VOML: A Framework for Modelling Virtual Organizations and Virtual Breeding Environment

NOOR JEHAN RAJPER*, AND STEPHAN REIFF-MARGANIEC**

RECEIVED ON 17.04.2014 ACCEPTED ON 17.03.2015

ABSTRACT

This paper presents the VOML (Virtual Organization Modelling Language) framework. VOML is a formal approach for specifying VOs (Virtual Organizations) and their VBEs (Virtual Breeding Environments).The VOML framework allows domain users to model a system in terms of their domain terminology and from that domain specific model IT community can derive a complete operational model closer to underlying execution environment. The framework is a collection of three sub-languages, each covering different aspects which are considered paramount at a particular level of VO representation. We present VOML and its underlying methodological approach in detail and demonstrate how to model VOs. Our focus will be on the methodological approach that VOML supports and on the language primitives that VOML offers for modelling VOs.

Key Words: Virtual Organizations, Modelling Languages Received.

1. INTRODUCTION

In frequently evolving business environments, cooperation and collaboration of autonomous partners to exploit or respond to emergent business opportunities is becoming a norm. This trend is due to the fact that it is not possible (or at least very costly) for most of the organizations to have all the skills and resources that might be needed in the future due to changing business context. Therefore, it is becoming a prerequisite for the survival of the organizations to cooperate and collaborate with other organizations in order to make-up for missing skills and resources, venture into new emergent domains or to compete with rival giant corporations. This collaboration is only possible if the organizations are flexible enough to evolve and adapt [1-8]. To cater for these demands the concept of VOs has emerged. VOs are ensembles that are dynamically created by sharing a number of resources in a distributed way to provide high-level functionalities, or services. A WBE is the organizational context in which VOs are created and operate. It lays down basic long-term agreements of cooperation among its participants (individuals or institutions) and characterizes the interoperable infrastructure used by the participants.

Formal models of a VO provide unambiguous description of VOs and the capability to reason about core aspects; which aids in understanding the behavior of VOs as well. Hence, there is a need to have a formal modelling language for VOs [9]. Different methodologies, paradigms, languages and architectures have been put forward in the recent years

* Assistant Professor, Institute of Mathematics & Computer Science, University of Sindh, Jamshoro.
** Senior Lecturer, Department of Computer Science, University of Leicester, UK

to tackle such challenges. However, most solutions focus one particular aspect of the problem and fail to accommodate the problem as a whole. Some efforts have been made at describing formal models of VOs which aim at representing and evaluating different characteristics of VOs using a rigorous modeling language, (what is called a structural model in this paper). Analysis and evaluation at this rigorous level demands more pragmatic descriptions of the model stripped away leading to model description that is not readily executable on the underlying execution environment. More concrete counterparts of the rigorously analyzed and verified models are usually derived manually. These concrete counterparts, (named operational models in our paper) describe in detail the coordination and communication aspect closer to IT community. This has usually lead to the introduction of discrepancies in the executable model and there is no systematic approach to assure that the abstract rigorous model and the underlying execution model are actually different views of the same system. This problem is compounded when the system represented by the model undergoes frequent reconfigurations during its execution.

So far there has not been any effort towards developing a richer and more expressible language which could not only express structurally adaptable dimension of VOs, but simultaneously their functional dimension. We have attempted to fill this gap with a consolidated modeling framework which besides paving the way for different kinds of analysis and evaluation, not only encompasses structural and functional dimensions but allows generating (semi-)automatically more concrete functional models from abstract structural models. The systematic generation of functional models out of structural models provides consistency and conformance between the two models of the same system. This paper presents languages for formal specification of VOs at a level which not only covers domain concepts abstractly but also captures the functionality (service) offered by the VO; which other specification languages such as [10-11] fail to capture. The domain specific constructs pave the way for VO models to easily adapt to the changing circumstances dynamically. Specifically, there are three different modelling languages each capturing a different aspect of VO. The first language named VO-Structural modelling language (VO-S for short) focuses on structural aspects and many of the characteristics peculiar to VOs such as relationship between two members, etc. The second language permits different reconfigurations on the structure of the VO. These reconfigurations change the core model itself. This language is called the VO-Reconfiguration (VO-R for short). The third language named VO-Operational modelling language (VO-O for short) describes operational models of VOs in more details, out of VO-S model.

Overview: In Section 2, a review is given related to work on VO modelling languages. Section 3 presents the methodological approach that the VOML framework has taken. In Section 4, the Structural modelling language (VO-S) has been introduced, and in Section 5 a discussion about operational modelling language (VO-O) is provided. Section 6 shows how VO-S is mapped to VO-O. Finally, Section 7 presents some conclusion and future work.

