

Structure and Behaviour of Virtual Organisation Breeding Environments

Laura Bocchi, José Fiadeiro, Noor Rajper and Stephan Reiff-Marganiec

Department of Computer Science, University of Leicester
University Road, Leicester LE1 7RH, UK

{lb148, jose, nr76, srm13}@mcs.le.ac.uk

This paper provides an outline of a formal approach that we are developing for modelling Virtual Organisations (VOs) and their Breeding Environments (VBES). We propose different levels of representation for the functional structures and processes that VBES and VOs involve, which are independent of the specificities of the infrastructures (organisational and technical) that support the functioning of VBES. This allows us to reason about properties of tasks performed within VBES and services provided through VOs without committing to the way in which they are implemented.

1 Introduction

This paper reports on on-going work towards a formal approach for modelling virtual organisations (VOs) and their breeding environments (VBES) in the sense of [10]. A VBE defines a base long-term cooperation agreement among a number of participants (individuals or institutions) and characterizes their interoperable infrastructure [9]. As such, a VBE represents the organisational context in which the creation and operation of VOs takes place; VOs are seen as ensembles that are formed dynamically to provide high-level functionalities, or services, by sharing a number of resources in a distributed way, using the new connectivity environments that are being made available through Global [15] and Grid Computing [14].

The purpose of developing a formal modelling approach echoes the challenge of building “Verifiable VOs” as raised in [7]. This implies that our approach is unavoidably partial: as in any formal account of the real world (which includes business), we need to operate on abstractions that are amenable to some form of mathematical representation and analysis. Our approach defines different levels of representation of VBES, VOs and their activities, which are essential for supporting several forms of analysis, from the properties of the coordination structures that are put in place through policies and workflows to the management of the resources that are shared within a VBE and used by their VOs.

There are multiple levels of abstraction at which formal methods can operate. In this paper, we abstract from the specificities of the infrastructures (organisational and technical, including IT) that support the functioning of VBES: we aim for an ‘infrastructure-agnostic’ account of the functional structures and processes that VBES and VOs involve. We concentrate on the functional and behavioural aspects in which partners and resources are involved without committing to the way in which they are effectively implemented. Therefore, we do not model the brokers (human or software) that procure services that can be used to create business or the communication networks that support interconnectivity — what we could call the VBE middleware.

We are also aware that the model that we present in this paper misses several aspects of VBES. For instance, we do not address at present the decision-making processes through which VOs are created within a VBE or the actual business goals that preside to the creation of a VBE (see [11] for an overview

of some of the formal approaches that have been proposed to address these issues). However, we do intend that the formal models that we propose can be used to inform such processes, for instance by supporting stochastic analysis on the usage that VOs can make on VBE resources or validation of functional properties that VOs offer through services, something that we are leaving for future work.

Our approach supports the definition of a structural and behavioural 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, what we call 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, what we call a *state configuration*. 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: 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 organised 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. ‘Real’ entities are only represented in state configurations: the other levels deal only with types of entities.

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

- A VBE consist 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* 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 (in the sense explained below).
- The current business configuration of a VBE, is understood as (1) the collection of additional (non-permanent) partners, that we call *associates*, and resources that are part of the VBE; (2) the tasks 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. 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 or discovered on the fly when needed, subject to service-level agreements, in order to be able to accommodate the needs of specific clients.
- The current state configuration of a VBE consists of ‘components’, connected through ‘wires’, that jointly operate the tasks 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. 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.

Part of the importance of distinguishing between these three levels is that we can account for two different kinds of change (admitting that the VBE level is invariant, the creation of which we do not model at present):

- Changes in the business configuration reflect the creation or deletion of tasks or VOs. Creating a new VO may involve identifying associates or the criteria that will need to be observed for

discovering such associates on the fly, depending on the nature of the customers that procure the service (in which case each service may involve different associates). 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 business concerns (which we do not model at present).

- Changes to the state configuration result from the launching of (instances of) 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 ‘timebands’ in the sense of [8], 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.

Given the limited space available, we focus only on the VBE and business configuration levels. In the sequel, we outline the formalisms and methodology that we are proposing for each level of representation and change, which we have adapted (and extended) from recent work on service-oriented modelling [12, 13]. Essentially, we use graph-based representations to formalise and establish relationships between the two levels — logic/process-based formalisms for the specification of activities and services.

