Adaptive Composite Service Plans for Ubiquitous Computing

Kevin Carey, Dave Lewis, Steffen Higel, Vincent Wade

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

Abstract: Ubiquitous Computing environments require dynamic adaptable application and network services because of their need to adapt to rapidly change context e.g. new users and devices being introduced, changes in connectivity, changes in user needs and requirements, etc. To support such environments, management systems need to dynamically adapt service execution and behaviour. One of the approaches being suggested currently is that of AI planning, where execution plans (sequences of service executions) can be generated dynamically. However, they often have being criticised for being inflexible since it only adapts to changes in context through re-planning. This paper explores an approach to make service plans more adaptable to changing context prior to as well as during execution time without re-planning. In our approach the service's non-functional adaptive behaviour is explicitly documented by a service developer in a run-time accessible model. This paper proposes the use of Finite State Machines (FSM) to describe the adaptive behaviour of a particular service and the formation of a composite FSM representing the adaptive behaviour of the constituent services. The paper then proposes policies to be used as rules by which adaptive behaviours can be specified prior to, or even during, execution. For a composite service, these policies are refined and applied to its constituent services, thus facilitating automatic refinement of high level policies to lower level policies executable appropriate constituent service. The paper presents a case study to illustrate the outlined approach and identify the strengths and challenges in applying this approach.

1. Introduction

Ubiquitous computing (ubicomp) environments aim to offer services of computer applications, embedded software and networked services in a highly flexible but integrated manner to their users [weiser91]. This requires adaptive services that modify their behaviour to the current task requirements of the user and adapt to the context in which the required service is executed. In addition however, such adaptive services must have their adaptive behaviour governed and constrained by (a) overriding user preferences and (b) policies of those responsible for the resources used.

Adaptiveness can be brought about by recombining existing application services in different combinations - a process called service composition. Service composition

links services in control and data flows using expressive techniques from workflow management and from the area of web service composition. However, in ubicomp service composition needs to be performed automatically in real-time. This presents a challenge for techniques which typically rely on humans for the design of composite services.

Artificial Intelligence (AI) planning is one technique that is receiving increased attention as a solution for automated service composition. AI planning techniques can automatically generate composite service plans consisting of simple set of actions. Each action can be supported by service invocations, given a set of required goals, a set of possible actions and a description of the initial state of the system. Current AI planning techniques uses service's inputs, outputs, preconditions and effects (IOPE) to dynamically generate composite service plans [sirin03].

However, AI Planners that separate planning and execution cannot devise plans that adapt to changes in state only knowable at runtime, e.g. user input, system events. Even AI planners that integrate planning and execution require continuous monitoring of composite service execution [ranaganathan04], limiting their applicability to ubiquitous computing environments. Consequently, in order for a particular service to react to context changes re-planning is required.

This paper explores an approach to make composite service plans more adaptable to service changing context prior or even during run-time without re-planning. With the intention of making the composite service plans more adaptable, the services used in these plans must contain adaptivity built in. In order for the adaptivity to be used the adaptivity is described in the form of FSM. The configuration of these FSMs i.e. the configuration of the service's adaptivity can be defined by the user prior or at run-time using policies. Thus, this paper describes an approach that allows adaptable services to be composed by an AI planner but which the user can then configure to obtain the adaptivity they want within the services within that plan.

This paper describes how FSM are used to describe the service's adaptivity and combined for a composite service in section 2.1, and how the policies for a composite service are refined and applied to its constituent services' FSM in section 2.2. In addition, it presents a case study in section 2.3 and implementation in section 2.4 to illustrate the outlined approach and identify the strengths and challenges in applying this approach. Experiment findings are shown in section 3 followed by conclusion and further work in section 4.

2. Policy-driven adaptive plans

Web Service Definition Language (WSDL) is a standardized service description language, which is very proficient at describing the functional aspects of services, enabling the definition of service operations and their input and output parameters. However, a richer semantic language is needed in order to reason about services that must be composed dynamically. OWL-based Web Service Ontology (OWL-S) [owls02] uses ontology based semantics to enhance web service descriptions. It aims to provide an unambiguous, computer-interpretable form, rich definitions of IOPE of

operations, as well as a rich set of control constructs for linking constituent services. Thus, OWL-S makes a good candidate for describing composite services.

