

# Defining customizable business processes without compromising the maintainability in multi-tenant SaaS applications

Malinda Kapuruge, Alan Colman and Jun Han  
 Faculty of Information and Communication Technologies  
 Swinburne University of Technology,  
 Melbourne, Australia  
 {mkapuruge, acolman, jhan}@swin.edu.au

**Abstract**— In Software-as-a-Service (SaaS) delivery model a vendor maintains a single application instance, which is used by multiple tenants. However, due to changing business requirements tenants expect customizations. Providing such customizations is trivial to retain tenants but a challenge to the vendor due to multi-tenancy. In this paper we present an approach to define such customizable business processes in multi-tenant SaaS applications without unnecessary and hard to maintain duplication of process definitions and deployments.

**Keywords**-SaaS; Cloud; SaaS; SOA; BPM; Middleware

## I. INTRODUCTION

SaaS vendors as business entities have to depend on third party service providers to provide certain functionalities as they alone might not be able to meet all requirements. Service Oriented Architecture (SOA) principles can be used to model such applications. Also Business Process Modeling (BPM) can be used to coordinate, package and deliver the business offerings. Tenants can define their business processes and later customize to suit renewed business requirements. But such customizations stem from a particular tenant’s perspective realized in a single application instance might lead to violations of goals of other tenants.

A naïve solution might be to allocate multiple isolated process definitions for each and every tenant so that customizations do not violate integrity of other tenants’ processes (process isolation). But in SaaS applications tenants share a common interest; hence, there will be many commonalities in those isolated process definitions. A separate process definition for each client leads to much duplication obviating any benefits from single-instance multi-tenancy[1]. As such the vendor possibly has to change multiple process definitions upon a change requirement and consequently maintenance can be challenging and costly. Therefore, a BPM approach for a SaaS application needs to fulfill three requirements.

Firstly, the **commonalities in behavior** must be captured to reduce unnecessary duplications. On the other hand a SaaS vendor cannot assume that *one size fits all*. Tenants’ requirements tend to slightly vary from each other. So the second requirement is the ability to **defining variations in behavior**. Third, it is important to **avoid invalid boundary crossings** to ensure appropriate process isolation. In this sense, when a modification is carried out in a common code from a particular tenant’s perspective, there shouldn’t be violations of goals of other tenants. Our approach defines a method for composing processes from modular units of behavior in order to meet these three requirements.

## II. APPROACH

Consider an expository scenario where a *Road Side Assistance Software System* is made available as a SaaS application (*RoSaaS.com*). Small and medium scale businesses such as *Travel Agents* and *Car Sellers*, who cannot afford to maintain such software systems and associated infrastructure, so can benefit by using *RoSaaS* on subscription basis. *RoSaaS* models the SaaS application as a composite of services which utilize a number of other collaborating services such as *Tow Trucks (TT)*, *Garages (GR)*, *Case Officers (CO)* and *Taxis (TX)*. *RoSaaS* uses BPM to model the coordination among these collaborating services and eventually deliver the assistance to *Motorists (MM)*. The *motorists* are the customers of tenants. Tenants, i.e., *Car Sellers and Travel agents* can customize the business processes to alter the way they need services to be delivered to their customers/motorists.

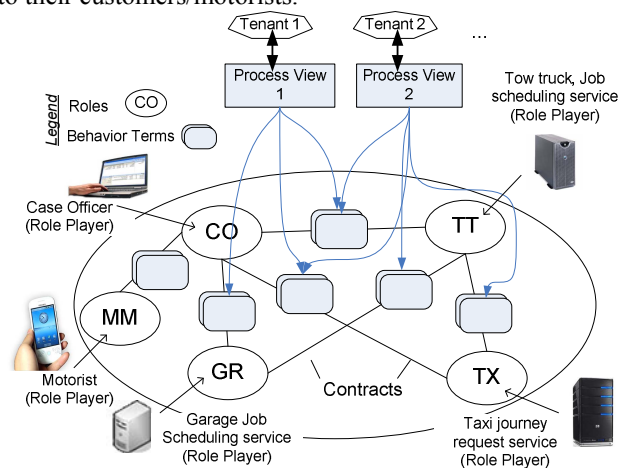


Figure 1: Process views refer to behavior terms

In our approach we envision the *RoSaaS* application instance as an **organization** which is an abstraction of underlying collaborators (service providers/consumers) and their collaborations. The organization captures the **roles** and the **contracts** among those roles[2]. We use ROAD design concepts to model such organization[2]. An overview is given in Figure 1. On top of this organizational structure, we define different behaviors which we call **behavior terms** [3]. Each behavior term specifies an accepted behavior (ordering of tasks) of the organization, e.g. *When towing is required CO should send the towing request to TC. When a car is towed, the TC should notify the CO. Upon that notification,*

CO should pay the TC and so on. These behavior terms are self-contained and groups the binary (point to point) interactions among roles.