2 A Model of Virtual Organisation Breeding Environments

As already mentioned, we see a VO as a dynamic ensemble of entities that operate over a communication and collaboration network through which they can share resources to offer services. Some of those entities and resources are provided by the VBE in which the VO was created; others are external to the VBE and co-opted or procured to satisfy the business goal of that particular VO. We define a (formal model of a) VBE to consist of:

- A collection of persistent partners, where a partner consists of:
 - Its name (individual or organisation, virtual or not);
 - A collection of attributes through which policies can be defined on the involvement of the partner in business configurations.
- A collection of persistent resources where a resource consists of:
 - Its identifier;
 - A collection of attributes through which the usage of the identified resource can be monitored.
- A collection of policies expressed over partners and resources that apply to all business configurations of the VBE.
- A collection of tasks that support the roles enacted within the VBE. A task is defined by a *task-module* consisting of:
 - Component specifications that are used in state configurations as interfaces to the partners (in which they are called *serves-interfaces*) or resources (in which they are called *uses-interfaces*) involved in the task;
 - Specifications of components and wires that jointly orchestrate the task.

- Mappings from the *serves-interfaces* (resp. *uses-interfaces*) to the partners (resp. resources) of the VBE complete the task definition.

Notice that we separate the *task-module* from the way it is used in the VBE. Effectively, task-modules are design primitives that define patterns that can be reused in the definition of VBEs.

As an example, we use a very simple scenario: a group of hotels, a car rental company and a guided-tour company decide to create a VBE — *visitUs* — to promote tourism in the local town.

- The partners of *visitUs* are the hotels (to make the example shorter, we model the case of two hotels, *grandHO* and *centralHO*), the car rental agency *carHI* and the guided-tour company *tourAG*;
- The resources include two systems: *registrationSY* supporting management activities of *visitUs* and a shared reservation system *reservationSY* supporting the business purpose (hotel bookings and so on).
- The policies establish criteria for the admission of transient partners (e.g. they need to operate within a given vicinity) and the use of the shared resources (e.g. the cost of maintaining the reservation system), for which their interfaces need to include parameters that capture these properties. The policies are expressed as first-order expression in the language of the parameters and the corresponding data types.
- The tasks include the process *managerRO* that supports the administrator role performed by *grandHO*, which connects to the registration system, and the process *memberRO* that allows each partner to use the reservation system. (Other roles might have been considered as discussed in [11].)

We use a graphical notation to depict task modules as illustrated in Figure 1 for the task *managerRO* that supports the administrator of *visitUs*. The specification of the component MO that orchestrates the task is Management Orchestrator and the wires are *RM* and *MR*. The specification of the serves-interface *MN* used by *grandHO* - the partner who performs the managerial role - is Registry Manager, and the specification of the uses-interface *RE* (which connects to *registrationSY*, the resource that supports the task) is Registry.

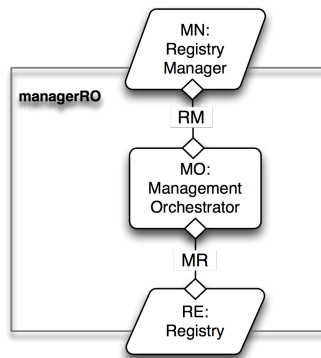
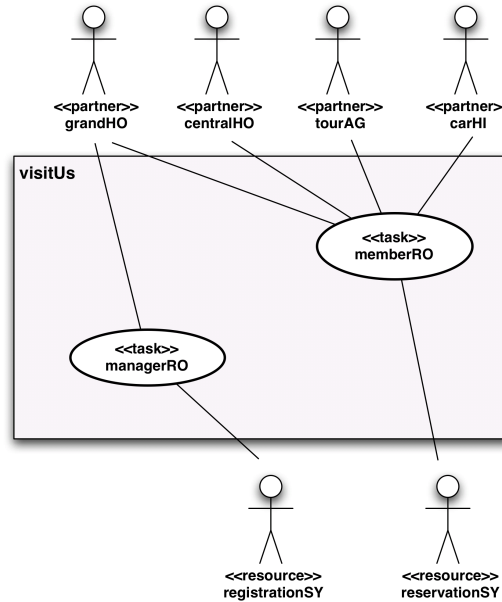


Figure 1: The task module *managerRO*

We also use a graphical notation to depict VBEs, which is inspired by use-case diagrams (though our usage of the notation is not necessarily faithful to its original purpose). As illustrated in Figure 2, we use stereotypes to identify the actors that correspond to partners and resources of the VBE. Each task is represented by a use-case.