2.1 Non-functional adaptive behaviour description – the Finite State Machine

Even though OWL-S and WSDL take a service-oriented approach, they have not shown how the non-functional adaptive behaviour of a service could be described. In particular, adaptive behaviour that is not captured through the definition of inputs and outputs are difficult to capture in OWL-S and WSDL. Service management typically governs such non-functional aspects of services (e.g. reliability logging, video resolution). The approach proposed in this paper uses Finite State Machine (FSM) models as a service management interface to expose the service's non-functional adaptive behaviour in a controlled abstract manner.

A (FSM) model shows possible *states* [uml03] that a system or component can have, and which *events* [uml03] can cause the state to change. Events can represent timers, counters, aspects of service invocation (e.g. service invoked, response sent), or changes in the context in which a service is operating. A change of state is called a transition. A *transition* [uml03] can also have an action connected to it that specifies what should be done in connection with the state transition.

In our approach we suggest that FSM models are used by the service implementers to expose the service's non-functional adaptive behaviour that is intended to be managed via adaptive behaviour rules at run-time. The generic FSM model for a request-respond atomic service is used that is composed of four states – IdleState, InputState, ProcessState and OutputState. These states are composite states that can contain several sub-state-machines that represent additional service behaviour that can be performed in that state. In addition to sub-state-machines, service implementers can also add events which could trigger a transition that would perform one of those sub-state-machines. However, typically in defining adaptive behaviour, these transitions are omitted by the implementer, but are made available as the actions of adaptive behaviour rules that can be bound to the service at runtime.

The FSM model for a composite service is not created by the service's implementer, but composed from the FSM models of the constituent services based on the service composition model. The FSM model for a request-respond composite service is also composed of 4 composite states with sub-state-machines and events. However the sub-state-machines and events are derived form the constituent services' FSM models and combined based on the service's service composition model. Figure 1 illustrates this composition by means of an example. In the example shown, the composite service composed of a sequence of 3 atomic services, where each atomic service has 2 sub-state-machines, one under InputState and the other under OutputState. The composite service's FSM will contain a sub-state-machine under InputState which belongs to the first atomic service's InputState, a sub-state-machine under OutputState which belongs to the last atomic service's OutputState and the other sub-state-machines will be located under ProcessState of the composite service's FSM model.

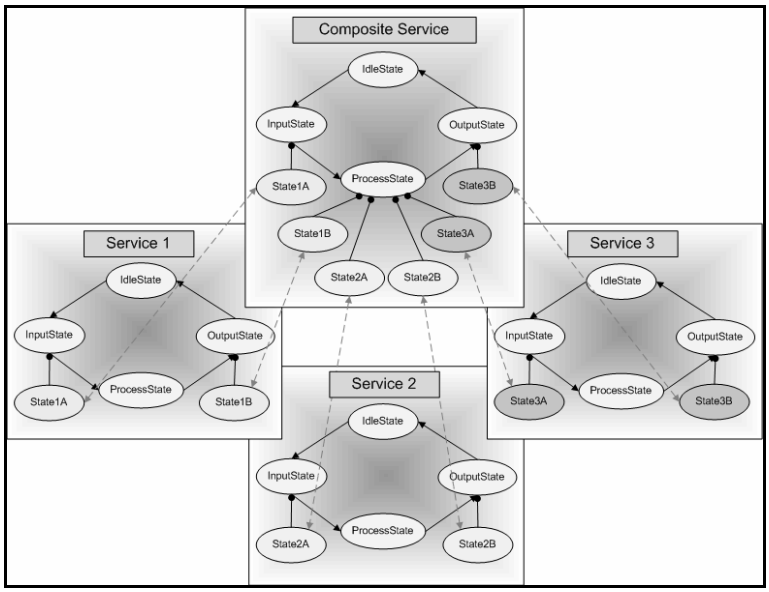

Fig. 1. An example of show a composition of Finite State Machines

2.2 Non-functional adaptive behaviour implementation

As the complexity of web services continues to increase the complexity, of their management systems, e.g. 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. Policy Based Management Systems (PBMS) were developed to address this, and other, problems [Lutfiyya97] [mont99] [wies94].