2. RELATED WORK

This work is related to [9] as it also focuses on modelling of VOs which are termed 'dynamic coalition' there. Dynamic coalitions are modelled using the VMD (Vienna Development Method) specification language [12] in which a VO specification consists of specific choices made in five orthogonal dimensions including membership, information representation, provenance, time and trust. The specific choices made in each dimension gives rise to VOs with unique properties. The analysis and verification of these unique properties is the main purpose of the [9]. Our work on the other hand we have developed a modelling

language for VBEs and VOs that incorporate domain notions and concepts as first class entities.

In [13] Agent Technology is used to capture VO notion by modelling VOs as system of cooperating agents. Whereas, we aim to develop a modelling language which is agnostic to any specific platform and technology. In [13] reconfiguration is limited to replacing one agent with another one having exact same behavior and capabilities, we on the other hand allow for dividing or sharing a task between members each having different behavior and capabilities but collectively equivalent to the task being shared. This is captured in the definition of tasks.

The field of dynamic adaptability can also be related to our work in general [14]. ASSL [10-11] is one such work that allows for dynamically adapting an autonomous system. However, business level requirements are abstracted away in the ASSL specification. We believe that business level requirements have an immense effect on the overall system (VO), in particular at the operational level. Our structural modelling language is able to capture these effects and helps direct its operational model (VO-O). Therefore, our modelling language provides constructs that on the one hand precisely describe a VO with enough abstraction that allow to easily restructure VO and on the other hand provides a complete operational model capturing the business level details.

3. VOML METHODOLOGY

Most modelling languages are dedicated to either the application aspect of the system i.e the actual goal or service (specification of business functionality) offered by the system, or the coordination and communication model close to the underlying execution environment. The advantage gained by this separation of concerns is the achieved simplicity which divides the complexity into easily manageable chunks. However, there are disadvantages as well; especially for domains where adaptability demands are so high that they require adaptations which change the core operational model of the VO itself. When the domain model and the reconfigurations (adaptations) that it may undergo are defined separately to its operational model (business functionality) then quite often both models cannot be easily combined in our experience. 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. A model describing the domain is abstract enough to give it the flexibility of making changes at the structural level; but fails to capture how these reconfigurations are reflected at the operational level. On the other hand the language which covers the operational aspects for particular application is able to define its process in concrete details such that it can readily be realized; but it comes at the cost of loosing the ability to reconfiguring itself dynamically; hence is inconsistent with its domain model's reconfigurations. In order to reduce complexity by adapting separation of concerns concept and at the same time to ensure that models focusing on different dimensions of the same system remain consistent VOML takes the incremental approach. First of all it defines different levels of representation for the VBE; each level focusing on particular aspects of the VBE and its VOs, and with each level being supported by an appropriate

language. Our approach [15] supports the definition of a structural and behavioral model of a fixed VBE based on three different levels of representation: (1) the definition of the persistent functionalities of the VBE; (2) the definition of the transient functionalities of the VOs that are offered by the VBE at a specific moment in time (a business configuration of the VBE) and (3) the ensemble of components (instances) and connectors that, at that time, deliver the services offered by the VOs present in the business configuration (a state configuration). The first level is invariant, i.e. it provides a representation of those aspects of a VBE that will not change; the business configuration at the level below captures the way the VBE is logically organized at that time in terms of VOs; 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. These different levels of representation then enable us to focus on different dimensions of VO 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-S (Structural Language). At this level we are also able to talk about the adaptability needs of VOs in general; we cover this dimension through our VO-R (Reconfiguration Language), which we are going to explore in the sequel. At the state configuration level we focus on the business functionality offered by any VO, in sufficient detail to allow for ready execution. We have developed VO (Operational Language) for describing this business functionality. Models at this level are derived from the information available in the VO-S models. One particular structural configuration of the VO 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. 1 [16].

Where M1, M2 and M3 represent structural models designed by domain experts who are proficient in their area but usually have no IT background. Therefore, VO-S language designed for domain experts at this level accommodates VO and VBE domain concepts as language constructs. Based on the structural description of a VO at that time, a more concrete level description of that VO can be generated in VO-O language. The VO-O model is represented through titles such as IM1.1 in Fig. 1, where IM stands for "Instance Model" and the first "1" in 1.1 refers to VO-S model named "M1" and the second "1" points to concrete instance named "1" of VO-O which conform to current configurations laid out at VO-S level model M1. The arrows between instance models represent reconfiguring one VO-O model to another VO-O model that is permitted by the current configurations at the VO-S level and the arrows between VO-S level and VO-O level models signify the different VO-O reconfigurations permitted by current VO-S model. Likewise, arrows between VO-S level models represent the reconfigurations