Figure 2: VBE diagram for *visitUS*

3 Business Configurations

The current business configuration of a VBE defines the types of tasks and the VOs that the VBE supports to meet its (current) business goals. The actual process through which a VBE decides on how to configure itself is not part of our present model as it depends on a number of business or organisational concerns that the formal methods that we are illustrating do not address. However, the kinds of qualitative and quantitative analysis that our approach provides should be able to inform that process and corresponding decisions, something that we plan to address in the future.



We define a VBE business configuration to consist of an extension of the VBE with:

- A collection of *associates* (transient partners), where an associate consists of:
 - Its name (individual or organisation, virtual or not);
 - A collection of attributes through which policies can be defined on the involvement of the associate in the VBE.
- A collection of transient resources, each of which consists of:
 - Its identifier;
 - A collection of attributes through which the usage of the identified resource can be monitored.
- A collection of policies expressed over associates and their resources that apply to their involvement in the VBE (e.g. the conditions that determine the cessation of their involvement).
- A collection of tasks that support the roles enacted by the associates within that business configuration of the VBE.
- A collection of VOs that define the services that the VBE provides in that business configuration. A VO is defined by a *VO module* consisting of:

- Component specifications that are used in state configurations as serves-interfaces to the partners or associates, or uses-interfaces to resources involved in the VO; typically, one such interface serves the coordinator of the VO.
 - Component specifications that are used as requires-interfaces for external entities or as the provides-interface for the customers of the VO. The specification of requires-interfaces identifies the behavioural properties that are expected of external parties to be eligible to be chosen as service providers for the VO. The specification of the provides-interface identifies the properties that customers can expect of the service offered by the module.
 - Specifications of the components and wires that model the (possibly distributed) process that orchestrates the services provided by the VO.
 - An internal configuration policy, which identifies the triggers of the external service discovery process as well as the initialization and termination conditions of the components and wires.
 - An external configuration policy, which consists of the variables and policies that determine the quality profile to which the discovered services need to adhere.
 - Mappings from the serves-interfaces (resp. uses-interfaces) to the partners or associates (resp. resources) of the VBE complete the VO definition.
- A collection of *external entities*, each of which represents a partner that may need to be co-opted to provide a service for one of the VOs that the VBE offers in that business configuration.
 - A collection of *customers*, one for each of VO, each of which defines the interface (interactions and functional properties) that the customer of the corresponding VO can expect.

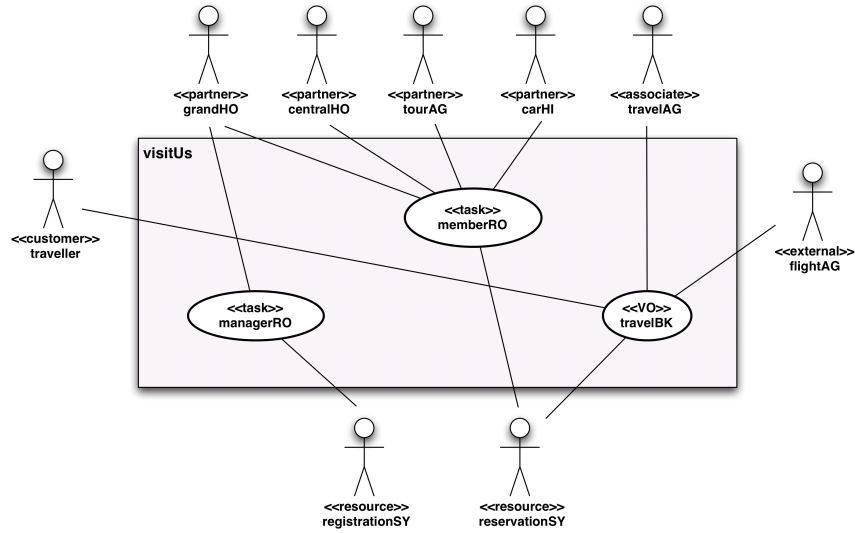
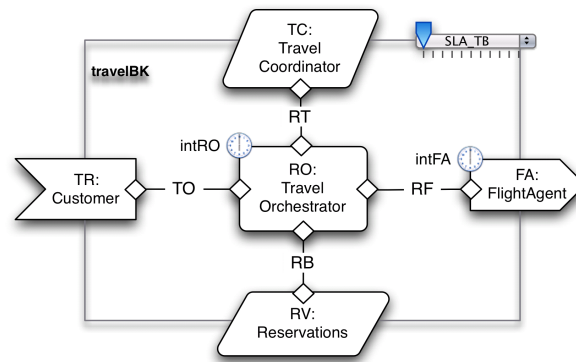
As for tasks, VO-modules are design primitives that define patterns that can be reused in the definition of multiple VBE business configurations. As an example, we consider a business configuration of *visitUs* in which a travel booking service is offered through a VO named *travelBK*. An associate named *travelAG* is admitted as a member of the VBE for managing that VO. Services offered through *travelBK* may require an external flight agent to be discovered according to the criteria specified in *flightAG*. A specific agent is not chosen as an associate in order to maximise customer satisfaction — each customer of the VO may express service-level policies (e.g. preference for a particular airline, or minimum cost, or proximity) that will be optimised when selecting the corresponding external partner.

