

ROAD4WS – Extending Apache Axis2 for Adaptive Service Compositions

Malinda Kapuruge, Alan Colman and Justin King
Faculty of Information and Communication Technologies
Swinburne University of Technology,
Melbourne, Australia
(mkapuruge, acolman, jmking)@swin.edu.au

Abstract— Service oriented architecture plays a prominent role in creating and utilizing business services in enterprise computing environments. The service composers produce value by aggregating lower-level re-usable services, scattered across the internet to create application level services. Web service middleware facilitates in defining service compositions in a comprehensive manner. However, in order to ensure the business viability amidst unpredictably changing business requirements, such compositions may need to adapt during the runtime. Such changes might vary from a minor regulation to a major re-structuring of the IT service composition. However, the complexity of the composition shouldn't increase and the runtime interruptions to the service delivery need to be kept to a minimum. In this paper we introduce ROAD4WS, which is a middleware extension to the popular Apache Axis2 web service engine. The extension brings together the modular adaptive architecture of the Role Oriented Adaptive Design (ROAD) with the web services deployment and consumption capabilities of the Apache Axis2 engine, in order to facilitate deploying adaptive service compositions.

Keywords- SOA; Service composition; Axis2; Middleware (key words)

I. INTRODUCTION

Middleware has conventionally been aimed at providing interoperability between legacy applications through functions such as reliable message delivery, end-point abstraction, protocol transformation, security and so on. Vendor implementations of such middleware in the form of message-oriented integration brokers or enterprise service buses (ESBs) have become increasingly sophisticated and complex. These products also commonly provide monitoring and management functions to handle the integration of heterogeneous applications and services.

Middleware products have also increasingly been supporting *compositional* functions such as brokerage, business process engines, event-driven processes, provisioning core services with billing services, and so on. Web services technology is a recent manifestation of long evolution of such middleware products. The stack of Web service standards has provided a common framework for implementing such functions. As business environments and their associated service-oriented ecosystems become increasingly volatile, compositional aspects of middleware need to be constantly updated. In order to support such change best practice implementations of middleware support composable modular mediators that implement various layers of the WS stack. This modular approach is more

amenable to adaptation than a heavy-weight monolithic middleware layer.

Apache Axis2 [1] is a popular web service engine that integrates a wide variety of web services standards. It does this by allowing its core engine to be extended in a modular way. Axis2 is capable of deploying web services and it also provides a client API to consume web services. However, while Axis2 supports modular extension, as of now there is no explicit support for deploying dynamically adaptable service compositions in Axis2.

In this paper we introduce an extension to Apache Axis2 web services engine, which allows easy deployment of service compositions that can be changed at runtime. In order to achieve that we use the Role Oriented Adaptive Design (ROAD) [2-4] to design the service composition, hence the extension is called ROAD4WS (Role Oriented Adaptive Design for Web Services). A ROAD composition provides rule-based mediation of interactions between a group of interrelated service consumers and providers. These rules define permissible interactions and enable domain specific business logic to be applied to such interactions. Furthermore the rules and the structure of the composition itself can be managed and altered at runtime. As a ROAD composite presents itself as a collection of web services at runtime, any alteration to the composite is dynamically reflected in the operations and policies presented at the service interface. In this paper we provide a detailed description on how the adaptable ROAD architecture is realized using the web services technology.

The rest of the paper is organized as follows. The section 2 contains a business example and the problem analysis. An overview of the ROAD and Axis2 is given in Section 3. Section 4 introduces ROAD4WS extension to Axis2 while Section 5 discusses the key features of ROAD4WS in detail. Section 6 presents the implementation. Related work is discussed compared to ROAD4WS in section 7 and we conclude the paper in section 8 by pointing to future work.

II. BUSINESS EXAMPLE AND PROBLEM ANALYSIS

A. Business Example

SwinniwS is a news portal that brings news bulletins, weather and sports updates to its customers. The *SwinniwS* uses updates from other service providers to compose the news reports. The collaborators, i.e., news, sports, weather updates providers are geographically distributed. *SwinniwS* uses web service technology to communicate with them. Such communications may include getting

news/sports/weather updates as well as publishing advertisements as part of news items and processing payments etc. Also SwinniW needs to monitor the interactions among the collaborators for quality control and billing requirements.

B. Problem Analysis

A service broker such as SwinniW integrates a series of information services, to provide value-added services and consequently earn some revenue. However, the success of the SwinniW as a business highly depends on the ability of the web services composition to withstand the runtime adaptation requirements. As an example, during the runtime,

1. The integrated external web services become unavailable. Consequently, new services need to be bound replacing the currently bound web services.
2. The business requirements of the SwinniW or its clients can change. Consequently the rules that regulate the interactions between parties to the composition needs to be altered, or the relationships between the parties may need to be structured. This needs to be done with no, or minimal, interruption to service delivery.

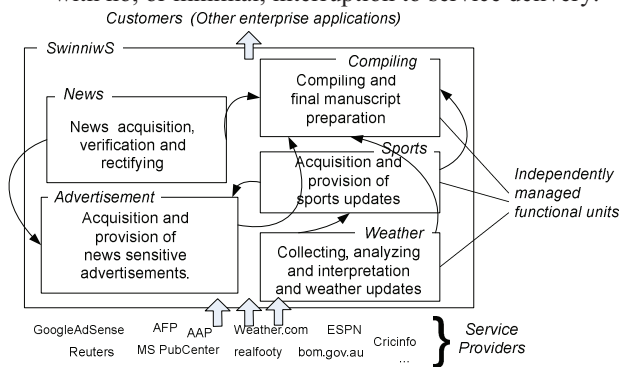


Figure 1: SwinniW composition and collaborators

In the business composition there are certain functional units (Figure 1), communicating with each other (shown by arrows) to fulfill a business requirement. These functional units need to be monitored and managed independently to avoid complexity of the overall system. Therefore such independently managed functional units (IMFUs) need to be clearly represented in the overall service composition. As an example, the weather updates provisioning can be managed independently. In this sense, how/who involved in providing, acquiring, analyzing, interpreting and structuring weather data is irrelevant to other functional units such as news filtering and compiling. In this sense, at a certain level of abstraction, these IMFUs represent the *Positions/Roles* of the overall composition. There are certain advantages of such a decomposition of functionalities.

a. The abstraction provided aid to keep the business logic in the base composition (SwinniW) as simple as possible. Only the interactions among these IMFUs need to be considered.

b. The changes can be applied to these isolated IMFU clusters without hindering the operations of the rest.

c. The functionalities of these isolated IMFUs can easily be outsourced as services to other business organizations if that is more efficient and profitable.

