# A Dynamic Reconfigurable Web Service Composition Framework Using Reo Coordination Language


Soheil Saifipoor
*Member of Young Researchers Club,*
*Faculty of Postgraduate Studies,*
*Department of Computer Engineering,*
*Islamic Azad University, Najafabad branch,*
*Isfahan, Iran*
*saifipoor@iaun.ac.ir*

Behrouz Tork Ladani, Naser Nematbakhsh
*Department of Computer Engineering,*
*University of Isfahan, Isfahan, Iran*
*{ladani,nemat}@eng.ui.ac.ir*



## Abstract

*Web services are self-contained, modular units of application logic which provide business functionality to other applications via Internet connections. Several models have been used to compose Web services which are mainly served at specification level and provide static data dependent coordination processes. Hence they can not support reconfigurable dynamic coordination processes in which participant Web services and the coordination process itself will not be known explicitly prior to execution and would be determined dynamically at run time. In this paper we present a framework to coordinate Web services using Reo coordination language. Reo is a channel-based exogenous coordination language which has a formal basis and supports loose coupling, distribution, dynamic reconfiguration and mobility. Given that Web services are inherently loosely coupled and primarily built independently, the channel-based structure of Reo and its reconfigurability will provide a reconfigurable coordination mechanism for Web service composition. The proposed approach is a distributed dynamic orchestration framework which uses Reo channels as a communication means between Web services and benefits from Reo reconfiguration property to provide a dynamic coordination process. Due to data independence property of Reo, the proposed model is a data neutral framework which is mainly focused on coordination. In this paper we also present a number of case studies by using the proposed framework and investigate its pros and cons through these case studies.*


## 1. Introduction

Web services are defined as self-contained, modular units of application logic which provide business functionality to other applications via Internet connections [10]. The need for fulfilling new business requirements by available Web services and modelling long-running interactions and complex dependencies among different Web services led designers to Web service composition models. Several organizations and business process modelling communities have proposed different Web service composition models and specifications.

Besides the lack of formal background in current Web service composition models, Web services are considered to be tightly coupled at coordination definition and the coordination process is not flexible enough to select Web services dynamically and change the coordination process at run time according to predefined constraints. In some cases the coordination model is the source of major shortcomings and can not support the specified requirements.

In this paper we are going to present a Web service composition framework using Reo coordination language. Reo [1] is a formal channel-based coordination language and presents composition of software components based on notion of channels. Reo enables designers to build complex coordinators, called connectors (circuits) out of simpler ones. The simplest connectors in Reo are a set of channels with well defined behaviours. Each channel has its own precise specification and behaviour. Reo specification can be implemented and executed in a Reo middleware. Reo also provides a reconfigurable framework in which the topology of the connectors can change at run time. Reo channels are independent of the exchanged data; hence it provides a data neutral model for composing software components. There are a number of middlewares for executing Reo circuits such as ReoLite [6], Mocha [5] and Eclipse coordination tool [9]. According to formal basis, visual expressiveness and reconfigurability of Reo, we have

proposed a framework to compose Web services by Reo coordination language.

The proposed model is a distributed dynamic orchestration framework which uses Reo channels as a communication means between Web services and benefits from Reo reconfiguration property to provide dynamic reconfigurable coordination process. In this dynamic orchestration framework a number of Web services are defined as partial orchestrators and will manage Reo circuits; hence the coordination is distributed between these Web services. Other Web services are Reo-enabled Web services which are aware of Reo channels and will communicate with other Web services through Reo channels. The distribution of Reo connectors among Web services helps designers, for example, to apply Quality of Service (QoS) consideration in target Web service selection at run time. The dynamic structure of this framework enables the coordinator to select the target Web services and other coordinator Web services dynamically and change the topology of the coordination process at run time. The paper is organized as follows: Section 2 outlines Reo coordination language and its properties. Section 3 makes a discussion on using Reo for Web service composition and compares Reo with other formal and informal specification techniques. In section 4 we propose the dynamic framework for Web service composition based on Reo. In section 5 a number of case studies are presented to investigate the pros and cons of the proposed framework. In section 6 we describe the related works and finally we conclude in section 7.

## 2. Reo Coordination Language

Reo is a channel-based exogenous coordination language based on the calculus of channels [1]. Reo consists of components that are connected via complex coordinators, called connectors or circuits, which coordinate their activities. Connectors are compositionally built out of simpler ones. The simplest connectors in Reo are a set of channels with well defined behavior supplied by the users. The connector imposes a specific coordination pattern without any knowledge about the software components internal communications.