We extend the diagrams used for VBEs to account for business configurations as illustrated in Figure 3.

We use a graphical notation similar to task-modules to depict VO-modules as illustrated in Figure 4 for *travelBK*. We use the symbol  to indicate the internal configuration policy as it applies to components and requires interfaces, and  for the external configuration policy. The module consists of a provides-interface *TR* for interactions with customers of the VO, a serves-interface for interactions with the coordinator of the VO, a uses-interface for interactions with the reservations system, and a requires-interface for the discovery of a flight agent.

A more formal definition follows where each node uniquely represents a specific interface of an entity (e.g., institution, participant, etc.) in a business configuration rather than the entity itself. A task- or VO-module M defines:

- A graph $graph(M)$.
- A distinguished subset of nodes $uses(M) \subseteq nodes(M)$.
- A distinguished subset of nodes $serves(M) \subseteq nodes(M)$ distinct from $uses(M)$.

Figure 3: VBE business configuration for *visitUS*Figure 4: The VO module *travelBK*

- In the case of a VO-module, a subset of nodes $requires(M) \subseteq nodes(M)$ distinct from $uses(M)$ and $serves(M)$.
- In the case of a VO-module, a node $provides(M) \in nodes(M)$ distinct from $requires(M)$, $serves(M)$ and $uses(M)$.
- A labelling function $label_M$ such that:
 - $label_M(n)$ is a component specification.
 - $label_M(e : n \leftrightarrow m)$ is a connector.

Component specifications and connectors are discussed in Section 4. In the case of a VO-module, we denote by $body(M)$ the (full) sub-graph of $graph(M)$ that forgets the node $provides(M)$, the nodes in $requires(M)$ and the edges that connect them to the rest of the graph. That is, $body(M)$ consists of all the elements that are internal to the VO.

A business configuration of a VBE also defines a labelled graph obtained by expanding its tasks and VOs with the bodies of the labelled graphs that correspond to their modules. Having such a formal representation for VBE configurations allows us to use graph transformations to formalise rules and policies to evolve the configurations, for instance the creation of new VOs. In Figure 5, we depict a business configuration that extends the one in Figure 3 with a new VO that offers arrangements for weddings as a service.