In addition to a modular structure, the functional structure of the composition needs to be separated from the management structure. In this sense, the functional units, which form the functional structure, need to interact among each other, e.g., gathering, exchanging information (news, sport etc.) and publishing them. The management system should constantly monitor and manage the functional structure, e.g., an engineer or a decision making software system monitoring and managing the interactions. This is to ensure the business objectives are achieved promptly responding to perturbations in the hosting environment.

In brief, a service composition approach that addresses services delivery in volatile marketplaces should be adaptable, modular, and support monitoring and management as separate concerns.

III. BACKGROUND

The ROAD4WS (Role Oriented Adaptive Design for Web Services) extension enables the deployment of adaptive service compositions that are designed as per the ROAD architecture[4]. The extension adds features such as dynamic composition deployment, dynamic service binding, dynamic business logic changes, and contract-based composition design etc. to the Axis2 web service engine. Before looking at how the extension works, we will briefly introduce the basics of ROAD architecture style and the Apache Axis2 engine.

A. Role Oriented Adaptive Design

Role Oriented Adaptive Design (ROAD)[4] is an architecture style to design self-adaptive composite software systems. The ROAD architecture brings a number of architectural attributes[5] that supports self-managed capabilities. One of the main benefits of ROAD architecture is that it clearly separates the management structure from the functional structure of a software system[5]. Also the contract-based composition design helps to modularize the complex service interactions.

ROAD envisages the software system as a real world organization which consists of interconnected *Roles (position descriptions)* of the organization[2]. External entities (*Players*) such as web services can be selected to perform the functionalities of these Roles. The mutual responsibilities of players are defined by the *binary contracts* among these roles. The ROAD runtime, which consists of such contracts, provides message routing, rule-based mediation and context evaluation capabilities. Once assigned to a particular Role, the players are obligated to perform duties of the Roles as per the defined contracts.

These contracts are monitored and managed according to rules defined by a special non-functional role called the *Organizer*. As well as writing rules that mediate interactions within the composite, the organizer is responsible for configuration of the composite by performing management activities such as define/abolish/modify contracts and

bind/unbind players. The domain in which a set of functional roles are under an Organizer role is called a *Self-Managed Composite* (SMC). An SMC is a modular runtime model of set of service-service relationships that controls and mediates interactions over those relationships. SMC models are not decomposable other than through specification of a role played by a separate system. They are intended to encapsulate a limited set of relationships at a particular level of abstraction rather than a global design of the function of a system. For a detailed discussion about ROAD concepts, we refer to [3, 4].

B. Apache Axis2

Apache Axis2[1] is an open source web services engine which provides functionalities to deploy and consume web services. The new modular design in Axis2 has made it possible to extend the capabilities of the engine with minimal or no changes to the core. As an example the engine allows adding custom Deployers, Modules that can further extend the service deployments and message processing capabilities. Modules such as Rampart¹, Sandesha² has already developed to extend Axis2 capabilities. The Axis2 engine can be deployed in a servlet container such as Apache Tomcat. Both SOAP and REST styles of web services are supported. With its built in XML info-set model AXIOM³, and pull-based parser, Axis2 provides a speedy and efficient parsing of SOAP Messages. Also Axis2 supports different transport protocols such as HTTP(S), SMTP, FTP. All these factors have made Axis2 suitable to serve as the basis in implementing ROAD4WS extension.

IV. ROAD4WS

With ROAD4WS, we bring together the adaptation and composite management capabilities of the ROAD as well as the service deployment and consuming capabilities of Apache Axis2 to provide a lightweight middleware solution to deploy and manage adaptable service compositions. In the following section we will first clarify how the ROAD4WS extension works. We then will highlight how ROAD4WS extends the capabilities of the Axis2 web services engine to deploy adaptable service compositions.

A. Overview

In order to understand how the ROAD4WS extension works, let's take the SwinniW scenario we presented in the section 2. The SwinniW business requirements need to be realized as a Self-Managed Composite[4], which we call *SwinniW SMC*. In ROAD4WS we identify four different phases of an SMC life-cycle as shown in Figure 2.

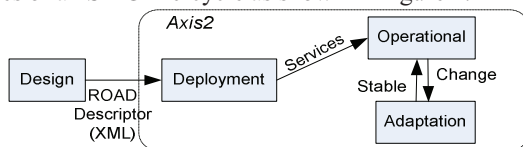


Figure 2: Life-cycle of a ROAD composite

¹ <http://axis.apache.org/axis2/java/rampart/>
² <http://axis.apache.org/axis2/java/sandesha/>
³ <http://ws.apache.org/axiom/>

Design phase:

1. The SMC designer identifies the roles of service providers and consumers in the composition. This includes, News Provider (NP), Weather-updates Provider (WP), Advertisement Provider (AP), Sports-update Provider, Compiler (CM) and Client (CL).
2. The SMC designer defines the contracts (interaction rules) among these roles, provided they need to communicate with each other. As an example, the contract CM-WP specifies the valid messages that should be exchanged, obligations of Role players and how the messages should be evaluated at runtime etc. These interaction rules are written in the Drools rules language,
3. Based on 1 and 2, the designer writes the complete composite descriptor. The designer may optionally use the ROADdesigner tool⁴ for this purpose. ROADdesigner is a graphical interface to design ROAD composites.

Deployment Phase:

4. The designed SMC descriptor that define the structure of the composite in XML and associated rule files are uploaded to Axis2 to be deployed by ROAD4WS. The ROAD infrastructure dynamically instantiates an SMC runtime that provides rule-based message mediation and routing as well as performance and context evaluation. A role object for each service is also created.
5. The ROAD4WS extension dynamically deploys the proxy web services based on the ROAD role objects. These proxy services expose permitted sets of operations to external service providers to invoke. The WSDL descriptions are dynamically generated to expose the service interfaces.
6. A special proxy service is created for the *Organizer* role. The organizer role player (an automated or semi-automated intelligent controller) can use this interface to manipulate the composition.

