

# Automated Policy-Refinement for Managing Composite Services

Kevin Carey, Dave Lewis, Vincent Wade  
Knowledge and Data Engineering Group, Computer Science, Trinity College Dublin  
{Kevin.Carey, Dave.Lewis, Vincent.Wade}@cs.tcd.ie

## Introduction

From e-commerce to ubiquitous computing, service composition is seen as a key technique in rapidly generating new, tailored functionality from existing service implementations. However, within a specific composite service deployment there may still be the opportunity for further run-time adaptation of the composite service's behaviour based on run-time adaptively built into the software components that implement. This may, for instance, cover non-functional behaviour such as quality of service. In this paper it is assumed that a implementer of service components provide runtime adaptively of service via policy rules that can be triggered by runtime events and which, given certain specified condition are satisfied, will cause specified actions to be taken by the component. Policy rules are declarative and can be loaded into a component at deployment or runtime. However, rules for specific service components will use terminology native to that service and so coordinating the aggregate adaptive behaviour of a composite service though manipulating the heterogeneous policy rules of the components implementing its constituent service is potentially complex. This paper describes an approach to automatically managing constituent service policy-rules of a composite service based. This management allows an administrator or a user to assign a goal, in the form of a high-level policy with event, condition and action, for the composite service and ensures that this goal is propagated to the constituent services by refining this goal into policies which are executed upon the service implementation. The approach taken exposes a service implementation's management controls as a finite state machine (FSM).

## Background

This section described some of the technologies and techniques underlying our approach.

### Web Services and Composite Services

Web services are self-contained, self-describing, modular applications that can be published, located, and invoked across the web. Web services perform functions, which can be anything from simple requests to complicated business processes [20].

Software functionality can be modelled as a well-defined service, which is the means by which their functionality is composed to define more complex services. Such a composition of services provides the well-defined context required for addressing adaptive behaviour, especially of non-functional behaviour such as quality of service, in complex, multi-organizational environments.

Service composition is the orchestration of a number of existing services to provide a richer composite service assembled to meet some user requirement in the form of a new service. A *composite service* is made up of several *constituent services* that are invoked according to some process flow model – service sequence description. Constituent services may be either *atomic services*, which cannot be subdivided, or other composite services.

There are several languages that take a service-oriented approach to integration. WSDL [32] is the primary one used for web service, enabling the definition of service operations and their input and output parameters. OWL-S [36][37] uses ontology based semantics to enhance web service descriptions with richer definition of inputs and outputs as well as of preconditions and effects of operations.

### **Finite State Machines**

Finite State Machines show all the possible states that a system or component can have, and which events can cause the state to change. An event can be another component that sends a message to it – for example, that a specified time has elapsed – or that some condition has being fulfilled. A change of state is called a transition. A transition can also have an action connected to it that specifies what should be done in connection with the state transition.

### **Policy Refinement and Policy-based Management Systems**

As the complexity of web services continues to increase the complexity of their management systems, including those for QoS management, must also increase. The management of such complex and distributed systems presents a number of significant challenges. Most notable among these challenges is how to ensure that the operation of the system matches the overall objectives of those using it. It was to address this, and other, problems that *Policy Based Management Systems* (PBMS) were developed [15] [16] [17].

Within a PBMS the desired behaviour of the managed system is specified in a policy. A policy may be defined as a definite goal or method of action to guide present and future decisions [18]. But in simple terms, it can be defined as a set of rules that express and reach a desired behaviour [19]. Policies can be specified at many different levels of abstraction from high-level business goals to policies for individual resources. Ideally, it should be possible for an organization to derive their system policies (low level policies) from their overall business goals (high level policies). This can be achieved through Policy Refinement, which is defined as the decomposition of policies relevant to a composite system into a set of policies that are executed in its constituent parts to implement the behaviour indented by the overall system level policy. Thus, in the context of service composition, policy refinement is capable of mapping high level goals of a composite service automatically down to low level policies which interact with the service and previously existing constituent services.

### **Architecture**

*In this section, an architecture for the management of composite services is described. This framework uses a policy-based approach for supporting the management of service behaviour of a composite service. The framework is composed of a policy refinement engine, a policy execution engine and a state machine. Furthermore, it works along side a workflow style composite service execution engine.*

### **Service Composition**

Our architecture assumes that the execution of the service composition is performed by a workflow engine. The workflow engine uses a sequence description for a

composite service to create a script with rules that activates the services according to the sequence description given.

