Policy-driven Business Management over Web Services

Stephen Gorton and Stephan Reiff-Marganiec
Department of Computer Science, University of Leicester, Leicester, UK
{smg24,srm13}@le.ac.uk

Abstract

Service-oriented Architecture allows for reusable services to be composed in such a way that business tasks or activities are easily satisfied. However, currently there is a lack of abstraction past the composition layer, and thus a gap between the service and business domains. We propose a method for depicting end-user business processes, that are further specified and refined by policies. Policies describe information that the end-user can specify, such as requirements, preferences and constraints. The result is a technique for end-users or business analysts to use, rather than software engineers.

1. Introduction

Service-oriented computing (SoC) enables the creation of customised software in a dynamic environment based on reusable services with well-defined interfaces available on the Internet. Services can be composed to create applications, depending on the functionality of services available. Whilst at the moment industrial use of SoC is often on an intra-enterprise level, there is the prospect of many more services being released and the emergence of a competitive service marketplace, shifting the attention from service infrastructure to service management [1].

While services benefit from software engineering advantages, it is our firm belief that their potential can only be reached if the the gap between the business domain and the software domain can be bridged. Our hypothesis is that services become more useful if there is some suitable method, abstract from technological details, to specify requirements which are then used by the system to discover and compose services into an executable application that satisfies the end-user's original goal. Our vision is that a user can enter a list of high-level rules that specify their requirements in the form of process notation and policies.

We present two methods of specification: a process notation specifying what *task* is executed when, and policies to encode further information about the requirements and constraints of the task. We have previously presented an approach for the process notation [3] and for policies [2], here we present the overall framework for specifying business requirements for Web Services.

2. Background

Our motivation for this work is to enable the end-user to have control over their software. This is based on work in telecommunications, where the focus was to give users control over their call settings using policies [6]. Here we also consider business process requirements

Business modelling refers to the specification of business processes and logic by business analysts (rather than software system analysts). In the world of Web Services, there are few options available for expressing these business requirements. Composition technologies such as BPEL can express sequence logic in service usage, but not at the more abstract business level. Graphical notations generally provide an intuitive method to define processes by a sequence of activities or tasks – the most widely-accepted universal process notation for business processes is the Business Process Modelling Notation [5]. BPMN can be used to model a BPEL process [10]. However, BPMN is not meant to capture business rules. We note that BPMN does not support the expression of non-functional business requirements. Business process modelling is often seen as an extension of workflow modelling, therefore some workflow languages are appropriate for modelling business processes. Among several notations, YAWL [9] is a workflow language that extends Petri nets to provide a powerful formal language with defined syntax and semantics.

Notations in general lack the ability to express any information that is not graphical, e.g. "this task should be completed in 2 days" or similar. Therefore we require a particular capability of expressing this further information, down to the sub-process (i.e. task or activity) level.

Policies are "information which can be used to modify the behaviour of a system" [4] without the need for re-compiling or re-deploying. In essence, a system's behaviour reacts to, or is constrained by policies. Policies are loosely coupled with the system they interact with. In order to maintain an effective IT environment, computer systems must adapt to changing business requirements. Policies have proven useful in evolving computing environments. We refine the definition of policy from [7] in the context of our Web Service management system as *"a high level statement as to how business requirements should be processed in the management system"*. Policies can be defined from different viewpoints. For example, an organisation may have general usage policies over software whereas project teams may have further policies specific to them, and even further an individual who has their own policies.

We consider two particular types of policy: Event-Condition-Action (ECA) rules and goals (ECAs without triggering events). Each policy encodes information about a particular business activity. These are well defined in the APPEL policy definition language [8].

Policies may be applied to a variety of problems within SoC. They are currently used for access control and to express reactive functionality. However, they have yet to be applied to a business management framework. Our work is aimed at developing a business policy framework for the management of Web Services in the business domain. The use of policies that we propose is orthogonal to a graphical business modelling language. Each task within a task map represents a unit of business activity that contributes to the satisfaction of the wider business goal. Tasks are subjected to external policy inputs, or global policies.

3. Graphical Notation

Our approach is based on a business goal that is satisfied by a number of objectives, where each objective is fulfilled by a set of tasks performed in a defined order. This process is described using a simple graphical notation. Each task is automatically matched to a Web Service, thus tasks can be defined independent of prior service knowledge. Non-functional attributes, overarching business constraints and business rules are expressed as policies.

While policies can contain rich information, it is generally accepted that graphical notations can be easier to construct, if the syntax is easy enough to use and remember. Thus we use graphical notation to depict the order of execution and transfer of data between tasks, although the latter aspect is not covered in this paper.

Our approach is based on a business goal that is satisfied by a number of objectives; each objective is fulfilled by a set of tasks performed in a defined order. Tasks are automatically matched to Web Services, thus tasks can be defined independent of prior service knowledge. Non-functional attributes, overarching business constraints and business rules are expressed as policies, see section 4.

A service is a computational entity that maps input data to output data, respects certain non-functional properties, might change a world condition and has a compensation action. In Fig. 3 we show how a service maps to a task. However, the mapping of tasks to services is included here for completeness: the business user is not required to be aware of this detail, as they work at the more abstract task and policies level.