Operational phase:

7. At runtime, functional players, which are distributed atomic web services or composite web services, can be bound to Roles. Functional players can exchange messages using the ROAD4WS extension by invoking the exposed interfaces.
8. The exchanged messages will be evaluated against the rules in the contracts. If valid, messages will be passed to other players or get rejected otherwise. Performance rules defined in the contracts are also evaluated.

Adaptation Phase

9. The organizer player manipulates the composition by changing contracts (states, operations, parameters), injecting rules, adding new roles, changing player bindings etc via the organizer interface exposed via the ROAD4WS.
10. The contracts affected by the adaptation activity can be temporarily suspended depending on the type/degree of

⁴ <http://www.swinburne.edu.au/ict/research/cs3/road/roaddesigner.htm>

change. Meantime, the rest of the contracts of the composition may function without any interruption.

11. Finally, the whole composition can be removed if no longer required.

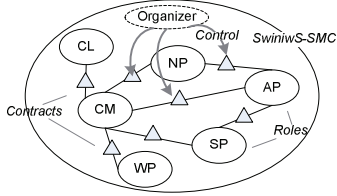


Figure 3: SwinniW SMC

The designed *SwinniW-SMC* is shown in Figure 3. The service interfaces are created for functional roles i.e., News Provider (NP), Advertisement Provider (AP), Weather Provider (WP), Client (CL), Compiler (CM), Sports update provider (SP) as well as for the special *Organizer* role to manage the composition[4, 6]. The Organizer manipulates the contracts according to changing business requirements. The contracts (CM-NP, NP-AP, CM-WP...) are established among the functional roles if they have mutual obligations. As an example the CM-NP contract defines the interaction protocol by which news updates should be provided for the compilation. Also the NP-AP contract is established to provide a communication channel between the advertisement provider and the news providers so that the provided advertisements are sensitive to the provided news items.

B. Web Service Deployment

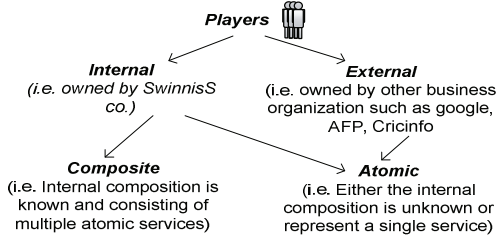


Figure 4: Player classification

The roles of the *SwinniW-SMC* are played by players (web services). These players are the actual functional entities that perform computations/activities. In this sense a role represents the “*requirement*” while player represents the “*computational unit*”. In ROAD4WS we classify players of an SMC as shown in Figure 4 from the deployment perspective. Depending on the type of players the manner in which they were designed and deployed can vary. We first, categorize the players owned by SwinniW (*internal*) or other business organizations collaborating with SwinniW (*external*). The internal players, i.e. owned by SwinniW, can either be *atomic* or *composite*. Since the information about the compositions of external players is unknown to SwinniW, all external services are considered as *atomic* too. In this sense, we consider the players are *atomic* when the internal composition is not known or it is irrelevant to further decompose.

As an example, the player of role AP could be played by an *external-atomic* player such as *GoogleAdSense* or

MicrosoftPubCenter. CM role could be played by an *internal-atomic* web service, deployed by SwinniW, whose whole purpose is to compose and layout the news items, weather updates and sports updates so that the customers (CM) can easily view them. WP role could be played an *internal-composite*, which integrates several other atomic web services deployed in weather stations and weather update interpreter services. In this sense, the WP player could be designed as an SMC too.

The complete picture of deployment environment is given in Figure 5. The Weather-SMC is playing the role WP. The Weather-SMC is consisting of its own roles such as weather updates Interpreter (IP), Local Weather updates provider (LW) and Overseas Weather updates provider (OW). Also the Weather-SMC defines contracts among those roles. The Weather-SMC too consists of an Organizer role to manipulate these contracts and the composition. Similarly the SP role could also be played by an internal composite, which could open up few other roles to external or internal atomic players.

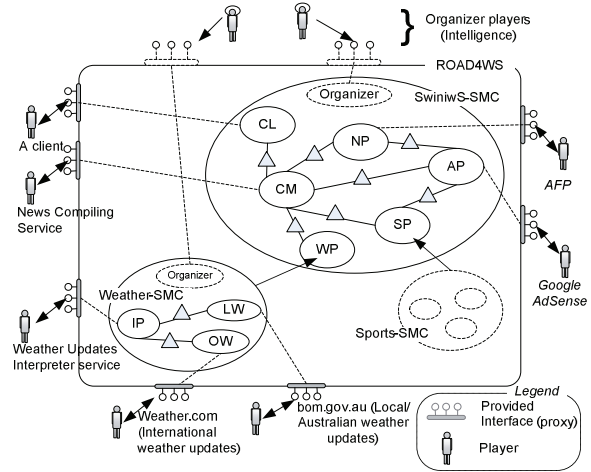


Figure 5. Composite deployment in ROAD4WS

Following is a step by step procedure of designing a business composition such as SwinniW.

1. First, all the identified IMFUs of a business composition are represented as Positions/Roles.
2. Then if a player of an IMFU is needed to be further decomposed, we represent it as an SMC (e.g. *Weather-SMC*) and it too is deployed via ROAD4WS.
3. If further decomposition is irrelevant, we represent them as single web service (e.g., *GoogleAdSense*, *News Compiling Service*).

This creates a hierarchical structure of SMCs until the *leaf nodes* are assigned with atomic web services. Further,

4. The *internal-atomic* services (e.g., *News Compiling Service*) could be deployed via Axis2 with available deployment options such as POJO (Plain Old Java Objects) or Archive-based (.jar).
5. In the case of *external-atomic* services (e.g., *GoogleAdSense*) only the endpoints are bound to the relevant roles.

All the functional roles defined in main SwinniwS-SMC and other SMCs are exposed as proxy web services as shown in Figure 5. The operations defined in these roles are exposed as web services operations. The external/internal atomic players can play these roles by invoking the exposed web services operations. Also the humans and decision making software systems can use the exposed Organizer interface to manipulate the SMC in a decentralized manner. The same organizer player might bind to play multiple Organizer roles if required or could be played by multiple players.