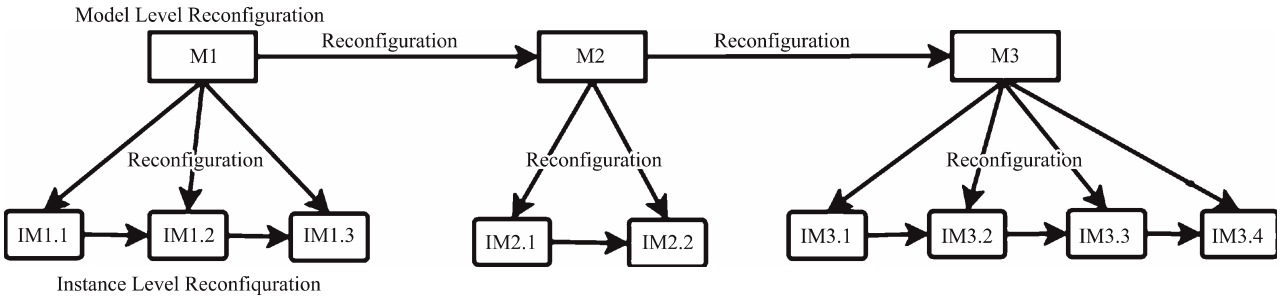

FIG. 1. VOML METHODOLOGY [16]

at the structural level reconfiguration. Structural level reconfigurations invalidate set of VO-O level reconfigurations that were permitted in the previous VO-S setting and offers whole new VO-O reconfiguration set that conforms to the new VO-S model, such as VO-S model configuration M1 permits three different VO-O level configurations namely IM1.1, IM1.2 and IM1.3 but, once M1 is evolved to M2 then the VO-O level configurations are IM2.1 and IM2.2.

4. VO STRUCTURAL LANGUAGE

The VO-S (Structural Modelling Language) defines the basic structure of the VO, its constituent elements, abstract process, and other details which define the essential structure of the VO and provide the basis for its operational models. The VO structural model consists of five basic elements: (1) Members, (2) Process, (3) Tasks, (4) VBE resource and (5) Data-Flow. We will discuss these in more detail next.

4.1 Members

We differentiate between three types of membership to VBE/VO as follow:

(i) A Partner is one who is permanent member of the VBE.

(ii) An Associate is one who is not a permanent (transient) member of the VBE, but rather has temporarily joined the VBE based on demands of some VO which requires some capability for which there is currently no member available on the VBE.

(iii) An ExtEntity is one which is neither a permanent nor a transient member of the VBE.External entities are transient members of VO and are discovered each time a VO launches a new instance; they leave the VO when the instance finishes its life.

Partners and associates are provided by the VBE to a VO and at the VO level they are permanent members of the VO. A permanent member of a VO exists beyond a single instantiation of its operational model; whereas external entities are discovered from the open universe. An excerpt for members description in VO-S is given in Fig. 2. In this example one partner named partnerX is involved in task named TourGuide and partnerY is assigned subtask hotel. An associate named associateA is performing the task FlightBooking.

4.2 Process

This element describes the workflow which leads to meeting the requirements of the customer of VO at the highest level of abstraction. It lists only those tasks that contribute towards achieving the goals of the VO. The other tasks on which the main tasks of the VO depend are specified by the tasks themselves in their supported by attribute. An excerpt given in Fig. 3 specifies a sample process in VO-S. The process description in Fig. 3 consists one of the VBE resources UsrDB and three tasks. The control flow between tasks is parallel and sequential (indicated by the leads to keyword) between VBE resource and the tasks.

```
Members
Partner ParnerX
{performsTask :Tour Guide}
ParnerParnerY
{performsTask :Hotel&Trasnport.hotel}
Associate AssociateA
{PerformsTaks :Flight Booking}
```

FIG. 2. VO-S MEMBER DESCRIPTION

```
Process
{
useAset (UsrDB) leadsToSatisfyTasks(Flight Booking,
Hotel&Transport Book, TourGuide)
}
```