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 decisions made within the system [alcantara03]. Put in simple terms, it can be defined as a set of rules that express and reach a desired behaviour [damianou01]. Policies can be specified at many different levels of abstraction from high-level business goals to policies enforced upon 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 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 intended by the overall system level policy. Thus, in the context of service composition, policy refinement is capable of mapping high level goals (human generated policies) for a composite service automatically down to low level policies (adaptive service behaviour rules) which interact with the individual constituent services.

In our approach once a FSM model is created for a composite service, adaptive behaviour rules (high level policies) can be added to the composite service as state-machine's transitions of its FSM, where the rule's action must match one of the sub-state-machines offered by the composite service's FSM model. In addition, the rule's event must be one of the events belonging to the composite service's FSM model. Lastly, the rule's conditions must only contain parameters belonging to composite service. Thus, only adaptive behaviour rules that don't contradict the constituent service developer's intentions and the service composer wishes can be applied to a composite service.

These composite service's rules are automatically refined into low level policy rules expressed as FSM transitions, in which are applied to the appropriate constituent service. This is achieved by the decomposition of the combined composite service's FSM model together with new the transitions i.e. composite service's policy rules, into constituent service's FSM models; each containing some of the composite service's transitions that can only affect this constituent service i.e. adaptive service behaviour rules (low level policies). An example is shown in case study section.

The approach described in this paper only allows the use of composite service user's policies (adaptive behaviour rules) that are expressed with the exposed composite service non-functional adaptive behaviours as FSM models. These models are modelled according to its constituent services' defined non-functional adaptive behaviours. Consequently, these policies are restricted to the constituent service developer's intentions for managing its service and they are refined via the decomposition of the FSM model.

2.3 Case Study

A model of a transport billing service is used as a case study to demonstrate the value of the outlined approach. The case study is of a public transport fitted with a ubiquitous device operating a service which has the ability to charge a user for their journey without any physical interaction from the user. Instead, a user's ubiquitous device automatically interacts with the transport and pays for the journey. The transport billing service is to be used in both trains and busses. The difference between trains and busses is that the train bills the user for the number of stations travelled and the bus bills the user for the duration of the journey. Another criterion for this service is the need to accommodate normal, student and child fares.

While this can be solved with multiple composite services or a composite service with complex composition, another approach is to use a composite service with a simple composition of adaptive and context aware services together with adaptive behaviour rules whose purpose is to govern the service's flexibility. Following this approach, the transport billing service is composed of a sequence of three atomic services:

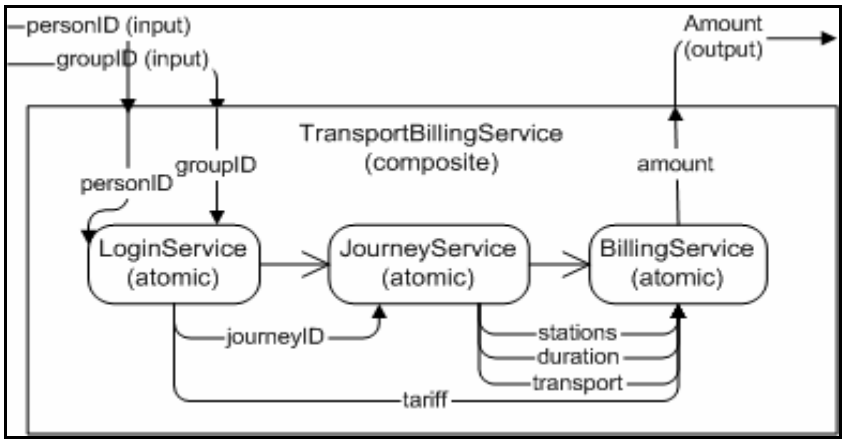

Fig. 2. Composite service and its constituent services

Using the service finite state machine composer tool (see implementation section), FSMs were added to each of the services together with events and sub-machine-states for the relevant services as shown in figure 3.