### **Describing the Service to be Managed**

Standardized service description languages, such as WSDL and DAMLS, are very proficient at describing the functional aspects of a service. But, these languages have not addressed describing the non-functional aspect of the service. The service management controls are typically classified as the non-functional aspect of the service. With a view to exposing the service's non-functional aspect in an abstract method, it is proposed for the use of finite state machine models. Service implementers use FSM to expose the, largely non-functional, elements of the implementation's behaviour that are intended to be managed via policy rules.

The states of a FSM represent the non-functional aspect configurations of the service it describes. A transition between two states in a FSM represents an action needed to be performed in the service to move from one state to the other. For example, actions may be to log an event or to indicate that the temperature has reached a threshold.

In the case of a composite service, the service's non-functional aspects should be described in terms the non-functional aspects of its constituent services. This is achieved by describing each desired non-functional aspect of the composite service as a FSM model and providing a description of how the states in this FSM model are related to the states in the constituent service's FSM model. The relationship information between the composite service's states and the constituent service's states is referred to as *state relationship information*. It is expected that state relationship information will be produced automatically or semi-automatically from the service composition model and constituent service FSMs. Currently, our experiment require the state relationship information to be generated by the designer of the service composition model by hand.

Consequently, the management controls for the composite service are exposed according to the service composer wishes and without contradicting the service developer's intentions for managing its service.

### **Managing the Service using Policies**

Policy is ideal for managing the services by the use of the FSMs, where it can monitor for a specific state and when this state is reached, it can perform the necessary action to reach the desired state.

In order to manage the composite service, the architecture has a state machine which keeps track of the states for the composite service's constituent services. First, prior to the execution of the composite service, the state machine loads the FSM models for each of the constituent services together with a default state for each of the FSMs. During the composite service execution, the state machine changes the current state of a FSM when an action for that state transition occurs on an active service. Policy comes into play when it sees a state it is expecting.

A solution to this is the use of policy refinement for the composite service. Thus, the architecture has a policy refinement engine that takes a high level policy, a goal, and refines it into low level policies, service policies, by way of intermediate policies.

### **Policy Refinement**

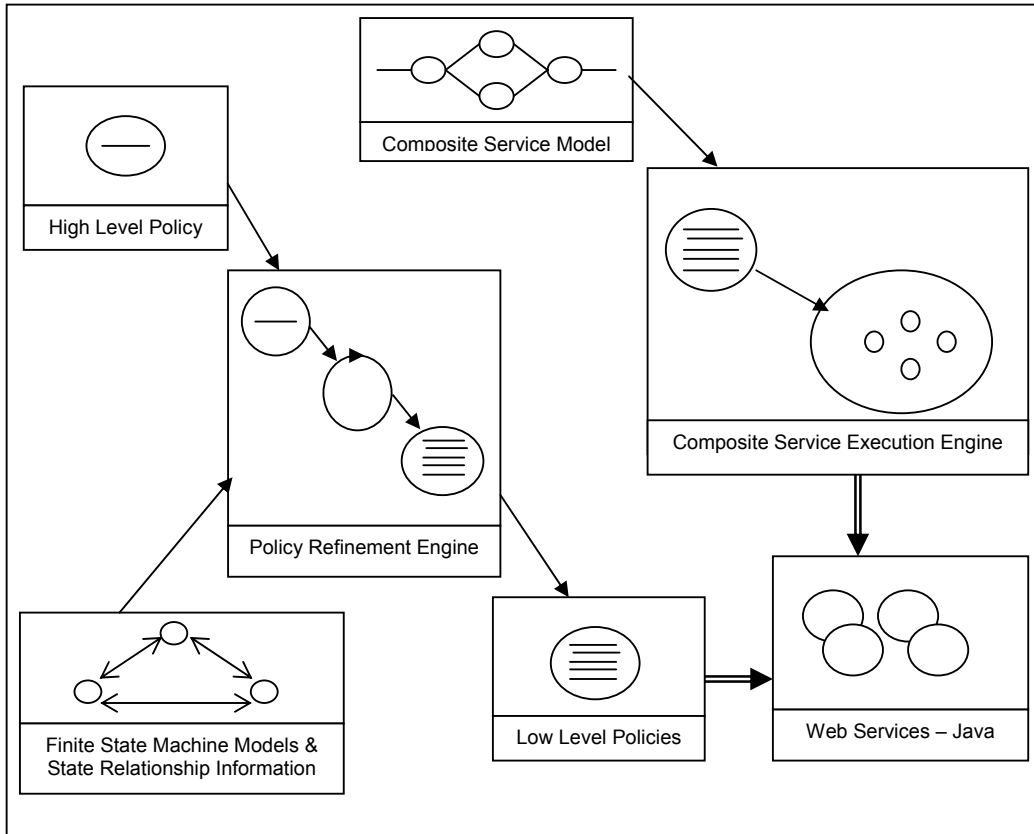
A goal, a high level policy with event, condition and action, expressed only with the states defined in the FSM for the composite service, when applied to the framework along with a composite service, it is refined by the framework into intermediate policies and subsequently into service policies. This occurs prior to the execution of the composite service.

