the consortium to expand for establishing specific VOs; and (4) a number of supporting tasks (VBEtask) that operate processes (management or otherwise) that serve the roles enacted within the VBE. These constituent elements are invariant, i.e. they are present in every business configuration of the VBE.

– **Business Configuration Level**
  The current business configuration of a VBE, is understood as (1) the collection of additional (non-permanent) members, that we call *associates* and *external entities (ExtEntities)*, and resources that are part of the VBE; (2) the tasks (VBEtasks) that support the roles of the new partners and their resources; (3) the VOs that the VBE currently supports; and (4) the policies that apply to their instantiation and their coordination at any given time. VBE support tasks and VOs may rely on complementary, transient partners (which we call ‘associates’) that join the VBE to provide specific

business services and remain in the VBE only while those services are required. Associates can be fixed at VO-creation time either at the discretion of customer or when a VBE does not have a suitable partner for a VO, while ExtEntities are discovered on the fly when needed, subject to service-level agreements, in order to be able to accommodate the needs of specific clients.

– **State Configuration Level**
  The current state configuration of a VBE consists of ‘components’, connected through ‘wires’, that jointly operate the VBEtasks and the services offered by the VOs that are running in the current state. These components include the shared resources of the VBE as well as those that are brought into the VBE by the associates and external entities. The topology of the configuration (the way components are wired together) reflects the policies established at the level of the current business configuration. At this level, one can determine levels of resource consumption or properties of a number of other parameters, including measures of quality of service.

3.1.2 The VOML Modelling Languages

The business configuration level and state configuration level, each comes with dedicated languages suited for the level of details that matter there. These languages are discussed in detail in the coming chapters, but very briefly, at the business configuration level we focus on the structure of the VOs at the highest level of abstraction using domain terminology. At the level below i.e. at the state configuration level we focus on the operational model of the VO providing sufficient details to realize the VO with respective to an underlying execution environment. The VO-R language triggers changes mostly at the VO-S model level, but some aspects could apply directly at the VO-O level. Clearly changes at the VO-S level impact upon the VO-O model.

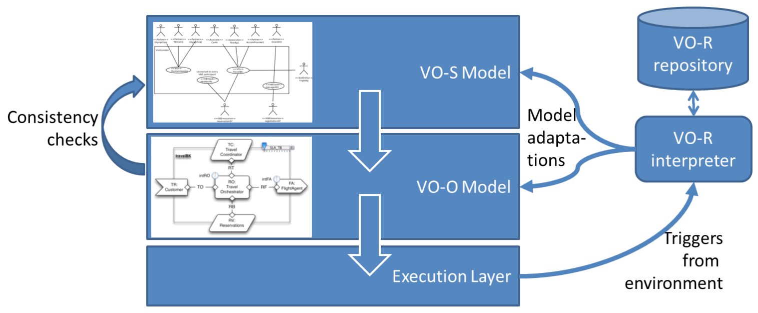


Figure 3.1: The Overview of the VOML framework

To clarify the relationship between different languages, we are going to describe the general description of architecture and process that shows how different models and languages relate to each other.

Very briefly, structural models present the general structure of the VO; these are mapped into operational models. The operational models are quite close to respective execution frameworks (such as agent based systems or service oriented systems) and can be mapped to these. The execution framework is monitored and any changes are reported back to the model levels where policies are activated to refine the VOs and ensure that they remain competitive. Reconfiguration rules are checked for consistency and furthermore any VO-model should be correct by construction (that is it should be a true refinement of the respective structural model) however this is further ensured by consistency checks. Figure 3.1 depicts this overview in a graphical manner.

3.1.3 Rationale Behind the VOML Framework

This different levels of representation enable us to focus on different dimensions of VOs at each level individually. For example at the business level we talk about the concepts which are specific to VOs irrespective of the functionality VOs offer. This dimension is captured through the VO-Structural Language (VO-S). At this level we are also able to talk about the adaptability needs of VOs in general; we cover this dimension through our VO-Reconfiguration Language

(VO-R). At the state configuration level we focus on the business functionality offered by any VOs, in sufficient detail to allow for ready execution. We have developed VO-Operational Language (VO-O) for describing this business functionality.

Models at the state configuration level are derived from the information available in the VO-S models. One particular structural configuration of the VO-S model gives way to a set of its operational (instance level) configurations. Changing the structural model through reconfiguration might invalidate some or all of different operational configurations possible from the previous structural model and allow for new set of valid operational configurations. This situation is shown in Fig 3.2.

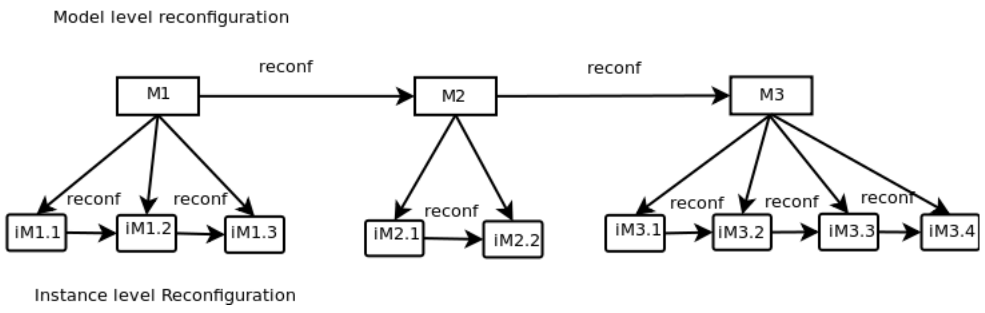


Figure 3.2: VO Reconfiguration

This particular organization of VOML framework also resolves the issue of fixed organizational structure and only looking at systems either at a very abstract level (with the highest level of reconfiguration flexibility) or a very concrete level (having a fixed organizational structure thus making any changes in the structure very difficult), but missing out the whole picture. This issue was discussed in detail in Chapter 1. This problem is resolved by the VOML framework; it gives us different levels to describe two different (but related) aspects of the same system (VO) using dedicated languages fit for the purpose. Furthermore, it provides the level of flexibility needed by the VO domain to get away with the limitation of fixed organizational structure (operational model). This has two fold effects: (1) we are able to address most of the agility and flexibility demands of VO by incorporating reconfiguration changes at the abstract level (VO-S), then (2) using mapping rules, we are able to make the required changes at the concrete level (VO-O). Part of the importance of distinguishing between these three levels is that we can account for two different kinds of change (assuming that the VBE level is invariant as we do not model the creation of VBEs) at the structural and operational level respectively:

- Changes in the business configuration reflect the creation or deletion of VBEtasks or VOs. Creating a new VO may involve identifying associates external entities or the criteria that will need to be observed for discovering such external entities on the fly, depending on the nature of the customers that procure the service (in which case each VO instance may involve different associates and external entities). Deleting a VO requires that the current state configuration is in a quiescent state relative to that VO, i.e. that none of the services offered by the VO is currently active. Changes at this level are triggered by external business concerns.
- Changes to the state configuration result from the launching of (instances of) VBE tasks or of services provided by one of the VOs present in the business configuration, which dynamically adds (or removes) components or wires to (from) the current state configuration. Changes at this level are triggered by the actions performed by or through the components and the communications exchanged through the wires that connect them.

Given the way levels are organised, these changes take place in different 'time-bands' in the sense of [15], i.e. the levels induce different granularities of time: the state-configuration changes take place within a fixed business configuration, meaning that business configurations induce a coarser time scale.

3.2 Basic VOML Modelling Elements

In this section we describe the basic elements the VBE comprises of. Each VBE consists of a number of participants, competencies, VBEassets (VBEtasks and VBEresources), VOs and policies concerning VBE's different aspects.

- **Participants:** Each VBE is a coalition of participants which could range from individuals to organizations. These participants are willing to cooperate by forming VOs. A participant in a VBE and consequently in a VO can be either a *Partner*, an *Associate* or an *ExtEntity*.

  A **Partner** is assumed to be a *permanent* member of the VBE. A partner is in the VBE because they are willing to cooperate in different VOs whenever an opportunity arises which requires their capabilities. It may be involved in more than one VO at any time and might be sitting idle at other times.

  An **Associate** on the other hand is one who has not joined the VBE before hand; rather it has been invited into the VBE because either its capabilities are required by one or more VOs/VBEtasks or it has been specifically demanded by the customer of the VO. An associate's participation in the VBE is bound to the duration in which its capabilities are required. When its capabilities are no longer required an associate has to leave the VBE (unless invited by the VBE itself to become a partner).

  Provision of partners and associates to VOs is the responsibility of the VBE. However, a VO can opt for a member from the open universe as well. A member of a VO from the open universe is termed **ExtEntity** in the model. Inclusion of an ExtEntity is acknowledgement of the fact that sometimes it is beneficial to choose members as per the customer demands or business goal of the VO. The customer only specifies its preferences rather than actually providing the member (in which case the member would be termed an associate). For example the flight booking agent of the TRavelBK case study is discovered dynamically in order to maximize the customer satisfaction.

  Associates and partners are chosen for VOs at its creation whereas ExtEntities are discovered dynamically from the open universe.

Distinguishing between partners, associates and external entities allows to not only identify permanent and transient members but also allows VBE to impose different constraint on them. For example a VBE can impose a governance rule that no associate is allowed to perform managerial task or there could be a rule in VBE that says if a particular VO task require a capability which is not listed in VBE competency list then an external entity could be hired.

- **VOs:** The VBE offers its services to the outside world through the creation of the VOs, it is the only offering made available for the VBE customers and hence the focal point of the VBE. A VO consists of a set of tasks that the members of the VO need to perform which leads to the satisfaction of the goal the VO is created to achieve. The managements aspects of the VO itself such as its creation, deletion, modification in structure, goal or membership, or any other aspect is managed by a special partner of the VO, called the coordinator of the VO. A VO also gets support from the VBE either by exploiting VBE's assets (VBEasset) such as its resources (VBEresources) or supporting tasks (VBEtasks). In the subsequent chapters the modelling languages for the description of VOs (and VBEs) are going to be discussed in detail.

- **Special Roles:** Following are two special roles that exists in VOML framework:

  - **VOcoordinator:** Every VO has one VOcoordinator who transforms the identified business opportunity into the goal that the VO has to achieve. The VOcoordinator is responsible for coming up with the plan to satisfy the goal, identify the skills, capabilities and resources needed to achieve it and selects the members for the VO. It is the VOcoordinator who steers the VO with several exclusive rights, such as changing the membership of the VO or modifying its goals (i.e., end products or services offered). In VOML the VOcoordinator consolidates several distinct organizational roles, such as broker, planner and coordinator.

  - **VBEmanager:** A VBEmanager is responsible for the management of the whole VBE. A VBEmanager is responsible for the creation, dissolution and updating VBE's competency lists, inviting or expelling members of the VBE and developing and code of conduct for VBE members, VO creation, VO termination and other aspects.

- **Competency:** A competency in a VBE mainly addresses the capabilities and capacities of the member entities [34, 9]. Members are chosen for a VO based on their competencies. A Capability refers to the *ability* of an entity (organization or otherwise) to manage and exploit its resources [33]. Whereas, capacity refers to dynamically changing properties of the capability i.e *availability* of resources used by the capability [34]. Hence, a competency is a catalogue of capabilities and capacities of VBE participants in a particular domain. A VBE usually has an associated list of commonly used capabilities in that particular domain. A member of the VBE must possess one of the capabilities mentioned in that list to become a member of the VBE. There might be a member available for a capability in the VBE or not at any particular time. If there is no member offering a particular competency required by some VO, and the capability is listed in the competency list of the VBE, then the member for that capability is included in the VBE as an associate if required in the business configuration.

- **VBEasset:** VBEasset consists of collection of resources and support tasks that are offered by the VBE to its participants and VOs to carry out different jobs and for the management of VBE itself. These assets are internal to the VBE and not accessible by entities outside the VBE. It consists of:

  - **VBEresource:** A VBEresource is one of the assets of the VBE that it offers to different VOs and Participants.

  - **VBEtask:** The VBEtask refers to processes that support the administration of the VBE, management of its resources and participant, roles of new participants and resources, etc. A VBEtask might also be created only to cater for the needs of a specific VO as well. The difference between VBEtask and VO is that VBEtasks are not available to the outside world and they only serve the VBE, its members or VOs. Whereas, a VO offers its services to the outside world and to provide that service a VO might use some of the VBEtasks.

- **Policies & Constraints:** Policies and constraints at the VBE level define the governance rules of the VBE. These policies lay down rules such as expected behaviour from the VBE members, rules for joining or leaving the VBE, penalties for violating the policies, etcetera. These policies and constraints also shape the kind of VOs or different relationship that can exists between members of the VBE or VOs, such a no two VOs in the VBE offer same product or service, implying a VBE without competition. Some policies might establish that a member can not be part of more than a certain number of VOs to ensure fairness to all its members.

3.3 The VOML Framework in the Context of the VO Life Cycle

Having provided an overview of our framework, we will now place this into the context of the different VO life cycle phases showing which parts are supported by languages from the framework, and how.
A VO inception starts at the business configuration model relying on functionalities and resources offered by a VBE available at persistent level. A VO is created when a *VO coordinator* identities a potential for a new business opportunity that falls under the niche domain in which the VBE operates. The identification process of business opportunity for VO creation is beyond the scope of this framework. However, the process for new VO creation is initiated by a VO coordinator when he sees an opportunity or it can be brought to him by external entity (future customer) with specific

demands. The VO coordinator can be any member of the VBE or the VBE might have set certain criteria for a VBE member to enact the role of VO coordinator through its policies and constraints. From the business opportunity the VO coordinator identifies the basic building blocks of new VO being created and describes these using the VO-S modelling language. This description specifies the required competencies, capabilities, abstract process description, members roles and relationships, and VBE assets required from VBE. A VO coordinator also selects *partners* and *associates* from the VBE for specific tasks of the VO.

A VO coordinator is a stakeholder of the business domain and does not necessarily possess skills to describe the concrete operational model which could actually be executed to satisfy the business opportunity. Hence, the structural model acts as a requirements specification and blue print for the operational model using the mapping rules provided in the Chapter 5. Using the VO-S model and mapping rules as guidelines an operational model is developed by the stakeholders in the IT community. This operational model is defined using the VO-O language and is quite close to respective execution frameworks (such as agent based systems or service oriented systems) and hence can be readily mapped to these. The operational model is defined at the state configuration level; each VO customer is bound and served by creating a new instance of VO-O model. External entities (member type) are also discovered from an open universe (outside the VBE) and bound to each specific VO-O instance. The termination of a VO-O instance occurs when the customer bound to that instance has been served and all the VBE assets consumed by that instance are released and external entities also leave. The termination of a VO-O instance does not imply dissolution of the VO itself; a VO is dissolved by the termination of the VO-S model which would be a business decision taken by the VO Coordinator according to rules and regulations of the VBE. A VBE advertises the services offered through its VOs available at the *business configuration* level. A VO exists as long as its VO-S model exists at the *business configuration level*. A VO serves a customer through a VO-O instance at the *state configuration* level and this instance ceases to exists when the customer is served.

A VO is evolved by reconfiguring it structural and operational models. The reasons for reconfiguration range from optimization of the VO, re-aligning the goal of the VO to address a change in the business opportunity or market circumstances, to avoiding the VO failing in providing its goal. All these reasons of reconfigurations are defined from the perspective of the business domain, and hence it is the responsibility of the VO coordinator to describe the situations where a VO needs to reconfigure and what course of action needs to be taken. For this purpose the language for reconfiguration (VO-R) is at the same level of abstraction as VO-S. During the life-time of the VOs the execution framework is monitored and any changes are reported back to the model levels where policies are activated to refine the VOs and ensure that they remain competitive. The changes made to the VO-S model affect the VO-O model in such a way that new instances will be following the new model, while any existing running instance will remain untouched.

3.4 Case Study

In this section we introduce the two case studies which are going to be used to explain the modelling constructs of the different languages and to demonstrate their effectiveness at capturing different features of the VOs. In this section we also justify the selection of these two case studies by exploring the different circumstances these VOs go through from creation to dissolution with respect to the criteria mentioned in the Section 1.2.

3.4.1 Running Example: TravelBK VO

Consider a VBE named *VisitLondon* (Figure 3.3) which offers different leisure and entertainment services to its customers, who are usually tourist. The current members (partners or associates) of the VBE are: a group of hotels, a car rental company, different travelling companies taking tourists in and out of London, telecommunication

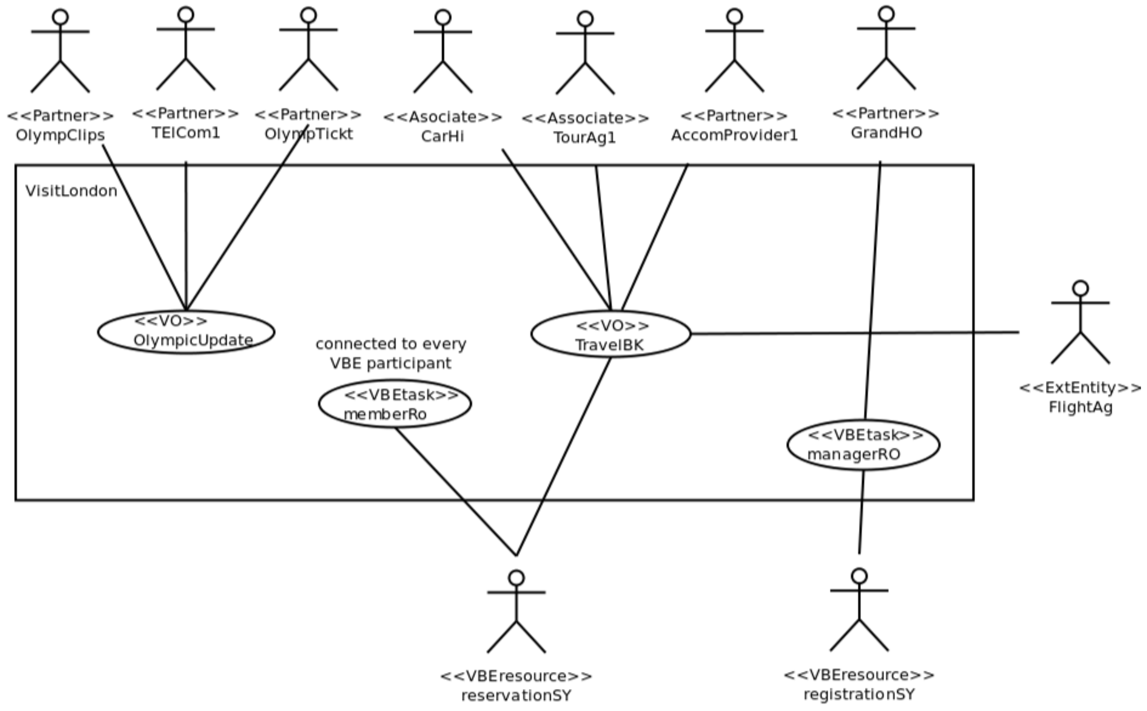


Figure 3.3: The *VisitLondon* VBE with two VOs: *TravelBK* and *OlympicUpdate*

providers, entertainment service providers, and a guided-tour provider for London.

This VBE has spawned a number of VOs, each representing a specific business service offered by the VBE to its customers. Of particular interest for this section is a virtual organization called *TravelBK*, an overview of this VO is depicted in Fig. 3.3. The goal of this VO is to provide travel itineraries for its customers and it is a compendium of three tasks: (1) The *TourGuide* task provides guided tours for the city; currently this task is performed by a partner called *TourAg*. (2) The *HotelBooking* task provides accommodation for the customers; it is currently performed by another partner called *AccomProviderX*. (3) The *TransportProvision* task books flight tickets for the customers which is performed by members whose involvement depends on customer preferences and hence they are specified as an *ExtEntity* (external entity). This VO uses a database *UsrDB* (a VBEresource) which provides customers' card details. *TourAg, AccomProviderX* and *UsrDB* are persistent and hence they remain unchanged for any instance of the VO.

This case study fits well with the criteria described in Section 1.2 on many accounts. It is a compendium of different services which do not fall into the same area of business and thus it is highly unlikely that a single entity (organization or otherwise) can offer all these services. This case study also provides a match for a variety of types of involvement of members. As it is impractical from a business perspective to keep changing the hotel (physical resource) or tour guides (human resource) quite frequently, the organizations tend to keep long-term relationships with members performing these tasks. Whereas the air travel needs to be customized as per customer criteria (cheap flight, particular airline, customer proximity). Hence, such services are not provided by a fixed business organization but selected on demand according to customer criteria. Hence, some members are relatively permanent and some are transient.
The reminder of the criteria is discussed in the section 6.4 through different scenarios when *TravelBK* VO has to go through different reconfigurations to face the influx of tourist for the London 2012 Olympic.

3.4.2 Evaluation Example: ChemicalPD VO

The second case study is a VBE named *ChemicalVBE*, which offers different services related to the field of chemical Industry. One of the VOs currently operating in this VBE refurbishes the existing chemical plants and if feasible, updates their batch processing process to a more current continuous operation process. The tasks involved in the VO are: *PreliminaryAnalysis* which analyses the details of the chemical reactions involved in the process, assessing the reaction kinetics and investigating if the conversion is feasible. The task *PilotPlantDesign* designs the new architecture of the plant. The *buildPilotPlant* task is responsible for building and operating the pilot plant to identify suitable modes of operation, potential problems with start-up and shutdown, etc. Finally, if everything goes right *BuildProcessPlant* the task builds the full-scale plant.
In addition to the above tasks there are a number of other (supporting) tasks such as providing different equipment needed or providing catalysts currently used by the re-

action processes. Besides the criteria described in Section 3.4.1 also observed in this VO, the other criterion that is likely to be encountered is the change in the goal of the VO. Its goal can change from refurbishing an old chemical plant to building a new plant from scratch and changing from a batch processing plant to a continuous one. These changes might affect the members, competencies and tasks. All these and further changes that this VO goes thorough are discussed in detail in Chapter 5.

3.5 Glossary

We briefly introduce and explain a number of terms used in the thesis that are widely used throughout the thesis and put the VOML into context. These terms will add towards understanding VOML framework.

- **Structural model:** Refers to the abstract description of constituent parts of a VO using domain terminology. This model only specifies what needs to be achieved in terms of end business goal, but not how to achieve it.
- **Operational model:** Focuses more on fine gain details of interactions taking place between different constituents described in the structural model and how those interaction are coordinated. This model is also typed with the business level aim of the VO in more details. Therefore, throughout this thesis the terms coordination and communication model are also used to refer to operational model.
- **Reconfiguration:** Reconfiguration of a VO-S implies a change of value in any of the attributes of tasks specification, modification of the capability list of a task or membership replacement. At the operational level however the reconfiguration mostly is reflected by termination of one VO-O instance and creating a new VO-O instance in place. This is because changes at the operational level (VO-O level) usually affect the coordination and communication model which represents specific layout of different components and coordination links between

them; at this level reconfiguration does not merely imply a change in attribute values. Discovery and binding of ExtEntity is the only exception, where reconfiguration is done on the running instances of a VO-O as the VO-O model anticipates that beforehand.

3.6 Summary

This chapter explained in detail the approach behind the VOML framework; a novel framework that we have developed for modelling Virtual Organisations (VOs). It provides different levels of representation for VBEs and VOs. At each level different aspects of VOs are captured using different modelling languages offered in the VOML framework.

This chapter also describes the pragmatic aspect of VOML methodology where VO life cycle model is described in details (from VOML perspective), different roles and activities involving the use of models (VO-S, VO-O and VO-R) during different stages is also described in detail.

Furthermore, this chapter also introduced the glossary of terms and two case studies; one of the case study is going to be used as running example in the rest of the chapters and other is going to used in the evaluation of the framework in Chapter 8.

Chapter 4

VO-S: VO-Structural Modelling Language

This chapter explains the Virtual Organization-Structural (VO-S) modelling language in detail and demonstrates how to model a system in terms of VO specific features using the VO-S modelling language.
VO-S defines the basic structural (computation-independent) model of the VO in terms of the domain concepts.

4.1 VO-S Modelling Constructs

The main modelling elements of VO-S are: process, tasks, competencies, members, resources and reconfiguration policies. These elements define the essential structure of the VO and provide the basis for its operational models. All of these are described in detail in the rest of the chapter. Specifically a VO-S model consists of the elements shown in table 4.1 with details explained in the respective sections.

Table 4.1: Main VO-S modelling constructs

Element	Section
Process	4.1.1
Tasks (AtomicTask, ComposableTask, ReplicableTask)	4.1.2
Competency (Capability, Capacity)	4.1.5
Members (Partner, Associate, ExtEntity, VOcoordinator, Customer)	4.1.9
VBEassets (VBEresource, VBEtask)	4.1.7
Data-Flow	4.1.8
Reconfiguration Policies & Constraints	6

4.1.1 Process

Figure 4.1 represents the description of the workflow which leads to meet the goal of the VO (requirements of the customer) at the highest level of abstraction. It specifies all the tasks that collectively lead to the realization of the business goal and the control flow between the tasks.
The *Process* description lists only those tasks that directly contribute to achieving the goals of the VO. These tasks however might rely on other *supporting* tasks to carry out the assigned responsibilities. These supporting tasks do not appear in the process description; rather each task lists in its `supportedBy` attribute all the supporting tasks it relies upon.

The following elements briefly describe the contents of the Process specification:

- **useAsset:** This element points to VBE provided resources or processes. From the VO's view point they are persistent and always available.
- **leadsTo:** This keyword refers to the control flow of the process. It has two identifiers one at left and one at the right. The identifier can be a VO task or VBE-asset. Semantically, it implies that after the completion of the task or VBEasset at left-hand side the control moves to the task or VBEasset at the right-hand side.
- **satisfyTask(s)():** The arguments for these elements can only be tasks, as the

name suggests. Semantically `satisfyTasks` implies that the exact control flow between the tasks that are the arguments of this element can be parallel or sequential at the VO-O level.
`satisfyTask()` on the other hand, takes only one argument as input and is used when the exact sequence is mandatory between different tasks and VBEassets.

```
Process
   {
      UseAsset UsrDB leadsTo satisfyTasks  (FlightBooking, Hotel&Transport,
TourGuide)
   }
```

Figure 4.1: VO-S Process Description

4.1.2 Task Specification

Task specification is the focal point of the VO-S modelling language. It not only describes the activities needed to satisfy the goal of the VO, but its description also captures the VO specific features. The different types of task in VO-S allow sharing of responsibilities between members by either dividing a task into subtasks or assigning more than one member to a task. The task specification also captures different kind of relationships that exist between members of a VO. It allows to impose restrictions on the way a VO can get restructured through reconfiguration. The competency needed to carry out the task is also part of the task description. As the VOcoordinator looks for VBE member's competencies to select the right partners for a new VO [33], the task description also shapes the member selection criteria. The type of membership the VO is going to have is also part of the task description.

A task in VO-S is divided into two parts, structural and business level, respectively called `Structure` and `Business Functionality`. The `Structure` specifies the VO-specific features irrespective of the business functionality offered (excluding the competency description) and directly or indirectly shapes the VO's different aspects such as the topology of the VO, kind of relationships between different members of the VO, possible valid (re-) configurations at a specific moment in time, resilience and agility of the VO (by task division and replication), etc.