FIG. 3. VO-S PROCESS SPECIFICATION

4.3 Task

We have put task specifications at the centre of our specification methodology; because this is where it is defined what service(s) is required by the VO from its members. The task requirements shape the behavior and competencies expected from the member who is going to perform the task, hence they control the VO membership. It is also the task description which helps in deciding the kind of restructuring that a task and hence a VO can undergo, having effects ranging from VO topology to member relationships. We are going to explain each of the above in detail next. Fig. 4(a) introduces an example task described using VO-S, as a reference example. Tasks are divided into two parts; one pertaining to the domain of the VO and the other pertaining to the actual business goal, respectively called STRUCTURE and BUSINESS FUNCTIONALITY. The STRUCTURE part provides primitives that describe VOs in terms of the concepts that are relevant at the domain level irrespective of the business functionality that is being offered by the task. This description also adds the flexibility to the VO to adapt to the changing environments through reconfigurations. The STRUCTURE part is further divided into TaskScope and ConfScope. TaskScope attributes provide a domain of discourse of configurations that a VO can undergo while conforming to the model description. It is the ConfScope part whose attributes decide exactly which configuration (from the domain of discourse) the VO is currently in. Consider the example of the allowed Members attribute in the TaskScope category. The value of this attribute specifies the maximum number of members a particular task can be shared by. At the operational model level this task can be 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 current Members attribute of ConfScope keeps track of exactly how many members are involved in the configuration at that particular moment in time. The BUSINESS FUNCTIONALITY part of task description consists of what the task is actually offering in terms of the business functionality. This part is mapped to the business protocol at the VO-O level which is discussed further in Sections 5 and 6. Elements of paramount importance of STRUCTURE part are further explained below:

Competency: Each task expects a certain set of capabilities that the VO members performing the task must possess. These capabilities are listed under the keyword Competency which list one or more capabilities required by the task. A Capability lists one or more resources it depends on and the capacity shows required quantity of that capability. VO-S uses this concept to allow a task to be shared by more than one member, which is explained in more detail while describing the task types.

Task Types: The two most common purposes suggested behind the formation of virtual organizations are (a) some entity (organization) posses some of the capabilities necessary to perform the task at hand but lacks a few i.e. it falls short of some others; and (2) an entity has all the capabilities required but lacks the required capacity(usually the case with SMEs (Small and Medium Organizations) competing for large jobs). VO-S offers provisions for such requirements by offering following three types tasks:

(i) AtomicTask: An AtomicTask is a task that must be performed by only one member. Any configuration of the VO which associates more than one member for this task is considered invalid.

(ii) ReplicableTask: A ReplicableTask 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 task, but both of them fall short of the amount of resources required (specified through the capacity attribute),then they can perform the task

collectively. Note that all the members are equally capable of carrying out the task individually; it is the amount (capacity) of resources required which forces them to cooperate.

(iii) ComposableTask: A ComposableTask can be performed by more than one members; here the criteria are a capability shortage of the members rather than the capacity. The task is actually divided into two or more different subtasks and each subtask can be performed by different member(s). Fig.4(b) shows a ComposableTask which has been divided into two subtasks (hotel and transport).

ComposableTask Hotel&Transport
STRUCTURE
TaskScope{
performedBy : Partner
allowedMembers : 3
allowedSubTasks : 2
}
ConfScope {
currentMembers : 1
currentState : atomic
}
Competency{
CapabilityroomReservation
{
resource:hotel.room
capacity : {totalRooms: 50}
}
CapabilitylocalTransport
{
resource: vehicle
capacity :{totalVehicles: 50}
}
}
BUSINESS FUNCTIONALITY
request:
from, to : location
checkin, checkout : date
name :usrData
reply:
amount :moneyValue
roomReservation.hconf :hcode
localTransport.taxiInfo: tCode

FIG. 4(a). VO-S COMPOSABLETASK IN ATOMIC STATE

ComposableTask Hotel&Transport
STRUCTURE
TaskScope{
performedBy : Partner
allowedMembers : 3
allowedSubTasks : 2
subTaskFlow: sequential
}
ConfScope {
currentMembers : 2
currentState : composed
currentSubTaks: hotel,
transport
}
Competency{
CapabilityroomReservation
{
resource:hotel.room
capacity : {totalRooms: 50}
} CapabilitylocalTransport
{
resource: vehicle
capacity :{totalVehicles: 50}
}
}
AtomicTaskhotel{
STRUCTURE
TaskScope
{competencies: roomReservation}
}
ConfScope {}
}
AtomicTasktransport{
STRUCTURE
TaskScope
{competencies: localTransport}
}
ConfScope {}
}
BUSINESS FUNCTIONALITY

FIG. 4(b). VO-S COMPOSABLETASK IN COMPOSED STATE

FIG. 4. A VO-S COMPOSTABLE TASK DESCRIPTION

Relationship between Members: In situations where more than one member VO is responsible for a task; a relationship between those members is implied. This element also has an effect on the workflow of the virtual organization. The relationship between members performing the same task can be of the type cooperation, but the members could also be competitors, based on the business demands. The relationship attribute of the task is used to describe the exact relationship between all the members who are sharing the responsibility of the same task. Cooperation relationship expresses that all the members are required for the task to satisfy its goal. An example of this could be the task of supplying the catalyst for a VO. 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.