The refinement of a high level policy to an intermediate policy is achieved with the aid of the composite service map, which describes how the states in the composite service's FSM are mapped to states of its constituent services. The high level policy is therefore refined into an intermediate policy, an aggregate policy expressed in terms of states from constituent services. The aggregate intermediate policy can have several events, conditions and action. By supporting aggregate intermediate policy and using state relationship information, it simplifies and speeds up the refinement process of the high level policy.

The transformation of an aggregate intermediate policy into service policy is more complex, it entails the creation of a service policy, a low level policy, for each of the intermediate's event, condition and action. The refinement of an aggregate intermediate policy also generates a service meta-policy. The service meta-policy is the backbone of the service policies, it ties them together.

For each condition of the intermediate policy, a service policy is generated that is triggered when the state specified by the intermediate condition is encountered, and as its action, it sends a notification to a service meta-policy that this condition has being reached. The same applies to the intermediate policy's events. Now, for each action of the intermediate policy, a service policy is generated that is triggered when it receives a notification from a service meta-policy and as its action, it performs the action needed to attain the desired state specified by the intermediate policy action. The generated service meta-policy for the aggregate intermediate policy expects notifications from all the 'condition' service policy and 'event' service policy as its condition and as its action it sends notifications to all the 'action' service policies.

The framework executes the service policies upon the constituent services during the execution of a composite service. Be aware of the fact that the action of an 'action' service policy will only take place when the service in which this action belongs to is active.



**Figure 1: Architecture model of policy-refinement for managing composite services**

## Implementation

### Overview

The framework was implemented according to the outlined architecture. The framework comprises of a policy refinement and execution engine accompanied by a composite service execution engine. These engines are implemented in a rule-based system style using Jess [21], where the services are written in Java.

### Composite service execution engine

The composite service execution engine is a simplified workflow engine for the purpose of executing a composite service. The composite service execution engine needs the sequence description for the composite service is about to execute, as well as, the location and the IOPE (Input, Output, Precondition and Effect) description for each constituent service. In this version of the engine, the service description is defined in a JESS template, where the intention for the final version is to retrieve the service description by parsing the service's WSDL file.

The composite service execution engine is implemented using Jess rules. The main Jess rule for the engine is triggered when the engine takes delivery of a sequence description for the composite service. When this rule is triggered, it generates workflow rules, expressed as JESS rules, for the execution of the composite service.

Once it is ready to execute the composite service, it triggers the workflow rule which starts the execution of the composite service by calling the first constituent service according to the composite service sequence. When the execution of this service is

completed it triggers the appropriate workflow rule which calls the next service until the execution of the composite service is complete. This composite service execution engine supports a composite service with sequential and even concurrent sequences, but not decision branching.

### **Policy Refinement and Execution Engine**

The policy engine is implemented in the same manner as the composite service execution engine, using a rule-based style with Jess rules. These rules work in combination to refine a goal into service policies. In order for the policy engine to execute, it requires information such as the service's name and its FSM models for each of the constituent service. It additionally needs the FSM for the composite service, as well as, the state relationship information which maps the states used in the composite service's FSM to the relevant states defined in the constituent services' FSMs.