The `Structure` is further divided into three parts namely `Competency`, `TaskScope` and `ConfScope`. The `TaskScope` lists those attributes which define the basic structure of the VO. Any reconfigurations which induce changes in this part corresponds to changing the structure of the task. For example making an atomic task a replicable one has huge consequence on the structure of the VO. The `Confscope` on the other hand, describes the current state of the VO with respect to the `TaskScope` attributes (structure of VO). For example, if the `TaskScope` says that the maximum number of members a particular task can be divided into is say three, then the `ConfScope` describes currently how many members are actually performing this task. The `TaskScope` also puts a check on the kind of reconfigurations the VO can currently go into, keeping the TaskScope attributes unchanged. The range of values that the ConfScope attributes can have are restricted by the values the attributes of TaskScope currently have. Consider the example of the `allowedMembers` attribute in the TaskScope category. The value of this attribute specifies the maximum number of participants a particular task can be shared/divided between. The task is performed by one participant in one configuration, two in another configuration, up to the value specified in the `allowedMembers` attribute in some other configuration. The `curentMembers` attribute of `ConfScope` keeps track of exactly how many participants are involved in the configuration at that particular moment in time.

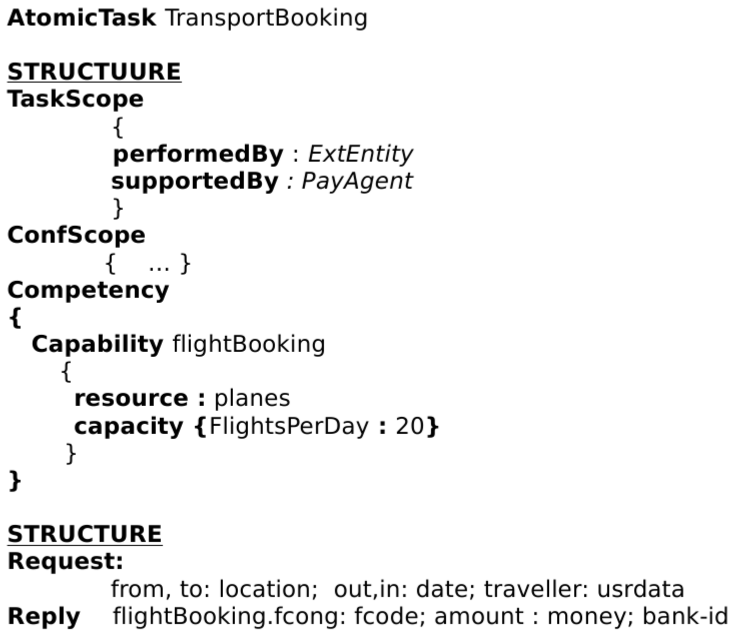
The `Competency` list the capabilities and capacities each task expects from its member(s) so that the task could be carried out. Hence, the competency part dictates which member is eligible to join the VO and who is not, but also what the capacity of the members should be, in addition to having the right capability.

The `Business Functionality` related data are defined in the detail in the Section 4.1.6).

4.1.3 Task Types

Three types of tasks are useful for modelling VOs; and each can be defined in the language.

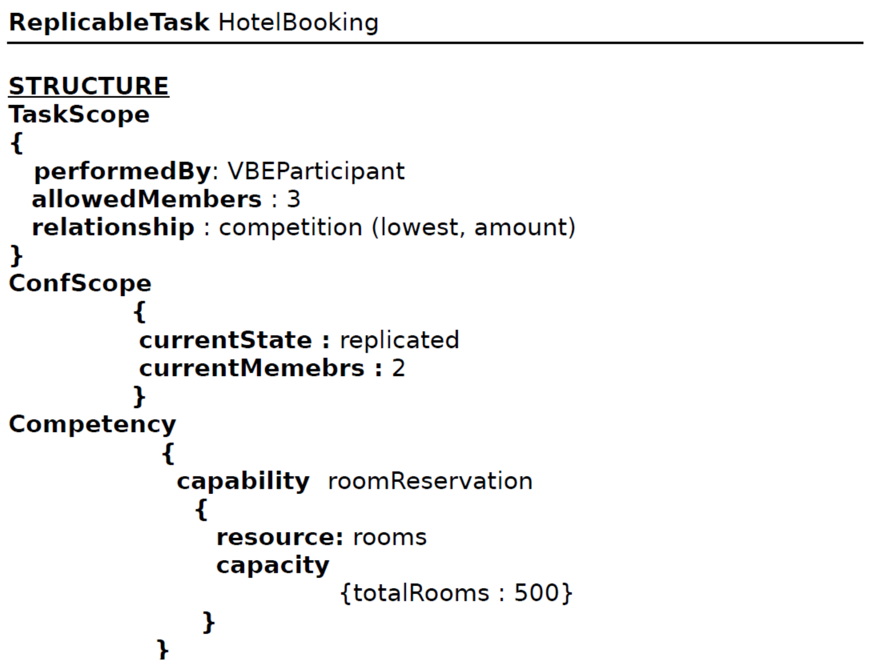
- **AtomicTask:** By making a task an `AtomicTask` it is implied that this task must be performed by only one participant. Any configuration of the VO which associates more than one participants for this task is considered invalid. The type of the task however can be changed through policies.



```
AtomicTask TransportBooking

STRUCTUURE
TaskScope
          {
          performedBy : ExtEntity
          supportedBy : PayAgent
          }
ConfScope
         {   ... }
Competency
{
  Capability flightBooking
     {
      resource : planes
      capacity {FlightsPerDay : 20}
     }
}

STRUCTURE
Request:
          from, to: location;  out,in: date; traveller: usrdata
Reply     flightBooking.fcong: fcode; amount : money; bank-id
```

Figure 4.2: AtomicTask description in VO-S

- **ReplicableTask:** A replicable task is one for which it is permissible to add more than one member if need be. For example if there are two members who both are eligible to carry out the replicable task, but both of them fall short of the amount of resources required (specified through the `capacity` attribute); collectively however they can overcome that hurdle, then it is permissible to involve both for that task. Note that both the members are equally capable of carrying out the task individually; it is the amount (capacity) of resources required which forces them to cooperate. Also worth mentioning is that the *capacity* shortage is not the only reason for which the task can be replicated. There can be many functional/non-functional criteria defined by coordinator which prefer replication of task over single members executing them. For example a single member available charges more to the VO than the two members collectively. In this situation it might be beneficial to replicate the task.



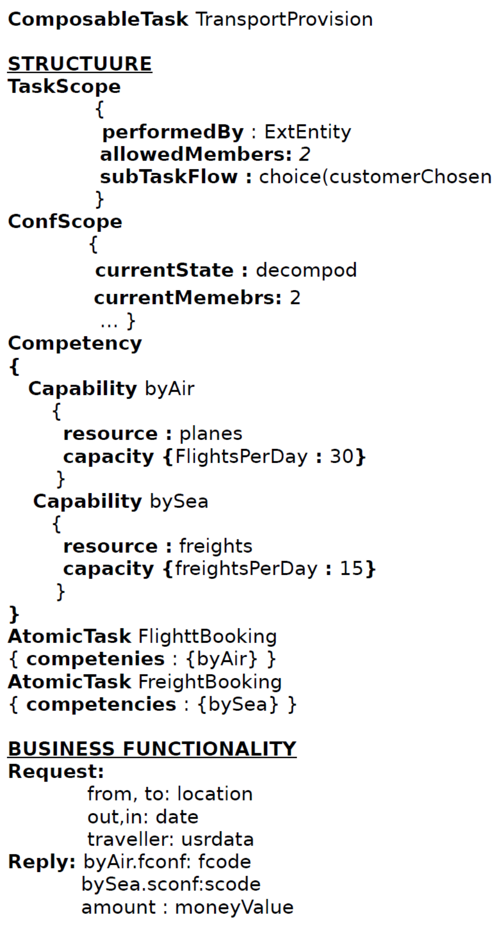
```
ReplicableTask HotelBooking

STRUCTURE
TaskScope
{
  performedBy: VBEParticipant
  allowedMembers : 3
  relationship : competition (lowest, amount)
}
ConfScope
         {
          currentState : replicated
          currentMemebrs : 2
         }
Competency
           {
             capability  roomReservation
              {
                resource: rooms
                capacity
                         {totalRooms : 500}
              }
           }
```

Figure 4.3: A ReplicableTask description in VO-S

- **ComposableTask:** Like ReplicableTask, a `ComposableTask` is one for which there can be more than one member if need be; but here the criteria is the *capability* of the members and not the capacity. Here the task is actually divided into two or more different subtasks and each subtask can be performed by different

VO members.

It is not always the case that a member falls short of capacity only; sometimes the members also fall short of the capabilities required by the task especially when there are many capabilities. In this case the task is decomposed into subtasks each having a subset of the original capabilities. This makes the number of capabilities required to carry out each subtask smaller which increases the probability of finding a member who possesses all the capabilities necessary to carry out the task.



```
ComposableTask TransportProvision

STRUCTUURE
TaskScope
          {
           performedBy : ExtEntity
           allowedMembers: 2
           subTaskFlow : choice(customerChosen
          }
ConfScope
         {
          currentState : decompod
          currentMemebrs: 2
           ... }
Competency
{
  Capability byAir
     {
      resource : planes
      capacity {FlightsPerDay : 30}
     }
   Capability bySea
     {
      resource : freights
      capacity {freightsPerDay : 15}
     }
}
AtomicTask FlighttBooking
{ competenies : {byAir} }
AtomicTask FreightBooking
{ competencies : {bySea} }

BUSINESS FUNCTIONALITY
Request:
         from, to: location
         out,in: date
         traveller: usrdata
Reply: byAir.fconf: fcode
        bySea.sconf:scode
        amount : moneyValue
```

Figure 4.4: A ComposableTask description in VO-S

4.1.4 The `Structure` of Task

As mentioned in Section 4.1.2 The `Structure` specifies the VO-specific features irrespective of the business functionality offered (excluding the competency description) and directly or indirectly shapes the VO's different aspects. In this section we explain the `Structure` in detail.

4.1.4.1 TaskScope Attributes

The following attributes constitute a `TaskScope` part:

- **performedBy:** This attribute dictates what type of member is permitted to perform the task. The values this attribute can have are given in the Table 4.2. Out of these values, `Partner, Associate` and `ExtEntity` are discussed in detail in section 4.1.9. The value `VBEparticipant` implies that under all circumstances the involved participant must be a member of the VBE. No one external to VBE can perform the task under consideration; `noPreference` implies that it does not matter to the VO at hands if the participant undertaking the task is a member of the VBE or not.

Table 4.2: Possible values for the **performedBy** attribute

Value	Explaination
`Partner`	Must be performed by a member who is a *permanent* member of VBE; under no circumstances it could be violated.
`Associate`	Same as Partner but here the member must either be provided on the discretion of the customer of the VO or invited into the VBE *temporarily* by the VBE itself when the VBE falls short of one of its competencies.
`ExtEntity`	The member be discovered from outside VBE and this member keeps changing with each VO instance.
`VBEparticipant`	Only Partner or Associate can be used here.
`noPreference`	No preferences implies any member can perform this task, be it a Partner, Associate or an ExtEntity.

- **allowedMembers:** It fixes the upper bound for the maximum number of participants a task can be shared between. This attribute is not part of atomic tasks.

- **relationship:** If a task is shared by more than one participant then this attribute specifies the type of relationship that exist between these participants. This attribute applies to RepicableTasks only and can have the value `competition` or `cooperation` as follows:

  - `cooperation`: If the relationship between the two members of the same task is defined as `cooperation`, it means they both are required for the task to satisfy its goal. An example of this could be the task of supplying the catalyst for the *ChemicalVO* example. If the quantity demanded by the customer is 500Kg, and each member is committed to provide 250Kg then the customer's demand is satisfied by providing 250Kg from one member and 250Kg from the other member. There is no competition between both the members.

  - `competition (comp-type, comp-attribute )`: Considering the same example now if the customer demands just 250Kg of catalyst then both members are able to satisfy the demands. If the relationship is set to value *competition(lowest, cost)* then both members will bid for that business opportunity. Criteria over which bidding is performed is specified in the *lowest* and *cost* i.e the bidder (member) with the lowest cost offer will be selected for that particular instance of VO. Note that the *comp-attribute* (cost in this case) must be part of the business functionally of the task.

- **currentSubTasks:** This attribute lists the names of subtasks any task is divided into at any particular moment in time.

- **subTask-Flow:** This attribute applies to `ComposableTasks` only; when new (sub)tasks appear the workflow between those subtasks needs to be identified so that their (subtasks') execution leads to the realization of the ComposableTask. This value lists the subtasks' names in the order in which they need to be carried out or in the case of subtasks having alternative capabilities, only one is chosen based on some criteria. For example Figure 4.4 list the situation when customer decides out of those subtask which subtask is going to serve the customer.

4.1.4.2 ConfScope Attributes

The following attributes constitute a `ConfScope` part:

- **currentMembers:** If the task is replicable or composable than it is possible to have a different number of participants involved in the task during different stages of the VO lifespan. The `currentMembers` attribute keeps track of the numbers of participants currently involved in the task.

- **currentState:** The `currentState` attribute keeps track of the current type of the tasks. It is not obligatory for a replicable or composable task to always involve more than one participants for such tasks. Replicability or composability is the facility that those tasks can benefit from if it is not possible for the task to be performed by single member for some reason.
  A ReplicableTask can be in one of the following states at any given time:

  - `atomic`: This state implies that currently the given replicable task is not in replicated state and only one member is solely responsible for carrying out the whole task.
  - `replicated`: This state implies that currently the given task is being shared between more than one member.

  Similarly, a ComposableTask can be in one of the following states at any given time:

  - `atomic`: This state implies that currently the given task is not divided into subtasks and only one member is solely responsible for carrying out the whole task.
  - `decomposed`: This state implies that currently the given task is being shared between more than one member and each member is carrying out one of

the subtasks of the ComposableTask.

4.1.5 Competency

A Competency is the combination of *capability* as the potential ability to perform a certain task, and the *capacity* as the availability of resources [22]. A task in VO-S specifies all the capabilities needed for the satisfaction of the functionality offered by the task and their capacity.

The competency not only helps the VO in deciding which members shall be part of the VO, but also in assigning different tasks to specific members. It also helps in identifying whether the VO is operational or not and whether some task (hence, VO) is deficit in some of the capacities or capabilities.

```
Competecy
{
   capability RoomReservation
              {
                resource hotel.rooms
                capacity
                         {totalRooms : 50}
              }
  capability LocalTransport
                  {
                   resource vehicles
                   capacity
                           {totalVehicles : 50}
                   }
```

Figure 4.5: Competency description in VO-S

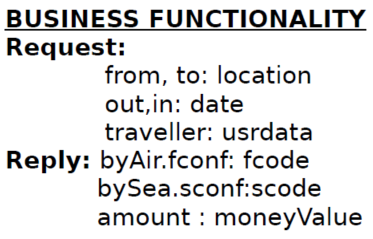
4.1.6 Business Functionality

The `Business Functionality` plays a major role in producing the operational (VO-O) model of the VO; it lists all the data items required to satisfy the goal of the task. It consists of two events: `request` and `reply`. The request event consists of all the data items that the task is going to need from the VO (other members) during its execution; whereas the reply event consists of all the data items that a task is supposed to return.

Further to the above, the data in each event category falls into one of the two groups: those which are prefixed with one of the capabilities of the task and those which are not. The capability prefixed data items represents those data items that are part of `Business Functionality` as long as that capability is part of the task, once the capability is replaced all the data items associated with that capability are removed from the `Business Functionality`. This division helps in identifying an individual `Business Functionality` for each subtask when the task needs to be decomposed as follow:

- A task is decomposed into subtasks each one having at least one unique capability inherited from the original task. This unique capability distinguishes one subtask from another.
- A subtask inherits only those capability prefixed data items from its parent task's `Business Functionality` to its own `Business Functionality`, when the capability is also inherited by the subtask.
- All the data items not prefixed with any capability are inherited by all the subtasks.

It also helps in not associating a data item to any subtask which does not need them. If a particular data item is required by all the subtasks then it is not prefixed with any capability. However, if the data is related to a particular capability then it is prefixed with the capability name. This way, if any capability is part of more than one subtask then the `Business Functionality` for all subtasks contains that data item. The Figure 4.6 lists the `Business Functionality` of the *TransportProvision* task. This task has two capability-prefixed data items. One is called *fconf*, which represents a receipt for booking a flight; it is prefixed with the capability *byAir*. The other capability-prefixed data item is *sconf*, which is the receipt for ferry booking; this data item is prefixed with the capability *bySea*. When the *TransportProvision* task is divided into two subtasks, one which books the flight and the other which books



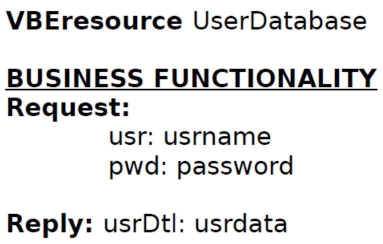
```
BUSINESS FUNCTIONALITY
Request:
          from, to: location
          out,in: date
          traveller: usrdata
Reply: byAir.fconf: fcode
         bySea.sconf:scode
         amount : moneyValue
```

Figure 4.6: A Business Functionality description in VO-S

the ferry; it is clear that *fconf* must be part of the flight booking subtask and *sconf* be part of the ferry booking subtask. Both of the data items are useless for the other subtask, so they must not be part of their `Business Functionality`. This is achieved by associating the data item *fconf* with the *byAir* capability and the *sConf* with the *bySea* capability. When specifying subtasks, it is also specified which capabilities the subtasks consists of; this way only data required by a subtask is identified. This concept has the added advantage of getting the flexibility of describing at run-time the number of subtasks a composable task can be divided into, rather than fixing the number and structure of subtasks at design time.

4.1.7 VBEassets

Besides tasks that are carried out by its members, a VO might need other basic resources and processes that are offered by the VBE; these are called *VBEassets*. As explained in Chapter 3 a VO has two types of VBEassets at its disposal: *VBEresource* and *VBEtask*. What is special about VBEassets is that they are always considered available for the VO to use without inviting any member for it to the VO. Unlike tasks, they do not have `Structure` part because VO does not put any specific constraints (demands) as it does on tasks; these are availed by VOs the way they are. The Figure 4.7 lists the VO-S description for a VBEasset called *UserDatabase* which is a database resource.



VBEresource UserDatabase

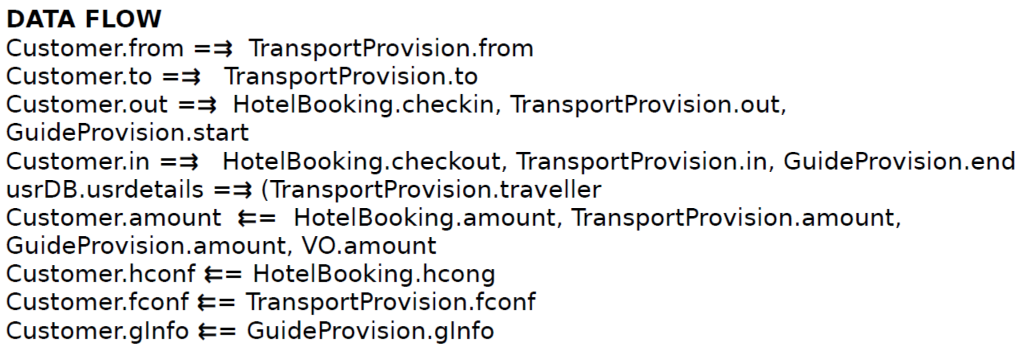
BUSINESS FUNCTIONALITY
Request:
usr: usrname
pwd: password

Reply: usrDtl: usrdata

Figure 4.7: A VO-S description of VBEresource

4.1.8 Data-Flow

The `Data-Flow` specifies the flow of data between different tasks. The data flow specifies for each task, VBEasset and the customer from where do they get values for their `request` data items; it also specifies for each `reply` data items where do their values gets dessimate to. This helps in associating a link between data items of different VO entities (task, VBEassets and customer). It also helps in realizing and verifying concrete orchestrations and transitions in the operational model.



DATA FLOW
Customer.from =⇉ TransportProvision.from
Customer.to =⇉ TransportProvision.to
Customer.out =⇉ HotelBooking.checkin, TransportProvision.out, GuideProvision.start
Customer.in =⇉ HotelBooking.checkout, TransportProvision.in, GuideProvision.end
usrDB.usrdetails =⇉ (TransportProvision.traveller
Customer.amount ⇇= HotelBooking.amount, TransportProvision.amount, GuideProvision.amount, VO.amount
Customer.hconf ⇇= HotelBooking.hcong
Customer.fconf ⇇= TransportProvision.fconf
Customer.gInfo ⇇= GuideProvision.gInfo

Figure 4.8: Description of Data-flow

Figure 4.8 lists the Data-Flow of the *TravelBK* VO.

- $\Longrightarrow$:- This operator takes either a single parameter at the left-hand side or an expression which evaluates to a single value. At the right hand side it takes one or more parameters. If there are more than one parameter at the left hand side

then they are always enclosed in a function which returns a single value. This operator assigns the single value at the left-hand side to each parameter on the right hand side. The following sentence from the Figure 4.8 assigns the value possessed by *out* data item of `Customer` to *checkin, out* and *start* data items of *HotelBooking*, *TransportProvision* and *GuideProvision* respectively.
Customer.out $\Longrightarrow$ (HotelBooking.checkin, TransportProvision.out, GuideProvision.start)

- $\Longleftarrow$:- This operator assigns a single value to the left-hand side operand. At the right-hand side if there are more than one data items then there is always an associated function with them which evaluate a single value from the right-hand side data items and assigns that to left-hand side data item. The default function is the **sum**(), which sums up all right-hand side data items; the *Customer.amount* in the Figure 4.8 is the example. In this figure the total cost the customer of the VO has to pay for the service offered by the VO is accumulation of the amount that all the members of the VO charges for their respective services. Following are some other functions:

  1. **`largest() :-`** Returns largest value
  2. **`smallest() :-`** Returns smallest value
  3. **`sum()`** Adds up all the values

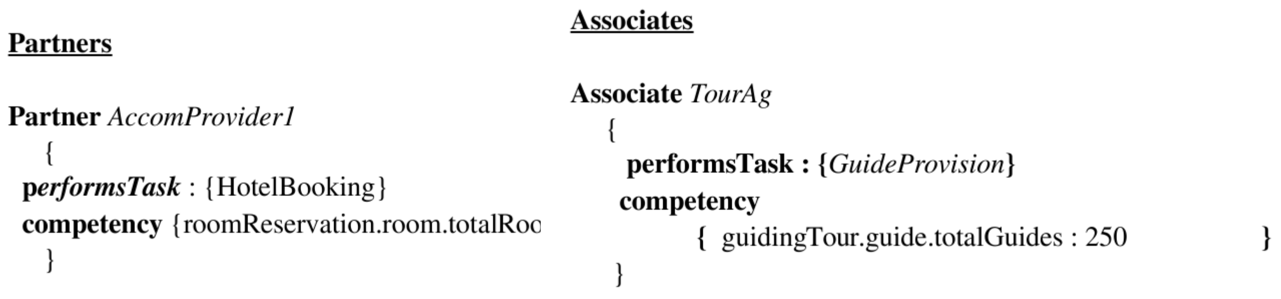
4.1.9 Members

The `Members` section of the VO-S model specifies the current members of the VO and the tasks these members are currently performing. In VO-S the `Members` section is divided into three further sections, one is called `Partners` section which as the name suggests list the current partners of the VO, and the other section is called `Associates` representing the current associates of the VO. Whereas, the `ExtEntities` list the current temporary members discovered from outside the VO and VBE.
The specification of each of these members comprises of two parts as follows:

- **performsTask:** This part lists the names of the tasks currently being performed by the member.

- **competency:** This part represent the capabilities that the member possesses and the amount of capacity shared by the member for the particular capability.

The Figure 4.9(a) lists the VO-S specification for a partner called *AccomProvider1* who currently performs only one task of *TravelBK* VO, named *HotelBooking*. The total number of rooms it provides is 500 which represents the capacity of the capability *roomReservation*. Whereas, the Figure 4.9(b) lists the VO-S specification for an associate called *TourAg* who currently performs only one task of *TravelBK* VO as well, which is *GuideProvision*. The capability this associate possesses is *guidingTour* and offers 250 guides to the VO in terms of capacity share.



```
Partners

Partner AccomProvider1
   {
 performsTask : {HotelBooking}
 competency {roomReservation.room.totalRoo
   }
```

```
Associates

Associate TourAg
   {
    performsTask : {GuideProvision}
   competency
         {  guidingTour.guide.totalGuides : 250                }
   }
```

(a) A VO-S Partner Specification

(b) A VO-S Associate Specification

Figure 4.9: A *Partner* and an *Associate* description in VO-S language

The classification of partners, associates and external entities is already discussed in detail in Section 3.2. In the following paragraph we discuss the other members of the VO.

4.1.9.1 VOcoordinator

Creation of new VO begins when one of the partners of the VBE identifying a business opportunity decides to pursue it by forming a VO. It decides the process required to satisfy the business opportunity and the competencies and VBE resources; it selects members for the VO from the VBE participants. The role of VOcoordinator is distinct from other members of the VO in having exclusive privileges, such as changing the membership of the VO, modifying its goals (i.e, end products) or terminating a VO. As such, it consolidates several distinct organizational roles, such as broker, planner and coordinator [26, 9].

The VOcoordinator contributes towards the achievement of the VO goal by supervising the progress of the tasks and members; rather than performing one of the tasks that contribute towards the purpose of the VO. Following this the behaviour of coordinator is not specified in the VO specification. The operations available to the VOcoordinator are the reconfiguration actions. However, at the structural level it is necessary to know that one exists and 'who' it is; hence it is listed in the `Members` part of the VO-S model; the only information available at the VO-S level is the VOcoordinator's identity.

4.1.9.2 Customer

A Customer is an entity outside the VO and usually outside the VBE as well that approaches the virtual organization to acquire a product/service offered by a VO. In the VO-S model there is no description of the customer; however, just like VOcoodinator it is assumed to exists when an instance of VO is created. A customer exists at the VO-O level only.

The type of membership characterizes the members' role and their type of participation (long-term or short-term) in the VO. Part of importance of this characterization is that now different set of policies can be applied to different sort of actors (members), for example ExtEntities can not be assigned any duties which are considered critical and such duties can only be assigned to partners are associates who have signed such contracts with the VBE. We distinguish three types of actors for carrying out the VO tasks: partners, associates and external entities; demand for partners and associates have been laid out in [23] whereas, need for external entities have been inspired by service-oriented paradigm.

4.2 Summary

In this chapter we introduced the VO-S modelling language in detail. We described the syntax and semantics of the main elements of the VO-S; elaborated the explanation with the help of specification snapshots from the *TravelBK* case study.

Chapter 5

VO-O Language & Mapping Methodology

While structural aspects of a VO (described using VO-S) are very abstract; its operational model is very concrete and quite close to respective execution frameworks (such as agent based system or service oriented system). We have dedicated a separate language for the operational model description which is named the *VO-Operational* (VO-O) language. The VO-O is inspired by and adapted from the SRML.
In this chapter we highlight the differences and adaptations made to the SRML for it to suit modelling of VOs and their breeding environments. Later on, this chapter lays down the guidelines which help map a VO-S description to its corresponding VO-O description.