A channel has precisely two channel ends. There are two types of channel ends: sink and source. A sink channel end dispenses data out of its channel and a source channel end accepts data into its channel. A connector is a set of channels and channel ends organized in a graph of nodes and edges. Channels are joined together in a node, so, a node is a construct which consists of a set of channel ends. Reo provides two types of operations: topological –ones that allow manipulation of connector topology and IO – ones that allow input/output of data. Reo enables components to connect and perform I/O on the connector, namely read, take and write. Topological operations are join and split. The semantic of a Reo connector is defined by the composition of the semantics of its channels and nodes. Users define the semantics of channels and Reo defines the semantics of nodes. Arbab has defined a set of complete channel types in [1], namely Sync, FIFO1, SyncDrain, LossySync, and Filter. Fig. 1 shows the visual notation for these channels.

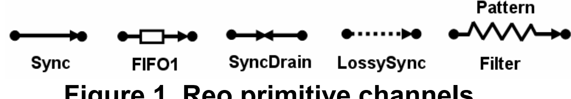

**Figure 1. Reo primitive channels**

The topology of connectors in Reo is inherently dynamic and reconfigurable; hence it helps software components to change the coordination process and their locations at run time. Reo provides a mobile model in which software components and channels can change their location without interfering the coordination process. Reo is a coordination model and as such has very little to say about the computational entities and the data which is exchanged on channels. These properties will provide a dynamic reconfigurable model in which we can compose Web services and consider them as Reo components. Reo also has a formal semantic, based on coinductive calculus of flow [1,4] and on constraint automata [2]. It also has a formal operational semantic that defines the rule of connectors in a distributed computing environment [3]. Fig. 2 shows three Reo connectors. These connectors can be trivially extended to handle any number of pairs.

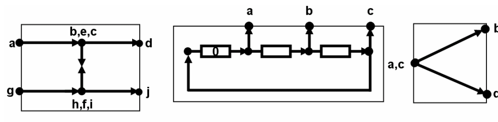

**Figure 2. Reo connectors (a) Barrier Synchronizer (b) Sequencer (c) Replicator**

As Reo is based on channels, users can define their own connectors out of simple channels to handle any coordination pattern. In Fig. 2(a) we have illustrated a barrier synchronizer connector. Here, the SyncDrain channel ‘ef’ ensures that a data item passes from ‘ab’ to ‘cd’ only simultaneously with the passing of a data item from ‘gh’ to ‘ij’ (and vice versa). If the four

channels 'ab', 'cd', 'gh', and 'ij' are all of type Sync, our connector directly synchronizes write/take pairs on the pairs of channels 'a' and 'd', and 'g' and 'j' [1]. In [1] Arbab has presented several connectors with full description of their mechanisms.

## 3. Web Service Composition and Reo Coordination Language

Web services are computational entities that are autonomous and heterogeneous (e.g. running on different platforms or owned by different organizations). Web services are described using Web service description language (WSDL [19]) and published and discovered according to predefined protocols [11]. A Web service has a specific task to perform and may depend on other Web services, hence being composite.

The composition of two or more Web services generates a new Web service which provides a collaborative behaviour for carrying out a new task. Web service composition can be *static* or *dynamic*. In static composition, Web services interact with each other on a pre-defined manner but in dynamic composition, Web services discover each other, interact and negotiate on the fly [11]. There are two approaches in static composition. In the first approach, referred as *orchestration,* Web services collaboration is coordinated by a coordinator which is a Web service. In the second approach, referred as *choreography,* there is no central coordinator but rather tasks are defined by specifying the interaction that should be undertaken by each participant Web services [12]. There are several orchestration languages such as BPEL4WS [18], BPML and some for choreography like WSCDL [17]. These languages are usually defined and standardized by business process modelling communities and lack a theoretical basis. This raises a number of challenges such as the need for guaranteeing the correct interaction of independent Web services. Consistency, completeness and deadlock free status are other issues which must be taken into consideration when a real world long-running Web service interaction is used. There are some cases in current standard Web service composition specification languages like BPEL4WS which address the ambiguity, inconsistency and incompleteness [26].

Besides the above issues, these languages are mainly served at specification level and provide static data dependent coordination processes. These models can not support reconfigurable dynamic coordination processes in which participant Web services and the coordination process itself will not be known explicitly prior to execution and would be determined dynamically at run time. Dynamic and reconfigurable coordination process for Web services has several applications such as the followings:

- Service Oriented Grid Computing [20]
- QoS consideration in Web service composition
- Semantic Web services and dynamic Web service selection [21]

On the other hand, there is no Web service composition specification language with a strong formal basis and expressive visual notation to help designers in definition and verification process. The mentioned drawbacks in current Web service composition specifications led us to introduce a dynamic distributed orchestration framework for Web service composition using Reo coordination language.

## 4. Dynamic Reconfigurable Web Service Composition Framework

In this section we are going to introduce the distributed reconfigurable framework for composing Web services using Reo coordination language. In this orchestration framework the coordination process is defined by Reo circuits. The circuits will be managed by a number of Web services which are dedicated to handling and executing the Reo circuits. Other participant Web services are Reo-Enabled ones which are aware of Reo channels and operations. These Web services can be considered as distributed components which communicate with other components through Reo channels.