C. Extending Axis2 capabilities

In order to facilitate the deployment of ROAD composites in a web services environment, we have extended the capabilities of the Apache Axis2 engine as follows.

Composition deployment: As of now there is no explicit support to deploy adaptive service compositions in Axis2. The ROAD4WS extension allows deploying adaptive service compositions that share a common adaptable runtime in a comprehensive manner.

Role and Contract based design: The Role orientation enables representing discernable functional requirements of a composition to be abstracted. The contract based design allows modularization of the business logic relevant to the composition. Such an abstraction and modularization helps to mitigate the complexity[7] resulting from runtime modifications. Such complexity is one of the obstacles to achieving adaptive software design[8].

Remote management: The composition and its behavior can be remotely managed to *regulate* and *reconfigure* the composition via the provided Organizer interface at runtime. ROAD4WS exposes this interface as a web service. In this sense, the *management* entity and the *functional* structures can be deployed in separate machines if needed.

Dynamic service bindings: In conventional Axis2, the service is written in a programming language such as Java or composition language such as BPEL. Method signatures (in Java) or partner-links (in BPEL) are exposed as WSDL interfaces. However, in the case of both Java and BPEL these interfaces are statically defined. In ROAD4WS these service interface definitions can be changed dynamically.

Event-based message synchronization: With Serendip[9] process modeling and enactment support, Axis2/ROAD4WS users can benefit from event-based synchronized message exchange among collaborating services. The message exchange can be synchronized with the events occurring in the composition via a declarative business process modeling and enactment approach.

V. AXIS2 WITH ROAD4WS

In Axis2 there are a variety of mechanisms that are available to deploy services such as archive-based, POJO-based etc. In this service deployment process, Axis2 hides the complexities such as WSDL generation, transport layer handling, SOAP/REST style conversations etc.[1]. Basically, Axis2 is designed to deploy and consume web services.

On the other hand the ROAD designs and creates self-managed composites[2]. In such a composite the

external/internal players collaborate to achieve a business goal by sharing a common runtime. In next few subsections we will investigate in detail, the features of ROAD4WS that brings ROAD and Axis2 together to support adaptive web service composition. We will clarify the concepts and techniques used with the examples from the SwinniwS business example.

A. ROAD to Axis2 Mapping

The ROAD4WS extension maps the ROAD concepts to Axis2 concepts as shown in the following table.

TABLE I. ROAD TO AXIS2 CONCEPTUAL MAPPING

ROAD	Axis2
Role	Proxy Services (WSDL)
Contract	N/A
Term	N/A
Term-Operation	Web service operation (<i>Operation Context</i>)
Player	Web-Service or a Web-Service Client
SMC	Group Proxy Services (<i>Service Group Context</i>)
Organizer	Proxy Service (WSDL)

Roles are exposed as proxy services and players can be implemented as web services/service-clients. *Operations* in *Contractual-Terms* are exposed as web service operations. An SMC is a collection of proxy services that share a common runtime and a common Organizer. Currently, there is no equivalent concept for a *Term* or for a *Contract* in Axis2. *Contracts* and *Terms* define the internal business logic of an SMC. We will now look further into contracts.

```
<?xml version="1.0" encoding="UTF-8"?>
<SMC ... rulesDir="data/rules/">
  <Roles>...</Roles>
  <Contracts>
    <Contract id="cm-ap" ruleFile="cm-ap.drl"
      type="permissive">
      <State>Incipient</State>
      <Terms>
        <Term id="cm-ap-t1" name="Advert provision"
          messageType="push">
          <Operation name="sendAdvertisement">
            <Parameters>
              <Parameter>
                <Type>String</Type> <Name>Title</Name>
                ...
              </Parameter>
            </Parameters>
            <Return>String</Return>
          </Operation>
          <Direction>AtoB</Direction>
        </Term>
        ...
      </Terms>
      <RoleAID>ap</RoleAID>
      <RoleBID>cm</RoleBID>
    </Contract>
    ...
  </Contracts>
  <PlayerBindings>
    <PlayerBinding id="cmbp" name="Composer Player Binding">
      <Endpoint>http://127.0.0.1:8080/axis2/services/
        Composer</Endpoint>
      <Roles> <RoleID>cm</RoleID> </Roles>
    </PlayerBinding>
    ...
  </PlayerBindings>
  <Description>Providing news bulletins</Description>
</SMC>
```

Figure 6: A contract between CM and AP

A sample contract between the CM and AP roles is given in Figure 6. A ROAD SMC descriptor consists of a number

of contracts similar to the highlighted. During the runtime a *Contract* can be in any of the states to name, *incipient*, *active*, *suspended*, *breached*, *terminated* or *discharged*[3]. The contract specifies multiple *Terms* that the both parties (player that perform CM and AP roles) should abide by. *Terms of a contract* also have execution states, which maintain the state of an interaction, e.g. “*AP has sent the advertisement and awaits confirmation from CM*”.

Operations defined in *Terms* are exposed as web service operations via WSDL. The direction of the message flow of a *Term* is defined by the property “*Direction*” as in Figure 6. In this sense, Role A, i.e. AP is supposed to initiate the conversation by sending a message. As there is a Return type of the operation *sendAdvertisement*, AP will receive a response message. The contract can also specify how the message should be evaluated using a rule file specified by attribute *ruleFile*. Currently Drools [10] is used to specify rules, where message and the contract states are inserted as facts. More details on ROAD composite descriptors and rule evaluations is beyond the scope of this paper and interested readers may refer to [11, 12].

B. Dynamic changes at runtime

The dynamic changes are handled in two different levels of the ROAD4WS as described in the table below.

TABLE II. CHANGE TYPES

Type	Description
Type 1 (Axis2 level)	Multiple SMC descriptors can be dropped or deleted from the specified file location to add or remove the ROAD composites during runtime without stopping the server. The ROAD4WS extension listens to such deployments/un-deployments.
Type 2 (SMC level)	These are the changes that occur within the composition. Usually carried out via the organizer interface. The ROAD4WS extension reacts to such changes in the composite and does changes in the Axis2 runtime accordingly.