5.1 Virtual Organization-Operational Modelling Language : VO-O

While VO-S is focused on covering structural features of the VOs, VO-O is focused on presenting a concrete *operational* view of VOs. The operational details of the VOs are derived from the information available at the structural (VO-S) level. Most of the derivation can be done automatically however, some parts of the VO-O need to be provided through manual refinement. A VO-O module derived through manual refinement only (i.e no part of the VO-O is derived automatically) has the advantage that it can be more flexible than the automatic one, in terms of division of `request` and `reply` parts of the `Business Functionality`. Each part can be divided into a number of operations (interactions) rather than a single operation that is gained from a direct mapping of the corresponding part at the VO-S level. It is worth mentioning that a VO-O module is mostly driven from information available in the `Business Functionality` part of the VO-S model. However, the `Structure` part of the VO-S model does influence the VO-O module to some extent, which will be made clear later on.

The main elements of VO-O are: module, Business Protocol(s), Layer Protocol(s), Business Role and Configuration Policies, which are described in detail in the remainder of this chapter.

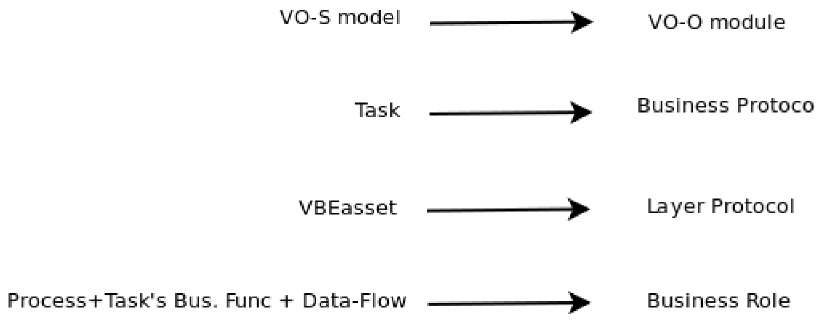


Figure 5.1: VO-S and VO-O relationship

The Figure 5.1 shows how the elements of a VO-S model relate to a VO-O module. A Business protocol is derived form the business functionality of the corresponding task. All the VBEassets at the VO-S level give rise to corresponding layer protocols in the VO-O module. VO-S Process description, interactions of all business and layer protocols (representing tasks and VBEassets of VO-S model) and the Data-Flow of the VO-S help drive concrete orchestration between different components of VO-O represented through business role of the VO.

5.1.1 Changes to the SRML

As VO-O is based on the SRML; it shares a common core in syntax and semantics. However, there are some key differences as follow:

- The main modelling primitive offered by the VO-O is called a *module*, with two specializations - VBEtask and VO modules. The rationale behind SRML's adaptation of modules can be easily reflected in the VBE, as there are some applications, processes and resources that support not only management of the VBE but also aid the VO life cycle. The VOs themselves are for the public which corresponds to a service module in the SRML, whereas we have replaced it with *VO module*. The activity module has been replaced by *VBEtask module*. For the rationale and further details [11] can be looked at.

- In the SRML all the members (service providers) are transient in the sense that each time a new instance is triggered (that is when new customer request comes in) all the members are discovered and bound to the instance; once the instance has served its goal all the members' association are terminated as well. For VOs we considers two types of members (1) those whose membership with the VO lives beyond single instantiation, that is persistent members; and (2) those whose membership is only limited to a single instantiation, that is transient ones.

- In the VO-O partners and associates sit at the 'top-end' of the module, but their behaviour is defined using a Business Protocol; SRML uses a Layer Protocol for 'top-end' entities. The reason for using business protocols rather than layer protocols is that layer protocols either represent passive entities such as those that are acted upon by other entities or entities which are served by the module (serves-interface); whereas business protocols represents entities whose nature of relationship with the VO (considering VOs as an individual entity) is of mutual interest, that is the entities (represented through business protocol) have joined the VO and offer their service(s) to the VO and in return they gain some benefit from the VO as well, be it financial or otherwise. Besides, such entities can leave VO any time they want or can be expelled by the VO as well. This kind of behaviour is represented by business protocols in the SRML; hence, this adaptation has been made.

- In the SRML entities at the 'bottom-end' of a module represent resources only, whereas, in VO-O bottom layer represents both VBE provided resources (VBEresources) as well as supporting tasks (VBEtasks) that exists in VBE, outside the VO.

- In VO-O partners and associates are assumed to be already available and hence the external configuration policies of the SRML for members are not part of the business protocol.

- In the SRML conversational interactions consists of five events (*request, reply, commit, cancel* and *revoke* events) whereas; VO-O has only request and reply events as VO members are obliged to provide what they have promised (part of the member definition) and hence there is no need for negotiation.

5.2 Methodology to Map VO-S to VO-O

In this section we describe in detail the methodology to map a VO-S description to its corresponding VO-O description. The mapping methodology consists of guidelines following which a VO-O description can be derived out of the VO-S description. We refer to these guidelines as mapping rules as they have the potential to be formalized and eventually help lead to the automation of the whole process of generating a VO-O model out of a VO-S description. As the aim of these guidelines is to be formalized enough to be transformed into mapping rules with the automation in mind, hence in this chapter mapping rules are also presented with that perspective in mind.

5.2.1 Methodology

The mapping rules which transform a VO-S description into a corresponding VO-O description follow a step-by-step methodology that needs to be respected to ensure a correct and consistent VO-O module conforming to the VO-S model description. For example step 6 of the methodology (given below), which refers to generating interactions part of a business role, can not be applied until all the business protocols and layer protocols have already been derived (in steps 2 and 3).
The rules apply to modules with only one component (internal) and respect the SRML constraint that all the communication is mediated through an orchestrator component. The following steps need to be followed for derivation of a consistent and conformant VO-O module from a VO-S model. These steps are explained in detail in Section 5.2.2 to Section 5.2.10.

Step 1: Copy data sorts from VO-S to VO-O.

Step 2: Generate Business Protocols for all the VO-S tasks (including supporting ones) and Layer Protocols for VBEassets.

Step 3: Generate the Business Protocol for Customer.

Step 4: Adjust supporting tasks at their respective positions in the `Process` description of the VO-S.

Step 5: Generate initiation condition (and termination condition when no refinement applied) for components and initiation triggers for external entities.

Step 6: Generate the Interactions part of Business Role.

Step 7: Generate Transitions skeletons of Business Role.

Step 8: Generate Orchestration part of Business Role.

Step 9: Generate wires and Interaction protocols.

Rest of the details such as external configuration policies (SLAs) and body of transitions for the VO are added in through refinement.
In the subsequent sections we are going to lay down the the rules which help transform a VO-S model into a corresponding VO-O module. The rules described transform a VO-S specification into a basic VO-O module description. By basic VO-O module specification we mean that business functionalities of all the tasks and VBEassets are transformed into business and layer protocols with only one interaction each. Business role (representing orchestrator), wires and initiation and termination conditions of orchestrator, initiation triggers for external entities (ExtEntities), state chart etc are also derived similarly (which can later be formalized into automated transformation rules, provided no refinement is done on any part of the VO-O module).
These rules also help us in proving the conformance between the two models.

5.2.2 Step 1: The Rules for DataTypes

The data types at the VO-O level represent the set of all the unique data types of each `Business Functionality`'s data items at the VO-S level. For example there are two data items called *out* and *in* in *TransportProvision* task of *TravelBK* VO whose data type is *date*. The *HotelBooking* and *GuideProvision* tasks each have two data items with same data type having names *out* and *in* (for the *HotelBooking* task) and *start* and *end* (for the *GuideProvision* task). Since all six of these data items have the same data type (*date*), in the `DataTypes` section of the VO-O module, there is going to be included only one data sort *date*. This implies that some of the data items in the *TravelBK* VO module have *date* as their data type. This is summarized in the following rules (guidelines):

1. Represent all the data items in the VO-S model having the same data type by a *single* corresponding data type in the `DataTypes` section of the VO-O module.

2. Each data item must have the same data type in the VO-O module that it had in the VO-S model.

The above rules cover step one of the methodology.

5.2.3 Step 2: The Rules for Generating VO-O Business and Layer Protocol from VO-S Task and VBEasset Specification

The VO-O business protocols and layer protocols are derived from the `Business Functionality` part of the VO-S tasks and VBEassets respectively. However, the `Structure` part of the VO-S description also effects the business protocol description, especially when tasks are replicable or composable.

The `Business Functionality` of the tasks and VBEassets in the VO-S language is divided into two events, `request` and `reply`. The order of both the parts does not affect their meaning i.e `request` means all the information (data items) required by the task/VBEasset to be carried out i.e. *incoming* data items and, `reply` means all the information (data items) returned by the task as a result of its completion i.e. *outgoing* data items.

Each `request` and `reply` parts gets transformed into different VO-O interaction types based on the business protocol, layer protocol or business role. Hence, it is worth mentioning the relationship that exists between VO-S `request` and `reply` data items and the corresponding VO-O interaction types.

The `request` data items of VO-S become:

- Data items associated with *reply* event (✉) of `s&r` interaction type.
- Data items associated with instantiation event (🔔) of `r&s` interaction type.
- Data items associated with `rcv` interaction type.
- Input data items to `ask/rpl` interaction type.
- Input data items to `tll/prf` interaction type.

All of these data items are considered *incoming* data items of the respective interaction type.

The `reply` data items of VO-S become:

- Data items associated with instantiation event (🔔) of `s&r` interaction type.
- Data items associated with reply event (⌧) of `r&s` interaction type.
- Data items associated with `snd` interaction type.
- Data items returned by the `ask/rpl` interaction type.
- Data item returned by `tll/prf` interaction type.

All of these data items are considered *outgoing* data items of the respective interaction type.

To aid the reader understand the relationship between the VO-S tasks/VBEassests and the corresponding VO-O business/layer protocols, we first give an overview of the overall protocol or layer generation from the corresponding tasks or VBEassets.

For the purpose of mapping rules (guidelines), we have divided (conceptually) a VO-S task/VBEasset and the VO-O business/layer protocol into three parts, as shown in the Figure 5.2. Part (1): Name of the VO-S task (its type) and VBEasset gets transformed into VO-O business and layer protocol. Part (2): From `request` and `reply` parts of the `Business Functionality`, `Interactions` part of business or layer protocol gets generated. Part (3): Once the interactions are generated, the `Behaviour` part of business or layer protocol is generated through refinement. There is no one to one mapping between `Behaviour` part of VO-O protocols and corresponding VO-S tasks or VBEassets specification; no information is available at the VO-S level which helps towards generating the `Behaviour` part of the business or layer protocol. The Rules (guidelines) for generation of `Behavior` part are described in Section 5.2.3.3.

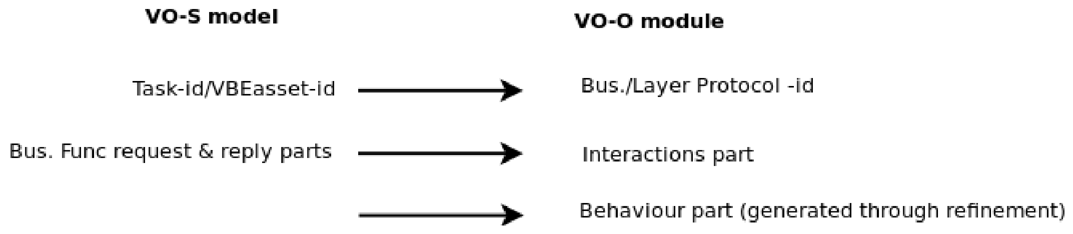


Figure 5.2: Relationship between Task/VBEasset and Business/Layer protocols

5.2.3.1 Part 1

1. `AtomicTask` *task-id* $\rightarrow$ `Business Protocol` *task-id* `is`

   i.e. A VO-S AtomicTask with *task-id* is syntactically written to `Business Protocol` *task-id* in VO-O module. An additional `is` keyword is appended at the end.

2. A `ReplicableTask` can be in two states, `atomic` or `replicated`; when in atomic state, the business protocol for a replicable task appears similar to atomic task's business protocol. However, in `replicated` state there exists separate business protocols corresponding to the number of members among which the replicated task is replicated (shared) at that time (i.e value of `currentMembers` attribute). At the VO-O level two or more business protocols each having an `Interactions` part consisting of the same set of `request` and `reply` data items are said to represent a replicable task at the VO-S level (in replicated state).

   (a) `ReplicableTask` *Task-id* $\rightarrow$ `Business Protocol` *task-id* `is`

   if and only if *task-id*.`currentState` = `atomic`

   A ReplicableTask with the `currentState` set to *replicated* has *n* business protocols with *n* equal to the number of `currentMembers`.

3. A `ComposableTask` can be in two states, `atomic` or `decomposed`. In atomic state `Business Protocol` for composable task is no different from an atomic task's business protocol. However, there does not exists any `Business Protocol` corresponding to a ComposableTask when in atomic state. Rather, there exists unique business protocols for each of the subtasks, the composable task currently consists of. Two rules, given below describe this:

   (a) `ComposableTask` *task-id* $\rightarrow$ `Business Protocol` *task-id* `is`

   if and only if *task-id*.`currentState` = `atomic`

(b) `ComposableTask` *task-id* $\rightarrow$ No business protocol exists.

Rules for Generating VO-O Business Protocol for subtasks of a ComposableTask: A subtask does not have its own `Business Functionality` part. It inherits the `Business Functionality` part from its parent task (the ComposableTask the subtask is part of) based on what competencies the subtask has inherited from the the parent task. In VO-S every data item is either prefixed with some capability or not. The data items that are prefixed with some capability exists in the business functionality of the task as long as the capability is part of the task. If the capability is deleted (through VO-R reconfiguration rules) the data item is deleted from the `Business Functionality` of the task as well. Every subtask inherits all the `request` data item from its parent's `request` data items, that are not prefixed with any capability. Out of capability-prefixed data items, the subtask inherits only those data items whose capability (prefixed) is part of competencies of the subtask as well. Same applies to the `request` event of the subtasks as well. Given below are the rules which state the above:

4. For ComposableTasks in composed state:

   (a) Each subtask's `request` event consists of all non-prefixed data items from `request` event of the corresponding `ComposableTask` and all prefixed `request` data items whose prefix matches subtasks' capability.

   (b) Each subtask's `reply` consists of all non-prefixed `reply` data items of the corresponding ComposableTask and all prefixed `reply` data items whose prefix matches subtasks' capability.

Once the `Business Functionality` part of each subtask has been identified, the subtask obeys all the mapping rules that apply to AtomicTasks.

5.2.3.2 Part 2: Rules for Generating `Interactions` Part

All the interactions of a business or layer protocol are derived from `request` and `reply` parts of the task's `Business Functionality`. These events can be divided into more than one interaction through refinement. Automated generation of `Interactions` part however, transforms the `request` and `reply` parts of VO-S into either one conversational interaction in the case of business protocol or a single synchronous interaction in the case of layer protocol.

Whether this transformation is done through automated or manual refinement, the following conditions must not be violated.

1. All the `request` data items must become *incoming* data items of a business or layer protocol.

2. All the `reply` data items must become `outgoing` data items of a business or layer protocol.

For the automated transformation from VO-S to VO-O the whole `Business Functionality` is replaced by one *conversational interaction* (`s&r` for customer or `r&s` for other members) at the VO-O level and each part (`request` and `reply`) of it in turn becomes event of the corresponding interaction at the VO-O level.

Example: Automation Rules to Generate `Interactions` part of a Business Protocol from VO-S Task: The mapping rules given below describe the `Interactions` part of business protocol for VO members (other than coordinator or customer) generated automatically from the `Business Functionality` of a VO-S task specification.

1. Place the word `Interactions` at the beginning (which symbolizes what follows next is interaction(s) of the business protocol).

2. Make the name of the interaction start with the name of the task (the interaction is part of), followed by hyphen symbol, and finally, append the word "interaction".

`Interactions`

task-id-interaction

3. Make `r&s` the default type of the interaction of business protocol representing a task.

   `r&s` *task-id*-interaction

4. Make all the `request` data items of task's `Business Functionality`, *incoming* data items of the VO-O interaction, by appending 🔔symbol before these data items.

   `r&s` *task-id*-interaction

   🔔*VO-S `request` data items*

5. Make all the `reply` data items of task's `Business Functionality`, *outgoing* data items of the VO-O interaction, by appending 🖂symbol before these data items.

   `r&s` *task-id*-interaction

   🔔*VO-S `request` data items*

   🖂*VO-S `reply` data items*

6. In the case of a business protocol representing a `ReplicapleTask`, add an appendix at the end of interaction name; the value of appendix of the interaction(s) is same as the value of appendix attached to the `Business Protocol` itself (the interaction is part of).

   The Figure 5.3(a) represents the `Business Functionality` in terms of VO-S task and the Figure 5.3(b) represents the same business functionality after getting transformed into `Interactions` part of a VO-O Business Protocol. In

BUSINESS FUNCTIONALITY
Request:
from, to: airport
out, in: date
traveller: usrdata

Reply: flightBooking.fconf: fcode
amount: moneyValue

```
r&s lockFlight
   🔔 from,to:airport,
      out,in:date,
      traveller:usrdata
   ✉ fconf:fcode
     Amount:moneyValue
```

(a) A VO-S Business Functionality (b) A Business Protocol's Signature part

Figure 5.3: A Business Functionality in VO-S and its corresponding VO-O transformation

this example, the transformation has been done manually so the default name of interaction has been replaced with `lockflight`.

Example: Automation Rules to Generate `Interactions` part of a Layer Protocol from a VBEasset Specification: The mapping rules given below describe the layer protocol for the VBEassets part of VO generated automatically from the `Business Functionality` of a VO-S VBEasset specification.

1. Place the word `Interactions` at the beginning (which symbolizes what follows next is interaction(s) of a layer protocol).

2. Make the name of the interaction start with the name of the VBEasset (the interaction is part of), followed by hyphen symbol, and finally, append the word "interaction".

   `Interactions`

   VBEasset-id -interaction

3. Make `ask/rpl` the type of the interaction of layer protocol representing a VBEasset.

   `Interactions`

   `ask/rpl` *VBEasset-id*-interaction.

4. Make data types of all the `request` data items of VBEasset's `Business Functionality`, *incoming* parameters of the VO-O interaction, putting them between small brackets "()" and appending it to the name of the interaction.

```
BEHAVIOUR
   initiallyEnabled lockFlight🔔?
   lockFlight🔔? ensures lockFlight✉!
```

Figure 5.4: VO-O Business Protocol Behaviour

`Interactions`
`ask/rpl` *VBEasset-id*-interaction(data types of VO-S `request`data items)

5. Make the data types of all the `reply` data items of VBEasset's `Business Functionality`, *outgoing* parameters of the VO-O interaction and append them at the end of incoming parameters of the interaction. To separate the incoming parameters from outgoing put a colon (:) between incoming and outgoing parameters.
   `Interactions`
   `ask/rpl` *VBEasset-id*-interaction$datatypesofVO-S$`request`$dataitems$:data types of VO-S `reply` data items.

5.2.3.3 Part 3: Rules for `Behaviour` Part of Business Protocol

The behaviour part consists of one or more sentences with constructs derived from temporal logic. The first sentence of the behaviour always starts with a *initiallyEnabled* construct. The interaction and its associated event type (incoming or outgoing) followed by the *initiallyEnabled* construct becomes the first point of communication with the member (who the business protocol represents) by the outside world (the orchestrator component in our case). The rest of the sentences start with the interaction and event-type which ended the previous sentence. This situation can be seen in Figure 5.4 where the second sentence starts with `lockFlight🔔?`, which is what the previous sentence ends with.

Example: Automation rules to generate `Behaviour` of Business Protocol from VO-S Task specification:

1. Append a "?" (representing action of receiving an interaction) at the end of the *instantiation* event (🔔) of the (first and) only interaction of a business protocol (unless business protocol is defined through manual refinement) and make it proceeded by `initiallyEnabled`.
   `initiallyEnabled` *interaction-id*🔔?

2. On the next line (second sentence) place the keyword `enables`. Make `enables` proceeded by the interaction and its event which ended the previous sentence. Make `enables` followed by the `reply` event (✉) of the (first) and only interaction and append "!" at the end of the interaction.
   `initiallyEnabled` *interaction-id*🔔?
   interaction-id🔔? `enables` *interaction-id*✉!

A composable task in an atomic state follows the rules of business protocol generation for atomic tasks; whereas in the decomposed state it is each of the subtasks that follows the rules of business protocol generation for atomic tasks.

5.2.4 Step 3: Business Protocol Rules for the Customer

In the VO-S model, there is no separate specification block for the customer of a VO like the rest of the members (through task specification) of a VO. A business protocol is mainly generated from the `Structure` part of a VO-S task specification, therefore we need to identify the Structure part for the customer first. This information is extracted from the `Data-Flow` part of the VO-S specification. In the the `Data-Flow` block, all the data items prefixed with the `Customer` becomes part of the `Structure` of a customer. The following rules, further separate the data items of the customer into *incoming* and *outgoing* events:

1. The data items associated with ⇇= operator become part of incoming-event of the `Customer`'s `Structure` block.

2. The data items associated with $=\!\rightrightarrows$ operator become part of outgoing-event of the `Customer`'s `Structure` block.

3. The data type for these items becomes the data types of the items at the other side of the operator (all the data items at the other side of operator have same data type).

Once the `Structure` part of customer figured out, then the business protocol for customer follows the same rules (for the generation of its interaction and behaviour part) as defined for the other business protocols except that a customer's business protocol can not be replicated or decomposed.

Example: Automation rules to generate `Interactions` of Customer's Business Protocol from VO-S Data-Flow: Mapping rules given below describe the `Interactions` part of business protocol for VO customer generated automatically from the `Business Functionality` identified from the `Data-Flow` part of the VO-S specification. The rules as follow:

1. Place the word `Interactions` at the beginning (which symbolizes that what follows next is interaction(s) of the business protocol).

2. Start the name of the interaction with the word `Customer`, followed by hyphen symbol, and finally, appending the word "interaction".

   `Interactions`

   Customer-interaction

3. Make `s&r` the type of the only interaction of the business protocol representing the customer.

   `Interactions`

   `s&r` *Customer-interaction*

4. Make all the `reply` data items of the customer's `Business Functionality`, *outgoing* data items of the VO-O interaction, by appending 🔔symbol before

these data items.

`Interactions`

`s&r` *Customer-interaction*

🕭*VO-S* `reply` *data items*

5. Make all the `request` data items of customer's `Business Functionality`, *incoming* data items of the VO-O interaction, by appending ✉symbol before these data items.

   `Interactions`

   `s&r` *Customer-interaction*

   🕭*VO-S* `reply` *data items*

   ✉*VO-S* `request` *data items*

The next paragraph describes the rules which generate the behaviour part of the customer's business protocol.

5.2.4.1 The `Behaviour` generation guidelines for business protocols representing customer of a VO

The guidelines and rules for generating behaviour part of the Customer's business protocol are same as described in the Section 5.2.3.3; only difference is that first sentence of behaviour represents publishing of the one of interactions (in automation case the only interaction) by appending the "**!**" symbol.

Example: Automation rules to generate `Behaviour` of Customer's Business Protocol:

1. Append a "!" (representing action of publishing an interaction) at the end of the *instantiation* event (🕭) of the (first and) only interaction of a business protocol (unless business protocol defined through manual refinement) and proceed it by `initiallyEnabled`.

   `initiallyEnabled` *interaction-id*🕭!

2. On the next line (second sentence) place the keyword `ensures`. Proceed `ensures` with the interaction and its event which ended the previous sentence. Follow `ensures` by the `reply` event (✉) of the (first) and only interaction and append "?" at the end of the interaction.

   `initiallyEnabled` *interaction-id*🔔!

   interaction-id🔔! `ensures` *interaction-id*✉?

5.2.5 Step 4: Adjusting Supporting Tasks in VO-S Process Description

If a task in the VO-S process description depends on some supporting task (s), then the supporting task must be carried out first before the main task, so that the main task gets all the required data needed by it to carry out the assigned responsibility.

Rules for adjusting the `Process` specification:

In the VO-S `Process` description append `satifyTask(`*sup_task-id*`)` before `satifyTask(` *dep_task-id* `).`

where :

sup_task-id = is a supporting task in the context of VO-S modelling language;

dep_task-id = is a main task in the context of VO-S modelling language;

sup_task-id is listed in the `supportedBy` attribute of the *dep_task-id*.

If a VO-S task depends on more than one supporting tasks then `satifyTasks` construct is used instead of `satifyTask`.

5.2.6 Step 5: Rules for Component and ExtEntity Triggers

The components (internal to module) are associated with *initiation state* and *termination state(s)* , whereas ExtEntities (requires-interfaces in the SRML terminology) are associated with *initiation triggers*. In the VO-O, the partners and associates do not have triggers; at the VO-O level they are already available. Only ExtEntity and (internal) components have triggers as in the SRML.

The following rules imply their automatic (or manual) generation from the VO-S description.

1. Make the *start state* of state chart, *Initiation* state of the component (orchestrator).

   int*component-id*🕒**init**: *s=start state of state chart*

2. Make the *last state* of state chart, *termination* state of the component (orchestrator). The last state of state chart is described in the Section 5.2.9

3. The *Initiation* trigger for an ExtEntity can be either publishing of their first interaction (associated with **initiallyEnabled**) or some other interaction event outside the ExtEntity protocol. Automatic generation always considers the first interaction of the ExtEntity itself, hence the value of trigger is always *default*. If a value other than the default is needed it is put in manually.

   Automatic generation:

   int*ExtEntity-id*🕒**trigger :** `default`

5.2.7 Step 6: `Interactions` Part Generation of Business Role (Orchestrator)

The component typed with *Business Role* is the most complex component of the VO-O module. A *Business Role* consists of three parts:

- The `Interactions` part lists all the interactions belonging to the orchestrator. The rules for generating these interactions are given in section 5.2.7.1.

- The *Orchestration* part lists all the local variables in use by the orchestrator and one variable representing the *state chart* of the VO (Module) with enumerated list of all the values that the variable can possess.

- The *Transitions* part specifies how the VO-O module represented as a state machine moves form one state to another, the events which gets published or invoked during each state, and the piece of work (computations) it performs during that state. The description of all this is specified through separate transition blocks. Each transition block represents the activities of a state (s).

The next three sections specify the rules which generate these parts (partially) dictated by the details specified at the VO-S level. The rest of the details are added manually.

5.2.7.1 Guidelines for Generation of `Interactions` Part of Business Role

The `Interactions` part of the orchestrator is extracted from all the `Business Protocols` and `Layer Protocols` of the VO-O module. Hence, the Business role generation can only takes place once all the business protocols and layer protocols have already been generated from VO-S tasks and VBEassets. The reason behind this is that, basically a business role's interaction part is the accumulation of all the interactions' duals defined in any business or layer protocols. More specifically:

For each interaction declared in any business protocol (VO-S Tasks and Customer) or layer protocol (VO-S VBEasset) description, there exists its dual interaction declared at the orchestrator side.