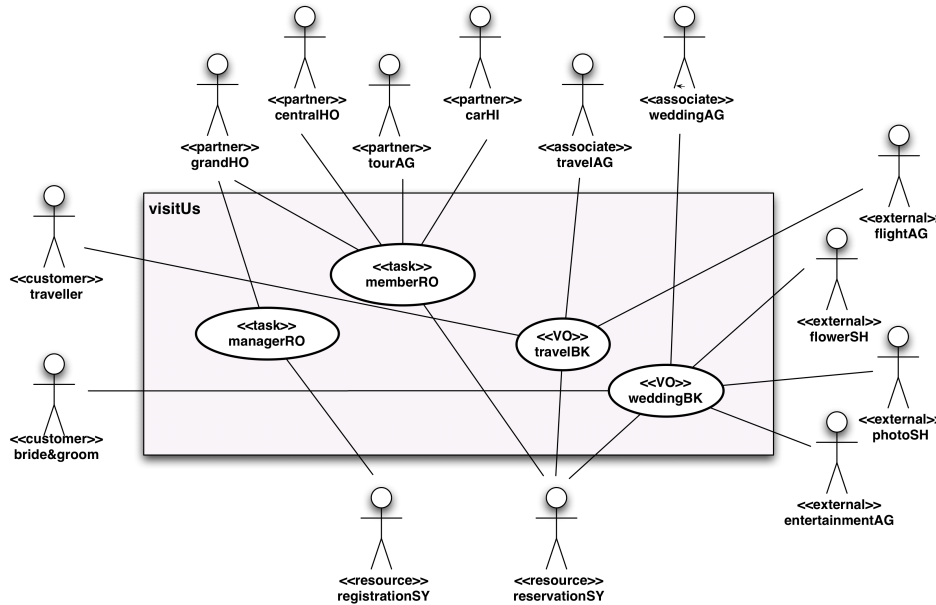


Figure 5: Another VBE business configuration for *VisitUS*

4 Component Specifications and Connectors

In order to account for the behaviour that, in state configurations (referred to as level 3 in Section 1), emerges from the interconnections established inside the ensembles that perform tasks or 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 $\langle \textit{signature}, \textit{behaviour} \rangle$ where:

- *Signature* declares the interactions in which the component may be involved.
- *Behaviour* is a formal model of the behaviour 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 that we have proposed for the service modelling language SRML [12]). We discuss below the provides-interface of *travelBK*, 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 adopted for service-oriented modelling [2]. The remaining specifications can be found in the Appendix.

In the formalism that we adopt, interactions can be either synchronous or asynchronous, one-way or two-way (i.e. conversational):

- *s&r/r&s* — conversational asynchronous interaction where the initiating party expects a reply from its co-party
- *rcv/snd* — one-way asynchronous receive/send
- *ask/rpl* — synchronization with the co-party to obtain/transmit data
- *tll/prf* — blocking requests to the co-party to perform an operation

In our example, the interface that *travelBK* offers to its customers specifies that the VO can engage in the interaction *bookTrip* (initiated by the customer) and send *payNotify* and *refund* to the customer.

```

BUSINESS PROTOCOL Customer is
INTERACTIONS
  r&s bookTrip
     $\ominus$  from, to : airport, out, in : date
     $\boxtimes$  fconf : fcode, hconf : hcode, amount : moneyvalue
  snd payNotify
     $\boxtimes$  status : bool
  snd refund
     $\boxtimes$  amount : moneyvalue
SLA VARIABLES
  KD : [0..100], PERC : [0..100]
BEHAVIOUR
  initiallyEnabled bookTrip  $\ominus$ ?
  (bookTrip  $\boxtimes$   $\wedge$  bookTrip  $\checkmark$ ?) ensures payNotify  $\ominus$ !
  (payNotify  $\ominus$ !  $\wedge$  payNotify.status) enables bookTrip  $\ddagger$ ? until today + KD < bookTrip.out
  (bookTrip  $\ddagger$ ?  $\wedge$  today + KD < bookTrip.out) ensures refund  $\ominus$ !
  refund.amount > bookTrip.amount * PERC / 100 after refund  $\ominus$ !

```

Interactions of type *r&s* and *s&r* are conversational (in the sense of [4]), i.e. they involve a number of events exchanged between the two parties:

<i>interaction</i> \ominus	The event of <i>initiating interaction</i> .
<i>interaction</i> \boxtimes	The reply-event of <i>interaction</i> .
<i>interaction</i> \checkmark	The commit-event of <i>interaction</i> .
<i>interaction</i> \boxtimes	The cancel-event of <i>interaction</i> .
<i>interaction</i> \ddagger	The revoke-event of <i>interaction</i> .

The meaning of these events should be self-explanatory: the *reply-event* is sent by the co-party, offering a deal or declining to offer one; in the first case, the party that initiated the conversation may either commit to the deal or cancel the interaction; after committing, the party can still revoke the deal, triggering a compensation mechanism. Events can have several parameters (for instance, the initiation event *bookTrip* \ominus carries data about airports and dates), and the corresponding reply event *bookTrip* \boxtimes carries reservation codes for the flight and the hotel as well as the total cost).

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 second property says that a commit event received during the validity period of the booking entitles the customer to receive a pay confirmation.

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 of the same name: all interconnections are externalised instead in what we call ‘wires’. A wire defines a connector through which two components can be interconnected so that they can interact. More specifically, a connector [1, 18] is a triple $\langle roleA, Glue, roleB \rangle$ where $roleA$ and $roleB$ are signatures and $Glue$ defines the protocol that coordinates the interactions identified in $roleA$ and $roleB$ — this may include routing events, superposing protocols for secure communication, or transforming sent data to the format expected by the receiver, inter alia. A wire interconnects two components through the connector by mapping $roleA$ to one component and $roleB$ to the other.