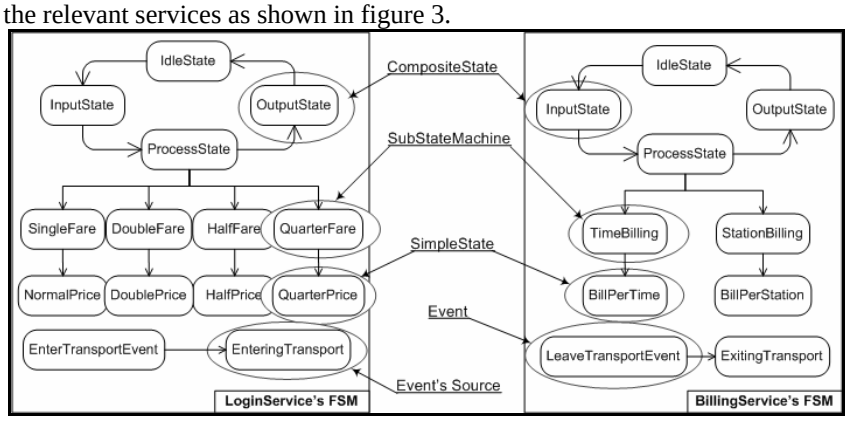

Fig. 3. Sub-State-Machines and Event for the relevant constituent services

Having added a FSM for each of the constituent services, a FSM is created for the composite service by combining the FSMs of its constituent services into a single FSM. Subsequently, the following adaptive behaviour rules were added to the composite service's FSM:

- Rule1: "When a user is leaving, bill the user per trip duration for buses" (event LeaveTransportEvent, condition transport=bus, action TimeBilling);
- Rule2: "When a user is leaving, bill the user per station for trains" (event LeaveTransportEvent, condition transport=train, action StationBilling);
- Rule3: "When an adult boards, set the fare's price to a single fare" (event EnterTransportEvent, condition GroupID=Adult, action SingleFare);
- Rule4: "When a student boards, set the fare's price to a half of a normal fare" (event EnterTransportEvent, condition GroupID=Student, action HalfFare);

- Rule5: “When a child boards, set the fare’s price to a quarter of a normal fare” (event EnterTransportEvent, condition GroupID=Child, action QuarterFare);

The adaptive behaviour rules are transformed into transitions and saved into the composite service’s FSM model. When the composite service’s FSM model is refined, these new transitions are propagated down to the relevant constituent service’s FSM in which it can then be used to influence the service’s behaviour at runtime. In this case, the adaptive behaviour rules 1 and 2 become transitions that will govern the behaviour of BillingService. Where the adaptive behaviour rules 3, 4, and 5 become transitions that will govern the behaviour of LoginService.

2.4 Implementation

In the implementation of the outlined approach, a service ontology model and a finite state machine ontology model were used. The service ontology (OWL-DL-compliant) model was created by [owls02], whereas the finite state machine ontology model was derived by the authors from the UML finite state machine model [uml03].