By dual, we mean the *complementary event* of an interaction, form the the pair of each interaction type. For example if the type of interaction is *s&r* then its dual is *r&s* with rest of the parameters (data items and their types) being the same at both sides.

The Guidelines (Rules) for Generation of `Interactions` part of orchestrator

1. Copy all the interactions from all business and layer protocols to the `Interactions` part of `Business Role`.

2. Change the type of every `Business Role` interaction to its complementary/dual interaction.

5.2.8 Step 7: Generating Business Role Transition Skeletons

Transitions are generated from the perspective of the orchestrator. In the case of refinement done on business or layer protocols the automated transitions generation is limited to the generation of its skeleton with only information about the triggering event of the transition. If the business and layer protocol generation is done by automated transformation (having one interaction per protocol), in this case things get simpler and the concrete control flow can be easily identified from the VO-S `process` description. In such case, the transition part can also include what goes in the *effects* and *sends* part of the transition. However, if any computation needs to be done at some stage such as comparing the service charges by two members, then this kind of computation is always done manually through refinement. The rules that help generating transitions (whether automated or manual) are as follow:

1. Generate one transition skeleton per interaction of `Business Role` where, the type of interaction is either `r&s` or `rcv`. The `triggeredBy` value of these transitions becomes the *instantiation* event of that interaction against which the transition is created (i.e. append 🔔symbol to the interaction name and assign it to `triggerdBy` attribute). For example if there is an interaction defined in orchestrator as *r&s bookTrip* than the transition dealing with this interaction is triggered by processing of `s&r` *bookTrip*🔔(which belongs to one of the business protocols of the module).

2. Generate one transition skeleton per interaction of `Business Role` where, the type of interaction is `s&r`. The `triggeredBy` value of these transitions becomes the *reply* event of that interaction against which the transition is created (i.e. append ⊠symbol to the interaction name and assign it to `triggerdBy` attribute). For example, if there is an interaction in the orchestrator called `s&r` *bookTrip*,

then there is going to be a transition in the orchestrator that is triggered by receiving of *bookTrip*⊠.

3. No separate transition (skeleton) is generated for interactions of the orchestrator which are initiated by the orchestrator itself (i.e `s&r, snd, ask, tll`).

4. All the interactions of type `s&r` and `snd` of orchestrator are attached to the `sends` part of one of the transitions of the orchestrator as follows:

   (a) If all the business and layer protocols consists of one interaction only then the attachment is done in accordance with the control flow identified in the Section 5.2.5.
   For example if following is described in the VO-S `Process` element:
   `...satisyTask(`*`taskA`*`) leadsTo satisfyTask(`*`taskB`*`)...`
   Then there is a transition in the orchestrator component which is triggered by the *processing* of the *reply* event (⊠) of the only interaction of `Business Protocol` representing *taskA*. In the `sends` part of this transition, the *instantiation* event (🔔) of the only interaction of `Business Protocol` representing the *taskB* is attached.
   transition ......
   triggeredBy: *`taskA-interaction`*⊠
   guardedBy :
   effects:
   sends: *`taskB-interaction`*🔔

If the business and layer protocols consist of more than one interaction, this implies that those protocols are generated manually through refinement. Hence, the control flow is much more complex. Therefore, in such cases the attachment to the `sends` part needs to be done manually as well. Care still must be taken that the abstract control flow (VO-S Process specification) is not violated. A

VO-O module is not valid if any of the orchestrator side outgoing interactions are left without associating them to `sends` part of any of the transitions.

5. Synchronous interactions do not get any corresponding transition either; they are called in the `effects` part of some transition as follows:

   (a) If all the business and layer protocols consists of one interaction only then the attachment is done in accordance with the control flow identified in the Section 5.2.5. That is the synchronous interaction of the layer protocol representing the VBEasset is attached to the `effects` part of that transition which is triggered by the *reply* event (⊠) of the interaction of the `Business Protocol` representing the task which was executed before the control flow transferred to the VBEasset in question.

   ...satifyTask(*taskA*) leadsTo useAsset(*assetA*)
   useAsset(*assetA*) leadsTo satifyTask(*taskB*)
   Above excerpt of VO-S `Process` description leads to following transition:
   transition ......
   triggeredBy: *taskA-interaction*⊠
   guardedBy :
   effects: *var = assetA-interaction(values for incoming data items):values for outgoing data items*
   ...
   sends: *taskB-interaction*🔔

In the case of business and layer protocols generated manually (through refinement), the control flow is much more complex. Therefore, the attachment is done manually. However, this manual attachment must still follow the abstract control flow.

5.2.9 Step 8: Orchestration Part of Business Role

The orchestration part of business role consists of local variables and a special variable representing the state chart of the orchestrator (provides an abstract view of the component). The rules are as follow :

- For the return types of each synchronous interaction (i.e the *output* data types associated with *ask/rpl* and *tll/prf* interactions) corresponding number of local variables are created with their data types matching the data types of *output* data items of synchronous interactions.

5.2.9.1 State Chart Generation

The values for state variable representing state chart of a VO represent the control flow of the transitions which is derived from information available in the `Process` description of VO-S specification and additional refinement done on the business and layer protocols, if any.

If all the business protocols and layer protocols of the VO-O module consists of only one interaction, then a basic state chart can be generated automatically through the rules given below. To describes these rules let us represent variable representing state chart as S and each state as S_n where n is a natural number.

The other thing to remember here is that each transition is guarded by one state and the execution of transition moves the S to some next state based on the information available at the VO-S `Process` description. The rules are as follow:

1. If n is the number of transitions then generate $n+1$ states for the state chart.

2. The transition whose `triggeredBy` value represents the receiving ("!") of (by orchestrator) the *initiation* event of business protocol representing the customer of VO is always considered first transition (the orchestrator engages in). Therefore, make S_1 the `guardedBy` value of this transition. Update the value of the state variable (S_i) to next state (second state in this case i.e S_2) in the `effects` part of the same transition. In the `sends` part *publish* the *initiation* event of

the interaction of that business protocol that appears first in the VO-S `Process` specification. Skip any VBEasset if it comes in the way.

3. Make the trigger for the next transition, the receiving of *reply* event of business protocol whose *instantiation* event was published in the previous transition. The value of of the `guardedBy` for this transition is the state that the state variable was moved to in the `effects` part of the previous transition (i.e S_2 in this case). Changes the value of state variable to next state (i.e S_3 in this case) in the `effects` part of this transition and for the `sends` part do the same as done for the `sends` part of previous transition.

4. Keep repeating step 4 for rest of the transitions as per the business protocols of tasks according to the tasks' position in the VO-S `Process` (i.e on first come first serve basis) until the transition which represents the receiving of *reply* event of business protocol for the task which is the last task in the VO-S `Process` description. In the `effects` part of that transition move the the state variable S to the its last state which is always S_{n+1} and in the `sends` part, publish the *reply* event of that interaction which represents the business protocol of the customer.

5. If there is a VBEasset described in the VO-S *Process* description, then attach the interaction of layer protocol (representing that VBEasset) in the `effects` part of the transition as per the position of the VBEasset in the VO-S `Process` description in the following ways:

   (a) If the VBEasset appears as the first element in the VO-S `Process` description then attach the layer protocol for the VBEasset in the *effects* part of the first transition i.e one that represents the receiving of `instantiation` event of customer business protocol by the orchestrator.

   (b) If the VBEasset appears at a position in VO-S `Process` description other than the first, then attach the layer protocol for the VBEasset in the *effects* part of the transition that represents the receiving of *reply* event of the business protocol of the task at appears before the VBEasset element in the VO-S `Process` description. In the `sends` part of the same transition publish the *instantiation* event of the business protocol that represents the task that appears after the VBEasset in the VO-S `Process` description or the publishing of *reply* event of the customer's business protocol if the VBEasset appears last in the `Process` description.

5.2.10 Step 9: Wire and Interaction Protocol Generation Rules

Before describing the rules which generate the wire, let us first review the structure of wires in general.
A wire looks like a table with five columns and *n* rows, where *n* corresponds to the number of interactions in a business or layer protocol and business roles. The Figure 5.5 represents a wire. Each row of a wire corresponds to a connector. Out of the first and last columns of the table, one describes the signature of one of the interactions of either the business or layer protocol and the other column describes one of the interactions of business role. The Second column describes the *RoleA* of the interaction protocol in the context of connector. Similarly, the fourth column describes the *RoleB* of the interaction protocol in the context of connector. The roles (*RoleA* and *RoleB*) of the connector describes the general signature which the interaction of the node (business or layer protocol) being attached to must have i.e each interaction of specific protocol or business role is the instantiation of the role associated with it. The middle (3rd) column identifies the interaction protocol which establishes a relationship between two parties (*RoleA* and *RoleB*).

A wire generated by default between two parties in the VO-O is always of type `Straight` which in the case of asynchronous interactions synchronises the events of the two interactions and in the case of synchronous interaction synchronizes the operations of both the parties.

TR Customer	c_1	**TO**	d_1	**RO** TravelOrchestrator
s&r bookTrip	S_1		R_1	**r&s** bookTrip
🔔 from	i_1		i_1	🔔 from
to	i_2	Straight.I(airport,airport,date,date,usrdata)O(fcode,hcode,guideinfo,moneyValue)	i_2	to
out	i_3		i_3	out
in	i_4		i_4	in
✉ fconf	o_1		o_1	✉ fconf
hconf	o_2		o_2	hconf
ginfo	o_3		o_3	ginfo
amount	o_4		o_4	amount

Figure 5.5: A Wire in VO-O language

5.2.10.1 Wire Generation Rules

1. Generate *n* wires where *n* represents total business and layer protocols of a VO-O module. Name each wire `W`*n*, where *n* is a numeric identifier for each wire.

2. Connect one business or layer protocol with the orchestrator (business role) per wire.

3. Fill left most column of each row with one of the interactions of chosen business or layer protocol and right hand side with one of the dual interactions of business role with respect to business or layer protocol interactions at the left hand side (or vice versa).

4. Map data items of every interaction of both parties to each other.

5. Generate the signature of a *straight* protocol in the middle column with respect to data items of interactions; a straight protocol consists of two parts: `I` which lists the data types of those data items that are provided by one party and received by another and part *O* lists the data types of those data items that are returned by one party to another in response to data items sent in the `I` part. *O* part follows `I` part, which is prefixed with `Straight.`

Further constraints can be specified for the wires manually, such as, encryption etcetera.

The interaction protocol consists of two parts; first part list the two roles `RoleA` and `RoleB` which list the data types of incoming and outgoing data items; the second part (*coordination*) synchronizes the interaction of both the roles' data types with each other.

5.3 Summary

In this chapter we first gave an overview of how the elements of VO-S modelling languages gets mapped to elements of the VO-O modelling language. We then described the adaptations made to the SRML for specifying the operational aspects of a VO. Finally, we described the guidelines using which a VO-S specification can be transformed into a basic VO-O specification. We also provided some examples showing transformation of some aspect of VO-S specification into corresponding VO-O specification. Where possible, we also described in the case of further refinement what constraints need to be followed such that the VO-O specification does not violate its corresponding VO-S specification. The details which could not be generated from the information available at the VO-S specification must be added in through manual refinement such as the external configuration policies and any computation done in the `effects` part of the transitions.

Chapter 6

VO-R: VO-Reconfiguration Modelling Language

In this chapter, we define the VO-R (VO-Reconfiguration) modelling language, which provides VOs with the ability to adapt to changing circumstances by reconfiguring the structure. VO-R offers a set of events, conditions and actions that can be exploited by any VO belonging to any domain. The VO-R is an extension of the APPEL policy language for the domain of VOs. The APPEL is an ECA (Event, Condition and Action) based policy language, detailed syntax of which is defined in the Section 2.3.1. The basic building block in APPEL is *Policy rule* which consists of an optional event, an optional condition, and an action. For a policy to get activated the event (if any) associated with it must be occur and the condition (if defined) must aslo be true. Actions affect the system to which the policies are applied.
How the set of ECAs defined in the VO-R language affect at the VO-S and the VO-R specification is described in detail in the Section 7.4. A typical APPEL policy looks as follows:

```
policy policy-name appliesTo task-id/member-id/VO-id
when optional trigger(s)
if   optional condition(s)
```

```
do    action(s)
```

The Virtual Organisations-Reconfiguration language (VO-R) is the specialization of APPEL to Virtual organisations. For VO-R we have added a number of events, conditions and actions specific to VOs. These events, conditions and actions are described in the reminder of this chapter.

6.1 Reconfiguration Events

This section explains the events which are part of VO-R language.

1. **EVENT:** `memberWithoutAnyJob` (*member-id*)

   Semantics: This event occurs when the `performsTask` attribute of the member is empty. This event occurs when either, (a) any member of a VO who has previously been assigned a task, now has no task assigned to it or, (b) it also occurs when a member remains without any task assigned to it for a certain duration of time, even when the member has not been assigned any task since the member joined the VO. The *member-id* is the name of the member to whom this triggers applies.

2. **EVENT:** `MemberLeft`(*member-id*)

   Semantic: This event occurs when the member *member-id* has left VO. It might occur as an indication to successful completion of *RemoveMember* action. The *member-id* is the name of the member who has left the VO.

3. **EVENT:** `CapacityDeficit` (*task-id, capability-id*)

   Semantics: The VO has all the required capabilities to carry out the task in question (*task-id*), but the member(s) assigned to the task *task-id* lacks the capacity required by the task. The *task-id* is the task which lacks the capacity and the *capability-id* points to the capability which the task is deficit in.

4. **EVENT:** `CapabilityDeficiency`(*task-id, capability-id*)

   The VO lacks a required capability, *capability-id* concerning a certain task *taks-id*. Unlike a capacity deficit, a capability deficit means that the task can not be conducted.

5. **EVENT:** `TaskWithoutAnyMember` (*task-id*)

   Semantics: The task *task-id* can not be carried out because no member has been assigned to it currently.

6. **EVENT:** `MemeberJoined` (*member-id*)

   Semantics: A new member *member-id* has joined the VO; no task has been assigned to the member yet.

7. **EVENT:** `NoMemberWithAllCapabilitiesFound` (*task-id*)

   Semantics: While looking for a member, no member was found who could provide all the capabilities required by the *task-id*.

8. **EVENT:** `NoMemberWithRequiredCapcityFound` (*task-id, [capability-id list]*)

   Semantics: Though some member has been found that can satisfy all the capabilities required by the task *task-id* but, even that member can not provide the required capacity demanded by certain capabilities listed in the *[capability-id list]*.

9. **EVENT:** `MemberTaskMismatch` (*member-id, task-id*)

   Semantics: If the member *member-id* assigned to task *task-id* provides a different set of capabilities than that needed by the task, this event occurs. This event can occurs when one or more capabilities of the task have been replaced by equivalent capabilities, but the already assigned member (whose competency matched the previous competency of the task) cannot provide the new equivalent capabilities. Another situation when it occurs is in replicable tasks where the maximum number of members has already been assigned and the task is being carried out as well but might still be deficit in some capacity (i.e. the task is being carried out with degraded performance); in this case the assignment of additional member to the task to cover the capacity deficit would fail. The event points to the actual cause. The *task-id* represent the task where this situation has occurred and the *member-id* to the culprit member.

10. **EVENT:** `MoreMembersAssignedThanPermissible` (task-id)
    Semantics: The *task-id* to which this event applies has been assigned more members than permitted by the *allowedMembers* attribute of the task or in the case of atomic tasks more than one member is assigned to the atomic task.

11. **EVENT:** `MemberRoleMismatch` (*member-id, task-id*)
    Semantics: This event occurs when the member *member-id*, being assigned to the task *task-id* does not fits the demand specified through the task's `performedBy` attribute or when the value of `performedBy` attribute has been changed through the VO-R action *changeRole*.

6.2 Conditions

This section explains the conditions available in the VO-R language.

1. `isTaskAtomic` (*task-id*)
   Semantics: This condition is true if the type of the *task-id* is `AtomicTask`, false otherwise.

2. `isTaskReplicable` (*task-id*)
   Semantics: This condition is true if the type of the *task-id* is replicable, false otherwise.

3. `isTaskComposable` (*task-id*)
   Semantics: This condition is true if the type of the *task-id* is composable, false otherwise.

4. `isTaskWithoutAnyMember`(*task-id*)

   Semantics: This condition is true if the *task-id* task has not been assigned any member, false otherwise.

5. `isMemberWithoutAnyJob`(*member-id*)

   Semantics: This conditions verifies that if the concerned member (*member-id*) has been assigned atleast one task or no task has been assigned to the member.

6. `isCapabEquivalent`(*newCapa-id, oldCapab-id*)

   Semantics: This condition compares the *newCapab-id* with the *oldCapab-id* and returns `true` if both the capabilities are equivalent, false in other case.

6.3 Actions

In this section we are going to explain the basic actions that are needed by any VO, to enact adaptations.

1. `RemoveMember` (*member-id*)

   Semantic: The membership of the member in question (*member-id*) ends as the consequence of this actions' execution.

2. `AddNewMember` (*member-id*)

   Semantic: A new member *member-id* becomes part of the VO; however, the member has not been assigned to any task of the VO yet.

3. `AddAltrCapab` (*task-id, currCapab-id, equivCapab-id, criteria*)

   Semantic: This action adds an equivalent capability with respect to an already present capability in the respective task. Capabilities are considered equivalent if they perform the same task but by using different means, for example the act of travelling could be achieved in many ways: by air, by road or by sea. The alternative capabilities become part of different sub tasks and at the VO-O level only one of these sub tasks is selected as per the *criteria*. The *task-id* represents the task (ComposableTask) to which this action applies, *currCapab-id* is the capability already part of the *task-id* and *equivCapab-id* is the new equivalent capability being added as an alternative capability to the *task-id*.

4. `DeleteAltrCapab` (*task-id, equivCapab-id*)

   Semantic: This action deletes an equivalent capability from the task only if there is more than one capability which is equivalent to some other activity in the task.

5. `ReplaceCapability` (*task-id, currentCapab, newCapab*)

   Semantic: One of the equivalent capabilities of the task is replaced by any of the capabilities which are considered equivalent to the capability which is being replaced. The *currCapab* is the capability being replaced and the *newCapab* is the capability is which replacing the previous one. The *task-id* is the name of the task to which this action applies. The `ReplaceCapability` only replaces one capability with another one only if the two capabilities are equivalent.

6. `AssignTask` (*task-id, member-id*)

   Semantic: Assigns member *member-id* to the *task-id*.

7. `UnAssignTask`(*task-id, member-id*)

   Semantic: This action undoes the effects of the `AssingTask` action by removing the member *member-id* member from the task *task-id*.

8. `MakeTaskComposable` (*task-id, [subtask list & the corresponding capabilities]*)

   Semantic: This action decomposes the task into two or more subtasks. The *task-id* is the task being decomposed and the second parameter is the list of subtasks and their corresponding capabilities into which the *task-id* is going to be divided into.

9. `MakeTaskReplicable` (*task-id, maxMembers, relation*)

   Semantic: This action changes the type of a task so that it can be shared by more than one member. The resulting type is `ReplicableTask`. The *maxMembers* parameter represents the maximum number of members the task is permitted to be shared between and the *relation* represents the relationship between these members (e.g. whether they are in competition or not).

10. `MakeTaskAtomic` (*task-id*)

    Semantic: This action changes the type of the task to *AtomicTask* so that, the task can be conducted by only one member, hence forbidding its sharing or decomposition. The *task-id* is the task being made atomic.

11. `ChangeRelationship` (*ReplicableTask-id, relation*)

    Semantic: This action only applies to replicable tasks where there is more than one member sharing the responsibility of conducting the task. The *relation* attribute shows whether all the members cooperate with each other or they are in competition with each other.

12. `changeRole` (*task-id, new-role*)

    Semantic: This action changes the type (partner, associate and external entity) of member permitted to perform the task by changing the value of the *performedBy* attribute.

13. `changeAllowedMembers`(*task-id, allowedMembVal*)

    Semantic: This action changes the number of members permitted to carry out a replicable or composable task *task-id*. The *task-id* is the task to which this action applies and *allowMembVal* is the value that is going to be assigned to the `allowedMembers` attribute of the task.

14. `searchMember`(*task-id*)

    Semantic: This actions looks for a new member outside the VO whose competency matches the competency required by the *task-id*.

15. `changeCapacity` (*task-id, capability-id, value*)

    Semantic: This action updates the capacity value of the *task-id*. The *capability-id* is the capability whose *resource*'s capacity is being updated.

6.4 The TravelBK Case Study

In this section we are going to show how VO-R reconfigurations effect the VO-S specification using some examples. For each example, we will present a scenario, show the reconfiguration rule(s) and describe the effects. We will also evaluate the scenarios against the criteria mentioned in Section 1.2 to identify the strengths and limitations of the VOML framework.

We consider the *TravelBK* case study introduced in Chapter 1. The Appendix A lists the full specification of TravelBK.

The VBE management has decided that *TravelBK* has a big role to play during the Olympics but needs to adapt to cope with the demands.

6.4.1 Scenario 1: More hotel beds are needed than the current provider can provide

The Olympics 2012 will bring an influx of people to London; these will all stay for a few days and hence will require accommodation. The current accommodation provider - that is the member responsible for the *HotelProvision* task - has a rather limited capacity, which is suitable for normal operation. A search for new members to provide accommodation shows that no member can individually meet the expected demand, but each provider could contribute to the demand and overall it could be met.

The current configuration specified *HotelProvision* as an atomic task, i.e. one that can only be offered by one member. A decision is made to specify the following reconfiguration policy which in turn will change the specification to cope with larger demand by making *HotelProvision* a replicable task.

```
policy MoreBeds appliesTo HotelProvision
when NoMemberWithRequiredCapacityFound (HotelProvision)
do   MakeTaskReplicable(HotelProvision, 3, [competition, cheapest])
```

Figure 6.1 show the respective part of the specification that will change when the rule is applied to the model presented earlier. Note in particular that the task is now replicable, allowing to be split between up to 3 members. Also, the members will be in competition with each other, that is the cheapest one having remaining capacity will be given priority in allocations. This scenario satisfies the criterion C4, by defining the task to be shared by more than one member and criterion C5 by describing the relationship between the members performing the *HotelProvision* task as competitors.

6.4.2 Scenario 2: Further means of transport are needed

Currently *by air* is the only means of transportation to bring tourists in and out of London – that is the current VO expects people to come by flying in. This is shown in the figure 6.2(a). The increase of people wishing to travel to London will put a strain on airlines and hence alternative means of transportation could allow to increasing the profit. It is hence decided to add diversity to the business by adding more options for the transportation task. This is achieved by a rule allowing the addition of *equivalent capabilities* as an alternative to by air – for example *by sea*. It is seen as equivalent because it offers a solution to the same task (namely transportation).

The following policy encapsulates the required change. Note that it performs two actions: one to add the alternative capability and the other to change the type of the task to ComposableTask so that both alternatives are available to instances of the VO. Hence, this scenario satisfies the criteria C6 and C4 respectively. Adding an equivalent capability also leads to the addition of a new member in the VO hence, addressing the requirements expressed by the criteria C4 and C5.

```
policy MoreTransportMeans appliesTo TransportProvision
do
   AddAlternateCapability (TransportProvision, byAir, bySea,
                                                      customerPref)
andthen
  MakeTaskComposable (TransportProvision, [[flightBooking, byAir],
                              [freightBooking, bySea]])
```

Figure 6.2(b) shows the resulting parts of the specification. Observe in particular that a new capacity has been added and then that the task type has changed. In addition, in the reply part of the business functionality, we now have an additional data item and in the dataflow there are changes to reflect that the data item is needed. Clearly in a specific running instances of the VO-O only one of the alternatives would occur.

Note that in this scenario we have not added a new task, but rather expended the scope of an existing one. This means that the overall purpose of the VO is maintained, but it has reacted to a new opportunity and adapted to a changing environment.

6.4.3 Scenario 3: One of the hotel partners has had a fire and had to withdraw their commitment

Sadly one of the hotels had a fire and cannot provide the promised accommodation. This is a kind of contract violation in the eyes of the VBE – clearly in business cases this happens and the VO is prepared to react to it as it has the following rule (this rule assumes that *memberX* is only involved in `HotelProvision`):

```
policy MemberQuits appliesTo memberX
do   UnAssignTask(memberX, HotelProvision)
andthen
         RemoveMember(MemberX)
```

This rule has two actions in a sequence, the second is only executed if the former is successful. Before a member can be removed the tasks assigned to that member need to be removed from him and then the member can be removed from the VO.

The application of the rule above might lead to a follow-on problem, namely that of a capacity or capability deficit. This situation is captured by further rules that describe how the VO should behave if such a deficit occurs (note that these rules can also handle the situation where due to increased demand more capacity is required).

```
policy MoreCapacityNeeded appliesTo HotelProvision
```

```
when CapacityDefecit (HotelProvision)
if   isTaskReplicable (HotelProvision)
do
          searchMember (HotelProvision)
andthen
      AddMember (memberX)
andthen
          AssignTask(HotelProvision, memberX)
```

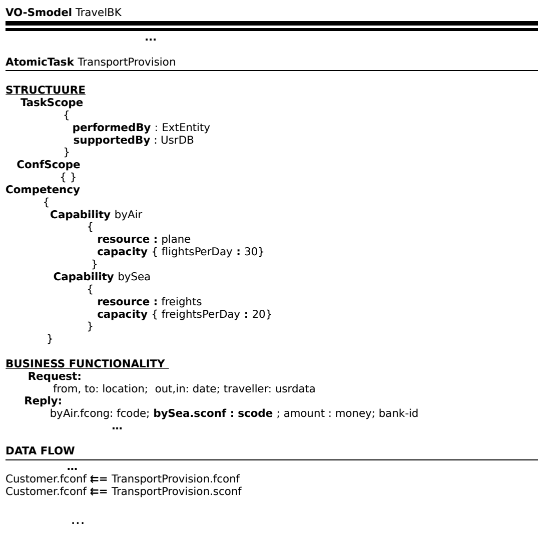
The last rule describes the process the VO should enact when a capacity deficit problem is encountered. It first checks the condition whether the task is replicable. It adds one more member to the task if the condition is satisfied. By doing so this scenario has satisfied the criterion C7 by preventing the VO from failing in the event of capacity deficit that occurred as a result of one member leaving the VO. This scenarios also shows the flexibility of framework to react to possible VO failure which could have occurred as a result of contract violation.

6.5 Summary

In this chapter, we first introduced the basic set of events, conditions and actions that we believe cover the core re-configuration demands of a VO, no matter which application domain the VO operates. However, there is potential to expand this core set of events, conditions and actions.
Then we validated some of these ECAs against different scenarios of our TravelBK case study to see if these ECAs satisfy the adaptation demands of TravelBK VO.