Type1 changes are carried out via the Axis2 deployment environment whilst the *Type2* are carried out via the organizer interface supplied with each SMC. The Organizer interface exposes its operations in Axis2 via WSDL to add or remove contracts/operations/terms/roles/bindings, injecting/deleting new rules to contracts etc. Examples for such operations are, *addNewContract()*, *removeContract()*, *addNewTerm()*, *removeTerm()*, *addNewContractRule()*, *removeContractRule()* etc. Such operations can be used to regulate or reconfigure the SMCs. Type 2 changes are more sophisticated and therefore let’s clarify how the composition is regulated and reconfigured with examples.

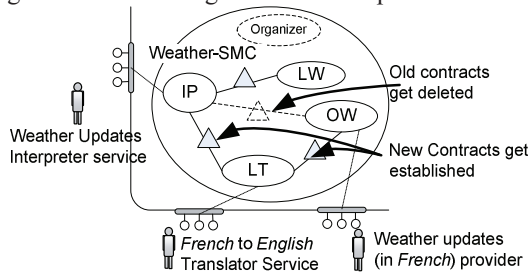


Figure 7: Changes within the composition (Type 2)

Example1 (*Reconfiguration*): Upon unavailability of usual OW player (i.e. the overseas weather updates provider), a new service needed to be bound. However, if the new service is providing updates in a language other than English, the updates need to be translated first, using a Language Translator (LT). Therefore new contract IP-LT and LT-OW are established and the old contract IP-OW is deleted as shown in Figure 7. Such reconfigurations are supported by the extension dynamically, easily and decentralized manner.

Example2 (*Regulation*): The operations of an existing contract can get changed. For instance, new *Contractual Term*[4] could be introduced to the contract IP-LW to get frequent and more specific updates upon natural disasters such as cyclones. Such regulatory changes to the contracts automatically get reflected in corresponding WSDL interfaces of adjoining contracts.

Moreover, the ROAD framework realizes these changes to contracts accordingly to the state of the contract [3] during the runtime. As an example, the organizer player can monitor the current state of the contract and decide whether the change should be applied immediately or after letting the current ongoing transactions to be completed (meanwhile no new conversations are allowed). Such mechanisms along with the modularity provided by the ROAD architecture means runtime modifications can be carried out safely and without interrupting the other parts of the composition.

C. Generating Interfaces

In ROAD compositions, there are multiple roles and hence require generating multiple WSDL interfaces for a given SMC. There are two types of interfaces for a given Role from the SMC perspective.

- *Provided Interface*: The interface provided by the composition exposing role operations. The player invokes provided interface to send messages to the composition. (Interfaces shown in Figure 5 and Figure 7 are provided interfaces).
- *Required Interface*: The interface that should be provided by the player exposing its functionalities. The composition invokes required interface to send messages to the player.

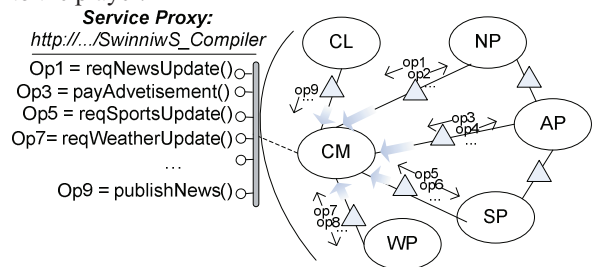


Figure 8: Operation aggregation in adjoining contracts

A role is an aggregation of the obligated operations defined in adjoining contracts as shown in Figure 8. As an example, Role CM is a position description that should abide by the obligations defined in bound contracts, CM-CL, CM-NP and so on. The little arrows in the figure denote the

direction of obligations. For example $op1=getNewsUpdate()$ is obliged to play by CM player. Consequently, the provided WSDL corresponding to the provided interface of CM role has a WSDL operation $getNewsUpdate()$ with suitable in/out parameters. Although not shown in Figure 8, similarly the other roles (e.g., AP) also will get its own interface including its obliged operations (e.g., Op4) defined in adjoining contracts (e.g., CM-AP, NP-AP, SP-AP).

The ROAD4WS extension needs to dynamically generate these interfaces for each and every identified role of the composition. Also re-generate the interfaces as the contractual agreements get changed during runtime. (Typically subject to course to business level negotiation phase before the contracts get changed). When the signature of an operation defined in a *Contractual Term* gets changed, the WSDL input and output parameters and message types need to get changed of the obligated Role. For example, if the parameters of operation $op1=getNewsUpdate(...)$ in contract CM-NP get changed, the WSDL corresponding to CM should also need to be changed as CM is the obligated role for this operation. Similarly when operations are added/deleted, WSDL operations are added/deleted from the service proxies.

Contractual Terms can specify either synchronous or asynchronous operations. For example, a “void $sendAlert()$ ” operation can be asynchronous whilst the “String $getNewsUpdate()$ ” is synchronous. Therefore message exchange patterns⁵ (MEPs) in WSDL interfaces need to be set accordingly. Depending on the in/out parameters of defined in operations of Contractual MEPs are allocated. These too will be re-adjusted according to the changes in *Contractual Terms*.

D. Message delivery

Players invoke the operations exposed in provided interface of the roles, creating messages. Such messages are routed across the composition to other players in collaboration. As an example in our scenario the messages need to be routed from AP to CM to provide advertisements. For this particular message delivery, we call AP is the *Origination Role* and the CM the *Destination Role*. The message is subjected to an evaluation against the contract, on its way from *Origination* to *Destination* as shown in Figure 9. The contract CM-AP precisely defines the type of messages that should be exchanged. Depending on the state of the contract[3] and the validness of the content, the message gets passed, held or rejected. As an example, if the contract is *temporarily suspended* the message will be either held or rejected depending on the rules specified in the contract CM-AP. In such situations the ROAD4WS will send an *AxisFault* back to the player of the *Origination Role* which contains reason for failure so that player can take a remedial action.

However, even when the message gets passed, players might not be available, e.g. *compiler service is offline due to network failure*. Also the message delivery to the CM needs to be carried out in synchronized manner, e.g. *advertisement*

needs to be sent after the related news item (e.g., another SOAP message) received from the NP. Due to such reasons, the messages get buffered in the *destination role* before the delivery to the current player.

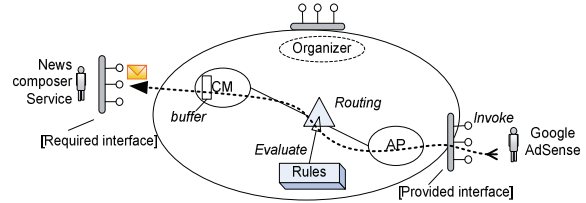


Figure 9: Message routing and contract evaluation

We have introduced two different methods of message delivery in ROAD4WS extension.