Service-level agreements are negotiated through policies using the c-semiring approach to constraint satisfaction and optimisation [5]. An example of a policy is:

$$\{TC.KD, TR.PERC\}$$

$$def_1(d,p) = \begin{cases} 1 & \text{if } d \in [0..100] \text{ and } 1 \leq d \text{ and } p \leq 90 \text{ and } p \leq 50+5*d \\ 0, & \text{otherwise} \end{cases}$$

The policy expresses that percentage p of the cost that is refundable (transmitted to the customer through the SLA variable $TR.PERC$) is bounded by the least of 90% and a linear function of the period d during which the deal can be revoked, which is established by the VO coordinator through the variable $TC.KD$.

5 Concluding Remarks and Further Work

In this paper, we have outlined a formal approach that we are defining for modelling structural and behavioural aspects of VBEs and VOs. Several levels of representation are proposed for VBEs that distinguish between (1) the persistent aspects of VBEs in terms of members, resources and tasks that involve them, (2) the possible business configurations of VBEs characterised in terms of the VOs that it creates to provide services and the additional (associate) members that are involved in the VOs, and (3) the state configurations of VBEs, which result from the services (instants) offered by the VOs at a given state.

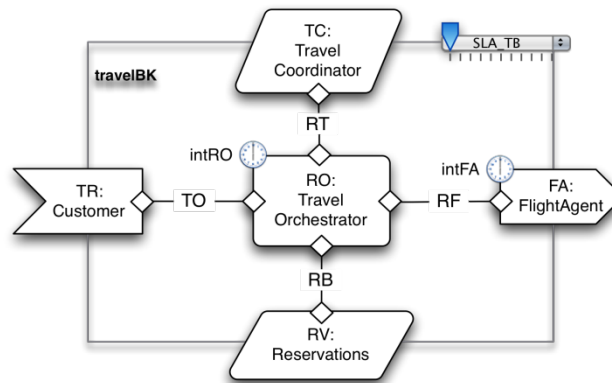
From a formal point of view, these levels of representation are graphs whose nodes are component specifications and the edges (wires) are connectors. Component specifications provide either interfaces for partners and resources to be involved in tasks and services offered through VOs, or orchestrations 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 (e.g. [13, 17]) in order to account for the evolution of VBE business configurations (as VOs are added, deleted or modified) and also their configuration states (as new services are created and bound to customers).

As formalisms for specification, we are using those put forward for service-oriented modelling in the SENSORIA project [1, 2, 12]. 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 [3], which we plan to extend to VOs. The proposed formal model also supports forms of quantitative analysis using the stochastic analyser PEPA [6, 16], which we intend to extend to VOs. Negotiation of service-level agreements is supported by techniques for constraint optimisation [5], which again we plan to use for the discovery of services from external partners that VOs may require.