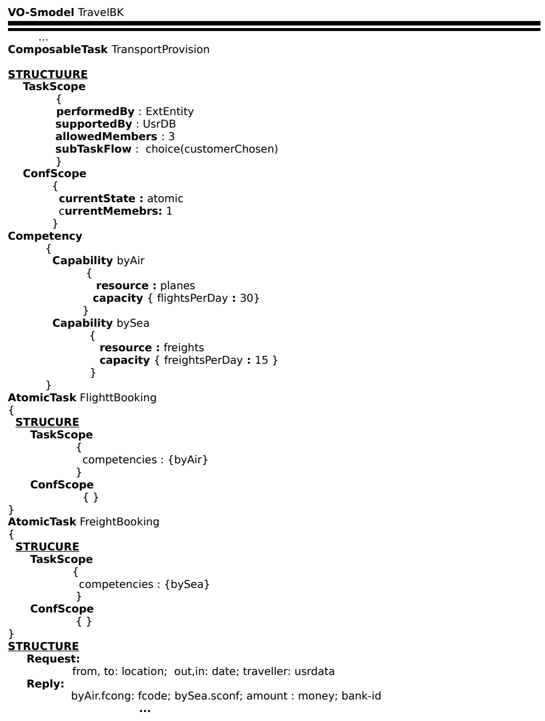
```
VO-Smodel TravelBK
==========================================================================
            ...

ReplicableTask HotelBooking
--------------------------------------------------------------------------

STRUCTURE
   TaskScope
            {
              performedBy: VBEParticipant
             allowedMembers : 3
             relationship : competition (cheapest)
            }
   ConfScope
             {
              currentState : replicated
              currentMemebrs : 2
            }
Competency
              {
                Capability  roomReservation
                        {
                         resource rooms
                         capacity  {totalRooms : 500}
                        }
              }

BUSINESS FUNCTIONALITY
     Request:
          from, to  : location;  out, in : date;  name : usrdata
     Reply :
          roomReservation.hconf : hcode;  amount : moneyValue
                         ...

Members
--------------------------------------------------------------------------

Partners

    Partner AccomProvider1
            {
             performsTask : {HotelBooking}
             competency {roomReservation.room.totalRooms  : 350}
            }

    Partner GrandHO
           {
            performsTask : {HotelBooking}
            competency {roomReservation.room.totalRooms  :250}
           }
                              ...
```

Figure 6.1: *TravelBK* VO after applying Rules from Scenario 1



```
VO-Smodel TravelBK
                              ...
AtomicTask TransportProvision

STRUCTUURE
   TaskScope
            {
              performedBy : ExtEntity
              supportedBy : UsrDB
            }
  ConfScope
           { }
Competency
        {
         Capability byAir
                 {
                   resource : plane
                   capacity { flightsPerDay : 30}
                  }
          Capability bySea
                 {
                   resource : freights
                   capacity { freightsPerDay : 20}
                 }
         }

BUSINESS FUNCTIONALITY
     Request:
          from, to: location;  out,in: date; traveller: usrdata
    Reply:
         byAir.fcong: fcode; bySea.sconf : scode ; amount : money; bank-id
                  ...
DATA FLOW
             ...
Customer.fconf ⇇= TransportProvision.fconf
Customer.fconf ⇇= TransportProvision.sconf

              ...
```

(a) *TravelBK* VO before applying scenario 2 rules



```
VO-Smodel TravelBK
       ...
ComposableTask TransportProvision

STRUCTUURE
   TaskScope
            {
            performedBy : ExtEntity
            supportedBy : UsrDB
            allowedMembers : 3
            subTaskFlow :  choice(customerChosen)
            }
   ConfScope
           {
            currentState : atomic
            currentMemebrs: 1
           }
Competency
         {
          Capability byAir
                 {
                   resource : planes
                  capacity { flightsPerDay : 30}
                 }
          Capability bySea
                  {
                    resource : freights
                    capacity { freightsPerDay : 15 }
                  }
         }
AtomicTask FlighttBooking
{
 STRUCURE
     TaskScope
              {
               competencies : {byAir}
              }
     ConfScope
               { }
}
AtomicTask FreightBooking
{
 STRUCURE
     TaskScope
             {
              competencies : {bySea}
              }
     ConfScope
              { }
}
STRUCTURE
    Request:
              from, to: location;  out,in: date; traveller: usrdata
    Reply:
              byAir.fcong: fcode; bySea.sconf; amount : money; bank-id
                        ...
```

(b) *TravelBK* VO after applying scenario 2 rules

Figure 6.2: *TravelBK* VO before and after applying rules from Scenario 2

Chapter 7

Formal Syntax and Semantics of VOML

Formally defined syntax and semantics provide the basis for VOs and VBEs modelled in the VOML framework to be formally analysed. Formal syntax of the languages that are part of the VOML framework is defined using xtext [4]. This chapter provides a detailed account of the syntax of the three languages in the VOML – VO-S, VO-O and VO-R – and discusses their syntactic integration. It also discusses the semantics of the three languages, their integration.

7.1 xtext

Formal syntactic definition of languages can be provided through a number of mechanisms – typically these are referred to as grammars. Traditionally BNF (Backus Naur Form) and variants thereof have been used to describe syntax; we use xtext in this work. ”xtext is a framework for development of programming languages and domain specific languages. It covers all aspects of a complete language infrastructure, from parsers, over linker, compiler or interpreter to fully-blown top-notch Eclipse IDE integration” [4]. Thus xtext could provide a natural avenue to build tool support in the eclipse framework for our languages; actually SRML of which we have drawn inspiration has a formal syntax in xtext and this has allowed for a fast development of the SRML toolset.

7.2 VOML Syntax

In this section we describe the formal syntax of the three languages which form part of VOML framework. The grammar is formally specified using xtext. The complete grammar is provided in the Appendix C; this section discuses the overall structure and organization of the grammar.

Given below is the entry rule for VOML framework:

```
Voml:
 (vosModel = VOSmodel)
 (vooModel = VOOmodel)
 (vorModel = VORmodel)
 "datatypes" (datatypes += DataTypes)+;
 DataTypes:
 "sorts" (datatype += DataType)*
         interval += Interval*;
DataType: name = SortName;
SortName : name = ID ;
Interval : "[" initialVal = INT ".." finalvalue=INT "]" ;
```

The *Voml* entry rule consists of single VOSmodel element, single VOOmodel element and single VORmodel element, each representing respectively the VO-S, VO-O and VO-R modelling languages. In the VOML infrastructure these three languages can be syntactically linked to each other based on the concept of name referencing [66] and inheritance. The xtext framework at the moment supports inheritance of only one other domain. Due to lack of developed infrastructure and inheritance limitation of xtext the syntax of the three languages of VOML is developed as one large modelling language.

The Voml rule also specifies one or more `DataType` elements. These elements are actually part of VO-S and VO-O models and hence should be described in the respective rules. However, since all the languages are described as part of one language the `DataTypes` rule has been moved to entry rule. This way redundantly describing the data types in the respective languages has been avoided. The `DataTypes` rule itself consists of zero or more `DataType` elements and zero or more `Interval` elements. The `DataType` rule allows to define user defined data types. The xtext has built-in support for a numeric data type `INT` and a string data type `STRING`. The `Interval` data type is mainly used by the VO-R models.

7.2.1 VOSmodel

VOSmodel: The VO-S modelling language starts with `VOSmodel` rule. This rule starts with keyword *VO-Smodel* and the name of the VO. The VOSmodel zero of more texttt-DataTypes elements, zero or `Tasks` elements and `VBEasset` elements, exactly one `Process`, `Taskspecification`, `AssetSpecification`, `DataDlow` and `Members` elements. The `Taskspecification` and `AssetSpecification` describe the complete specification of tasks and VBE assets mentioned in the `Tasks` and `VBEasset` elements of the VOSmodel rule.

```
VOSmodel:
 "VO–Smodel" name = ID
 "datatypes" (datatypes += DataTypes)*
 "tasks" (tasks += Tasks)*
 "vbe" "assets" (vbeAssets += VBEasset)*
 "process" (process = Process)
 taskSpecification = TaskSpecification
    assetSpecification = AssetSpecification
 "data−flow" dataFlow = DataFlow
 "members" (members = Members);
```

7.3 Task Specification

The atomic, replicable and composable tasks are defined as generalization of an abstract rule `TaskType`. Given below is the description of atomic, replicable and composable tasks with the rules `AtomicTask`, `ReplicableTask` and `ComposalbeTask`. All these rules begin with keywords respective to tasks followed by the name of the tasks, which is actually reference to another rule `Tasks`. The `Tasks` rule lists names of all the tasks part of VO. The referencing provides consistency check by linking the task names described at the start of the VO-S model with their specification later in the specification. All three rules consists of one element representing *structure* part of task specification (with names rule names `Structure`, `RepStructure` and `CompStructure` for atomic, replicable and composable tasks respectively) and one element for *business functionality* part with the rule name `BusinessFunctionality`. The rule representing the structure part of the respective task types in turn comprises of three elements, one for *TaskScope* part (with rule names `TaskScope`, `TaskScopeForRep` and `TaskScopeForComp` for `Structure`, `RepStructure` and `CompStructure` respectively), `ConfScope` and `Competency`. The `RepStrucutre` in addition contains one more optional element for subtasks of composable tasks.

```
TaskType : AtomicTask | ReplicableTask | ComposableTask;
AtomicTask :"atomic" "task"   name = [Tasks]
          (structure = Structure)
          "business" "functionality" (businessFuc =
               BusinessFunctionality);
Structure : "structure" (taskScope = TaskScope)
          "confScope" "{" (confScope = ConfScope) "}"
          "competency" "{" (competency = Competency) "}"   ;
                      ...
ReplicableTask : "replicable" "task"  name = [Tasks]
                (repStructure = RepStructure)
               "business" "functionality" (businessFuc =
                    BusinessFunctionality);
```

```
RepStructure :"structure" (repTaskScope = TaskScopeForRep)
          "confScope" "{" (confScope = ConfScope) "}"
           "competency" "{" (competency = Competency) "}"   ;
                    ...
ComposableTask :"composable" "task" name = [Tasks]
         (compStructure = CompStructure)
         "business" "functionality" (businessFuc =
             BusinessFunctionality) ;
CompStructure : "structure" (taskScope = TaskScopeForComp)
        "confScope" "{" (confScope = ConfScope) "}"
         "competency" "{" (competency = Competency) "}"
         (subTasks = SubTaskSpecification)? ;
```

The description of different task types in the VO-S language differs in having some of the attributes different in the *TaskScope* part while most of the attributes are common in all the *TaskScope* part. Hence in the syntax we have defined one rule named `TaskScopeAttribute` which list all possible attributes that can be in any of the task types. The rules `AtomicTaskScopeAttribute`, `RepTaskScopeAttribute` and the `CompTaskScopeAttribute` each return `TaskScopeAttribute` rule customized for their individual type. The rule `AtomicTaskScopeAttribute` represents all the attributes part of an atomic task's taskscope from the rule `TaskScopeAttribute`, the `RepTaskScopeAttribute` for replicable tasks and the `CompTaskScopeAttribute` for composable tasks from `TaskScopeAttribute` rule.

```
TaskScope: "taskScope" "{" (taskScopeAttr +=
    AtomicTaskScopeAttribute)*  "}";
TaskScopeAttribute: SupportedBy |  PerformedBy| SupportsTask|
    AllowedMembers |  Relationship | SubTaskFlow |subTasks=
    CurrentSubTasks;
AtomicTaskScopeAttribute returns TaskScopeAttribute :  SupportedBy |
      PerformedBy| SupportsTask;
TaskScopeForRep: "taskscope" "{" (repTaskScopeAttr +=
    RepTaskScopeAttribute)* "}";
```

```
RepTaskScopeAttribute returns TaskScopeAttribute : AllowedMembers |
     PerformedBy | SupportedBy | SupportsTask |  Relationship ;
TaskScopeForComp:"taskScope" "{" (comTaskScopeAttr +=
    CompTaskScopeAttribute)* "}";
CompTaskScopeAttribute returns TaskScopeAttribute : SubTaskFlow |
    SupportsTask| PerformedBy | SupportedBy  | AllowedMembers |
    subtaskList= CurrentSubTasks  ;
```

7.3.1 VORmodel

The `VORmodel` rule consists of zero or more `PolicyRuleGroup` elements besides the name of the `VORmodel`. The `PolicyRuleGroup` in turn consists of one or more policies (represented by `PolicyRule`) grouped by policyGrouping operator (represented by `PolicyOpr`). The `PolicyRule` in turn consists of optional `Location`, optional `Triggers`, optional `Conditions` and one `Actions` elements. The VO-R model is linked to VO-S model by referencing to members, tasks and capabilities defined in VO-S models as can be seen by the `MemberWithoutAnyJob` rule which is referring to members. The complete specification of `VORmodel` is given in appendix C.

```
VORmodel :
         "VO-R" name=ID
         (policyRuleGroup+=PolicyRuleGroup)*;
         PolicyRuleGroup: policyRule += PolicyRule (policyOpr +=
             PolicyOpr policyRule += PolicyRule)*;
PolicyRule: ("policy" policyName=ID)?
             ("appliesTo" (location = Location)*)?
             ("when" (triggers = Triggers))?
             ("if" (conditions = Conditions))?
             "do" (actions = Actions);
Location : name = [Random];                          ...
MemberWithoutAnyJob: "memberWithoutAnyJob" "(" memberId=[Members] ")
    ";
```

7.3.2 VOOmodel

The `VOOmodel` rule represents partial syntax for the VO-O modelling language. The `VOOmodel` rule consists of `VOmodule` and `Specification` elements out of which `VOmodule` specify the components, whereas the specification (business role, protocols, wire protocol, etc) of components defined through the `Specification` rule. Both parts of the VO-O module (i.e the components and respective specification) are then liked as shown in line 12 of the listing below by referring to `BusinessProtocol` rule in the `RequiresInterface` rule.

```
1  VOOmodel:
2        (module = VOmodule)?
3        ("specification" (specifications+= Specification)+)?;
4
5  VOmodule : "service" name=ID "is"
6   ("components" (components += Component)*)?
7   ("requires" (requires += RequiresInterface)*)?
8   ("provides" provides += ProvidesInterface)?
9   ("uses" (uses+=UsesInterface)*)?
10  ("wires" (wires+= Wire))?;
11              ...
12 RequiresInterface : "requiresInterface" name=ID ":" businessProtocol
       =[BusinessProtocol]
```

It is worth mentioning here that the VO-S and VO-R are linked to each other through the VO-R language syntax making references to members, tasks, VBE assets and the VO itself but same does not apply to the VO-S and VO-O parts. The reason behind this is that VO-R policies directly work on the VO-S constituents, hence it must know the exact names whereas VO-S constituents might not be mapped to VO-O components on a one to one basis such as replicable task is represented by more than one components when in replicated state and with exactly one components when in atomic state. This situation gets even complicated for composable tasks when they have been decomposed into subtasks. The exact link (mapping) between VO-S constituent and VO-O constituent each named uniquely is actually created through these mapping rules.

7.4 Semantics

We aim to define the semantics of the VOML in future. Here we review briefly the semantics of the APPEL and the SRML on which the VO-R and the VO-S languages build by extension and adaptation. From the perspective of the VOML framework we also discuss the semantics of the new extensions to the languages, and discuss how the languages may be linked.

The VO-S language defines the basic structure of a VO. It does not cover behavioural aspects of the VOs while in operation (execution) and serving a customer. This behaviour is represented at the VO-O level in the form of interactions and the order of interaction execution.

The SRML provides mathematical semantics for modelling business logic of services. The business protocol constructs are defined over the UCTL [62] and the interactions in the SRML are captured by a model of computation. For detailed description of mathematical model of the SRML see [6]. Whereas, the APPEL has been given formal semantics based on the δDSTL(x) [50]. The VO-O semantics is a subset of the SRML semantics and semantics of the VO-R is adapted from APPEL. In the VO-O language all the partners and associates of a VO are represented through *serves* components. Therefore, the VOML distinguishes from the SRML by associating business protocols with *serves* components and allowing more than one serves components in the VO module which is adapted from *service* module in the SRML.

Besides, the VO-O does not allow external configuration policies for the business protocols associated with serves components as partners and associates (serves components) in the VO-O are not discovered at run-time (VO-O instances) rather, they are already chosen from the VBE at the VO-S level. Other difference is that the conversational interactions are limited to *request* and *reply* events.

The semantic integration of these languages is the aim of future work but conceptually there is no overlap between the three languages as each language represents different aspects of a VO. More specifically, the APPEL defines a transformation relation on the VO-S model at design time and at run-time for VO-O models. Activation of policies ends in modification to the VO-S models and the VO-O instances. Similarly, the VO-S defines design time transformational relation on the VO-O models. Each time a change is made in the VO-S model it is propagated to the VO-O model and new instances of the VO-O then reflect these changes. The Figure 7.1 describes the semantic integration of the VOML framework at conceptual level.

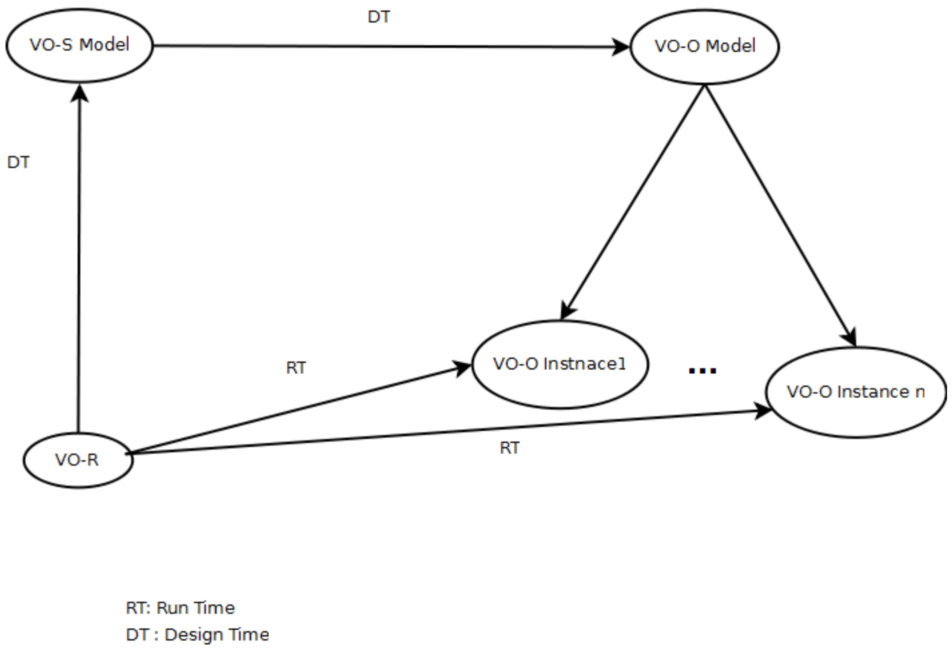


Figure 7.1: Semantic integration of VO-S, VO-R and VO-O at conceptual level

7.5 Summary

In this section we described the syntax of the the VOML sub languages. Furthermore, we also lay down the conceptually the methodology that we intend to use in future to give formal semantics to the sub languages and the issues that could raised due to such integration.

Chapter 8

Evaluation and Discussion

In the course of this work, we have so far shown the example of TravelBK VO. We will now apply the VOML to specify a *ChemicalPD* (Chemical Plant Development) VO to show that VOML lends itself well to model virtual organizations of different complexity and different nature.
This chapter presents our work towards modelling structural and reconfiguration aspect of *ChemicalPD* using the VO-S and VO-R languages respectively. The full VO-S description of *ChemPD* VO is shown in the Appendix B. The ChemPD model in its initial configuration (where all tasks are atomic) is shown in the Appendix B.

8.1 Chemical Process Development Case Study

To assess the practicality of VOML, we have adapted a case study which is based on actual chemical process development (CDP), conducted as part of the GOLD project [5, 27, 14, 3]. As the goals of the GOLD project are focused on providing a SOA based middleware for VOs, access control policies and analysing different properties of the dynamic coalitions (VOs); we only focus on the aspects of this case study that fall into structural and reconfiguration scope. Thus, we extracted the required details only from the publications and technical reports available.

Consider a VBE named *ChemicalVBE* which offers different services related to the field of the Chemical industry. One of the VOs currently operating in this VBE is called *ChemicalPD* which serves existing batch processing plants (that produce a widely-used chemical) which are approaching the end of their serviceable life. *ChemicalPD* comes into play when the owner(s) of the plant decide to phase out/sell the existing plant and build a new chemical plant in place that produces the same chemical. In doing so *ChemicalPD* also helps in converting the plant from batch to more modern continuous operations if possible. The basic project plant in the original case study is divided into following four phases:

- First phase consists of preliminary laboratory level investigation to determine whether conversion from batch to continuous operation is feasible or not. During this phase preliminary trial of downstream processing separation methods is also performed to investigate its compatibility with the continuous operation.
- Based on the results of phase 1, a design for the pilot plant is built during second phase.
- In third phase a small-scale model pilot plant is build to identify suitable modes of operation, potential problems with start up shut-down, etc.
- In the final phase, full scale production plan is built.

8.1.1 Task Structure of the *ChemicalPD*

In order to aid the reader in understanding the task and membership specification of ChemicalPD, VO we have borrowed the Figure 8.1 showing the structure of the chemical development project at the end of phase 1 from [5]. In this figure the box labelled *RE* represents the member performing the laboratory-level preliminary investigation (phase one). The *Laboratory* and *Simulation* labelled boxes show the services that the RE must provide to the VO. The *Eau de cham* box represents the owner and decision maker of the project at each stage. *PP* represents the member responsible for the design of pilot plant and the two *EV* boxes represent the equipment suppliers of the VO. Solid lines between boxes represent those members who remain relatively fixed during different phases of the VO whereas, dashed lines represent those members whose membership might change as the project evolve and unanticipated circumstances might demands changes the in the project plan. For example the EV might change from one providing filtration technology products to one providing alternative centrifuge technology.

The ChemicalPD VO-S model specification consists of four main tasks; each corresponds to one of the phases mentioned above. In particular, the *PreliminaryAnalysis* task corresponds to phase one of the original case study (performed by RE). The Laboratory and Simulation boxes of the figure in turn correspond to VO-S notion of capabilities that the member performing the PreliminaryAnalysis task must posses. The *PlantDesign* task corresponds to the phase two, *PilotPlantBuilding&Operation* and *FullScalePlantConstruction* corresponds to phase three and four respectively.

The Eau de Chem corresponds to the customer of the ChemicalPD in our case. Similarly, PP and and two EVs correspond to the tasks *PlantDesign* and *EquipmentProvision* in ChemicalPD. The solid arrows in this figure represent the relationships which remain relatively fixed and the dashed arrow represent the more uncertain ones. The above mentioned tasks depend on a number of other (supporting) tasks, such as providing different off-the-shelf process equipment according to supplied specifications, supplying custom-built equipment not available from the off-the-shelf vendors and providing catalyst currently used by the reaction processes. These tasks are considered supporting tasks by the ChemicalPD as their role is limited to supplying the material/equipment. The *CatalystProvision* task provides the catalyst required to the ChemicalPD. In the ChemicalPD model of the case study we have only considered one kind of equipment provider named *EquipmentProvision*, performing both the off-the-shelf and custom-built equipment, as these are fairly identical.

It is worth reminding the reader here that in VO-S the tasks that appear under the umbrella of `Process` description are considered main tasks and those that do not are

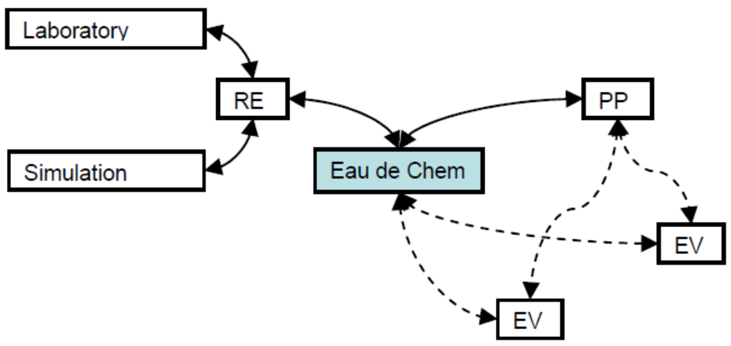


Figure 8.1: The structure of the Chemical Development Project at the end of Phase 1 of original case study

considered supporting tasks. VO-S does not explicitly tag the tasks with any construct. However, a main task specify the supporting tasks, it depends through its *supportedBy* attribute. By doing so, we have attempted to keep the vocabulary of the VO-S language as minimal as possible.

The significance of these different tasks is that we are able to differentiate between the core tasks carrying out some part of the goal (of VO) and those tasks that do not directly form part of the goal. This differentiation may play a key role in assigning different types of members to different tasks; which eventually provides the flexibility to specify different policies for members carrying out those tasks. One such policy could be to try to keep the members performing the main tasks in VO as longer as possible. Whereas, for the members of supporting tasks the VO might afford to have selfish policies which benefit the VO but not the supporting tasks' members and the VO might change those members as many times as it wants. Hence, we are able to differentiate the significance of different tasks and later associate different policies according to the importance of respective tasks in the VO using VOML.

8.1.2 Membership Types of ChemicalPD

In this section we are going to describe the membership structure of the ChemicalPD VO and will try to justify the rationale behind this structure in particular and types of membership identified in VOML in general using the concepts developed at the beginning of the case study and as depicted by the figure 8.1.

1. The members for the main tasks (PreliminaryAnalysis, PilotPlantDesign, BuildPilotPlant and BuildFullscalePlant) have been chosen to be of type *partner*. The competency requirements for these tasks are mainly the same for different type of chemical plants. The solid lines in the Figure 8.1 also indicate to it. Besides, it allows VBE to create multiple (instances of) ChemicalPD VO quickly, as many of its aspects (in this case members for all the main tasks) are going to be same for each instantiation of ChemicalPD. Another benefit of having persistent

members is that with time the ChemicalPD is going to grow more efficient as its persistent members and coordinator keep learning from its past experiences.

2. The catalyst used in the chemical reactions (for the production of the chemical) plays a vital role. The chemistry of the reaction and the separation method both get effected by the properties of the catalyst. However, the goal of ChemicalPD itself is not associated with catalyst itself; the catalyst is required by the tasks satisfying the VO goal. Hence the task of catalyst provision (CatalystProvision) is included in the ChemicalVO as a supporting task. Since every chemical plant has its own specific catalyst supplier with long term relations and contract and the catalyst also varies as per the chemistry used by the chemical plant, this implies that catalyst and its supplier varies from one chemical plant development to another (different ChemicalPD instances) so it can not be the partner of the ChemicalPD. However, the catalyst provider is also not eligible to be considered an external entity as external entity is discovered from the outer open universe (of VBE), but the customer of the ChemicalPD wants a catalyst supplier with whom the customer already has ties and once the plant is operational, the same catalyst supplier is going to supply the catalyst. These restrictions (by the VBE) and requirements (by the customer) make the member supplying catalyst eligible to be an associate who comes from outside the VBE but not from the open universe, rather on customer's recommendation.

3. The equipment providers (both vendors and manufactures) lend very well to be external entities, as they keep changing from chemical plant to chemical plant and this equipment might not be required once the plant is operational

The above membership types clearly satisfies one of the characteristics highlighted by the original case study and that is the relationships between VO members are of different types, some are relatively fixed and some are more uncertain. Our framework provides concrete constructs to represent different types of relationships between the members. The GOLD architecture does highlight these differences but at the architecture level both of relationship types have been implemented with services which in our opinion do not clearly differentiate between them.