- **Pull-based:** Messages are kept in the Role to be pulled by the player. As an example, the player invokes the $getNextMessage()$ operation of the *Provided Interface* of the corresponding Role and retrieve the next available message.
- **Push-based:** Messages will be pushed to a pre-specified endpoint. The ROAD4WS pushes the message to the Player by invoking the corresponding operation of the *Required Interface* at the player web service.

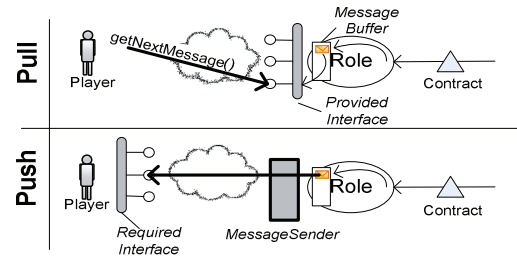


Figure 10: Pull- and Push-based message delivery

These two modes of message delivery are shown in the Figure 10. **Pull-based** message delivery is useful for players that cannot keep a static endpoint or players that interact with the composition only as clients. In both these cases the composition cannot send the messages to the players and consequently have to wait until the message being pulled. In contrast **push-based** message delivery is useful for players that can provide an endpoint to the composition. Once a message is received and evaluated via the contracts, it will be routed to the destination role player’s endpoint. The contracts may define the number of attempts in which a message should be pushed to the destination player before being flagged as a failed delivery. Upon failure, ultimately an *AxisFault* is thrown.

The push and pull based message delivery mechanisms are implemented via the available Axis2 functionalities such as custom *Message-Receiver*s and *Client API*[1]. The ROAD4WS extension registers a dedicated *Message-Receiver* for each Role to deliver the messages in pull-style. The Axis2 *operational clients* are being used to push the messages towards the *Required Interface*.

⁵ <http://www.w3.org/2002/ws/cg/2/07/meps.html>

Such buffering is important for synchronizing the messages too. As the messages get routed across the contracts, the contracts trigger events. An event is an occurrence that is meaningful to the composition. These events are being used to schedule the message delivery among players to get the obligated tasks accomplished[9]. As an example in SwinniWS the messages flowing from NP and WP towards the CM might need to be synchronized. As shown in Figure 11, the messages m1,m2 are routed across to CM role from NP and AP roles respectively. The messages get evaluate by contracts CM-NP and CM-AP and get buffered in the CM role. According to the evaluation events e1 (=newsItemOK) && e2(=AdvertisementOK) get triggered. Once the event pattern, $e1 \wedge e2 \Rightarrow true$ the messages are synthesized and are pushed to the CM player. Details on message synchronization and synthesis are beyond the scope of this paper.

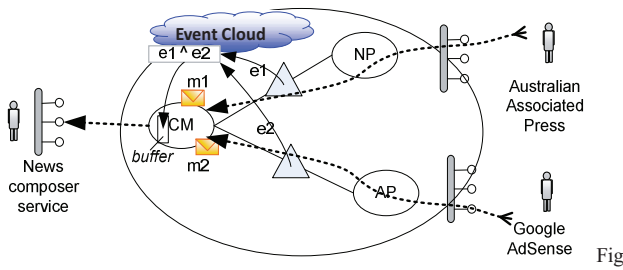


Figure 11: Message synchronization with events

VI. IMPLEMENTATION

The overall picture of how ROAD4WS positions itself in Axis2 is given in Figure 12. The ROAD4WS is implemented by extending the functionalities available in Axis2 such as *modules*, *custom deployers*, and *message receivers*.

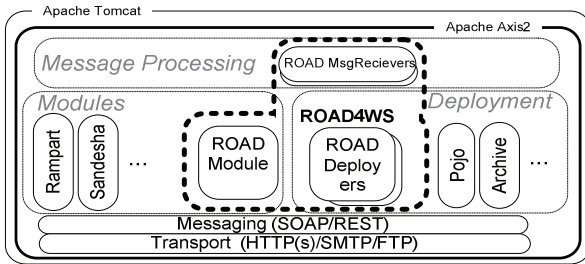


Figure 12: ROAD4WS in Axis2

For example, the ROAD Deployers register themselves with *Axis2-kernel* to listen to deployment changes in the Axis2 repository in order to dynamically deploy the composite descriptors. The ROAD Module extends the core functionalities of Axis2 engine to enact self-adaptive service compositions designed according to ROAD. The ROAD *Message Receivers* intercepts messages targeted towards the deployed proxies of an SMC to get them routed across contracts. This use of extensible features avoids modifications to Axis2 engine and thereby allows easy integration of ROAD4WS with already installed Axis2 environments. The servlet container could be any that is supported by Axis2 such as Apache Tomcat.

The SwinniWS composition can be deployed in Axis2 shown in Figure 13. The SMC files such as *SwinniWS_smc.xml*, *weather_smc.xml* are dropped to *road_composites* folder while the internal atomic services such as *Compiler*, *Weather Interpreter* can be deployed using Axis2 built-in deployment mechanisms such as POJO and Archive-based. ROAD4WS-Admin interface can be used to remotely deploy/undeploy the SMCs.

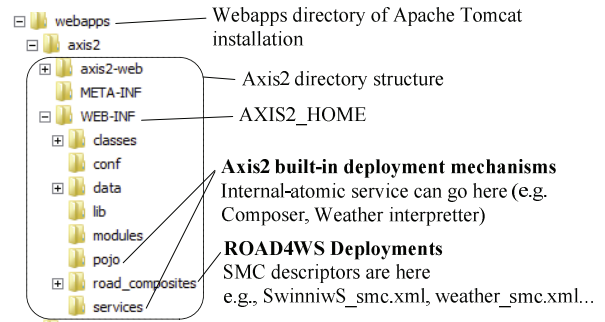


Figure 13: SwinniWS Deployment in Axis2

Once deployed, the functional and organizer provided-interfaces are available in Axis2 web services engine as shown in Figure 14. Depending on the modifications to the contracts, the available operations get changed.