Competition(comp-param) is a relationship where members compete. Considering the same example now if the customer demands just 250Kg of catalyst then both members are able to satisfy the demands. If the members are in a competition relationship then all the members will bid for that business opportunity.

The criteria over which bidding is performed are specified through the comp-param. For example if the comp-param is cost then the bidder with the lowest cost offer will be selected.

4.4 VBEresource

Besides tasks that are carried out by partners and associates, a VO might need other basic resources that are offered by the VBE. These are represented with the VBEresource keyword by VO-S. What differentiates between tasks and VBE resources is that VBE resources are made available to all the VOs of the VBE. Any VO can use them but can not specify criteria over the VBE resources. They are considered to be always available. At the VO-O level they are mapped to the layer protocol which effectively assumes those assets are persistent entities. The VO-S description for a VBEresource consists only of the BUSINESS FUNCTIONALITY part as a VO does not have any control over it and it cannot specify any other criteria over it as well.

4.5 Business Functionality

This part consists of all the data required at the operational level for the functionality (service) to be carried out. It is divided into two parts Request and Reply. The Request part lists all the data which is required by the member who is performing the task and the Reply part is the data provided by the member to the VO once the task has been performed. Each data item can be prefixed by a capability name as well. This helps in associating the data item with the corresponding subtask, in case the task is divided. Data items which are not prefixed with any capability name are associated with all subtasks. Let's assume there is a composable task named Hotel&Transport in some VBE which provides two services: a hotel booking service and a transport provision service. This task has two data items called hconf which can be thought of as a receipt for booking the hotel and taxiInfo which is the receipt for transport booking. But when this task is divided into two subtasks one which books the hotel and the other which books the transport; it is clear that hconf must be part of the hotel booking subtask and taxiInfo be part of the transport booking subtasks. Both of the parameters are useless for the other subtask, so they must not be part of their BUSINESS FUNCTIONALITY. This is achieved by prefixing the parameter hconf with the roomReservation capability and the taxiInfo with the local Transport Provision capability in the BUSINESS FUNCTIONALITY part.

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.

4.6 Data-Flow

Data-Flow specifies the relationship between data items in such a way that it helps in realizing concrete orchestrations, transitions and wires in operational model. It consists of one or more sentences as shown in Fig. 5. In this example a data item named 'to' from the customer is assigned to the corresponding data items of three tasks named Hotel&Transport, TourGuide and FlightBooking.

5. VO-O LANGUAGE

For the operational model we realized that another language already developed by our colleagues could meet most of the needs of the VO-O with minor extension and adaptation. This language is called SRML (Sensoria Reference Modelling Language) [17] which is aimed at service oriented systems. We will return to discuss the differences after defining the concepts used for VOs, as we can then refer to them while highlighting key differences between SRML and VO-O.

5.1 VO Module

A VO is defined by a `VO module' at VO-O level; it consists of:

(i) Component specifications that are used in state configurations as serves interfaces for the VO-S tasks performed by the partners and associates), or uses-interfaces to VBEresources involved in the VO.

(ii) Component specifications that are used as requires-interfaces for ExtEntity(external entities) or as the provides-interface for the customer of the VO. The specification of requires-interfaces identifies the behavioral properties that are expected of external parties to be eligible to be chosen as ExtEntity (service providers) for the VO. The specification of the provides-interface identifies the properties that customers can expect of the service offered by the VO-module.

(iii) Specifications of the components and wires that model the (possibly distributed) process that orchestrates the services provided by the VO.

(iv) An internal configuration policy, which identifies the triggers of the discovery process for the ExtEntity.

(v) An external configuration policy, which consists of the competency constraints that determine the quality profile to which the external entities need to adhere.

Costomer.to ==> Hotel&Transport.to,
Flight Booking.to, Tour Guide.to

FIG. 5. VO-S PROCESS DESCRIPTION

VO-modules are design primitives that define patterns that can be reused in the definition of multiple VBE business configurations. We use a graphical notation to depict VO-modules as illustrated in Fig. 6 for a VO named travelBK. In order to account for the behavior that emerges from the interconnections established inside the ensembles that deliver services through VOs, we need a uniform representation of the entities and resources involved, which in our approach we do in terms of component and wire specifications. A component specification is a pair <signature, behavior> where:

(i) Signature declares the interactions in which the component may be involved.

(ii) Behavior is a formal model of the behavior of the entity that the component represents expressed

in terms of the interactions identified in the signature and a number of parameters that reflect resource consumption or quality-of-service attributes.