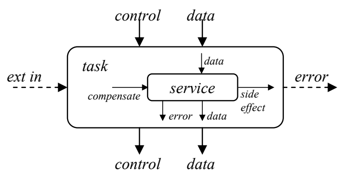
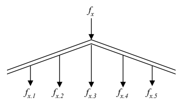


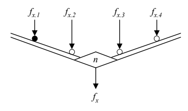
Figure 1. Services map to Tasks.

A task map maps a set of control flows to a set of tasks. Thus it depicts process order of task execution. It is defined by a set of operators, a set of control flows, and a set of data flows. We might also refer to "sub" maps, which essentially are task maps, just that they form parts of other task maps.

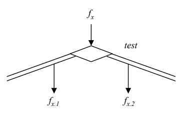
An operator is a function on a control flow, with the ability to split and merge the flow as defined in this section. A control flow is an execution sequence of entities and a data flow is a route of data between tasks. Entities are ordered in the task map showing their relative execution order.



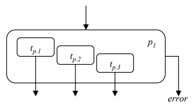
(a) Flow Split.



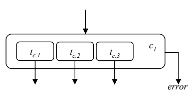
(b) Conditional Merge.



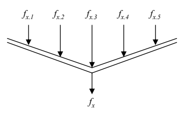
(c) Flow Junction.



(d) Strict Preference.



(e) Random Choice.



(f) Flow Merge.

Figure 2. Operators.

Our notation includes operators (Fig. 2) for splitting control flow over many branches (a), synchronising two or more flows into one (b), diverting control flow down one one branch according to some test (c) and for merging more than one flow where synchronisation is not an issue (f). The strict preference operator (d) attempts tasks in a specific order until one completes. The output flow is dependent on the task that was completed. The random choice operator (e) attempts a set of tasks simulaneously. When one task completes or reaches a commit phase, all other tasks in the set are cancelled. As with strict preference, the output flow is dependent on the task that was completed.

Synchronisation between flows is only an issue when two or more flows are active when they are to be merged. In this case, a conditional merge operator can specify which of the incoming flows are mandatory (i.e. have to complete) or optional, plus a further option to specify how many flows must complete (at least the number of mandatory flows) before the control flow can continue.

4. Policy Framework

Our work is based on the APPEL policy description language [8], which is allows expression of ECA rules and goals. APPEL is a practical and comprehensive policy language for the call control domain [8]. APPEL was chosen because Web Services have many similarities to telecommunications features.

A basic APPEL policy defines a number of policy rules, each of which specifies a set of triggers, a set of conditions and a set of actions. The first two sets may be empty, but the last cannot be, thus APPEL has the ability to express ECA rules and user goals.

The policy itself has an owner and may be applied to a user, a set of users or a domain (identified through email-like addresses). Policies can specify modalities through the preferences **must**, **should** and **prefer**, plus their negations. No specification of preference would indicate that the user was neutral about a subject.

An event can be defined as either a message being passed in a system, a specified event triggers a policy. In the context of Web Service management, we define simple events to be: message events (which occur when a message is sent or received), time events (which can be either absolute, periodic or relative time events), change events (occur when properties are changed either internally or externally), service events (generated before a service is invoked or afterwards) and interaction events (occur when a service operation returns a response). An absolute time event is one where a prescribed time is reached and can be specified with a standard XML timestamp object. A periodic time event represents a regular period of time, with a starting time and either duration or ending time. Relative time events represent an absolute time that can be calculated according to some criteria, usually a start time and duration. Complex events can be built from simple events using event composition primitives (e.g. conjunction), resulting in event (or trigger) groups.

Conditions are boolean values that must equate to `true` for the policy to be triggered. An action is a step towards fulfilling a business goal. In the context of Web Services, an action may include the invocation of a service. Actions may be atomic or composite. Action composition primitives include `and`, `or`, `andthen` and `else` as defined by the APPEL language.

Each task in a task map has requirements attached to it in the form of a policy. Tasks are also subject to external policy inputs, which could influence the behaviour of the task and potentially overrule any originally intended behaviour. We allow external inputs to reflect different stakeholders in a project. For example, a project team might build a software application then submit it to the company's software director for approval. The director may see a function and request that it is subject to change or further constraints. There are similarities here with the Aspect-oriented programming (AoP) paradigm.

In addition to task policies and external inputs, overarching business constraints may be placed over entire task maps through the use of global policies. These policies may include preferences such as preferred suppliers and core non-functional constraints (e.g. "we refuse to use services originating from country X" or "we should not use a service from an untrusted supplier").

Policies and task maps are combined at runtime and dynamically bound to services to take a user to their required applications. At design time, policies are defined along with the task map. We note at this stage that much of the functionality of the runtime engine is hypothetical and assumed, as our main focus is on defining the requirements.

