Towards a Task-Oriented, Policy-Driven Business Requirements Specification for Web Services

Stephen Gorton and Stephan Reiff-Marganiec

Department of Computer Science, University of Leicester
University Road, Leicester LE1 7RH, UK
{smg24, srm13}@le.ac.uk

Abstract. Dynamic assembly of complex software is possible through automated composition of web services. Coordination scripts identify and orchestrate a number of services to fulfil a user or business goal. There exists a need for expressing high level business requirements in such a way that is accessible by businesses. Current solutions fail to include specifications at the appropriate level of abstraction. Our approach defines a graphical notation to depict a business goal in terms of objectives, which are refined by tasks. The specifics of each task as well as overarching business constraints are expressed by policies.

1 Motivation

The advent of Service-oriented Architecture (SoA) makes software "on demand" a distinct possibility. The relatively recent introduction of web services means that automated composition of services can be achieved. Solutions already exist for service discovery and description, though these may be far from complete. Composition solutions also exist, with the Business Process Execution Language (BPEL[1]) the de facto standard.

Attempts to bridge the gap between the business domain and the service domain are often made by expressing business logic through composition or other technologies, but there is a distinct lack of tools which can express precise requirements specifications at the business level. While existing solutions tackle aspects such as functionality and sequencing of business activities, none are complete to encompass all information required at the business level.

The problem that we address in this paper regards business process modelling and analysis, and our goal is to develop a modelling language to accurately express a complete set of business requirements, through the use of policies, in terms of web service usage. One particular aspect is that the notation should be suitable for use by business users (not IT experts) and that it should be simple to use to encourage changes when demand arises.

[1] http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/

2 Background

Service-oriented Architecture, and its implementation as Web Services, make the vision of just-in-time assembly of applications a distinct possibility. SoA refers to a system architecture where a number of independent services can be composed at runtime into larger applications in order to respond to immediate business needs or goals. For details about SoA we refer to Alonso et al. [1]. The automatic composition (i.e. identifying plans for composing services such that they fulfil some desirable goal) and ways for end-users to express these goals (requirements) are two aspects that still must be addressed. In this paper we concentrate on the latter.

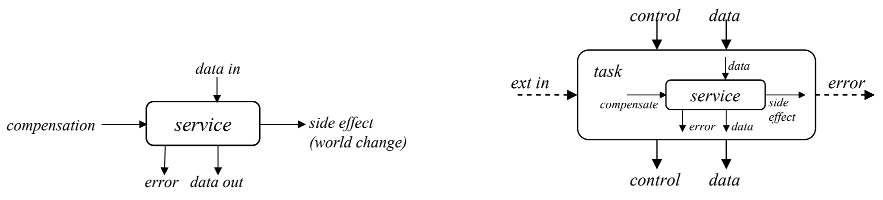
We consider the flow of the business process and the description of the business policies as parts of the requirements specification. Task flow is usually captured in a way that describes the operative nature of the business by using task maps or work flow languages. Task flow is obtained through business modelling as this requires a certain understanding of the business processes involved. Existing composition technologies such as BPEL can express sequence logic in service usage, but they are aimed at the IT level. The Business Process Modeling Notation (BPMN) [2] addresses the problem of expressing business requirements. However, the BPMN specification [2] states that it was "constrained to support only the concepts of modelling that are applicable to business processes", thus not supporting organisational structures and resources, functional breakdowns, data and informational models, strategy and business rules. We believe that BPMN has too many shortcomings to be considered as a complete business solution for expressing business requirements for a web service-based application. In particular, we note that BPMN does not support the expression of non-functional business requirements.

Business policies on the other hand express rules that are of a more generic nature; often they do not apply to a specific business process but rather to specific tasks or the way that the business operates overall. Policies are descriptive in their nature. Policy description languages [3] have been used to express quality of service constraints or access control, that is to describe very low level properties of systems. The Appel policy language [4] has been defined to express end-user rules in telecommunications systems and we are extending this in our ongoing work to interact with the task maps discussed in this paper.

Our approach builds on the conceptual ideas of BPMN by using a simpler graphical notation, but adding policies to express precise business requirements.

3 Overview of Approach

Our graphical notation is intended to act as a modelling agent for businesses who choose to use web services. The process of requirements elicitation begins with the specification of the business goal. This goal is broken down into objectives that are fulfilled by tasks, which represent atomic business activities. The goal is then expressed in terms of a task map and policies. Now there exists an accurate model for the business requirements.