References

- [1] J. Abreu, L. Bocchi, J. L. Fiadeiro, A. Lopes (2007): *Specifying and composing interaction protocols for service-oriented system modelling*. In: J. Derrick, J. Vain (eds) *Formal Methods for Networked and Distributed Systems*. LNCS, vol 4574. Springer, pp. 358–373.
- [2] J. Abreu, J. Fiadeiro (2008): *A coordination model for service-oriented interactions*. In: D Lea, G. Zavattaro (eds) *Coordination Languages and Models*. LNCS, vol 5052. Springer, pp.1–16.
- [3] J. Abreu, F. Mazzanti, J. Fiadeiro, S Gnesi (2009): *A model-checking approach for service component architectures*. In: D. Lee, A. Lopes, A. Poetzsch-Heffter (eds) *FMOODS-FORTE'09*. LNCS, vol 5522, Springer, pp. 212–217.
- [4] B. Benatallah, F. Casati, F. Toumani (2004): *Web services conversation modeling: A cornerstone for e-business automation*. *IEEE Internet Computing* 8(1): pp. 46–54
- [5] S. Bistarelli, U. Montanari, F. Rossi (1997): *Semiring-based constraint satisfaction and optimization*. *Journal of the ACM* 44(2): pp. 201–236
- [6] L. Bocchi, J. Fiadeiro, S. Gilmore, J. Abreu, M. Solanki, V. Vankayala (2009): *A Formal Model for Timing Aspects of Service-Oriented Systems*. (Available from www.cs.le.ac.uk/jfiadeiro).
- [7] J. Bryans, J. Fitzgerald (2008): *The verifiable virtual organisation: a position paper*. In: *Proc. Formal Aspects of Virtual Organisations 2008*. Newcastle University CS-TR-1098.
- [8] A. Burns, G. Baxter (2006): *Time bands in systems structure*. In: D. Besnard, C. Gacek, C. B. Jones (eds) *Structure for Dependability: Computer-Based Systems from an Interdisciplinary Perspective*. Springer, pp. 74–88.
- [9] L.Camarinha-Matos, H. Afsarmanesh (2004): *The emerging discipline of collaborative networks*. In: *Proc. Virtual Enterprises and Collaborative Networks 2004*. Kluwer, pp. 3-16.
- [10] L.Camarinha-Matos, H. Afsarmanesh (2003): *Elements of a base VE infrastructure*. *Journal of Computers in Industry* 51(2): 139–163.
- [11] L. Camarinha-Matos, H. Afsarmanesh (2007): *A framework for virtual organization creation in a breeding environment*. *Annual Reviews in Control* 31: 119–135.
- [12] J. L. Fiadeiro, A. Lopes, L. Bocchi (2006): *A formal approach to service-oriented architecture*. In: M. Bravetti, M. Nunez, G. Zavattaro (eds) *Web Services and Formal Methods*. LNCS, vol 4184. Springer, pp. 193–213.
- [13] J. L. Fiadeiro, A. Lopes, L. Bocchi (2008): *An Abstract Semantics of Service Discovery and Binding*. Submitted. (Available from www.cs.le.ac.uk/jfiadeiro)
- [14] I. Foster, C. Kesselman (eds) (2004): *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann
- [15] Global Computing Initiative, <http://cordis.europa.eu/ist/fet/gc.htm>
- [16] J. Hillston (1996): *A Compositional Approach to Performance Modelling*. Cambridge University Press
- [17] D. Hirsch, U. Montanari (2001): *Two graph-based techniques for software architecture reconfiguration*. *Electronic Notes in Theoretical Computer Science* 51, pp. 177–190.
- [18] M. Shaw, D. Garlan (1996): *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, London

Appendix – The *TravelBK* VO-module



TRAVELBK consists of:

- TR – the provides-interface of the module, of type *Customer*;
- FA – a requires-interface (for a flight-booking service), of type *FlightAgent*;
- RO – the component that orchestrates the business process, of type *TravelOrchestrator*;
- RV – a uses-interface for a resource that provides a registration system, of type *Reservations*;
- TC – a serves-interface for the partner that plays the role of coordinator of the VO, of type *TravelCoordinator*;
- TO, RB, RF, RT – wire-interfaces typed by connectors that establish the required interconnections.

VO *TravelBK* is

COMPONENTS

```

RO: TravelOrchestrator
  intRO ⌚ init: s=START ∧ hconf=NILL
  intRO ⌚ term: s=END_UNBOOKED
                ∨ (s=CONFIRMED ∧ today≥bookTrip.out) ∨ s=END_COMPENSATED
  
```

PROVIDES

```

TR: Customer
  
```

REQUIRES

```

FA: FlightAgent
  intFA ⌚ trigger: hconf=hcode
  
```

SERVES

```

TC: TravelCoordinator
  
```

USES

```

RV: Reservations
  
```

EXTERNAL POLICY (partial)**SLA VARIABLES**

TC.KD, TR.PERC, FA.MAX, TR.KD

CONSTRAINTSC₁: {TC.getFlightCommission(FA.serviceId),FA.MAX}

$$\text{def}_1(d,p) = \begin{cases} 1 & \text{if } d \leq p \\ 0 & \text{otherwise} \end{cases}$$

C₂: {TC.KD,TR.PERC}

$$\text{def}_2(d,p) = \begin{cases} 1 & \text{if } d \in [0..100] \text{ and } 1 \leq d \text{ and } p \leq 90 \text{ and } p \leq 50 + 5 * d \\ 0 & \text{otherwise} \end{cases}$$

WIRES (partial)