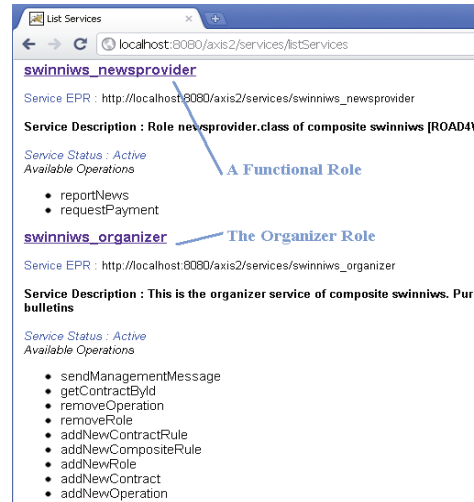


Figure 14: Exposing functional and organizer interfaces

VII. RELATED WORK

A plethora of approaches proposed for integrating web services to create service compositions in the past. In this section we will discuss some of the selected approaches due to space limitations. For clarity in explanation we will divide those into three main categories, BPEL-based, ESB and the Generic (Other) approaches.

A. BPEL-based integration approaches

WS-BPEL[13] has been the de-facto standard for integrating the distributed web services according to a well-defined

orchestration model. Many vendors support WS-BPEL. However, runtime adaptation of BPEL is problematic. As a result, many approaches have been proposed to improve the adaptability of WS-BPEL and thereby the service composition. A-ActiveBPEL[14] brings adaptation to WS-BPEL based service composition via semantic modifications to service invocation primitives. Robust-BPEL[15] allows self-healing and self-optimizing capabilities to WS-BPEL while TRAP-BPEL[16] further improves such capabilities by providing a *generic proxy* that adds self-adapting behavior to service composition. In the two layered architecture is proposed by Akkiraju et al.[17] an abstract business process flow specification is first given in the higher layer via semantically annotated BPEL4WS. Second, the lower layer captures the service discover, integration and execution. However, in these approaches the adaptation is limited to selecting and integrating service endpoints. Compared to ROAD4WS less-attention has been paid to allowing restructuring the composition at runtime.

More advanced adaptation capabilities to WS-BPEL language have been introduced via well-known techniques such as *Aspect-oriented* and *template-based* software development. AdaptiveBPEL[18] allows adapting a service composition via a policy driven approach. The policies are used to adapt the service composition via *Aspect Viewing*. Navarro et al. uses *aspects* to modularize the cross-cutting concerns in composing decentralized web services[19]. Similarly in Patus[20] and Ao4BPEL[21] aspects are used to dynamically adapt the composition. Several *template based* approaches have also been proposed[22, 23]. Geebelen et al.[22] proposes a controller that selects pre-defined templates and integrate with master process. In [23] Rajaram et al. propose selecting and enacting templates based on context/input parameters to integrate with a master process.

However, a common worry about *Aspect-oriented* and *Template-based* approaches is that such approaches assume a *fixed part* and a *volatile part*. For example the *master-process* or the *point-cuts defined business process* represents the fixed part. The *aspects* or the *parameters* those get replaced or changed, represent the volatile part. Such a distinction of fixed and volatile parts might be possible for some change requirements but might not be sufficient for changes where a major re-structuring is mandatory in adapting the composition. In contrast, ROAD4WS allows a greater level of runtime re-structuring capabilities of composition. Also the modular architecture isolate the different functional units so that the maintenance does not affect the composition as a whole.

B. ESB-based integration approaches

Enterprise Services Bus (ESB) is a very common style of integrating the distributed enterprise applications via web services. The idea is to provide a unified, simple interface to consume services/applications without worrying about the complexity of underlying service interfaces. Usually the enterprise applications are connected via a common message mediation channel known as a *Bus* to facilitate peer to peer

communication. Mule⁶, Synapse⁷, ServiceMix⁸ are example ESBs available at present.

In that sense, both the ROAD4WS and ESBs share a common interest i.e., to integrate enterprise applications via web services. However, the key benefit of using ROAD4WS over an ESB is its ability to decompose and modularize the different functional units of an enterprise application. This is vital for realizing large and complex business organizations in a well-manageable manner. As shown in Section IV, the SwinniS application is decomposed into a several SMCs until the leaf level Roles are assigned with *atomic services*. When required, these atomic services can be further decomposed.

Another advantage of ROAD4WS, compared to ESB is that its *organizational oriented view*[4] compared to the *bus oriented view*[24]. In ROAD architecture, all the collaborators perform duties as assigned by the organization. Just like human organizations, the composition is designed as a hierarchical, rule-based system, where different services have a position to play defined by its contracts (*responsibilities*) with other positions. The organization itself represents a business requirement abstracting the underlying service-service interactions in the form of self-managed composites (SMCs). This philosophy of designing service compositions provides the potential for better alignment of technical service abstractions with business services to model and adapt the composition as a whole. Rather than adapting the manner in which peer-to-peer service provisioning is carried out as in ESB, a specific goal-oriented composition as a whole is adapted.

The separation of management structure from functional structure also prevents management concerns getting muddled with functional code. In our approach the designer is already provided with the concepts and tools to design compositions amenable to runtime change while keeping these concerns separate.

C. Other integration approaches

VC-bus[25] promotes component/service re-use in all levels of an application including software, middleware and hardware. The approach is providing the ability to vertically bind the components in application development. The term *vertical bind/composition* is used in contrast to *application level service (horizontal) composition* as we know it. As an example, the service integration via an ESB is considered as horizontal composition. However, the approach fails to capture and represent the service-service interactions to the depth that our approach captures. As shown, the *Contractual Terms* captures the mutual obligations of two Roles of a composition. It precisely defines the types of messages, permissions and obligations, which cannot be seen in VC-Bus approach.

eFlow [26] provides a way to enact and manage composite e-services modeled as processes. The approach provides the ability to dynamically modify the processes.

⁶ <http://www.mulesoft.com>

⁷ <http://synapse.apache.org/>

⁸ <http://servicemix.apache.org>

However there is no explicit representation of peer-to-peer provisioning of services among participating entities. This representation is hidden in the work-flow based process descriptors making it difficult to adapt the composition. Also the enactment is based on an underlying process engine based on underlying services. However, with ROAD4WS the designer (Axis2 user) is capable of abstracting the underlying operations with roles and contracts giving an indirection between the execution entity (player) and the business requirement (role).