Given the space available, we are not able to define in detail the formalisms that we use in component specifications (these are similar to those proposed for the SRML (Service Modelling Language) [17]). Fig. 7 shows a VO-O description of provides-interface of some VO, which is of type Customer. This specification is what we call a business protocol: it uses patterns of typical business conversations, which are abbreviations of sentences of a temporal logic that we have adapted from SRML [17].

In the formalism that we adopt, interactions can be either synchronous or asynchronous, one-way or two-way (i.e. conversational); Table 1 summarizes the options. In our example, the interface that the VO offers to its customers specifies that the VO can engage in the interaction bookTrip (initiated by the customer). Interactions of type r&s and s&r are conversational in the sense that they expect a reply from the receiving party.

TABLE 1. INTERACTION TYPES

r&s	The interaction is initiated by the co-party, which expects a reply. The co-party does not block while waiting for the reply.
s&r	The interaction is initiated by the party and expects a reply from its co-party. While waiting for the reply, the party does not block.
rcv	The co-party initiates the interaction and does not expect a reply.
snd	The party initiates the interaction and does not expect a reply.
ask	The party synchronizes with the co-party to obtain data.
rpl	The party synchronizes with the co-party to transmit data.
tll	The party requests the co-party to perform an operation and blocks.
prf	The party performs an operation and frees the co-party that request it.

TABLE 2. CONVERSATIONAL INTERACTIONS

Interaction 🔔	The event of initiating interaction.
Interaction ✉	The reply-event of interaction

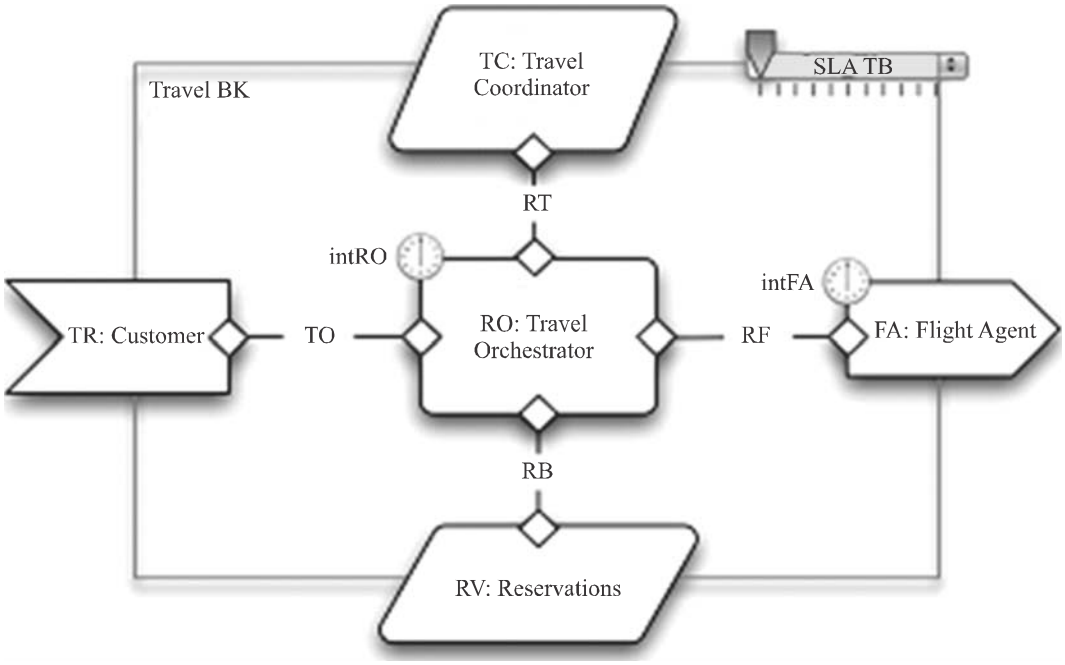

FIG. 6. THE VO MODULE, TRAVELBK [17]

Events can have several parameters (for instance, the initiation event bookTrip🔔 carries data about airports and dates), and the corresponding reply event bookTrip✉ carries reservation codes for the flight and the hotel. These events are used as atomic formulae in the language that we use to specify the properties that a customer can expect from the service. For instance, the first property specifies that the VO is ready to receive the initiation event of bookTrip. The declaration of the interactions in a signature is local to the component, i.e. all interaction names are local. This implies that there are no implicit relationships between components that result from the accidental use of the same name: all interconnections are externalized instead in what we call 'wires'. A wire defines a connector through which two components can be interconnected so that they can interact identical to their role in SRML.