8.1.2.1 Company Acquisition and Coordination Difficulties

During the course of chemical plant development in the original case study, one of the events that occurred was that the Eau de Chem company became involved in the company acquisition having the consequence of an additional reporting structure put in place across new organizational boundaries. Many more companies and management groups became involved. Another difficulty faced by Eau de Cham was that they were involved in many more projects at the same time. The resources needed to maintain all the projects with agility, the management efforts required to coordinate the various members and projects and effective communication between different members turned out to be major difficulties faced by most chemical companies.
All of these difficulties prove the fact that having a VO alone is not the solution as suggested by most of the research on the VOs. There needs to be a more stable network dedicated to the coordination, management and quick creation of different projects, where each project (VO in our case) is managed by its own coordinator. The name used for such a structure is called VBE in the VOML framework. On contrary to the path taken in the original case study where Eau de Chem is managing many projects and formed different teams for each project in a single VO, we have created two structures one named *ChemicalVBE* where Eau de Chem is a stakeholder and each project of Chem de Chem now becomes an individual VO. Now, the Eau de Chem has its defined set of responsibilities and the management and coordination responsibilities of each VO are managed by the VO's own coordinator. Having the VBE in place now allows Eau de Chem to exploit its experience and expand its business by allowing the offer of services to the outside world in the form VOs.

8.1.3 Dynamic Nature of ChemicalPD

The chemical process development life-cycle is highly dynamic as unanticipated changes may occur at any time with consequence ranging from structure to competencies and membership. It is crucial that ChemicalPD remains agile and flexible to face both anticipated (through risk analysis) and unanticipated changes.

8.1.3.1 Goal Modification

In this section we have merged two of the original case study's scenarios under the umbrella of goal modification as, both of the scenarios call for similar changes. Before going to describe these scenarios, it is worth recalling that in our work we call the changes in capabilities as the modification to the goal. The goal modification does not radically alter the goal rather the path to achieve the (high level) goal has been modified in some way.

Scenario 1: Conversion not feasible One of the reasons forcing changes in the VO is when for some reason the upgrade from batch operation technology to more modern continuous operation is not possible. The chemical reaction(s) involved during the process may not be compatible with the process being operated in a continuous fashion.

Consequence: This leads to modifying the goal set in the beginning by now focusing on a batch design, with a possible view to upgrading the existing plant. The VOML accommodates for this modification through two changes in the *ChemicalPD* (1) changing the capabilities of the *EquipmentProvision* task of vendors and manufactures to suit the demands of batch processing plants, and (2) by changing the membership of the *ChemicalPD* for vendors and equipment to match the new capabilities.

Scenario 2: Downstream processing problems While upgrading from batch process to continuous process, the new operating conditions unexpectedly affect the downstream recovery of the catalyst. One such problem reported in the

original case study occurred during phase one (*PreliminaryAnalysis*) when the chemistry required for continuous operation turned out to be suitable but, the filtration technology proved to be problematic for the downstream separation of the catalyst. A new separation method using centrifuge technology was therefore initiated with a different equipment supplier.
consequence: This involves (1) change in the capabilities of the equipment provider from one having capability to provide filtration technology equipment to one offering centrifuge technology and, (2) a modification to the VO structure through the replacement of a specialist equipment provider by a new one.
Both of the above scenarios require the same set of reconfigurations in VOML, we are presenting here the reconfigurations and their effects on the *ChemicalPD* model using the downstream processing problem. The downstream processing problem effects the *EquipmentProvision* task by changing its capability from *FiltrationEquipmentProvider* to *ContinuousEquipmentProvider*. This change is reflected through the *changeSeparationTech* policy. The *relinquishTask* and *changeVOmembership* policies unassigns the current member assigned to the task and then expels it from the ChemicalPD, if it is not required elsewhere. The policies are given below:

policy *changeSeprationTech* **appliesTo** *EquipmentProvision*
if `isCapabEquivalent (` *ContinuousEquipmentProvider, FiltrationEquipment-Provider* **)**
do `ReplaceCapability (`*EquipmentProvision, FiltrationEquipmentProvider, ContinuousEquipmentProvider*`)`

policy *relinquishTask* **appliesTo** *EquipmentProvision*
when `MemberTaskMismatch (`*EV1, EquipmentProvision*`)`
do `UnAssignTask (`*EquipmentProvision, EV1*`)`

policy *changeVOmembership* **appliesTo** *EV1*

when `MemberWithoutAnyJob (`*EV1*`)`

do `RemoveMember(`*EV1*`)`

```
AtomicTask EquipmentProvision

STRUCTURE
TaskScope
        {
         performedBy :  ExtEntity
        }
ConfScope
        {
           ...
        }

Competency
           {
            Capability FiltrationEquipmentProvider
                       {
                       resource   FiltrationEquipment
                       capacity
                             {  quantity : 2000 }
                        }

           }
```

(a) *EquipmentProvision* task before capability change

```
AtomicTask EquipmentProvision

STRUCTURE
TaskScope
        {
         performedBy :  ExtEntity
        }
ConfScope
        {
           ...
        }

Competency
           {
            Capability ContinuousEquipmentProvider
                       {
                       resource   ContinuousEquipment
                       capacity
                             {  quantity : 2000 }
                        }

           }
```

(b) *EquipmentProvision* Task after capability change

Figure 8.2: *EquipmentProvision* Task before and after capability change

The policy *changeSeprationTech* replaces the *FiltrationEquipmentProvider* capability with its equivalent capability, the *ContinuousEquipmentProvider*, only if the two capabilities are considered equivalent (in the VBE Competency description). This constraint is verified through the `isCapabEquivalent` condition of the VO-R languages. Note that, the task of providing equipment still remains the same, just the kind of equipment has been altered in some way. This situation however, can lead to a mismatch between the capabilities required by the task from its member and the member currently assigned to it, as the member still possess the the FiltrationEquipmentProvider capability. The policy *relinquishTask* takes care of this situation. This policy gets triggered when the event `MemberTaskMismath` gets fired and takes back the responsibility of the EquipmentProvision task from the current members (EV1). The policy *changeVOmembership* eventually expels the EV1 from the *ChemicalPD*.

Scenario 3: External Event The chemistry and reaction process depends heavily on a particular catalyst. As this catalyst is a naturally occurring substance and its properties vary considerably with the geographical location where it is found. The supplier of the catalyst ceased to operate during the course of plant development. The quest for a new catalyst supplier was complicated by the fact that the catalyst supplied by the new supplier has to be compatible to the original one, that is it must have similar properties.

consequence: Many catalyst sources temporarily joined the *ChemicalPD* VO and phase one (PreliminaryAnalysis) chemistry (reaction processes) was restarted to ensure that the catalyst was an adequate replacement. This process continued until an appropriate supplier was found that worked well with the selected operations method (continuous method). One instance mentioned in the case study is that a new catalyst supplier's catalyst worked well with the old batch plant but not in continuous operation laboratory trials. Hence, the supplier had to leave the *ChemicalPD* and a new supplier was looked for.

Considering this example, the changes that the VO-S model has to undergo to account for the above situation are as follow:

In the VO-S specification the catalyst supplier is an associate because it is recommended by the customer of the VO (Eau de Chem in this case), but when new catalyst sources are tried, the customer might not have any ties with them. Hence, in this case the customer might not be able to suggest new supplier sources. This leads to looking for adequate suppliers from the open universe, making them temporary members of the VO, investigating if the catalyst supplied by the new supplier is adequate choice and based on the result, expelling the temporary member or continue working with it. This situation of temporary membership is dealt naturally by the VOML's *ExtEntity* type of membership i.e. they are invited into ChemicalVO as external entities rather than associate and once an appropriate supplier is found it's membership is turned from external entities to associate again and continues its support in the next stages of the ChemicalVO. In short, we need to make provision for the following kind of changes:

1. Change of membership type from associate to external entity. This change is covered by the *membersRoleUpdate* policy. This policy gets activated as the result of triggering of the event *MemberLeft*.
2. Invite external entity. This change is achieved through the *findNewCatalystSource* policy, which first searches for the new catalyst source, then adds it to ChemicalVO and finally assigns the new member to the Catalyst-Provison task through `SearchMember`, `AddNewMember` and `AssignTask` actions respectively. *memberFound* is a special keyword referencing the name of the member identity who had been returned by the searchMember action.
3. Expel external entity (if not suitable) or, change type from ExtEntity to Associate (if suitable).

policy *membersRoleUpdate* **appliesTo** *CatalystProvision*
when `MemberLeft(`*CP*`)`
do `changeRole (`*CatalystProvision, ExtEntity*`)` **policy** *findNewCatalystSource*
appliesTo *CatalystProvision*
do `SearchMember (`*CatalystProvision*`)`
andThen `AddMember(`*memberFound*`)`
andThen `AssignTask(`*CatalystProvision, memberFound*`)`

policy *newCatalystSourceNotSuitable* **appliesTo** *CatalystProvision*
do `UnAssignTask(`*CatalystProvision, tempMem-id*`)`
andThen `RemoveMember(`*tempMem-id*`)`

policy *newCatalystSourceSuitable* **appliesTo** *CatalystProvision*
do `ChangeRole` (*CatalystProvision, Associate*)

Scenario 4: The Need for Additional outsourcing of Specialist Services As the chemical plant development progressed, unanticipated changes kept occurring from time to time, most of which required the previous tasks to be performed again, which in turn lead to the delay in the completion of project. The potential cost for missed opportunity was therefore very great, especially at the end of the project when the full scale plant needed to be built. The solution found for this problem was to increase the labour and expedite the plant construction. However, the contractor did not have the required amount of labour. Hence the need for additional contractors and one more contractor was added to the VO. Collectively the existing and additional contractor could finish the project in the specified time. This kind of situation can also be seen in the figure 8.1 where there are two equipment providers (labelled as *EV*).

Consequence The provision for sharing a task between more than one member when the members lack the capacity is to replicate the task specification and this is acheived through changing the type of the task to *ReplicalbeTask*. Given below are the policies which account for both of the mentioned changes and the VO-S excerpt for the task *FullScalePlantConstruction* which can now be replicated between more than one member is given in the Figure 8.3. The *expediatePlantConstruction* policy increase the total labour to 1000 which triggers the event `CapacityDeficit` which in turn is handled by *addMoreLabour* policy by making the task replicable.

policy *expediatePlantConstuction* **appliesTo** *FullScalePlantConstruction*
do `ChangeCapacity`(*FullScalePlantConstruction, FullPlantBuildingCapability.labour, 1000*)
policy *addMoreLabour* **appliesTo** *FullScalePlantConstruction*
when `CapacityDeficit`(*FullScalePlantConstruction*)

do `MakeTaskReplicable(`*FullScalePlantConstruction, 2, cooperation* `)`

Scenario 5: Division of Responsibility This scenarios covers the case widely quoted in VO literature as one of the reasons and benefit of VO approach. The issue is when different organizations having specific capabilities want to expand their business by forming synergies with other organizations by creating Virtual organizations (or even VOs within VOs). We have approach this problem by allowing a task to be subdivided into smaller tasks and then each subtask can be assigned to different members. This situation occurs in the *ChemicalPD* when the member performing the preliminary analysis leaves the VO and no members can be found who satisfy all the capabilities of the PreliminaryAnalysis task. This situation of not being successful in finding a single member with all capabilities is triggered by the event `NoMemberWithAllCapabilitiesFound` used in the policy *createSubTasks* below. The decision is made to divide the task into two sub tasks, one task performing the laboratory experiments with the capability *LaboratoryService* and other one with the capability *SimulationService*. These are respectively called *LabExpTask* and *SimExpTask* and are represented in the action `MakeTaskComposable`. Since the *criteria* parameter is omitted from the action which specifies the relationship between the subtasks, the default relation between the two subtask is of type *cooperation*. The cooperation relation between subtasks imply that any sub task can be carried out first or both subtasks can be performed parallely in terms of control flow at the operational model (VO-O) level.

policy `createSubTasks` **appliesTo** `PreliminaryAnalysis`
when `NoMemberWithRequiredCapabilityFound`
do `MakeTaskComposable(PreliminaryAnalysis, [[LabExpTask,LaboratoryService` `], [SimExpTask, SimulationService]])`

8.2 Assessment of Framework According to Criteria

The criterion C1 which is about expressibility of the language in terms of domain concepts as constructs is evaluated in general (without referring to specific parts of specific case study) in the Section 8.3 below. The criterion C2 is shown through different policies that reconfigure the VO model either at structural level or at operational level. The criterion C3 is shown through scenario 1 and scenario 3. In scenario 1 the goal of the VO is modified by requiring different set of capabilities for *Equipment-Provison* task and the respectively the competencies of the member carrying out the task. This Scenario also satisfies the criterion C6. Scenario 3, in particular shows the ability of the languages to change the association of member from one member type to another and the concept of adding temporary member of a VO whose membership is circumstantial.

The autonomy of members (C8) is preserved by interface style communication where all the required details are provided to the member (through tasks) rather then directly performing actions on the associated resources and the members provide back the outcome of carrying out the task.

The criterion C9 is accomplished by keeping the structural model and operational model separate. The Mapping rules guide changes to the operational model whenever any change occurs in the structural model that requires the operational model to undergo some changes so that the operational model reflects the structure existing at the VO-S level.

The criterion C4 can be observed in the scenario 4 where replication and sub-division of tasks occurs. The criterion C5 is achieved by the `relationship` attribute of `TaskScope` part. The criterion C7 points to the situations where a VO can fail and how can such situations be addressed in the VOML framework. In the next section we discuss some of the situations which can lead to the failure of VO in accomplishing its goal or failing to perform.

8.2.1 Handling of Failure

The very concept of VOs was coined to cope with failure such as by seeking business partners with complementary competencies when one organizations fails to achieve a business opportunity, dedicating more resources (physical, human skills and others) to expedite the progress, evolving structurally as well as operationally to align itself to changing business environments, etc. Though VOs by their very nature are resilient to many failures and resist them as long as possible, there still are situations when eventually VOs might fail. In this section we are going to discuss the most common causes of VO failure, which one of those failure are handled by VOML framework and how. We will also consider the limitations of the VOML framework with respect to failures. Given below are most frequent causes of the failure:

1. **Contract violation:** Contract violation occurs when the any members of the VO fails to provide the promised service in any way such as by providing resources less in quantity then actually promised, by poor quality of service, by failing to provide on time, etc. The VOML framework does not specifically specify the repercussions of contract violations. However, some policies in VO-R might be a consequence of how some of the contract violations are dealt at the VBE management level. Hence, the VOML framework can specify the situations through VO-R policies both at VBE level and VO level that describe how to recover from contract violations. The contract violations are not captured through VO-S and VO-O levels though and their resolution is at the discretion of VBE and VO management as well.

2. **Unforeseen external event:** Unforeseen events were described in detail throughout the chapter on the *ChemicalPD* case study. We have also shown through the case study the weaknesses and strengths of handling such events. Even though the VOML framework does not provide provisions to explicitly specify such situations we have shown through different scenarios that VOML can handle most of those failures quite naturally. An example is when the supplier of the

major catalyst provider ceases to operate suddenly and the VO has to look for new members and until it is decided that the new catalyst is fit for the chemistry the member providing that catalyst is included in the VO as temporary and once the decision is taken only then the catalyst provider's membership is changed to permanent one by making it associate of the VO. The only limitation in this case is that the policy describing the event and what to do in such event must be described in advance. So while the VOML framework is rich enough to deal with many different failures, it is the vision and sharpness of management people in the business domain that matters more. The more visionary the management is the more unforeseen situations can be described in advance and the more robust and resilient the VO becomes.

3. **ICT infrastructure failure:** Problems in communication links, such as broken communication lines or delays in communication lines might fail VOs by satisfying the VO goal when the circumstances of business environment has changed and the business opportunity has already been lost.

4. **Fixed organizational structures:** A VO needs to continuously reconfigure itself to keep aligned with the changing business environment. An operational model satisfying the VO goal with close to execution environment level details is by virtue limited in its capability to adapt. By far the most easily made reconfiguration on such model is dynamic discovery and binding of components at run-time. The environments for which VOs are appropriate require far more flexibility then that. Hence, a VO can fail if it is not flexible enough to reconfigure itself. The VOML framework provides a solution to this problem by keeping the structural aspect of the VO separate from the operational aspect. The structural aspect is abstract enough to allow for far more flexibility in changing different aspects of VO description, and then using mapping guidelines to derive concrete operational description of VOs. This way, a VO defined through the VOML framework provides more flexibility to change different aspects of the VO description at different levels of abstraction leading to more robust and

resilient VOs.

8.3 Comparison with the ARCON Reference Model

In this section we evaluate the VOML framework with respect to the ARCON reference model introduced in Section 2.1.3.1. ARCON is a reference model which provides basic elements any manifestation of VOs or VBEs (termed CNOs in ARCON) possesses. This reference model explores the CNOs in detail and attempt to provide abstract models that capture the most fundamental concepts that CNOs possess. Hence, evaluating the VOML framework in the light of ARCON reference model would help better understand the VOML framework.
The overview of the ARCON reference model is described in the Section 2.1.3.1. In this section we are going to compare and contrast the VOML framework with respect to different aspects of the ARCON as following:

Life cycle stages: The life cycle stages of CNOs consists of creation, operation, evolution, metamorphism and dissolution. The VOML framework covers, operation and evolution stages of VO life cycle using different modelling languages and by modifying goal of the VO by changing the capabilities of the tasks. Metamorphism is captured by abandoning current VO and creating new VO. Creation and dissolution stages of life cycle are not currently captured by the VOML framework.

Environmental characteristics: Environmental characteristics perspective is further divided into two parts: (a) the environment internal to the VOs and VBES (*Endogenous characteristics*) and (b) the environment external to the VBEs and VOs (*Exogenous interactions*). Our works falls under the *Endogenous characteristics* of the second perspective, which in turn consists of four dimension as follow:

Structural dimension addresses the composition of the VBEs and VOs. It consists of participants, relationships among them and the tasks performed by the participants. These characteristics are captured through VO-Structural (VO-S) language.

Componential dimension focuses on tangible/intangible elements of VBEs and VOs such as resources (human, software, hardware) or conceptual resources such as knowledge. The VOML framework provides competency, capability and VBEasset constructs to capture physical aspects of the resources. Intangible resources such as skills possessed by humans can be captured by capability construct of the VO-S.

Functional dimension addresses base functions/operations available and time sequenced flow of executable operations related to different phases of the life cycle. The VOML framework allows abstract control flow constructs in VO-S in the form of `Process` elements and relationship between replicable tasks and subtask flow between subtasks of a composable task. Then, at the VO-O level, a concrete process is defined from these elements with an exact sequence of operations.

Behaviour dimension addresses the principles, policies, and governance rules that drive or constrain the behaviour of VBEs and VOs members. So far VOML does not provide such policies except for the evolution and reconfiguration of VOs through VO-R language.

VOML has not attempted to capture the *Exogenous Interactions* which consist of *Market, Support, Societal* and *Constituency dimensions* because to the most part these interactions refer to elements with respect to VBEs which is the focus of this PhD. However, the aspect such as interaction with the customers of VOs are covered at the state configuration level through VO-O language.

Modelling intent This perspective is divided into three layers: (a) *General representation layer* this layer includes the most general concepts common to different manifestations of VBEs and VOs, irrespective of the domain. We captures such modelling concepts by proving constructs general to all domain through VO-S modelling language such as different types of tasks, competency specification, etcetera and VO-R languages which allows any sort of VOs can be reconfigured by identifying basic events, conditions and actions applying to all domain irrespective of particular application.

(b) *Specific modelling layer* includes more detailed models focused on different manifestations of VOs such as their topology. VOML develops concrete models using VO-S and it indirectly defines the topology using the `Process` elements of the VO-S languages and the relationship that exists between replicated or composable tasks. The topology in our framework is based in the information flow in VO rather the control and management structure put into place.

(c) The *Implementation Modelling layer* represents models of concrete VBEs and VOs. VOML have dedicated VO-O modelling language for the purpose.

In General VOML is able to capture most of the basic elements defined in the ARCON reference model especially with respect to VOs (and its different manifestations) with enough details.However, we can not claim that VOML realizes each and every aspect of the ARCON model.

8.4 Evaluating the VOML Framework with Respect to Questions set out at the Start

One of the questions that we started out (in Section 1.4.3) was to know the generality of the modelling languages developed in the VOML framework. We have modelled two case studies using VOML framework; one case study (travel itineraries) is commonly found in literature and used as proof of concepts, another case study is an adaptation is of a real world situation which is about construction of a chemical plant [14]. Based on the confidence gained from modelling these two case studies we are hopeful that

the framework is generic enough to model different application domains of VOs.
The other two questions that we needed answer to was concerning the agility demands of VOs and getting away with the fixed organization structure of the systems (VOs). The VOML framework has approached these issues in two: (a) firstly a dedicated language has been developed which reconfigures the VOs specified in VOML framework, (b) secondly, the level of agility required by VOs demands that the VOs should be able to structurally change themselves altogether to cope with the new realities of the business environment they are operating in. In this regards the constructs has been made available in the VO-S language (that define the structure of the VOs) which are capable to me modified having the impact on the structure of the VOs model; these structural changes however, do not change the business aspect of the VO (i.e the service(s) offered). These constructs are the provisions for the tasks to be changed from being performed by one member only to multiple members, decomposing a task into smaller manageable tasks, changing topology by changing the relationship between VO members, etc.
Most of the reconfigurations are applied to the VO-S models from which VO-O models are generated whose instances execute to serve the business purpose of the VOs. The reconfigurations applied to VO-S models only affect the new instances of the VO-O models. No reconfigurations that can change the structure of the VOs are applied while any VO-O instance is running. However, the reconfigurations that only make changes to the VO-O instances only are applied to the individual instances and their affect does not persist beyond the VO-O instance on which these reconfigurations were applied. This reconfiguration only applies to the joining and leaving of external entities.

8.5 Summary

In this chapter we attempted to evaluate the VOML framework and its corresponding modelling languages for a more realistic case study of chemical plant development. The original case study worked as requirements that the service based middleware must exhibit to cope with dynamic and complex nature of VOs. We captured different complexities and flexibilities demanded of VOs mentioned explicitly or implicitly and than tried to come up with VOML based solution to it or modelling them through VOML's modelling languages.

Furthermore, we evaluated the VOML framework with ARCON reference model is provides basic elements that any manifestation of VOs and VBEs exhibit. Comparing it with ARCON further strengthens the VOML framework.

We also evaluated the framework with the criteria identified in the Section 1.2 in the context of the case study and identified the strengths and weaknesses of the framework. We also discussed how the failures are handled in the framework.

We concluded the chapter by revisited the questions raised in the beginning and looked examined whether the framework has been successful in answering those questions and how has that questions approached by the VOML framework.

```
AtomicTask : FullScalePlantConstruction

STRUCTURE
TaskScope
          {
           performedBy: Partner
           supportedBy: EquipmentProvision, PlantDesign
          }
ConfScope
          {
               ...
          }

Competency
{
              Capability  FullPlantBuildingCapability
                       {
                         resource labour
                         capacity  {totalLabour :500}
                        }
}
```

(a) FullScalePlantConstruction Task before Replication

```
ReplicableTask : FullScalePlantConstruction

STRUCTURE
TaskScope
          {
           performedBy :  Partner
           allowedMembers : 2
           relationship : cooperation
          supportedBy : EquipmentProvision, PlantDesign
          }
ConfScope
          {
          currentMemebrs : 1
          currentState : atomic
          }

Competency
{
              Capability  FullPlantBuildingCapability
                       {
                         resource labour
                         capacity  {totalLabour :1000}
                        }
}
```

(b) FullScalePlantConstruction Task after Replication

Figure 8.3: *FullScalePlantConstruction* Task before and after Replication

Chapter 9

Conclusion

In this chapter, we conclude the thesis by having some concluding remarks and presenting further extensions to our work.

9.1 Concluding Remarks

In this dissertation, we put forth the VOML - a new framework for modelling VOs. The VOML is a compendium of sub languages each focusing on a particular dimension of VOs at a particular level of its representation. Through these sub languages and different levels of representations the VOML exposes an incremental approach where domain level details are defined using terminology familiar to different stakeholders of the VOs and the VBEs. The domain level description (modelling) is raised to highest level of abstraction which describes basic characteristics of the VOs and the VBEs. This description exists at the business configuration level of the representation. At the state configuration level of the representation, operational models of VOs are defined from the information available in the domain level description.
We have defined three novel languages for the VOs, 1) the VO-S (VO-Structural) modelling language for modelling structural aspects of a VO, 2) the VO-O (VO-Operational) modelling language for describing operational aspects of a VO, and 3) the VO-R (VO-Reconfiguration) language concerned with reconfigurations at the struc-

tural and operational level. The VO-R aids provisions to the VOML framework to cater to the agility and flexibility demands of the VOs. It defines a set of events, conditions and actions (ECAs) which apply to all the VOs irrespective of the application. These ECAs enable a VO to reconfigure itself in face of unpredictable events and changes in the business environment. The VO-R is an extension of the APPEL policy language which has built in support to be extended to different domains.
The VOML framework also provides guidelines (mapping rules) to establish a link between the structural (VO-S) and the operational (VO-O) models of a VO. Using these guidelines an operational model of a VO is derived from the VO-Specification.
In this work, the VOML was applied to two case studies to validate its applicability. One of the case studies was used as running example and offered travel itineraries as a VO service (called the *TravelBK* VO). The other case study is an adaptation of a case study carried out in a realistic setting which is about developing a chemical plant [14].
To the best of our knowledge, the VOML is currently the only solution which 1) specifies the VOs using domain terminology, 2) reconfigures the VOs to cope with the dynamically changing circumstances, 3) generates operational model closer to underlying execution environment, and 4) shows the affects of structural reconfiguration on the operational model. Although other attempts at modelling VOs exists, their focus is on analysis of different characteristics of VOs or providing formal models. The VOML complements those aspects by providing languages which can be used to describe the application (business) side of the VOs.
The VOML framework is generic and can accommodate most of the manifestations of VOs.

9.2 Future Work

This section puts forward a few ideas for extending the work presented in this thesis.

Currently the VOML framework does not provide any tool support for transforming a VO-S model into its corresponding VO-O description and a tool to verify that a VO-O model would be desirable. This will require the formalization of mapping rules (that are currently in the form of guidelines) for the purpose of automation. We should also need to define the formal operational semantics of the languages as well for that purpose.

The other avenue worth exploring is semantic integration of the sub languages based on the ideas mentioned in the Chapter 7. The VOML framework is capable of modelling structural aspects of VBEs; the suitability of VO-S for describing the behavioural aspects of VBEs such as creation and deletion of VOs, inviting and expelling members from VBE is another direction worth investigating. More precisely, investigating in detail to examine weather the VBE can be treated just like a VO or there needs to be special considerations for modelling VBEs. In this PhD, we have described policies related to reconfiguration of VOs. Future work should be concerned with describing different management and governance policies of the VBE; such as describing policies which trigger to VBE that a certain VO is not profitable and hence should be dismantled, or indicating and constraining the use of VBEassets by a VO over a certain threshold. Extensions to VO-R to express different governance rules of VBEs is another aspect we are planning to explore in future. Another future direction we are intending to follow is to provide formalism describing architectural evolution of VBE to account for the evolution of VBE business configurations. Example of which can be formalising the effect on the structure of VBE after some evolution such as creation of new VOs, their termination or modification. From a formal point of view, the different level of VBE representation are graphs whose nodes are component specifications and the edges (wires) are connectors. Component specifications provide either interfaces for members and resources to be involved in tasks and services offered through VOs, or orchestrations of those services, or requirements for external services, or properties offered to customers of VOs. Choosing graphs as formal models allow us to use techniques that have been proposed for formalising architectural aspects of system structure and evolution (eg, [43]) in order to account for such evolutions.