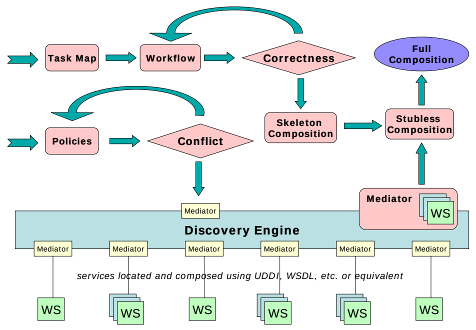


Figure 3. The process execution model

The policies and task map are processed in different ways before being merged to create a full service composition. The task map is parsed and transformed into a workflow document, to enable verifying correctness. Once this has been achieved, the workflow can be transformed into a composition script, such as BPEL, using stubs in the place of service details. At the same time, policies are checked against each other and all policy conflicts will be resolved, either through negotiation or overruling. Policies can then be passed to an intelligent discovery engine that searches for services that satisfy each task. This engine will require some form of semantic mediation between user-defined data types and service-specified data types. With details of each (composite) service, the stubs in the composition can be replaced and a full composition is generated.

Noting that tasks are atomic and should represent a single unit of business activity, we presume that the discovery engine can find a suitable service that will satisfy the requirement. In the event that a service does not exist, the engine may attempt to find or create composite services that can perform the task. Obviously, we must assume that a suitable amount of services already exist to be able to serve our purposes.

5. Simple Example

To provide a simple example, we define a policy that influences the results of a task concerned with service access – in this case that enough credit must be available before the service access is granted (the XML closing tags have been left of for brevity):

```
<policy name="servicecharge" owner="admin@service.com"
        applies_to="everyone">
  <preference>must
  <policy_rule>
    <trigger>access_attempt
    <conditions>
      <condition>
        <parameter>UserType
        <operator>eq
        <value>PrePaid
      <and>
      <condition>
        <parameter>credits
        <operator>gt
        <value>0
    <action>authorise
```

This goal policy essentially states that if the user is a prepay customer he must have credits before access is allowed. Note that the modality is ”must”, therefore the policy must always be enforced.

6. Conclusions and Further Work

We have noted that the modelling notations available for expressing business processes are all lacking in some way. Whilst some have the ability to express non-functional requirements of activities, they lack formalisms. While others have the latter, they lack other capabilities such as non-functional requirements specification.

In this paper we have presented a framework for managing business use of Web Services. This framework includes a graphical notation for depicting when activities, called *tasks*, are executed. These tasks are individually defined and their requirements are expressed in policies. Each task has one main policy attached to it, but may be exposed to overarching business constraints or external task inputs. Policies are written by the end-users with the assistance of a wizard, since we do not expect them to be able to use low-level XML code. This work leverages recent results from the telecommunication domain, where policies are already used to specify rich context information for the end user.

We believe this work is of benefit to the management community because it focusses on making the Service-oriented Architecture (SoA) accessible to businesses by abstracting away implementation details such as composition, messaging and security. Using the presented method, the business value of SoA is increased. While there is already a coupling between Business Process Management and SoA, none have attempted to enable them to work together in such a dynamic way.

Our further work includes defining more formally the structure of policies and task maps. We also plan to look into how policies and task maps interact and work together at design time. Finally, we also plan to define a mapping between task maps and a suitable workflow language, as assumed previously.

Acknowledgements

This work is funded by the IST-FET IST-2005-16004 project SENSORIA (SE for Service-Oriented Overlay Computers).

References

[1] F. Casati, E. Shan, U. Dayal, and M.-C. Shan. Business-oriented management of web services. *CACM*, 46(10):55–60, 2003.

[2] S. Gorton and S. Reiff-Marganiec. Policy support for web service business requirements. In *LA-Web 2006*, pages 199–202. IEEE Proceedings, 2006.

[3] S. Gorton and S. Reiff-Marganiec. Towards a task-oriented, policy-driven business requirements specification for web services. In S. Dustdar, J. Fiadeiro, and A. P. Sheth, editors, *BPM*, LNCS, pages 465–470. Springer, 2006.

[4] E. Lupu and M. Sloman. Conflicts in policy-based distributed systems management. *IEEE Trans. Software Eng.*, 25(6):852–869, 1999.

[5] Object Management Group (OMG). *Business Process Modeling Notation (BPMN) Specification*, Feb 2006.

[6] S. Reiff-Marganiec. Policies: Giving users control over calls. In M. D. Ryan, J.-J. C. Meyer, and H.-D. Ehrich, editors, *Objects, Agents, and Features*, volume 2975 of *LNCS*, pages 189–208. Springer, 2003.

[7] S. Reiff-Marganiec and K. J. Turner. Use of logic to describe enhanced communications services. In D. Peled and M. Y. Vardi, editors, *FORTE*, volume 2529 of *LNC*, pages 130–145. Springer, 2002.

[8] S. Reiff-Marganiec, K. J. Turner, and L. Blair. APPEL: The ACCENT policy environment/language. Technical Report CSM-164, University of Stirling, Jun 2005.

[9] W. M. P. van der Aalst and A. H. M. ter Hofstede. YAWL: yet another workflow language. *Inf. Syst.*, 30(4):245–275, 2005.

[10] S. A. White. Using BPMN to model a BPEL process. *BPTrends*, 2005. http://www.bptrends.com, accessed on 15/03/06.