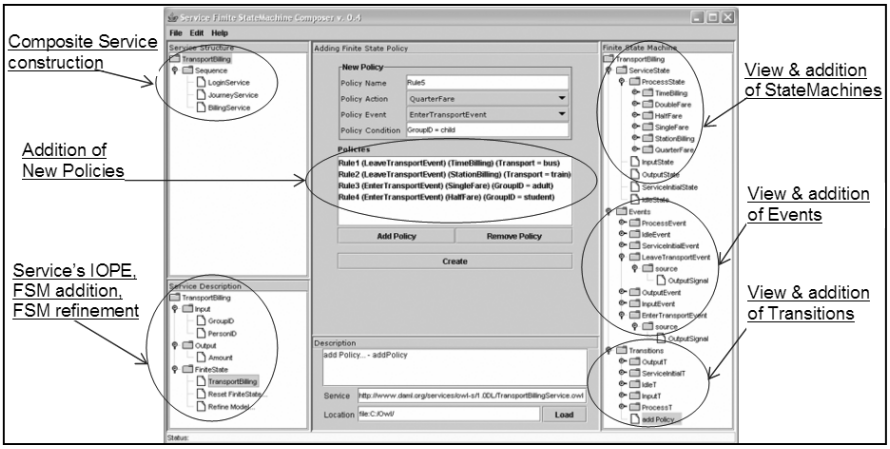

Fig. 4. Screenshot of Finite State Machine Composer tool

A tool (shown in Figure 4) was implemented to facilitate the creation of FSM-based adaptive behaviour model for atomic services, and subsequently to use these in the generation of a composite FSM for a composite service as outlined in the architecture. This tool was written in Java and used Jena [carroll04], a Semantic Web Framework for Java, to reason about the relevant ontology instance models. It also provides a GUI for a user which facilitates the user in defining ontology based FSM models for existing OWL-S services. To support the development of a constituent service, an FSM can be added to any OWL-S atomic service and then the relevant sub-state-machines and events can be added to this FSM. In supporting a composite service user, the tool reads a composite service ontology instance model from a file and through its GUI allows a user to see the composite service structure and details. When adding a FSM to a composite service, this FSM will become a combination of

the sub-state-machines of its constituent services. This tool only allows a user to add policies – adaptive behaviour rules – to a FSM for a composite service and the ability to refine these policies to its constituent services. Below is a snippet of the OWL generated for the FSM of BillingService:

```
<fsm:SignalEvent rdf:ID="LeaveTransportEvent">
 <fsm:signal>
  <fsm:Signal rdf:ID="ExitingTransport"/>
 </fsm:signal>
</fsm:SignalEvent>
<fsm:CompositeState rdf:ID="ProcessState">
 <fsm:subvertex>
  <fsm:SubmachineState rdf:ID="StationBilling_substate">
   <fsm:submachine>
    <fsm:StateMachine rdf:ID="StationBilling_smachine">
     <fsm:submachineOf rdf:resource=
      "#StationBilling_substate"/>
     <fsm:top>
      <fsm:CompositeState rdf:ID="StationBilling">
       <fsm:topOf rdf:resource=
        "#StationBilling_smachine"/>
       <fsm:subvertex>
        <fsm:SimpleState rdf:ID="BillPerStation">
         <fsm:container rdf:resource="#StationBilling"/>
        </fsm:SimpleState>
       </fsm:subvertex>
      </fsm:CompositeState>
     </fsm:top>
    </fsm:StateMachine>
   </fsm:submachine>
   <fsm:container rdf:resource="#ProcessState"/>
  </fsm:SubmachineState>
 </fsm:subvertex>
</fsm:CompositeState>
```

A support service composer tool (similar look and field as Figure 4) was developed to allow a user to describe a composite service detailing its composition structure and its IOPE as well as of its constituent services. The service composer tool helps the user in visualising the service that is being composed and ensures the user creates appropriate models. It also allows a user to save the ontology description for this particular service into an OWL file and load it at a later stage. Below is part of the (process) OWL file generated for TransportBilling Service:

```
<process:CompositeProcess rdf:ID="TransportBilling">
 <process:hasInput rdf:resource="#GroupID"/>
 <process:hasInput rdf:resource="#PersonID"/>
 <process:composedOf>
  <process:Sequence>
   <process:components>
    <process:ProcessComponentList>
     <rdf:rest>
      <process:ProcessComponentList>
       <rdf:rest>
        <process:ProcessComponentList>
```

```
          <rdf:first rdf:resource="#BillingService"/>
          <rdf:rest rdf:resource="http://www.w3.org/
           1999/02/22-rdf-syntax-ns#nil"/>
         </process:ProcessComponentList>
        </rdf:rest>
        <rdf:first rdf:resource="#JourneyService"/>
       </process:ProcessComponentList>
      </rdf:rest>
      <rdf:first>
       <process:AtomicProcess rdf:ID="LoginService">
        <process:hasOutput rdf:resource="#JourneyID"/>
        <process:hasInput rdf:resource="#PersonID"/>
        <process:hasInput rdf:resource="#GroupID"/>
        <process:hasOutput rdf:resource="#Tariff"/>
       </process:AtomicProcess>
      </rdf:first>
     </process:ProcessComponentList>
    </process:components>
   </process:Sequence>
  </process:composedOf>
  <process:hasOutput rdf:resource="#Amount"/>
</process:CompositeProcess>
```