As formalisms for specification, we are using those put forward for service-oriented modelling in the SENSORIA project. Together with the graph-based representation of business configurations, these formalisms can be used for inferring emergent properties of VOs. Model-checking techniques have been used for verifying properties offered by services, which we plan to extend to VOs. The proposed formal model also supports forms of quantitative analysis using the stochastic analyser PEPA [42], which we intend to extend to VOs.Another aspect of future work is performing some performance modelling based on variants of VO-S models, but considering how certain VO-R rules will allow for the system to be better adaptable to cope with changes. This could lead to a set of rules that are useful to have in the repertoire of any VO.

Appendices

Appendix A

TravelBK : A travel itinerary offering VO

VO-Smodel TravelBK

```
        ...
AtomicTask TransportProvision

STRUCTUURE
TaskScope
            {
            performedBy : ExtEntity
            }
ConfScope
           {    ... }
Competency
{
   Capability byAir
      {
       resource : plane
       capacity {flightsPerDay : 30}
      }
}

BUSINESS FUNCTIONALITY
Request:
            from, to: location
           out,in: date
           traveller: usrdata
Reply:
           flightBooking.fcong: fcode
          amount : money; bank-id

AtomicTask HotelBooking
```

```
STRUCTURE
TaskScope
{
performedBy: Partner
}
ConfScope
             {
              }
Competency
                 {
                  capability roomReservation
                           {
                            resource: rooms
                            capacity
                                     {totalRooms : 500}
                            }
                }

BUSINESS FUNCTIONALITY
Request:
            from, to:location
            out, in:date
            name : usrdata
```

Reply :
roomReservation.hconf : hcode
amount : moneyValue

AtomicTask GuideProvision

STRUCTURE
TaskScope
{
performedBy: VBEParticipant
}
ConfScope
{ ... }

Competecy
{
capability guidingTour
{
resource: guide
capacity
{totalGuides : 250}
}
}

BUSINESS FUNCTIONALITY
{
Request :
start, end: date
name : usrdata
Reply :
amonut: moneyValue
guidingTour.gInfo : guideInfo
}

DATA FLOW
Customer.from **=⇉** TransportProvision.from
Customer.to **=⇉** TransportProvision.to
Customer.out **=⇉** HotelBooking.checkin, TransportProvision.out, GuideProvision.start
Customer.in **=⇉** HotelBooking.checkout, TransportProvision.in, GuideProvision.end
usrDB.usrdetails **=⇉** (TransportProvision.traveller
Customer.amount **⇇=** HotelBooking.amount, TransportProvision.amount, GuideProvision.amount, VO.amount
Customer.hconf **⇇=** HotelBooking.hcong
Customer.fconf **⇇=** TransportProvision.fconf
Customer.gInfo **⇇=** GuideProvision.gInfo

Members

Partners

Partner *AccomProvider1*
{
performsTask : {HotelBooking}
competency {roomReservation.room.totalRooms : 500}
}
<u>Associates</u>

Associate *TourAg*
{
performsTask : {*GuideProvision***}**
competency
{ guidingTour.guide.totalGuides : 250 **}**
}

Appendix B

ChemicalPD : A Chemical plant developing VO

Tasks

```
PreliminaryAnalysis
PlantDesign
PilotPlantBuilding&Operation
FullScalePlantConstruction
VendorEquipProvision
ManfEquipProvision
CatalystProvision
```

Process

```
                {
                 satistifyTask ( PreliminaryAnalysis) leadsTo  satistifyTask (
PlantDesign)

                 satistifyTask ( PlantDesign) leadsTo  satistifyTask
(PilotPlantBuilding&Operation)

                 satistifyTask (PilotPlantBuilding&Operation) leadsTo
                                                          satistifyTask
(FullScalePlantConstruction)
              }
```

ComposableTask PreliminaryAnalysis

STRUCTURE

```
TaskScope
              {
                performedBy: Partner
                allowedMembers : 2
                allowedSubtasks : 2
                currentSubTasks : noSubTasks
                subTask-Flow :  none
                supportedBy : EquipmentProvision, CatalystProvision
               }
ConfScope
              {
               currentMembers : 1
               currentState: atomic

               }
Competency
                 {
Capability LaboratoryService
              resource  xEquipment
              capacity {
                          simulExp : 8
                         }

Capability SimulationService
              resource simSoftware,
              capacity  {
                          handleData : 40 TeraByte
                           }
                }
```

BUSINESS FUNCTIONALITY

```
{
Request :
                reactionDetails
                  catalystProperties
                  sepTechnology
Reply :
               LaboratoryService.feasibilityReport ,
               LaboratoryService.sepTrialReport
               SimulationService.
                KineticMoel
}

AtomicTask  PlantDesign

STRUCTURE
TaskScope
            {
             performedBy : Partner
            supportedBy : EquipmentProvision, PreliminaryAnalysis
             }
ConfScope
            {
                  ...
            }

Competency
{
             Capability  ChemicalPlantArcitectingService
                       {
                          resource plantArcitects
                          capacity totalArcitects  :15
                       }
}

BUSINESS FUNCTIONALITY
{
Request :
                  kineticMoel
                  equipSpec
                  probDescription
Reply :
                   plantDesign
}

AtomicTask : PilotPlantBuilding&Operation

STRUCTURE
TaskScope
            {

             supportedBy : EquipmentProvision, CatalystProvision,
                           PlantDesign
            }
ConfScope
           {

           }
Competency
               {
```

```
      Capability  PilotPlantBuildingCapability

          {
              resource labour
              capacity  {totalLabour :100}
              }

             Capability OpearitonAnalysisCapability
              {
              resource : operatoinalAnalyser
              capacity  {analysisResults : n days }
               }
                }

BUSINESS FUNCTIONALITY
{
Request :
                 plantDesign : esgnDemands,
                 sepEquip  : sepEquip-id,
                 processDetails : prcDetails,
Reply :
         PilotPlantBuilt : bool ,
         operationalDataAnalysisReport
}

AtomicTask : FullScalePlantConstruction

STRUCTURE
TaskScope
            {
              performedBy: Partner
              supportedBy: EquipmentProvision, PlantDesign
             }
ConfScope
           {
                 ...
           }

Competency
{
                 Capability  FullPlantBuildingCapability
                            {
                              resource labour
                              capacity  {totalLabour :500}
                             }
}
BUSINESS FUNCTIONALITY
{
Request :
              plantDesign : esingDetails,
              sepEquip : sepEquip-id
Reply :
              fullScalePlantReady : bool
}
```

```
AtomicTask EquipmentProvision

STRUCTURE
TaskScope
          {
           performedBy :  ExtEntity


          }
ConfScope
          {
              ...
          }

Competency
             {
              Capability FiltrationEquipmentProvier
                          {
                          resource   FiltrationEquipment
                          capacity
                                 {  quantity : 2000 }
                          }

               }

BUSINESS FUNCTIONALITY
{
Request  :
           equipSpec : specDocType
           equipQuantity : Quantity
Reply  :
          equipDetails : specDoc
          quantConf : bool
}

AtomicTask  CatalystProvision

STRUCTURE
TaskScope
          {
          performedBy : ExtEntity
          supportsTask :
          }
ConfScope
          {
                 ...
          }
 Competency
         {
             Capability calatystXprovision
                      {
                       resource  catalystX
                       capacity {quantity : 500KG}
                      }
         }
```

```
BUSINESS FUNCTIONALLITY
{
Request :
              ctlstQuantity : Quantity
Reply :
              confirmation :  bool
}

Members

Partners

VOCoordinator chemCoor

Partner REC
    {
  PerformsTask : PreliminaryAnalysis
Competency
                  {
                   LaboratoryService.simulExp : 8
                  SimulationService.handleData : 40 TeraByte OR licence : 60
                  }
    }

Partner  PPD
    {
     PerformsTask : PlantDesign
     Competency
                      {
                        ChemicalPlantArcitectingService.arcitecureDelivery : x-days
                      }
     }

Associates

Associate TC
    {
      PerformsTask :  PilotPlantBuilding&Operation,
                      FullScalePlantConstruction
Competency
                 {
                  PilotPlantBuildingCapability.constructionTime : x days
                 OpearitonAnalysisCapability.analysisResults : n days
                 FullPlantBuildingCapability.constructionTime : x days
                 }
     }
    }

AtomicTask EquipmentProvision

STRUCTURE
TaskScope
             {
              performedBy :  ExtEntity
              }
ConfScope
            {
                ...
```

```
        }

**Competency**
               {
                 **Capability** ContinuousEquipmentProvier
                               {
                               **resource**   ContinuousEquipment
                               **capacity**
                                       {  quantity : 2000 }
                                }
  }
```

Appendix C

VOML Syntax

```
grammar org.xtext.example.mydsl.Voml3 with org.eclipse.xtext.common.
    Terminals

generate voml3 "http://www.xtext.org/example/mydsl/Voml3"

// Project: voml3
// lang: Voml3
//ext: voml
Voml:
 (vosModel = VOSmodel)
 (vooModel = VOOmodel)
 (vorModel = VORmodel)
 "datatypes" (datatypes += DataTypes)*;
 DataTypes:
 "sorts" (datatype += DataType)*
          interval += Interval*;

DataType: name = SortName;
SortName : name = ID ;
 Interval : "[" initialVal = INT ".." finalvalue=INT "]" ;


VOSmodel:

    "VO–Smodel" name = ID
 "tasks" (tasks += Tasks)*
 "vbe" "assets" (vbeAssets += VBEasset)*
 "process" (process = Process)
 taskSpecification = TaskSpecification
    assetSpecification = AssetSpecification
 "data-flow" dataFlow = DataFlow
 "members" (members = Members) ;

Tasks: name = ID;
VBEasset : name = ID;

Process: "{" processStatements += ProcessStatement+ "}" ;

ProcessStatement: processOpernad += ProcessOpernad "leadsTo"
    processOpernad += ProcessOpernad;

ProcessOpernad : SingleSatisfyTask | MultiplSatisfyTask | UseAsset;

SingleSatisfyTask: "satisfyTask" "(" task = [Tasks] ")" ;
MultiplSatisfyTask : "satisfyTasks" "(" task += [Tasks] ("," task +=
     [Tasks])+ ")" ;
UseAsset : "useAsset" "(" assets += [VBEasset] ("," assets += [
    VBEasset])* ")"  ;


/* ************************************************ */
/* ************** Task Specification *************** */
/* ************************************************* */
TaskSpecification :(tasksSpec += TaskType)*;

TaskType : AtomicTask | ReplicableTask | ComposableTask;
AtomicTask :"atomic" "task"   name = [Tasks]
           (structure = Structure)
```

```
        "business" "functionality" (businessFuc =
            BusinessFunctionality)
        ;
Structure : "structure" (taskScope = TaskScope)
        "confScope" "{" (confScope = ConfScope) "}"
        "competency" "{" (competency = Competency) "}"   ;

/* ****************************************************** */
/* ***************** TaskScope Attributes Definition ******* */
/* *****************************************************   */
TaskScope: "taskScope" "{" (taskScopeAttr +=
    AtomicTaskScopeAttribute)*  "}";
TaskScopeAttribute: SupportedBy |  PerformedBy| SupportsTask|
    AllowedMembers |  Relationship | SubTaskFlow |subTasks=
    CurrentSubTasks;
AtomicTaskScopeAttribute returns TaskScopeAttribute :  SupportedBy |
      PerformedBy| SupportsTask;
 SupportedBy: "supportedBy" ":" taskName = [Tasks] ;
 SupportsTask: "supportsTask" ":" taskName = [Tasks];
Relationship:"relationship" ":" relation = Relation;
enum Relation:  cooperation = "cooperation" | Competition |
    custChosen= "customerChosen" ;

Competition: "competition" "(" comType = CompetitionType compAttr =
    CompetitionVariable  ")";
enum CompetitionType: lowest = "lowest" | highest = "highest" |
    quickest = "quickest" | fcfs = "firstComeFirstServe"  ;
CompetitionVariable: name = ID  ;
AllowedMembers:"allowedMemebrs" ":" totalMembers = INT ;
PerformedBy: "performedBy" ":" name = MemberType;
enum MemberType: partner = "partner" | associate = "associate" |
    extEntity = "extEntity" | vbeParticipant = "vbeParticipant" |
    noPref= "noPreference";
SubTaskFlow: Competition;
 enum CurrentSubTasks: noSubTasks = "noSubTasks" | CurSubTaskList;
```

```
CurSubTaskList: subTaskName += SubTaskName  (”,” subTaskName +=
    SubTaskName)+ ;
SubTaskName: name = ID;
/* ******************************************************* */
/* ***** TaskScope Attributes for ReplicableTask Definition ******/
/* *********************************************************/

TaskScopeForRep: ”taskscope” ”{”
                                    (repTaskScopeAttr +=
                                        RepTaskScopeAttribute)*
                                  ”}”;
RepTaskScopeAttribute returns TaskScopeAttribute :  AllowedMembers |
     PerformedBy | SupportedBy | SupportsTask |  Relationship ;

/* **********************88******************************** */
/* **** TaskScope Attributes for ComposalbeTask Definition *******
    */
/* *********************************************************
    */


TaskScopeForComp:”taskScope” ”{”
                               (comTaskScopeAttr += CompTaskScopeAttribute
                                   )* ”}”;
CompTaskScopeAttribute returns TaskScopeAttribute : SubTaskFlow |
    SupportsTask| PerformedBy | SupportedBy  | AllowedMembers |
    subtaskList= CurrentSubTasks  ;

/* ***************************************************** */
/* ************* ReplicableTask Definition ************ */
/* ****************************************************** */

ReplicableTask :”replicable” ”task”  name = [Tasks]
         (repStructure = RepStructure)
```

```
        "business" "functionality" (businessFuc =
            BusinessFunctionality);

RepStructure :"structure" (repTaskScope = TaskScopeForRep)
         "confScope" "{" (confScope = ConfScope) "}"
         "competency" "{" (competency = Competency) "}"    ;


/*
    ************************************************************************
     */
/* ******************* ReplicableTask Definition
    ************************ */
/*
    ************************************************************************
     */

ComposableTask :"composable" "task" name = [Tasks]
         (compStructure = CompStructure)
         "business" "functionality" (businessFuc =
             BusinessFunctionality)

         ;
CompStructure : "structure" (taskScope = TaskScopeForComp)
        "confScope" "{" (confScope = ConfScope) "}"
         "competency" "{" (competency = Competency) "}"
         (subTasks = SubTaskSpecification)?
             ;
SubTaskSpecification: (subTasks += SubTask)+;
SubTask: "atomic" "task" name = [SubTaskName] "{" "competencies" ":"
     "{" (subTaskCompetencies = SubTaskCompetencies) "}" "}";
SubTaskCompetencies: subTaskCompetency += [Capability] (","
    subTaskCompetency += [Capability] )*;


```

```
/* ************************************************************ */
/* ********************** ConfScope Attributes Definition ***** */
/* ************************************************************
    */

ConfScope: CurrentMemebrs | currState = CurrentState;
CurrentMemebrs: "currentMemebrs" ":" currentMembs = INT;
 enum CurrentState: atomicState = "atomicState" | replicatedState =
     "replicatedState" | composedState = "composedState" ;

/* **************************************************** */
/* **************** Competency Definition **************** */
/* ***************************************************** */

Competency : (capabilities += Capability)* ;
Capability: "{" "capability" name = ID
              "resource" ":" capabilityResource = CapabilityResource
               "capacility" "{" capacityName = ID ":" capacityType =
                   INT  "}"
             "}"
 ;
CapabilityResource:  name = ID;

BusinessFunctionality: request = RequestPart reply = ReplyPart;
RequestPart: "request" ":" (requestDataItem += RequestDataItems)*;
ReplyPart: "reply" ":" (replyDataItem += ReplyDataItems)*;
RequestDataItems: (capId=[Capability] ".")? variableName = ID ":"
    type = [DataType] ;
ReplyDataItems: variableName = ID ":" type = [DataType] ;


/* ***************************************************** */
/* ************************ Data-Flow Definition ******** */
/* **************************************************** */

```

```
DataFlow : dataFlowStmts += DataFlowStatement*;
DataFlowStatement:leftOperand = Operand operator = DataFolowOperator
     rightOperand += Operand;
DataFolowOperator: ForwardArrowOpr | BackArrowOpr;
ForwardArrowOpr: opr = "==>";
BackArrowOpr : opr = "<==" ;

Operand : operandName=ID ;

/* ******************************************************* */
/* ****************** Member Definition ******************* */
/* ******************************************************* */

Members: "partners" (partner += Partner)*
         "associates" (associates += Associate)*
         "VOcoordinator" (coordinator = VOcoordinator);

Partner: "partner" name = ID
        "performsTask" (performsTask += MemberPerformsTaskList)*
        "competency" "{" competencyList += MemberCompetencyList "}"
        ;


Associate: "associate" name = ID
        "performsTask" (performsTask += MemberPerformsTaskList)*
        "competency" "{" competencyList += MemberCompetencyList "}"
        ;
VOcoordinator: coordinatorName = ID;
MemberPerformsTaskList: "{" tasksPermedByMember =
    TasksPerformedByMember "}" ;
MemberCompetencyList:"{" contrComp += MemberContributedCompetency
    ("," contrComp += MemberContributedCompetency )* "}" ;// name=ID;

TasksPerformedByMember: taskName+= [Tasks] ("," taskName+= [Tasks]);
```

```
MemberContributedCompetency: compName = [Capability] ":" value = INT
    ;

/* ************************************************** */
/* ******************* VBEasset Definition *********** */
/* *************************************************** */

AssetSpecification: (assetTypes += AssetType)*;
AssetType: VBEresource | VBEtask ;


VBEresource: "vbe" "resource" name = [VBEasset]
        "business" "functionality" (businessFuc =
            BusinessFunctionality);

        VBEtask: "vbe" "task" name = [VBEasset]
        "business" "functionality" (businessFuc =
            BusinessFunctionality);

/* ********************************************************* */
/* ********************** VO-R Definition ****************** */
/* ********************************************************* */

VORmodel :
        "VO-R" name=ID
        (policyRuleGroup+=PolicyRuleGroup)*;

Random: Member | Task | VO;
Member: name =[Members] ;
Task: name = [Tasks] ;
VO: name =ID ;

PolicyRuleGroup: policyRule += PolicyRule (policyOpr += PolicyOpr
    policyRule += PolicyRule)*;
 PolicyOpr :Guarded | UnGuarded | Parallel | Sequential ;
 Guarded : "g" "(" condition += Conditions")";
 UnGuarded : unguarded = "u";
 Parallel: parallel = "par" ;
 Sequential : seq = "seq";

PolicyRule: ("policy" policyName=ID)?
            ("appliesTo" (location = Location)*)?
            ("when" (triggers = Triggers))?
            ("if" (conditions = Conditions))?
            "do" (actions = Actions);

Location : name = [Random];Triggers : trigger += VORTrigger ("or"
    trigger += VORTrigger)?;
Conditions:conditionOperandWrapper += ConditionOperandWrapper (
    condOpr += ConditionOpr conditionOperandWrapper +=
    ConditionOperandWrapper)*;
ConditionOperandWrapper: (notCondition ?= "not")? condition +=
    Condition  ;
Actions: action += Action (actionOpr += ActionOpr action += Action)
    *;

VORTrigger:
    MemberWithoutAnyJob
  | MemberLeft
  | CapacityDeficit
  | CapabilityDeficiency
  | TaskWithoutAnyMemebr
  | NoMemebrWithAllCapabilitiesFound
  | MemberJoined
  | NoMemberWithRequiredCapacityFound
  | MemberTaskMismatch
  | MoreMembersAssignedThanPermissible
  | MemberRoleMismatch;


```

```
MemberWithoutAnyJob: "memberWithoutAnyJob" "(" memberId=[Members] ")
    ";
MemberLeft : "memberLeft" "(" memberId = [Members] ")";
CapacityDeficit: "CapacityDeficit" "(" taskId = [Tasks] ","
    capacityId = ID  ")" ;
CapabilityDeficiency : "CapabilityDeficiency" "(" taskId = [Tasks]
    "," capabilityId = [Capability] ")" ;
TaskWithoutAnyMemebr: "TaskWithoutAnyMember" "(" taskId = [Tasks] ")
    ";
NoMemebrWithAllCapabilitiesFound : "NoMemebrWithAllCapabilitiesFound
    " "(" taskId = [Tasks] ")";
MemberJoined :"MemberJoined" "(" memberIDId = [Members] ")" ; //nr:
NoMemberWithRequiredCapacityFound : "
    NoMemberWithRequiredCapacityFound" "(" taskId = [Tasks] "," "["
    capabilityList = ID"]" ")" ;
MemberTaskMismatch :"MemberTaskMismatch" "(" memberID = [Members]
    "," taskId = [Tasks] ")" ;
MoreMembersAssignedThanPermissible :"
    MoreMembersAssignedThanPermissible" "(" taskId = [Tasks]  ")" ;
MemberRoleMismatch:"MemberRoleMismatch" "("  memberId =[Members] ","
     taskId = [Tasks]    ")";

/* ******************************************** */
/* ****************** CONDITIONS ************** */
/* ********************************************* */

Condition: name = ConditionName;
enum ConditionOpr: and = "and" | or = "or" | not = "not";
ConditionName:
    IsTaskAtomic
  | IsTaskComposable
  | IsTaskReplicabe
  | IsTaskWithoutAnyMember
  | IsMemberWithoutAnyJob;

```

```
      IsTaskAtomic  :"IsTaskAtomic" "(" taskId = [Tasks] ")" ;
  IsTaskComposable: "IsTaskComposable" "(" taskId = [Tasks]  ")" ;
  IsTaskReplicabe: "IsTaskReplicabe" "(" taskId = [Tasks]  ")" ;
  IsTaskWithoutAnyMember: "IsTaskWithoutAnyMember" "("taskId = [
      Tasks]  ")" ;
  IsMemberWithoutAnyJob: "IsMemberWithoutAnyJob" "(" taskId = [Tasks
      ] ")" ;

/* ********************************************** */
/* *************** ACTIONS ********************* */
/* *********************************************** */

Action: name = ActionName;
enum ActionOpr: and = "and" | or = "or" | notthen = "andthen" |
    orelse ="orelse" ;
ActionName:
         RemoveMember
       | AddNewMember
       | AddAltrCapab
       | DeleteAltrCapab
       | ReplaceCapability
       | AssignTask
       | UnAssignTask
       | MakeTaskComposable
       | MakeTaskReplicable
       | MakeTaskAtomic
       | ChangeRelationship
       | ChangeRole
       | ChangeAllowedMembers
       | SearchMember
       | ChangeCapacity   ;

        RemoveMember :"RemoveMember" "(" memberId = [Members] ")";
        AddNewMember :"AddNewMember" "(" memberId = [Members] ")";
        AddAltrCapab :"AddAltrCapab" "(" taskId = [Tasks] ","
            currentCapabId=[Capability] "," equivCapabId=ID ","
            criteria = ID")";
        DeleteAltrCapab :"DeleteAltrCapab" "(" taskId = [Tasks] ","
            equivCapabId=[Capability] ")";
        ReplaceCapability :"ReplaceCapability" "(" taskId = [Tasks]
            "," currentCapabId=[Capability] "," newCapabId=ID ")";
        AssignTask : "AssignTask" "(" taskId = [Tasks] "," memberId =
            [Members] ")";
        UnAssignTask : "UnAssignTask" "(" taskId = [Tasks] ","
            memberId = [Members] ")";
        MakeTaskComposable : "MakeTaskComposable" "(" taskId = [Tasks]
             "," "[" subTasksAndCorrespondingCapabs = ID"]" ")";
        MakeTaskReplicable : "MakeTaskReplicable" "(" taskId = [Tasks]
             "," maxMembers = INT "," relation= ID  ")";
        MakeTaskAtomic : "MakeTaskAtomic" "(" taskId = [Tasks] ")";
        ChangeRelationship : "ChangeRelationship" "(" replicableTaskId
             = [Tasks] "," relation = ID ")";
        ChangeRole : "ChangeRole" "(" taskId = [Tasks] "," newRole =
            ID ")";
        ChangeAllowedMembers : "ChangeAllowedMembers" "(" taskId = [
            Tasks] "," allowedMemberVal = INT ")";
        SearchMember : "SearchMember" "(" taskId = [Tasks]  ")";
        ChangeCapacity : "ChangeCapacity" "(" taskId = [Tasks] ","
            capabId = [Capability] "," value=INT ")";

/* ************************************************** */
/* ***************** VO-O Definition ***************** */
/* ************************************************** */

 VOOmodel:
      (module = VOmodule)?
      ("specification" (specifications+= Specification)+)?;

VOmodule : "service" name=ID "is"
```

```
 ("components" (components += Component)*)?
 ("requires" (requires += RequiresInterface)*)?
 ("provides" provides += ProvidesInterface)?
 ("uses" (uses+=UsesInterface)*)?
 ("wires" (wires+= Wire))?
;

ModuleNode: Component | ExternalInterface | LayerInterface;

Component: name=ID ":" businessRole = [BusinessRole]
 "{" (internalConfigurationPolicy += InternalConfigurationPolicy)*
     "}";

InternalConfigurationPolicy: internalConfigurationPolicyName=[
    Component] "[" expType = ComponentInternalConfigurationPolicy "]"
     ":" internalConfigurationPolicyguard = StateFormula;
enum ComponentInternalConfigurationPolicy: init = "init" | term = "
    term";

ExternalInterface : RequiresInterface | ProvidesInterface;
RequiresInterface : "requiresInterface" name=ID ":" businessProtocol
    =[BusinessProtocol]
 "{"
   (internalConfigurationPolicyName = ID "[" "trigger" "]" ":"
       internalConfigurationPolicyGuard = ModuleTrigger)?
 "}"
;
ModuleTrigger: operand += ModuleTerm (operantor += BinaryOperator
    operand += ModuleTerm);
ModuleTerm: Default | ModuleInteractionRef | Term |
    ParanthesizedModuleTerm ;
Default : default = "default";
ParanthesizedModuleTerm: "(" target = ModuleTerm ")";
ModuleInteractionRef: name=[ModuleNode] "." term = MTerm ;
MTerm : Term |  interactName = [ASyncInteraction];//
```

```

ProvidesInterface:   "providesInterface" name=ID ":"
    businessProtocol=[BusinessProtocol] ;
LayerInterface: UsesInterface | ServesInterface;
UsesInterface:  "usesInterface" name=ID ":" layerProtocol = [
    LayerProtocol];
ServesInterface: "servesInterface" name=ID ":" businessProtocol=[
    BusinessProtocol];

/* ****************************************************** */
/* ********************* SPECIFICATION ******************** */
/* ****************************************************** */

Specification : NodeSpecification | InteractionProtocol;
NodeSpecification: BusinessRole | BusinessProtocol | LayerProtocol;

BusinessProtocol: "business" "protocol" name=ID  "is"
 "parameters" (prm += ParList)*
 "interaction" (interactions += Interaction)*
 "behaviour" (behaviour = Behaviour)? ;
 ParList : name = ID ":" type=[SortName];
 /* ************************************************** */
 /* **************** INTERACTION DEFINITION ************ */
 /* ************************************************** */

 Interaction: ASyncInteraction  | SyncInteraction ;

 ASyncInteraction: ConvInteraction | NonConvInteraction ;

 ConvInteraction: convIntrType = ConvInteractionType name = ID
     //("[" val = INT"]" )?
                (convInterEventAndParam +=
                    ConvInteractionEventAndParam)*;