SELF-SERV[27] makes use of the concept *service communities*, which are containers of alternative services. As such once a job is received the job is delegated to appropriate member of the community. The approach shares some common characteristics, for example the ability of external services to register themselves with a community is similar to a player binding to a Role. ROAD4WS too uses similar routing mechanism to divert the messages to appropriate roles as shown earlier. However, again in this approach too there is a lack of explicit representation of service-service interactions to the depth ROAD has tackled.

VIII. CONCLUSION AND FUTURE WORK

In this paper we introduced ROAD4WS, which is an extension to *Apache Axis2* to deploy and consume ROAD self-managed composites. With this extension, Axis2 users can benefit from capabilities of deploying adaptive and self-managed service compositions in Axis2 environment, which is not available at present. The extension allows the service composite author to avoid writing lower level code to integrate web services. These composites can be managed and adapted to face changing business requirements.

As future work, we are extending the WS-Policy language to expose the semantics of contractual agreements in WSDL interfaces. Also, areas of interest such as context-awareness, service-packaging and business process modeling support are currently being investigated based on presented work.

ACKNOWLEDGMENT

This work was partially supported by the Smart Service Cooperative Research Centre. We also acknowledge the technical contribution made by J. Margathe, E. Eldwin, M. Talib and S. Kukunoor.

REFERENCES

- [1] S. Perera, et al., "Axis2, Middleware for Next Generation Web Services," Proc. International Conference on Web Services, 2006, pp. 833-840.
- [2] A. Colman and J. Han, "Roles, players and adaptable organizations," Applied Ontology, vol. 2, pp. 105-126, 2007.
- [3] A. Colman and J. Han, "Using role-based coordination to achieve software adaptability," Science of Computer Programming, vol. 64, pp. 223-245, 2007.
- [4] A. Colman, "Role-Oriented Adaptive Design," PhD Thesis. , Swinburne University of Technology, Melbourne, 2007.
- [5] A. Colman, "Exogenous management in autonomic service compositions," In Proceedings of the Third International Conference on Autonomic and Autonomous Systems 2007 (ICAS 2007), Athens, Greece, IEEE Computer Society Press, pp. 19-25, 2007.
- [6] J. King and A. Colman, "A Multi Faceted Management Interface for Web Services," Australian Software Engineering Conference, IEEE Computer Society, vol. 0, pp. 191-199, 2009.
- [7] B. Meyer, "Applying 'design by contract'," Computer, vol. 25, pp. 40-51, 1992.
- [8] J. Han and A. Colman, "The Four Major Challenges of Engineering Adaptive Software Architectures," Proc. Computer Software and Applications Conference, 2007. COMPSAC 2007. 31st Annual International, 2007, pp. 565-572.
- [9] M. Kapurige, A. Colman, and J. Han, "Controlled flexibility in business processes defined for service compositions. ," Proc. Services Computing (SCC) - In Proceedings, Washington DC., 2011.
- [10] P. Browne, JBoss Drools Business Rules: Packt Publishing, 2009.
- [11] J. King, et al., "ROADfactory 1.0: Implementation overview and usage guide. <http://www.it.swin.edu.au/personal/acolman/pub/roadfactoryTR1.pdf>," Swinburne University. 2011.
- [12] A. Colman, "Rules semantics for ROAD Self-Managed Composites. ," http://www.ict.swin.edu.au/personal/acolman/pub/ROAD_TR1102.pdf, Swinburne University, 2011.
- [13] "BEA, IBM, Microsoft, SAP, AG, Siebel Systems, Business Process Execution Language for Web Services V1.1 specification ", ed, 2003.
- [14] F. A. A. Lins, C. dos Santos, and N. S. Rosa, "Adaptive web service composition," SIGSOFT Softw. Eng. Notes, vol. 32, pp. 1-8, 2007.
- [15] O. Ezenwoye and S. M. Sadjadi, "Enabling Robustness in Existing BPEL Processes," In Proceedings of the 8th International Conference on Enterprise Information Systems, pp. 95-102, 2006.
- [16] O. Ezenwoye and S. M. Sadjadi, "TRAP/BPEL: A Framework for Dynamic Adaptation of Composite Services.," WEBIST'2007, Barcelona, Spain, 2007.
- [17] R. Akkiraju, et al., "Executing abstract Web process flows," presented at the International Conference on Automated Planning and Scheduling (ICAPS), 2004.
- [18] A. Erradi, P. Maheshwari, and V. Tosic, "Policy-driven middleware for self-adaptation of web services compositions," presented at the Middleware 2006, Melbourne, Australia, 2006.
- [19] L. Navarro, et al., "Modularization of Distributed Web Services Using Aspects with Explicit Distribution (AWED)," in On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE, ed, 2006, pp. 1449-1466.
- [20] M. Braem, et al., "Isolating Process-Level Concerns Using Padus," in Business Process Management, ed, 2006, pp. 113-128.
- [21] A. Charfi, "Aspect-Oriented Workow Languages: AO4BPEL and Applications - PhD Dissertation," Darmstadt University of Technology, Darmstadt, Germany, 2007.
- [22] K. Geebelen, S. Michiels, and W. Joosen, "Dynamic reconfiguration using template based web service composition," Proc. Proceedings of the 3rd workshop on Middleware for service oriented computing, Leuven, Belgium, 2008, pp. 49-54.
- [23] K. Rajaram and C. Babu, "Template based SOA framework for dynamic and adaptive composition of web services," Proc. Networking and Information Technology (ICNIT), 2010 International Conference on, pp. 49-53.
- [24] (2005) Sonic ESB: an architecture and lifecycle definition. Sonic White Paper.
- [25] R. Mietzner, et al., "Combining horizontal and vertical composition of services," Proc. Service-Oriented Computing and Applications (SOCA), 2010 IEEE International Conference on, 2010, pp. 1-8.
- [26] F. Casati, et al., "Adaptive and Dynamic Service Composition in eFlow," in Advanced Information Systems Engineering, ed, 2000, pp. 13-31.
- [27] B. Benatallah , et al., "Declarative Composition and Peer-to-Peer Provisioning of Dynamic Web Services," Proc. Data Engineering, CA, USA, 2002, pp. 297--308.