A sample behavior term is given in Figure 2. Multiple business processes can refer and re-use these behavior terms. The objective is to capture the **commonalities in behavior** of the organization so that there is less redundancy in modifications/maintenance. For more details about behavior modeling please refer to [3].

```

BehaviorTerm TowingBehavior{
  Task SendTowReq{
    EPre TowReq;
    EPost TowReqSent & PickupLockKnown;
    PerfProp 2h;
    Roblig CO;
  };
  Task Tow{
    EPre PickupLockKnown & DestinationKnown;
    EPost CarTowed & TowAcked;
    PerfProp 24h;
    Roblig TC;
  };
  Task PayTow{
    EPre CarTowed & TowAcked;
    EPost TCPaid;
    PerfProp 2h;
    Roblig CO;
  };
  //More tasks...

  Constraint c1: (TowReqSent>0) ->[4h,24h](CarTowed>0);
  Constraint c2: (CarTowed>0) ->(TCPaid>0);
  //More constraints...
};

```

Figure 2: A Sample behavior term

Note that in our approach, a process definition a collection of such behavior terms[3]. In this sense, it is just a view provided for a tenant. These views can be customized during runtime. However, the actual customization occurs in a single organizational structure. This brings the challenge to meet the requirements described above.

In order support **defining variations in behavior** we use **behavior term specialization** technique. Meaning, a behavior term can extend/specialize already defined behavior term as shown in Figure 3. The child behavior term only needs to specify the additional properties (e.g. a new task, a different pre-condition for a task) and rest of the properties are inherited from the parent. As such if a tenant requires a variation in behavior, the *RoSaaS* designer can define a new behavior term via specialization and use the child/specialized behavior term in tenants process view instead of the parent. Also different variations of towing behaviors could co-exist in the organization.

```

BehaviorTerm TowingVariant extends TowingBehavior{
  Task PayIncentive{
    EPre CarTowed;
    EPost IncentivePaid;
    PerfProp 2h;
    Roblig GR;
  };
};

```

Figure 3: An speacalized behaviour term

To **avoid invalid boundary crossings** we use constraints defined in two different levels. First, there are constraints defined in the behavior terms to safeguard the collaborator goals known as **behavior level constraints**. Second, the process views(tenants) can also specify additional constraints

to safeguard the goals of tenants known as **process level constraints**. The Figure 4 shows, upon modification in a shared behavior term, the new behavior is validated against these two different levels of constraints. In this way, it is possible to detect the possible violations of goals of other tenants and collaborators of a modification request. Also only the relevant set of constraints is being used for validation without unnecessary restrictions compared to use of global constraints.

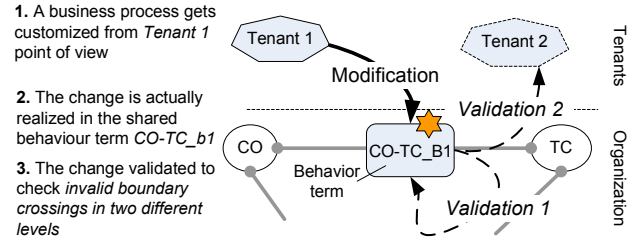


Figure 4: Validation of constraints

The table below shows the link between the **concepts** and the **purpose** they serve to fulfill the three requirements.

Concept	Purpose
Behavior term	Groups related tasks and constraints to capture commonalities in behavior
Behavior Specialization	Allow variations in composite behavior with lesser redundancy
Two Level Constraints	Avoid invalid boundary crossings of (a) other tenants (b) collaborators/undelying service providers

### III. CONCLUSION

Different BPM approaches for SaaS applications have been proposed in the past[4]. Also techniques such as *template-based* and *aspect orientations* have been used to capture commonalities and variations of processes [5, 6]. Nonetheless little attention has been paid to achieve the customization without compromising the maintainability of business process modeling in single-instance multi-tenant[1] environments. Our BPM approach for BPM in SaaS applications overcomes those limitations.

### REFERENCES

- [1] G. Chang Jie, et al., "A Framework for Native Multi-Tenancy Application Development and Management," in Enterprise Computing, CEC/EEE . , 2007, pp. 551-558.
- [2] A. Colman and J. Han, "Roles, players and adaptable organizations," Applied Ontology, vol. 2, pp. 105-126, 2007.
- [3] M. Kapuruge, A. Colman, and J. Han, "Controlled flexibility in business processes defined for service compositions," in Services Computing (SCC), 2011.
- [4] R. Mietzner, F. Leymann, and M. P. Papazoglou, "Defining Composite Configurable SaaS Application Packages Using SCA, Variability Descriptors and Multi-tenancy Patterns," in Internet and Web Applications and Services(ICIW), 2008, pp. 156-161.
- [5] K. Michiel, et al., "VxBPEL: Supporting variability for Web services in BPEL," Information and Software Technology, vol. 51, pp. 258-269, 2009.
- [6] A. Charfi and M. Mezini, "Hybrid web service composition: business processes meet business rules," in International conference on Service oriented computing, New York, NY, USA, 2004, pp. 30-38.