(a) Service as a computational entity (b) Services map to tasks

Fig. 1. Services

The task map (and policies) are read by a parsing engine, which searches Internet directories for web services that satisfy the requirements. Once all services have been located, their descriptions are returned. A coordination engine generates a coordination script according to the descriptions and flows in the task map.

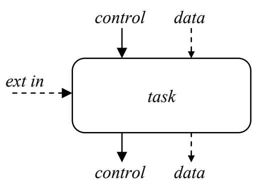
To define the goal, the business must define the objectives that would satisfy it and the tasks required to satisfy each objective, along with the execution sequences of the tasks. A business goal is likely to be defined at a very high level and thus cannot be easily formalised. Functionality is the core requirement for each task. Functionality can usually be more accurately expressed at the atomic task level, whereas non-functional requirements may be expressed at the composite task level, such that they can propagate through to any subtask. Our approach uses policies to encode rules describing the operation of the business as well as the constraints that apply to certain tasks.

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 (e.g. undo as in [5]). In Fig. 1(a), we see how a service is graphically represented, and in Fig. 1(b) how a service maps to a task (or composite task). Note that the service may be of a composite nature (i.e. composed of other services).

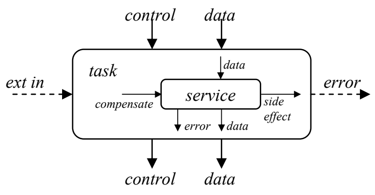
4 Graphical Modelling Constructs

A task is a business activity that contributes to an objective and thus the wider business goal. Each task fulfils a functional requirement. Each task must have a control input and a control output. It has external inputs representing a policy that affects this particular task. Once control has reached a task's input, the task's triggers are activated. On completion of the task, the control leaves through its output channel. Composite tasks are task sub-maps, enabling the designers to separate concerns over aspects of the business goal.

A flow is a sequence of entities (tasks or operators) in the task map and can either be a control flow or a data flow. All tasks inside a task map are subject to policies that are centrally specified by the consuming business or governing law that either restrict the service selection or change the shape of the task map. An example of the former is that a corporate policy may state that the use of

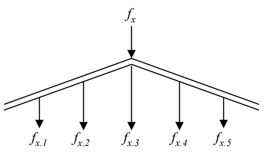


(a) Simple task

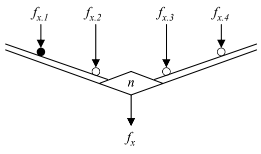


(b) Composite task

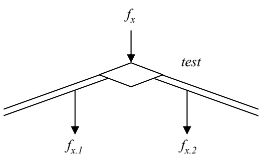
Fig. 2. Tasks



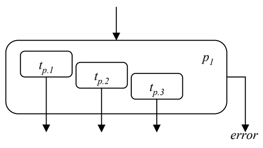
(a) Flow Split



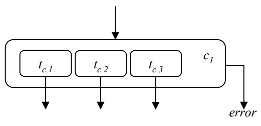
(b) Conditional Merge



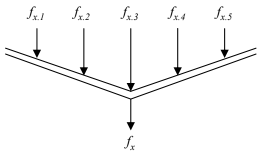
(c) Flow Junction



(d) Strict Preference



(e) Random Choice



(f) Flow Merge

Fig. 3. Operators

a direct competitor's services is forbidden. An example of the latter is a policy that requires the obtaining of at least 3 quotes before a purchase can be made. Note that data and control flow can be independent and we can have partial data flows.

In addition to tasks and flows, which can express simple sequencing, we define operators that are functions on control flows. These further enable a business to accurately model their business goal.

Flow Split. The flow split operator takes a control flow input and produces a set of control flow outputs. In Fig. 3, the operator is pictured with one input and five output flows. When the active control flow reaches the operator, control is distributed amongst the outgoing flows such that each flow progresses simultaneously. For example, in a typical customer-supplier-warehouse example, a product dispatch may involve simultaneously notifying the customer of the dispatch whilst ordering a stock replacement.

Conditional Merge. The conditional merge operator takes a set of active input control flows and, subject to business-defined constraints, merges them with synchronisation to a single output flow. We allow to specify mandatory and optional flows (the filled or empty circles in the graphical notation). Also, the notation allows to specify the number of flows that must reach the operator before proceeding. For example, when looking for airline ticket quotes, one might request quotes from three suppliers, including the preferred supplier. Before booking, we might say that we must have a quote from the preferred supplier, plus one more.