A goal, a high level policy, is refined by a Jess rule which is triggered when the states used in a high level policy matches the states from the composite service's FSMs. This Jess rule creates an aggregate intermediate policy which uses the relevant states from the constituent services' FSMs according to the state relationship information. Aggregate intermediate policies are still an abstract policy and cannot be executed upon a service. It needs to be further refined into service policies which can be executed upon the constituent services.

An aggregate intermediate policy is also refined by Jess rules. Firstly, an aggregate intermediate policy is broken into parts, one for each event, condition and action the aggregate intermediate policy contains. These policy parts are handled by the other Jess rules, which are designed to generate a service policy for each policy part. A policy part containing the information for an event of an intermediate policy is used to create an *event service policy*, an executable policy which handles a single event. A *condition service policy*, an executable policy which handles a single condition, is created from a policy part containing the information for a condition of an intermediate policy. And a policy part containing the information for an action of an intermediate policy is utilised to generate an *action service policy*, an executable policy which handles a single action. In addition to these policies, a *service meta-policy* is also generated from an intermediate policy. A *service meta-policy* ties *event service policies*, *condition service policies* and *action service policies* together.

After an expected event occurs, the relevant event service policy sends a notification to the appropriate service meta-policy. When a nominated condition is met, the appropriate condition service policy sends a notification to the appropriate service meta-policy. Once the service meta-policy has received all the expected event and condition notifications, it sends notifications to all the pertinent action service policies. As soon as an action service policy receives a notification, it performs the instructed action by way of the state machine if the service in which this action belongs to is active. The name of the high level policy, in which these service policies derived from, is used as an identifier for the notifications between these service policies.

A state machine is implemented to support the execution of service policies. Firstly, the state machine registers the default state for each of the FSM from all the

constituent services. When an action service policy is triggered, it notifies the state machine to perform a specific action on a service. The instructed action is only performed by the state machine on the service if this action can be performed by the service in the current state. After performing the instructed action, the state machine updates its relevant state.

### **Results & Conclusion**

This paper shows how a composite service can be managed in an automated fashion with the use of a policy refinement approach. The results of the implemented architecture shows that a goal assigned to a composite service is refined into policies which enacts on the constituent service implementations to assure that each service is managed according to the management goal bestowed on the composite service.

It is shown how policy refinement is achieved for a composite service with the use of FSM models and state relationship information, and how FSM models and state relationship information exposes the management controls of the services without contradicting the service component developer's intention.

For further work, it is intended to automate the process of creating state relationship information possibly with the assistance of semantic descriptions to aid in the automated mapping. Furthermore, a policy confliction detection engine would be required to ensure that there are not any conflicts between the low level policies.

### **References**

- [15] Lutfiyya, H. L., Bauer, M. A., Marshall, A. D., and Stokes, D. K. A Policy-Driven Approach to Availability and Performance Management in Distributed Systems. 27-8-1997.
- [16] Mont, M. C., Bladwin, A., and Goh, C. POWER Prototype: Towards Integrated Policy-Based Management. HPL-1999-126. 18-10-1999.
- [17] Wies, R., "Policies in Network and Systems Management - Formal Definition and Architecture," *Journal of Networks and Systems Management*, vol. 2, no. 1, pp. 63-83, Mar.1994.
- [18] Alcantara, O. D. and Sloman, M. QoS Policy Specification A mapping from Ponder to the IETF. 2003.
- [19] Damianou, N., Bandara, A. K., Sloman, M., and Lupu, E. C. A Survey of Policy Specification Approaches 2002.
- [20] [www.ibm.com](http://www.ibm.com).
- [21] <http://herzberg.ca.sandia.gov/jess/>.
- [22] Alaettinoglu, C., Villamizar, C., Gerich, E., Kessens, D., Meyer, D., Bates, T., Karrenberg, D., and Terpstra, M. Routing Policy Specification Language (RPSL). 1999.
- [32] Christensen, E., Curbera, F., Meredith, G., and Weerawarana, S. Web Services Description Language (WSDL) 1.1. 15-3-2001. W3C.
- [36] DAML-S. DAML-S: Semantic Markup for Web Services. 2002.
- [37] McIlraith, S. A., Son, T. C., and Honglei Zeng, H. Semantic Web Services. 2001. IEEE Intelligent Systems.