In this framework we have two types of coordination process: Dynamic and Static. In dynamic coordination processes, the target Web services and the coordination process will not be known explicitly prior to run time. In static one, the target Web services and the coordination process are precisely defined prior to process execution. In this paper we mainly focus on the dynamic coordination processes. We believe that the static process is subsumed by the dynamic one. In this framework we have two categories of Web services: (1) *Reo-Enabled Web Services* (2) *Reo service provider Web Services*. Reo-Enabled Web Services (REWS) are common Web services which are defined by users and expose their business operations by their WSDL interfaces. Besides their common business logic, these Web services have a Reo channel handler which manages the Reo channels. Reo channel handler is a local Reo middleware which holds and controls Reo channels and provides the channel operations through WSDL interface of the Web

service. According to Reo terminology, we can consider this category of Web services as Reo components. These components have their own business logic, besides the Reo channel handling ability. The primary form of this category of Web services and their Reo channel handler operations are illustrated in Fig. 3. Note that the corresponding operations along with their return values and required parameters are shown in this figure.

**Reo-Enabled Web Service**
This Web service has its own operations and interfaces

Reo Channels Handler
- Accept/Reject **Connect** (*Che_ID*)
- Accept/Reject **Disconnect** (*Che_ID*)
- Data/Null **Read** (*Che_ID*)
- Data/Null **Take** (*Che_ID*)
- Accept/Reject **Write** (*Che_ID*, Data)
- Accept/Reject **Join** (*Che_ID, Che_ID*)
- Channel List **GetChannelEndsID**()

**Figure 3. Reo-Enabled Web service**

The other category of Web services in this framework is Reo service provider Web Service (RSWS). These Web services will hold Reo circuits and manage internal and external requests on channels. They have several tasks such as managing the internal circuits, sending requests to external Web services and receiving external Web service requests on channel ends.

We can define two types of RSWSs: (1) *Dynamic Reo service provider Web services* (2) *Static Reo service provider Web services*. In static RSWS, the Reo circuit is defined prior to execution and we do not need to define the circuit dynamically. Fig. 4 illustrates a static RSWS which handles a simple 2-pair SynchBarrier Reo connector.

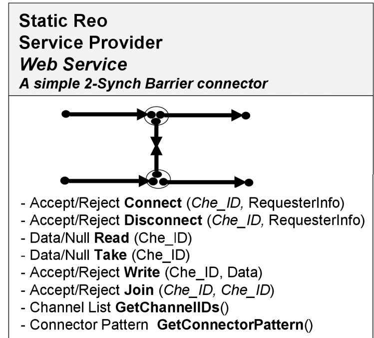


**Figure 4. Static Reo service provider Web service**

In dynamic RSWS, the structure of the Reo circuit is not defined prior to execution and the external Web services should request for constructing a circuit; hence these Web services do not have a fixed number of channel ends. According to the structure of this framework, the target Web services which are going to be coordinated by this dynamic connector structure will also be specified at runtime. Therefore we need a comprehensive internal protocol to manage such dynamic structure. This protocol will define the Reo circuit, the target Web services and the data which is exchanged on channels. Fig. 5 illustrates the required operations for such a dynamic reconfigurable coordinator.

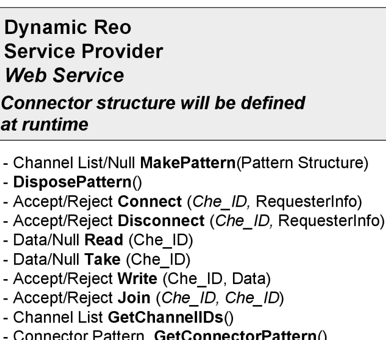


**Figure 5. Dynamic Reo service provider Web service**

According to Figures 3, 4 and 5, there are a number of operations which are common between these Web services. The following are the definition of these common channel operations:

***Connect:*** An external Web service calls this operation, passes a channel end Id and asks the target Web service to get connected to the channel end. If the target Web service accepts the request, it will acknowledge the source Web service by returning an Accept message, otherwise the target Web service will acknowledge by sending back a Reject message.

***Disconnect:*** An external Web service calls this operation, passes a channel end Id and notify the target Web service that it will release the channel end. After that, the source Web service can not operate on the disconnected channel end.

***GetChannelIDs:*** An external Web service would call this method to get the channel end Ids which are held by the target Web service. The target Web service will return the list of its channel ends in response to this request.

***Join:*** An external Web service calls this operation and passes two channel end Ids and asks the target

Web service to merge them together into a new node. One of these channel ends should be located on the target Web service. If the target Web service accepts the operation, it will acknowledge by sending back an Accept message.