Flow Junction. A flow junction operator diverts the control flow down one of two possible output routes according to a binary test.

Strict Preference. A strict preference operator attempts to execute a series of tasks in a defined order, progressing when one of the tasks is completed. The task with highest priority is attempted first. Each task in the operator specifies its own output flow which is followed when its parent task is completed.

Random Choice. Choice is similar to preference, but without priorities attached to included tasks. When control reaches this operator, all tasks may be attempted simultaneously. When a first task reaches a commit stage all others are cancelled.

Flow Merge. Flow merge is an operator that takes a set of control flow inputs and maps to a single output flow. In order to preserve synchronisation, we say that only one flow of the incoming set must be active, with all others inactive. This may be the result of a prior junction, preference or choice operator.

5 Evaluation

Our approach has been to simplify the requirements specification process for non-IT experts working in the business domain. Despite the existence of other methods, we believe that our method has the following advantages when applied at the business level:

- *Expressiveness:* Our language is able to express as many or as few requirements as is deemed necessary by the business. Task maps are an easy method to understand and, with the aid of a wizard, policies are easy to construct. Despite being at a higher level of abstraction, the task map can be automatically mapped into service coordination scripts. We also include operators in our notation that are non-existent in current notations, e.g. preference, thus increasing the expressiveness for end users.
- *No Binding:* All tasks are expressed without the knowledge of services that are available. The job of matching services to tasks is performed automatically by a search engine, based on ontologies and richer semantic descriptions of web services, which is out of the scope of this paper (there is active research in this area which has led to some preliminary results; most ideas are centred around planning algorithms).
- *Change:* If some aspect of the business goal needs changing to cater for a new or changed business requirement, it can be done with relative ease by altering the task map or underlying policies. The service coordination script is generated automatically, which is subject to any changes made to the specification.
- *Technology Compatibility:* Though not an immediate aspect of business versatility, our solution is able to take advantage of current solutions that exist, e.g. BPEL as the coordination script. In this respect, a business always has the option of altering their executable coordination script before proceeding.

- *Composition Views:* We add that our solution can generate different views that are customized to different stakeholders. In particular, a project manager may be more interested in (composite) task requirements whereas the IT director may be more interested in the global or business-wide constraints. Further low level views include control flow views and data flow views.
- *Workflows:* Our notation is able to support many of the workflow patterns as described in [6].

The conciseness of this paper does not allow to present details on the issues of cancellation, negotiation and how standard workflow patterns are supported.

6 Conclusions and Further Work

We have presented a notation for describing business requirements at an abstract level. A business goal is defined in terms of objectives which are further refined by tasks. Tasks are organised into a task map. Policies define complete requirements and specifications for tasks, and are more generic in that they can be used throughout the task map, providing information to many parts of a business goal, and even across multiple goals. We firmly believe that this solution is able to fill the gap between service levels and business levels.

Our further work includes refinement of the ideas presented on policies, based on the APPEL policy language [4]. We also propose that a workbench be designed to enable designing of task maps and policies through the use of a graphical user interface.

Acknowledgements

This work is funded by the IST-FET IST-2005-16004 project SENSORIA (Software Engineering for Service-Oriented Overlay Computers). Further thanks to Marie-Claude Gaudel for her advice on cancellation and undoing.

References

1. Alonso, G., Casati, F., Kuno, H., Machiraiu, V.: Web Services: Concepts, Architectures and Applications. Springer (2004)
2. Object Management Group (OMG): Business Process Modeling Notation (BPMN) Specification. (2006)
3. Lupu, E., Sloman, M.: Conflicts in policy-based distributed systems management. IEEE Trans. Software Eng. **25**(6) (1999) 852–869
4. Turner, K.J., Reiff-Marganiec, S., Blair, L., Pang, J., Gray, T., Perry, P., Ireland, J.: Policy support for call control. Computer Standards and Interfaces (2005)
5. Gaudel, M.C.: Toward undoing in composite web services. LRI, Paris-Sud University and CNRS, Orsay, France (2004)
6. van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow patterns. Technical Report FIT-TR-2002-03, Queensland University of Technology, Brisbane (2002)