Differences between VO-O and SRML: As indicated, VO-O is based on SRML, but there are some key differences. For a start, (a) in SRML all the members (service providers) are transient in the sense that each time a new instance is triggered (when new customer comes in) all the members are discovered and bounded to the instance; once the instance has served its goal all the members' association gets terminated as well. For VOs we considers two types of members (a) those whose membership with VO goes beyond single instantiation - persistent ones; and (2) those whose membership is only limited to the single instantiation (same as SRML service providers) - transient ones. (b) In VO-O partners and associates sit at the `top-end' of the module, but their behavior is defined using Business Protocol ; SRML uses Layer Protocol for `top-end' entities.(c) In VO-O entities at the `bottom-end' of a module still represent resources but those resources are provided by the VBE only. VBEresource use layer protocol just as in SRML. (d) In VO-O partners and associates are assumed to be already available and hence the external configuration policies of SRML for members are not part of the business protocol. (e) In SRML conversational interactions consists of five events request, reply, commit, cancel and revoke whereas VO-O has limited those events only to request and reply as VO members are obliged to provide what they have promised (part of the member definition) so rest of the events are not needed.

```
Business Protocol Customer is
Interactions
s&r bookTrip
from, to: airport
out, in : date
traveler :userdata
Reply:
fconf :fcoded
Behavior
InitiallyEnbabled bookTrip?
bookTrip? Ensures bookTrip!
```

FIG. 7. BUSINESS PROTOCOL DESCRIPTION IN VO-O

6. MAPPING VO-S (DOMAIN) MODEL TO VO-O (BUSINESS) MODEL

Consistency between different models representing specific dimensions of the same system is paramount. One of the contributions of this paper is to provide such consistency with the help of mappings which relate some piece of information available at one model to another model. These mappings help in automatically generating the basic skeleton of the VO-O model from VO-S model. Due to space constraints we are only going to discuss how an AtomicTask at VO-S level is mapped to a business protocol at the VO-O level, as this is seen as one of the most interesting happenings due to the centrality of task. Fig. 8 shows VO-S description of an atomic task; this VO-S description gets mapped to VO-O description of Fig. 9 description.

Recall that the VO-O language focuses on the operational dimension, the business protocol only talks about the

BUSINESS FUNCTIONALITY part of the VO-S task description. A business protocol specification mainly consists of signature and behavior pair.

Interaction Part: Though BUSINESS FUNCTIONALITY can be divided into more than one interaction at the VO-O level (provided the union of all the parameters of all the requesting and replying interactions is the same as the data items of the BUSINESS FUNCTIONALITY's Request and Reply parts, this examples looks at a case where the whole BUSINESS FUNCTIONALITY is replaced by one conversational interaction at the VO-O level and each part of BUSINESS FUNCTIONALITY in turn becomes a corresponding event of the interaction at the VO-O level. The name of the interaction starts with the name of the task, followed by hyphen symbol, then appending the word "interaction". A bell symbol is appended before the request (instantiation) parameters and an envelope symbol is appended before the reply parameters.

```
AtomicTask Flightbooking
Structure
…
Business Functionality
Request:
from, to: airport
out, in : date
traveler :userdata
Reply:
fconf :fcoded
```

FIG. 8. VO-S ATOMICTASK DESCRIPTION

```
Business Protocol Flightbookingis
Interactions
S&R lockFlight
from, to: airport
out, in : date
traveler :userdata
Reply:
fconf :fcoded
Behavior
InitiallyEnbabled lockFlight🔔?
bookTrip🔔? Ensures lockFlight ✉!
```

FIG. 9. VO-O BUSINESS PROTOCOL DESCRIPTION

The VO customer is one, on whose request the VO is created. The rest of the parties only become involved after the creation. This means that all the parties involved in the VO are passive, except the customer of the VO. This fact is the underlying argument for considering the interactions (of every component) of type r&s and of type s&r for the customer. Behavior part. The Behavioral part always starts with the initiallyEnabled keyword which lists the first communication that the business protocol is going to receive or send. In VO-O '?' symbolizes the processing of the interaction and '!' symbolizes the triggering of the interaction. The task's first interaction is always triggered by some entity external to the task, hence `?' is appended at the end of the interaction name, which implies that the member is waiting for the communication to get triggered. initiallyEnabled is always followed by the initiation event of first interaction (usually the only interaction). All the business protocols representing member's tasks wait for their first interaction to get triggered hence a ?is appended at the end of the interaction name. The next line in the given example ensures that once the instantiation event of the interaction has been received by the member, it is guaranteed that the member will send its reply.