```

```
NonConvInteraction: nonConvnInrType = NonConvInteractionType name =
     ID // ("[" val = INT"]" )?
                    (nonConvInterEventAndParam +=
                        NonConvInteractionEventAndParam)* ;

   ConvInteractionEventAndParam:
                    convInterEventType = ConvInteractionEventType
                parameters += Parameter ("," parameters += Parameter)
                    * ;

  NonConvInteractionEventAndParam:
                    nonconvInterEventType =
                        NonConvInteractionEventType
                parameters += Parameter ("," parameters += Parameter)
                    * ;

Parameter: name = ID ":" type = [SortName];
InteractionType:   ASyncInteractionType | sync =
    SyncInteractionType;

ASyncInteractionType : conv = ConvInteractionType | nonConv =
    NonConvInteractionType ;
enum ConvInteractionType : rands = "r&s" | sandr = "s&r";
enum NonConvInteractionType : snd = "snd" | rcv = "rcv" ;
enum SyncInteractionType: ask = "ask" | rpl = "rpl" | tll = "tll" |
     prf = "prf";

ASyncInteractionEventType: convIntrEventType =
    ConvInteractionEventType | nonConvIntrEventType =
    NonConvInteractionEventType;
enum ConvInteractionEventType :   request = "Crequest" | reply = "
    reply";

 enum NonConvInteractionEventType: req =  "nonCrequest" ;
SyncInteraction:
```

```
                syncIntrType = SyncInteractionType name = ID
                syncInputInteractionParam +=
                    SyncInputInteractionParam*
                (syncOutputInteractionParam +=
                    SyncOutputInteractionParam*)?;

 SyncInputInteractionParam: "(" (parameters += Parameter (","
    parameters += Parameter)*)? ")";
 SyncOutputInteractionParam :  ":" parameters += Parameter (","
    parameters += Parameter)* ;

  /* ************************************************ */
 /* *************** BEHAVIOUR DEFINITION ************** */
 /* ************************************************ */

 Behaviour: intiallyEnabledStatement = InitiallyEnabled
        (otherStatements += BehaviourStatement)* ;

 InitiallyEnabled: "initallyEnabled" initEnabledEvent = ProcessEvent
    ;

// Event : interactName = [ASyncInteraction] ("[" val = INT"]" )?
    "[" interactionEvent = ASyncInteractionEventType "]";
  Event : interactName = [ASyncInteraction] "[" interactionEvent =
     ASyncInteractionEventType "]";
 PEvent: ProcessEvent | PublishEvent;

 ProcessEvent : event = Event "?";
 PublishEvent: event = Event "!";
 BehaviourStatement: Enable | Ensure ;
 Enable : enableSubject = BehaviourFormula "enables" enableObject =
    ProcessEvent;
 Ensure: ensureSubject = BehaviourFormula "ensure" enableObject =
    PublishEvent ;
 BehaviourFormula: operand += BehaviourWrapper (operator +=
     BinaryOperator operand += BehaviourWrapper)*;
 BehaviourWrapper: (negative ?= UnaryOperator)? ("(")? operand +=
     BehaviourTerm (")")?;
 BehaviourTerm : Term | LocalVar | SlaVar | envrFunc = EnvrFunc |
     InteractionParameter | PEvent;
ParanthesizedBehTerm: "(" target = BehaviourFormula ")";
 InteractionParameter:  name= [Interaction] "." parameter =
     InteractionParam;
 InteractionParam: def = DefaultParam | DefinedParam ;
 DefinedParam: paramName  =[ParList];
   enum DefaultParam : replyParam = "Reply" ;
 Term:
  StringConstant | IntegerConstant | bool = BooleanAtom ;
 enum BooleanAtom: true = "true" | false = "false";
 IntegerConstant: name=INT;
 StringConstant: name=STRING;
 LocalVar : name = ID ":" type = [SortName];
 enum EnvrFunc : todayParam = "today" | nowParam = "now" ;
 SlaVar : name = ID "[" initialVal =INT ".." finalVal=INT "]";

 /* ****************************************** */
 /* *********** OPERATOR DEFINITION ************ */
 /* ******************************************* */

 Operator : BinaryOperator  | unaryOpr = UnaryOperator;
 BinaryOperator: arthmOpr = ArthmOpr | cmprOpr = CmprOpr | cntrlOpr
     = CntrlOpr;
enum ArthmOpr : addition = "+" | subtraction = "-" | multiplication
    = "*" | division="/" | power = "^" ;
enum CmprOpr: equals = "==" | not_equals = "!=" | lessThan = "<" |
    moreThan = ">" | lessThan_or_equal = "<="| moreThan_or_equal=">="
     ;
enum CntrlOpr : conjunction="&" | disjunction = "|" ;
enum UnaryOperator : not_boolean = "!"  | negative_num = "unary-
    minus" ;

 /* ************************************************** */
 /* ************** LAYER PROTOCOL ********************* */
 /* *************************************************** */
  LayerProtocol: "layer" "protocol" name=ID "is"
 "parameters" (prm += ParList)*
 "interaction" (interactions += Interaction)*
 "behaviour" (behaviour = Behaviour)?;
/* ****************************************************** */
 /* *************** BUSINESS ROLE  ************************ */
 /* ***************************************************** */

 BusinessRole: "business" "role" name=ID "is"
 "interaction" (interactions += Interaction)*
 ("orchestration" orchestration = Orchestration)? ;

 LocalVarDefinition:
  var += LocalVar ("," var += LocalVar)*;

 StateMachine: name = ID ":" "[" stateLIteralValue =
     StateLiteralValues "]";
 StateLiteralValues :  stateLiteral += StateLiteral+ ;
 StateLiteral : name = ID;
 SlaVariable:  name=ID ":" type=Interval;
 VariableName: name = ID;
 Orchestration:
 ("local" (stateMachin = StateMachine localVar = LocalVarDefinition)
    *)?
 ("sla" "variable" (slaVariables += SlaVariable)*)?
 (transitions += Transition)*;
  // Transition: "transition" name = ID("[" val = INT"]" )? "is"
   Transition: "transition" name = ID "is"
   "triggeredBy" (trigger = Trigger)?
```

```
    "guardedBy" (guard = StateFormula)
          "effects" (effects = Effect)
          "sends" (sends = Send)? ;
  Trigger: Event | SF ;
   SF: (parameter += StateFormula "indent")*  parameter +=
       StateFormula ;

  /*******************************************************/
  /* **************** StateFormula **************************/
  /* *******************************************************/
   StateFormula : opernad += OperandWrapper (operator +=
       BinaryOperator operand += OperandWrapper)* ;
   OperandWrapper: operand = BasicOperand;
   BasicOperand :  envFunc = EnvrFunc |LocalVariable | Term |
       InteractionParameter ;
   LocalVariable : var = [LocalVar];

  /************************************************/
  /* *************** EFFECT part **********************/
  /* ************************************************/

  Effect: (esf += ESF "indent")* esf += ESF ;
  ESF: (parameters += ESFElement "implies")* parameters +=
      ESFElement;
ESFElement : (operand += EFFOperandWrapper operator +=
    BinaryOperator)* operand += EFFOperandWrapper;
EFFOperandWrapper: (negative ?= UnaryOperator)? operand = ETerm;
ETerm: Term | ETermSpot | Event | SyncEvent | LocalVar  ;
SyncEvent:"sycInter" syncInter = [SyncInteraction] "(" (args =
    SyncIntrArgumentList)? ")";
SyncIntrArgumentList: parameters += Argument ("," parameters +=
    Argument)* ;
     Argument: name = [SyncInputInteractionParam];
     ETermSpot:
       "post" "(" localVar = [LocalVar] ")";

    /*****************************************************************/
  /* ****************** SENDS part ******************************/
 Send:  ssf += SE ("indent" ssf += SE )*;
 SE: (parameterLeft = SSFElement "implies")? right = SSF;
  SSF: Event | PropertyAssignment | PostAssignment;
  PropertyAssignment: etermL = InteractionParameter "=" etermR =
      ETerm;
  PostAssignment: post = ETermSpot "=" sync = SyncEvent ;
  SSFElement: ESendFormula;
  ESendFormula: operand += ESFTerm (operator += BinaryOperator
      operand += ESFTerm);
  ESFTerm:  ETerm | EParathesizedFormula;
EParathesizedFormula: "(" target = ESendFormula ")";

     /*
         *****************************************************************/

  /* ************************* Wire DEFINITION
      ***********************/
  /*
      *****************************************************************/


Wire: name=ID "{"
               "nodeA" wireRoleA = [ModuleNode]
               "nodeB" wireRoleB = [ModuleNode]
               (connectors += Connector)*
               "}" ;

Connector: "connector" "{"
             "attachmentA" "{" attachmentInteraction = [Interaction]
             "=>"
             attachmentAIPInteraction = [IPInteraction]
```

```
                (attachmentAParameterMappings +=
                    AttachmentAInteractionParameterMapping)*
                "}"
                "attachmentB" "{" attachmentInteraction = [Interaction]
                "=>"
                attachmentBIPInteraction = [IPInteraction]
                (attachmentBParameterMappings +=
                    AttachmentBInteractionParameterMapping)*
                "}"
                 "interaction protocol" ip = [InteractionProtocol]
                            "}" ;
AttachmentAInteractionParameterMapping :
                 interactionParameter = [InteractionParameter] "=>"
                     ipInteractionParameter = [IPInteractionParameter];

AttachmentBInteractionParameterMapping :
                 interactionParameter = [InteractionParameter] "=>"
                     ipInteractionParameter = [IPInteractionParameter];


        /*
            *****************************************************************/

   /* ********** INTERACTION PROTOCOL DEFINITION
       ************************/
   /*
       ******************************************************************/


   InteractionProtocol: "interaction protocol" name = ID "is"
                       ("roleA" interactionRoleA = IPInteraction)?
                       ("roleB" interactionRoleB = IPInteraction)?
                       "coordination" ?;

   IPInteraction: interactionType = InteractionType name = ID
```

```
                (connectorParameters += IPInteractionParameter (”,”
                    connectorParameters += IPInteractionParameter)*)?;
IPInteractionParameter: (returnType ?= ”return”)?
 (interactionEventType = ConvInteractionEventType)?
name = ID ”:” ipInteractionParameter = [SortName];
```

Bibliography

[1] Conoise, http://www.conoise.org visited on: 2008.

[2] Ecolead, http://ecolead.vtt.fi, visited on 22/19/39.

[3] GOLD Project http://www.neresc.ac.uk/projects/gold/projectdescription.html visited on: 8/9/2011.

[4] Xtext http://www.eclipse.org/xtext/ visited on: 2012.

[5] A. Wright A. Conlin, H. Hiden. A chemical process development case study as a source of requirements for the GOLD project. Technical Report No. CS-TR-968, School of Computing Science, University of Newcastle upon Tyne, June 2006.

[6] J. Abreua. *Modelling Business Conversations in Service Component Architectures*. PhD thesis, University of Leicester, July 2009.

[7] Hamideh Afsarmanesh and Luis M. Camarinha-Matos. Federated information management for cooperative virtual organizations. In Abdelkader Hameurlain and A Tjoa, editors, *Database and Expert Systems Applications*, volume 1308 of *Lecture Notes in Computer Science*, pages 561–572. Springer Berlin / Heidelberg, 1997.

[8] Hamideh Afsarmanesh and Luis M. Camarinha-Matos. Future Smart-Organizations: A Virtual Tourism Enterprise. In *WISE*, pages 456–461, 2000.

[9] Hamideh Afsarmanesh and Luis M. Camarinha-Matos. A framework for management of virtual organization breeding environments. In Luis M. Camarinha-

Matos, Hamideh Afsarmanesh, and Angel Ortiz, editors, *Collaborative Networks and Their Breeding Environments*, volume 186 of *IFIP International Federation for Information Processing*, pages 35–48. Springer Boston, 2005.

[10] Stefano Bistarelli, Ugo Montanari, and Francesca Rossi. Semiring-based constraint satisfaction and optimization. *JOURNAL OF THE ACM*, 44(2):201–236, 1997.

[11] Laura Bocchi, José Luiz Fiadeiro, and Antónia Lopes. A Use-Case Driven Approach to Formal Service-Oriented Modelling. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation*, volume 17 of *Communications in Computer and Information Science*, pages 155–169. Springer Berlin Heidelberg, 2009.

[12] Laura Bocchi, José Luiz Fiadeiro, Noor Rajper, and Stephan Reiff-Marganiec. Structure and behaviour of virtual organisation breeding environments. In *FAVO*, pages 26–40, 2009.

[13] P. Briol. *BPMN, the Business Process Modeling Notation Pocket Handbook*. LULU PR, 2008.

[14] J.W. Bryans, J.S. Fitzgerald, C.B. Jones, and I. Mozolevsky. Formal modelling of dynamic coalitions, with an application in chemical engineering. In *Second International Symposium on Leveraging Applications of Formal Methods, Verification and Validation*, pages 91–98, Nov. 2006.

[15] Alan Burns and Gordon Baxter. Time bands in systems structure. In Denis Besnard, Cristina Gacek, and Cliff B. Jones, editors, *Structure for Dependability: Computer-Based Systems from an Interdisciplinary Perspective*, pages 74–88. Springer London, 2006.

[16] Luis M. Camarinha-Matos and H. Afsarmanesh. Design of a virtual community infrastructure for elderly care. In *Collaborative Business Ecosystems and Virtual Enterprises*, pages 439–450. Kluwer Academic Publishers, 2002.

[17] Luis M. Camarinha-Matos and H. Afsarmanesh. Creation of Virtual Organizations in a breeding environment. In *Fundamental Approaches to Software Engineering*, May, 2006.

[18] Luis M. Camarinha-Matos, H. Afsarmanesh, C. Garita, and C. Lima. Towards an architecture for virtual enterprises. *Journal of Intelligent Manufacturing*, 9:189–199, 1998.

[19] Luis M. Camarinha-Matos, H. Afsarmanesh, and M. Ollus. *Virtual Organizations Systems and Practices*. Springer, 2005.

[20] Luis M. Camarinha-Matos, H. Afsarmanesh, and M. Ollus. Ecolead And CNO Base Concepts. In Luis M. Camarinha-Matos, Hamideh Afsarmanesh, and Martin Ollus, editors, *Methods and Tools for Collaborative Networked Organizations*, pages 3–32. Springer US, 2008.

[21] Luis M. Camarinha-Matos, H. Afsarmanesh, and RJ Rabelo. Infrastructure developments for agilevirtual enterprises. *International Journal of Computer Integrated Manufacturing*, page 235 254, 2003.

[22] Luis M. Camarinha-Matos and Hamideh Afsarmanesh. Collaborative networks: a new scientific discipline. *Journal of Intelligent Manufacturing*, 16:439–452, 2005.

[23] Luis M. Camarinha-Matos and Hamideh Afsarmanesh. The ARCON modeling framework. In *Collaborative Networks: Reference Modeling*, pages 67–82. Springer US, 2008.

[24] Luis M. Camarinha-Matos, Hamideh Afsarmanesh, Nathalie Galeano, and Arturo Molina. Collaborative networked organizations - concepts and practice in manufacturing enterprises. *Computers and Industrial Engineering*, 57(1):46 – 60, 2009.

[25] Luis M. Camarinha-Matos, Hamideh Afsarmanesh, and Martin Ollus. Ecolead: A Holistic Approach to Creation and Management of Dynamic Virtual Organizations. In Luis M. Camarinha-Matos, Hamideh Afsarmanesh, and Angel Ortiz, editors, *Collaborative Networks and Their Breeding Environments*, volume 186 of *IFIP International Federation for Information Processing*, pages 3–16. Springer Boston, 2005.

[26] Luis M. Camarinha-Matos, Ivan Silveri, Hamideh Afsarmanesh, and Ana Inês Oliveira. Towards a framework for creation of dynamic virtual organisations. In *in Camarinha-Matos, Luis M. et al. (Eds.) Sixth IFIP Working Conference on Virtual Enterprises PRO-VE 2005*, pages 26–28. Springer, 2005.

[27] Adrian Conlin, Nick Cook, Hugo Hiden, Panos Periorellis, and Rob Smith. Gold architecture document. Technical Report CS-TR-923, School of Computing Science, University of Newcastle upon Tyne, July 2005.

[28] J. Cummings, T. Finholt, I. Foster, and C. Kesselman. Beyond being there: A blueprint for advancing the design, development, and evaluation of Virtual Organizations . Technical report, Final Report from Workshops on Building Effective Virtual Organizaions, 2008.

[29] Nicodemos Damianou, Naranker Dulay, Emil Lupu, and Morris Sloman. The Ponder Policy Specification Language. In *LECTURE NOTES IN COMPUTER SCIENCE*, pages 18–38. Springer-Verlag, 2001.

[30] Alessandro D'Atri and Amihai Motro. Virtue: a formal model of virtual enterprises for information markets. *Journal of Intelligent Information Systems*, 30:33–53, 2008.

[31] Giovanna Di Marzo Serugendo, John Fitzgerald, Alexander Romanovsky, and Nicolas Guelfi. A metadata-based architectural model for dynamically resilient systems. In *SAC '07: Proceedings of the 2007 ACM symposium on Applied computing*, pages 566–572, New York, NY, USA, 2007. ACM.

[32] H.E. Eriksson and M. Penker. *Business modeling with UML: business patterns at work*. Omg Series. John Wiley & Sons, 2000.

[33] Ekaterina Ermilova and Hamideh Afsarmanesh. Competency and Profiling Management in Virtual Organization Breeding Environments. In *Network-Centric Collaboration and Supporting Frameworks*, volume 224 of *IFIP International Federation for Information Processing*, pages 131–142. Springer Boston, 2006.

[34] Ekaterina Ermilova and Hamideh Afsarmanesh. Modeling and management of profiles and competencies in VBEs. *Journal of Intelligent Manufacturing*, 18:561–586, 2007.

[35] J. L. Fiadeiro, A. Lopes, and L. Bocchi. A formal approach to service component architecture. In *Services and Formal Methods, Third International Workshop*, pages 193–213. Springer, 2006.

[36] John Fitzgerald, Peter Gorm Larsen, Paul Mukherjee, Nico Plat, and Marcel Verhoef. *Validated Designs For Object-oriented Systems*. Springer-Verlag TELOS, Santa Clara, CA, USA, 2005.

[37] John S. Fitzgerald, Peter Gorm Larsen, and Marcel Verhoef. Vienna development method. In Benjamin W. Wah, editor, *Wiley Encyclopedia of Computer Science and Engineering*. John Wiley & Sons, Inc., 2008.

[38] Ian Foster, C Kesselman, J Nick, and S Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. *Open Grid Service Infrastructure WG Global Grid Forum*, 22, 2002.

[39] Ian Foster and Carl Kesselman. *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, November 2003.

[40] Ian T. Foster. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. In *Proceedings of the 7th International Euro-Par Conference Manchester*

on Parallel Processing, Euro-Par '01, pages 1–4, London, UK, 2001. Springer-Verlag.

[41] Stephen Gorton and Stephan Reiff-Marganiec. Policy Support for Business-oriented Web Service Management. In *Proceedings of the Fourth Latin American Web Congress*, pages 199–202, Washington, DC, USA, 2006. IEEE Computer Society.

[42] Jane Hillston. *A compositional approach to performance modelling*. Cambridge University Press, New York, NY, USA, 1996.

[43] Dan Hirsch and Ugo Montanari. Two Graph-Based Techniques for Software Architecture Reconfiguration. *Electr. Notes Theor. Comput. Sci.*, 51:177–190, 2001.

[44] C. B. Jones J. W. Bryans, J. S. Fitzgerald and I. Mozolevsky. Dimensions of dynamic coalitions. Technical Report No. CS-TR-963, School of Computing Science, University of Newcastle upon Tyne, May 2006.

[45] Lalana Kagal, Tim Finin, and Anupam Joshi. A policy based approach to security for the semantic web. In Dieter Fensel, Katia Sycara, and John Mylopoulos, editors, *The Semantic Web - ISWC 2003*, volume 2870 of *Lecture Notes in Computer Science*, pages 402–418. Springer Berlin / Heidelberg, 2003.

[46] Jeffrey O. Kephart. Research challenges of autonomic computing. In *Proceedings of the 27th international conference on Software engineering*, ICSE '05, pages 15–22, New York, NY, USA, 2005. ACM.

[47] Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. *Computer*, 36:41–50, January 2003.

[48] K.J. Turner L. Blair, S. Reiff-Marganiec. Appel: The accent policy environment/language. Technical Report CSM-164. 2005.

[49] E.C. Lupu and M. Sloman. Conflicts in policy-based distributed systems management. *IEEE Transactions on Software Engineering*, 25(6):852–869, Nov/Dec 1999.

[50] Carlo Montangero, Stephan Reiff-Marganiec, and Laura Semini. Logic-based Conflict Detection for Distributed Policies. *Fundam. Inf.*, 89:511–538, December 2008.

[51] Mohammad Reza Nami and Seyed Naser Hashemi. Investigating a new formal model for autonomous virtual organisation using RAISE method. *IJNVO*, 7(6):505–513, 2010.

[52] Mohammad Reza Nami, Mohsen Sharifi, and Abbas Malekpour. A Preliminary Formal Specification of Virtual Organization Creation with RAISE Specification Language. In *Proceedings of the 5th ACIS International Conference on Software Engineering Research, Management & Applications*, SERA '07, pages 227–232, Washington, DC, USA, 2007. IEEE Computer Society.

[53] D Nguyen, S Thompson, J Patel, L Teacy, N Jennings, M Luck, V Dang, S Chalmers, N Oren, T Norman, A Preece, P Gray, G Shercliff, P Stockreisser, J Shao, W Gray, and N Fiddian. Delivering services by building and running virtual organisations. *BT Technology Journal*, 24:141–152, 2006. 10.1007/s10550-006-0029-6.

[54] T. J. Norman, A. Preece, N. R. Jennings, M. Luck, V. D. Dang, T. D. Nguyen, V. Deora, J. Shao, W. A. Gray, and N. J. Fiddian. Agent-based Formation of Virtual Organisations. *Knowledge Based Systems*, 17:2004, 2004.

[55] Jigar Patel, W. T. Luke Teacy, Nicholas R. Jennings, Michael Luck, Stuart Chalmers, Nir Oren, Timothy J. Norman, Alun Preece, Peter M. D. Gray, Gareth Shercliff, Patrick J. Stockreisser, Jianhua Shao, W. Alex Gray, Nick J. Fiddian, and Simon Thompson. Agent-based virtual organisations for the Grid. In *Proceedings of the fourth international joint conference on Autonomous agents and*

multiagent systems, AAMAS '05, pages 1151–1152, New York, NY, USA, 2005. ACM.

[56] Ricardo Rabelo, Sergio Gusmeroli, Cristina Arana, and Thierry Nagellen. The ecolead ict infrastructure for collaborative networked organizations. In *Network-Centric Collaboration and Supporting Frameworks*, volume 224 of *IFIP International Federation for Information Processing*, pages 451–460. Springer Boston, 2006.

[57] Ricardo J. Rabelo, Luis M. Camarinha-Matos, and Rolando V. Vallejos. Agent-based brokerage for virtual enterprise creation in the moulds industry. In *Proceedings of the IFIP TC5/WG5.3 Second IFIP Working Conference on Infrastructures for Virtual Organizations: Managing Cooperation in Virtual Organizations and Electronic Busimess towards Smart Organizations: E-Business and Virtual Enterprises: Managing Business-to-Business Cooperation*, pages 281–290, Deventer, The Netherlands, The Netherlands, 2000. Kluwer, B.V.

[58] Stephan Reiff-Marganiec and Noor Rajper. Modelling virtual organisations: Structure and reconfigurations. In *PRO-VE*, pages 297–305, 2011.

[59] Burak Sari, Tayyar Sen, and S. Kilic. Formation of dynamic virtual enterprises and enterprise networks. *The International Journal of Advanced Manufacturing Technology*, 34:1246–1262, 2007.

[60] Giovanna Di Marzo Serugendo, John S. Fitzgerald, and Alexander Romanovsky. MetaSelf: an architecture and a development method for dependable self-* systems. In Sung Y. Shin, Sascha Ossowski, Michael Schumacher, Mathew J. Palakal, and Chih-Cheng Hung, editors, *SAC*, pages 457–461. ACM, 2010.

[61] Jianhua Shao, W Alex Gray, Nick J Fiddian, Vikas Deora, Gareth Shercliff, Patrick J Stockreisser, Timothy J Norman, Alun Preece, Peter M D Gray, Stuart Chalmers, Nir Oren, Nicholas R Jennings, Michael Luck, Viet D Dang, Thuc D

Nguyen, Jigar Patel, W T Luke Teacy, and Simon Thompson. Supporting formation and operation of virtual organisations in a grid environment. In *THE UK E-SCIENCE ALL HANDS MEETING 2004*, pages 376–383, 2004.

[62] Maurice ter Beek, A. Fantechi, S. Gnesi, and F. Mazzanti. An action/state-based model-checking approach for the analysis of communication protocols for service-oriented applications. In Stefan Leue and Pedro Merino, editors, *Formal Methods for Industrial Critical Systems*, volume 4916 of *Lecture Notes in Computer Science*, pages 133–148. Springer Berlin / Heidelberg, 2008.

[63] Andrzej Uszok, Jeffrey Bradshaw, and Renia Jeffers. Kaos: A policy and domain services framework for grid computing and semantic web services. In Christian Jensen, Stefan Poslad, and Theo Dimitrakos, editors, *Trust Management*, volume 2995 of *Lecture Notes in Computer Science*, pages 16–26. Springer Berlin / Heidelberg, 2004.

[64] Emil Vassev and Joey Paquet. Assl - autonomic system specification language. In *Proceedings of the 31st IEEE Software Engineering Workshop*, SEW '07, pages 300–309, Washington, DC, USA, 2007. IEEE Computer Society.

[65] Emil Iordanov Vassev. *Towards a framework for specification and code generation of automatic systems*. PhD thesis, Concordia University, Montreal, P.Q., Canada, Canada, 2008.

[66] Jos Warmer. A model driven software factory using domain specific languages. In *Proceedings of the 3rd European conference on Model driven architecture-foundations and applications*, ECMDA-FA'07, pages 194–203, Berlin, Heidelberg, 2007. Springer-Verlag.

[67] Maciej Witczyski and Adam Pawlak. Virtual organizations in the electronics sector. In Luis M. Camarinha-Matos, Hamideh Afsarmanesh, and Martin Ollus, editors, *Virtual Organizations*, pages 221–232. Springer US, 2005.