TR Customer	c ₁	TO	d ₁	RO TravelOrchestrator
s&r bookTrip ⚠ from to out in traveller travcard ⊠ fconf hconf amount	S ₁ i ₁ i ₂ i ₃ i ₄ i ₅ i ₆ o ₁ o ₂ o ₃	≡	R ₁ i ₁ i ₂ i ₃ i ₄ i ₅ i ₆ o ₁ o ₂ o ₃	r&s bookTrip ⚠ from to out in traveller travcard ⊠ fconf hconf amount
rcv refund ⚠ amount	R ₁ i ₁	≡	S ₁ i ₁	snd ackRefundSnd ⚠ amount
RO TravelOrchestrator	c ₂	RF	d ₂	FA FlightAgent
s&r bookFlight ⚠ from to out in traveller ⊠ fconf amount	S ₁ i ₁ i ₂ i ₃ i ₄ i ₅ o ₁ o ₂	≡	R ₁ i ₁ i ₂ i ₃ i ₄ i ₅ o ₁ o ₂	r&s lockFlight ⚠ from to out in traveller ⊠ fconf amount

END MODULE

SPECIFICATIONS**LAYER PROTOCOL** TravelCoordinator **is****INTERACTIONS**

rpl getFlightCommission(FAid:serviceId):moneyvalue

LAYER PROTOCOL Reservations **is****INTERACTIONS**

rpl availability(out,in:date):hcode
prf book(hconf:hcode)
prf cancel(hconf:hcode)

BUSINESS ROLE TravelOrchestrator **is****INTERACTIONS**

r&s bookTrip
 Ⓐ from,to:airport,
 out,in:date,
 traveller:usrdata
 travcard:paydata
 ☒ fconf:fcode,
 hconf:hcode,
 amount:moneyvalue
ask findHotel(out,in:date):hcode
t1l bookHotel(hconf:hcode)
t1l cancelHotel(fconf:hcode)
snd ackRefundSnd
 Ⓐ amount:moneyvalue
s&r bookFlight
 Ⓐ from,to:airport,
 out,in:date,
 traveller:usrdata
 ☒ fconf:fcode
 amount:moneyvalue

ORCHESTRATION

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

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

transition FlightAnswer

triggeredBy bookFlight☒
guardedBy s=QUERIED
effects bookFlight.Reply ⊃ s'=FLIGHT_OK
 ∧ ¬bookFlight.Reply ⊃ s'=END_UNBOOKED
sends bookFlight.Reply ⊃ bookTrip☒ ∧ bookTrip.Reply=True
 ∧ ¬bookFlight.Reply ⊃ bookTrip☒ ∧ bookTrip.Reply=False

transition TripCommit

triggeredBy bookTrip✓
guardedBy s=FLIGHT_OK
effects s'=CONFIRMED ∧ bookHotel(hconf)
sends bookFlight✓ ∧ payNotifyⒶ

```

transition TripCancel
  triggeredBy bookTrip*
  guardedBy s=FLIGHT_OK
  effects s'=END_UNBOOKED  $\wedge$  cancelHotel(hconf)
  sends bookFlight*

transition TripCompensate
  triggeredBy bookTrip†
  guardedBy s=CONFIRMED  $\wedge$  today<bookTrip.out
  effects s'= END_COMPENSATED  $\wedge$  cancelHotel(hconf)
  sends bookFlight†  $\wedge$  ackRefundSnd!
       $\wedge$  ackRfundSnd.amount=bookTrip.amount*PERC/100

transition ConfirmBookTripTimeOut
  triggeredBy now $\geq$ bookTrip.UseBy
  guardedBy s=FLIGHT_OK
  effects s'=END_UNBOOKED  $\wedge$  cancelHotel(hconf)
  sends bookFlight*

```

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  $\geq$  bookTrip.out

```

BUSINESS PROTOCOL Customer is

INTERACTIONS

```

s&r bookTrip
  ! from,to:airport,
  out,in:date
  ☒ fconf:fcode,
  hconf:hcode,
  amount:moneyvalue

rcv refund
  ! amount:moneyvalue

```

SLA VARIABLES

```

KD:[0..100], PERC:[0..100]

```

BEHAVIOUR

```

initiallyEnabled bookTrip!
(bookTrip $\bar{?}$   $\wedge$  bookTrip✓?) enables bookTrip†?
until today+KD $\geq$ bookTrip.out
(bookTrip†?  $\wedge$  today+KD  $\geq$  bookTrip.out) ensures refund!
refund.amount=bookTrip.amount*PERC/100 after refund!

```

INTERACTION PROTOCOL Straight.I(d₁) is

ROLE A

```

snd S1
  ! i1:d1

```

ROLE B

```

rcv R1
  ! i1:d1

```

COORDINATION

```

S1 = R1
S1.i1=R1.i1

```

...

END SPECIFICATIONS