3. Experiment findings

While current AI planning techniques can dynamically generate composite service plans, it was observed that in terms of adaptivity support, they can only provide support for service adaptivity by re-planning the composite service plan [sirin03]. Although this might not be a problem in some circumstances, it poses a problem for composite services in ubiquitous computing environment.

The proposed approach supports adaptivity without the need for re-planning. It exposes the service's adaptivity potential to the user through FSM for an atomic service as well as for a composite service. The service composition model provides scaffolding for the auto generation of a composite service's FSM. Thus, the composite service's adaptivity is described by its FSM which is the combination of the constituent services' FSM. The composite service's FSM is a composite adaptivity description which is extrinsically linked to its constituent services. The service's adaptivity can be configured prior or during execution using policies. This provides the level of dynamicity required in ubiquitous computing environments. For a composite service, these policies are refined and applied to its constituent services, configuring its adaptivity.

4. Conclusions and Further Work

This paper showed how non-adaptive service composition plans can be supplemented by policy-based techniques to allow context-aware adaptivity in service execution. It describes how the proposed approach can expose non-functional

adaptive behaviours of the constituent services to the composite service level using FSM. The FSMs are generated as part of the development process of the constituent service's implementation. The approach allows adaptive behaviour rules, in the form of user policies, to be imposed on a composite service without contradicting the constituent service developer's intentions for managing its service. It also showed how adaptive behaviour rules for composite services are mapped into transitions that will govern the adaptive behaviour of the relevant constituent services.

Our future work in this area will investigate the benefits and drawbacks of using our proposed approach and extend it accordingly. We will exploit ontological reasoning to provide a better method of aggregating constituent service FSMs and investigating how to control exposure of different levels of complexity present in the aggregate FSM in order to match to the user's level of experience.

We are also investigating in parallel, how similar results can be achieved without requiring design time modelling of adaptive behaviour. Instead, post-development wrapper patterns model how conditional aspects of a service's behaviour can be handled in a particular ubiquitous computing environment. The conditional logic in these patterns are mapped to taxonomies of simple service parameters that can be used to select services included in AI planning process thus exploiting existing plan optimisation techniques.

5. References

[alcantara 03] Alcantara, O. D. and Sloman, M. QoS Policy Specification A mapping from Ponder to the IETF. 2003.

[carroll04] Carroll, J., Dickinson, I., Dollin, C., "Jena: Implementing the Semantic Web Recommendations", in Proc. of World Wide Web Conference 2004, 17-22 May 2004, New York, NY, USA

[damianou01] Damianou, N., Dulay, N., Lupu, E., Sloman, M., (2001) "The Ponder Policy Specification Language" ,Proc. Policy 2001: Workshop on Policies for Distributed Systems and Networks, Bristol, UK, 29-31 Jan. 2001, Springer-Verlag LNCS 1995, pp. 17-28

[Lutfiyya97] 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.

[mont99] Mont, M. C., Bladwin, A., and Goh, C. POWER Prototype: Towards Integrated Policy-Based Management. HPL-1999-126. 18-10-1999.

[owls02] "OWL-S: Semantic Markup for Web Services", The DAML Service Coalition, http://www.daml.org/services/, October 2002

[ranaganathan04] Ranaganathan, A, Campbell, R., "Autonomic Pervasive Computing based on Planning", Proceedings of the International Conference on Autonomic Computing, IEEE, 2004

[sirin03] Sirin E., Parsia B., "Planning for Semantic Web Services", 3rd International Semantic Web Conference, 2003.

[weiser91] Weiser, M. (1991), "The Computer of the 21st Century", Scientific American, vol. 265, no.3, September 1991, pp 66-75.

[uml03] – OMG - Unified Modelling Language 1.5, March 2003.

[wies94] 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.