7. CONCLUSIONS

In this paper we put forward a new and promising modelling language for VO-VOML. It is a compendium of sublanguages each focusing on a particular dimension (here the domain and business levels) of VOs at a particular level of its representation. Through these sublanguages VOML exposes an incremental approach where domain level details are defined in isolation of the functionality at the business configuration level using the VO-S language,

but at the same time provides enough details that allow the specification of a corresponding and consistent operational model using the VO-O language at its state configuration level. A third language, concerned with reconfigurations at the structural and operational level will complete the picture. The formally defined models will allow for quantitative and qualitative analysis that can help in making decisions for the creation, evolution or termination of VOs (besides business motives), for instance by supporting stochastic analysis on the usage that VOs can make of VBE resources or validation of functional properties that VOs offer through services. We are also investigating tools for VOML, mainly a compiler to automatically generate VO-O skeletons from VO-S descriptions.

ACKNOWLEDGEMENT

Authors would like to thanks University of Leicester, UK, for providing facilities and technical support to complish this study.

REFERENCES

[1] Camarinhamatos, L.M., Silveri, I., Afsarmanesh, H., and Oliveira, A.I., "Towards a Framework for Creation of Dynamic Virtual Organizations" 6thIFIP Working Conference on Virtual Enterprises, Camarinhamatos, L.M., et.al. (Editors), Volume 186, pp. 26-28, Springer, 2005.

[2] Cummings, J., Finholt, T., Foster, I., and Kesselman, C., "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 Organizations,2008.

[3] Foster, I., "The Anatomy of the Grid: Enabling Scalable Virtual Organizations", International Journal on High Performance Computing Applications, Volume 15, No. 3, pp. 200-222, USA, 2001.

[4] Mamarinhamatos, L.M., Afsarmanesh, H., and Ollus, M.,(Editors), "Virtual Organizations Systems and Practices", Springer, 2005.

[5] Esposito, E., and Pietro, E., "Investigating Virtual Enterprise Models: Literature Review and Empirical Findings", International Journal of Production Economics, Volume 148, pp. 145-157,Elsevier, 2014.

[6] Noran, O., "Collaborative Disaster Management: An Interdisciplinary Approach", Computers in Industry, Volume 65, No. 6, pp. 1032-1040, Elsevier, 2014.

[7] Coates, K., and Kimberly, F., "A Case for Collaborative Networks for Clinical Nurse Educators", Nurse Education Today, Volume 34, No. 1, pp. 6-10, Elsevier, 2014.

[8] Ovidiu, N., "Collaborative Networks in the Tertiary Education Industry Sector: A Case Study", International Journal of Computer Integrated Manufacturing, Volume 26, pp. 29-40, Taylor & Francis, 2013.

[9] Bryans, J., Fitzgerald, J., Jones, C., and Mozolevsky, I., "Formal Modelling of Dynamic Coalitions, with an Application in Chemical Engineering", IEEE 2ndInternational Symposium on Leveraging Applications of Formal Methods, Verification and Validation, pp. 91-98,Cyprus, 2006.

[10] Vassev, E., and Hinchey, M., "ASSL: A Software Engineering Approach to Autonomic Computing", IEEE Computer, Volume 42, No. 6, pp.90-93, 2009.

[11] Vassev, E., and Paquet, J., "ASSL - Autonomic System Specification Language", IEE Software Engineering Workshop, pp. 300-309, 2007.

[12] Fitzgerald, J., Larsen, P.G., Mukherjee, P., Plat, N., and Verhoef, M., "Validated Designs for Object-Oriented Systems", Springer-Verlag TELOS, Santa Clara, CA, USA, 2005.

[13] Norman, T.J., Preece, A., Jennings, N.R., Luck, M., Dang, V.D., Nguyen, T.D., Deora, V., Shao, J., Gray, W.A., and Fiddian, N.J., "Agent-Based Formation of Virtual Organizations", Knowledge Based Systems, Volume 17, pp. 103-111, UK, 2004.

[14] DiMarzo, S.G., Fitzgerald, J., Romanovsky, A., and Guel, N., "A Metadatabased Architectural Model for Dynamically Resilient Systems", Proceedings of ACM Symposium on Applied Computing. pp. 566-572, ACM, New York, USA, 2007.

[15] Laura, B., Jose, F.N.R., and Reiff-Marganiec, S., "Structure and Behaviour of Virtual Organization Breeding Environments", EPTCS, Volume 16, pp. 26-40, 2010.

[16] Rajper, N.J., "Virtual Organization Modelling Languages", Ph.D. Thesis, 2012, http://hdl.handle.net/2381/10942.

[17] Fiadeiro, J.L., Lopes, A., and Bocchi, L., "A Formal Approach to Service Component Architecture", Proceedings of 3rd International Conference on Web Services and Formal Methods, pp.193-213, Vienna, Austria, Springer, 2006.