In this framework, Web services communicate with each other through Reo channels. Therefore they should construct the required channels, send the channel Ids to target Web service, ask the Web service to get connected to the channel ends and finally communicate through the channels. The channel communication is done by means of Reo channel operations. The behaviour of the provided operations such as Write, Read and Join which are exposed by these Web service WSDL interfaces depend on the type of the channel. We consider two categories of Reo channels: Synchronous and Asynchronous channels. According to channel types, we define different mechanisms for operation calls in these categorises.

For synchronous channels such as *Sync, Filter, SyncDrain* and *LossySync* the operations on channel ends are implemented by request-response model of Web service operation call. Assume a REWS (called source Web service) which holds the source end of a Sync channel. The sink end of this channel is located on another REWS (called target Web service). When an active instance in the source Web service calls the Write operation on the source channel end, this call will be mapped by Reo channel handler to a Write operation on the target Web service. The Write operation invocation has channel Id and data as its parameters. It will notify the target Web service that a Write operation is triggered on the source channel end. If there is any Take or Read operation suspending on the sink channel end, the Write method will pass the data to the suspended active instance and acknowledge the source Web service by sending back an Accept message. On the other hand, if there is no Take or Read operation suspending on the sink channel end, the Write operation will acknowledge the source Web service by sending back a Reject message; therefore the active instance in source Web service will become suspended. The Take operation by an active instance in the target Web service will also call the Take operation on the source Web service. If there is any Write operation suspending on the source channel end, the Take method will send back the suspended data to the target Web service. Otherwise if there is no Write operation suspending on the source channel end, it will acknowledge the target Web service by sending back a NULL message. This way, the Take operation in the target Web service will become suspended.

For asynchronous channels like FIFO, we have the following operational mechanism. In these channel types such as FIFO1, there is a buffer in the source Web service which holds the data. Any Write operation on the source channel end by an active instance in the source Web service will call the Write operation on the target Web service. If there is a Take operation suspending on the sink channel end, the Write operation will hand over the data and acknowledge the source Web service by returning an Accept message. If there is no Take operation suspending on the sink channel end, the Write operation will return a Reject message as its return value and save the data in the channel buffer.

When an active instance in the target Web service calls the Take or Read operation on the sink channel end, Reo channel handler will map the call to a Take or Read operation on the source Web service. After that, the source Web service will check the channel buffer. If there is a data item in the channel buffer, the Take or Read operation will return the data item to the target Web service as the return value of the operation. When the buffer is empty, the Take or Read operation will acknowledge the target Web service by a NULL message and the active instance in the target Web service will become suspended. We can also define operational mechanisms for other channel types and use them as a means of communication between Web services. Table 1 shows the service-oriented mechanisms for other primitive Reo channels.

The Reo channel handler in REWSs manages the internal channel ends and maps the internal requests to Web service operations according to the channel definition. As illustrated in Figures 4 and 5, operations in RSWSs are the same as REWSs. But as the coordination process is not defined in these Web services and should be constructed at run time, a number of specific operations are defined in these Web services. These Web services accept the coordination process which is defined by Reo circuits and coordinate the participant Web services. The following describe these specified operations:

***MakePattern:*** This method will receive a Reo connector pattern and construct the circuit. After constructing the Reo circuit, the Web service will ask the participant Web services to get connected to the constructed channel ends to read or write data on them.

***DisposePattern:*** This method will cause the target dynamic RSWS to discard the Reo circuit. In this way, the target Web service will become ready for constructing new requested patterns.

**Table 1. Service-oriented mechanism for implementing operations of primitive Reo channels**

| Reo channel | Service-oriented operational mechanism |
|---|---|
| **SyncDrain**<br>Source Web service a ●→←● b Target Web service | -Write on 'a' by an active instance in the source Web service will call the Write operation on the target Web service. If there is any Write suspending on 'b', the operation will return an Accept message. Otherwise, it will return a Reject message and the Write operation on 'a' will become suspended.<br><br>-Write on 'b' by an active instance in the target Web service will call the Write operation on the source Web service. If there is any Write suspending on 'a', the operation will return an Accept message. Otherwise it will return a Reject and the Write operation on 'b' will become suspended. |
| **LossySync**<br>Source Web service a ●·····→● b Target Web service | -Write on 'a' by an active instance in the source Web service will call a Write operation on the target Web service. The Write operation will return an Accept message in any status of 'b'. If there is any Take suspending on 'b', the active instance in the target Web service will get the data. If not the data will be discarded. |
| **Filter**<br>Source Web service a pattern b Target Web service | -This channel has the same mechanism as Sync channel except that the Write on 'a' by an active instance in the source Web service will call a Write operation on the target Web service if the data matches the channel pattern. |

According to the structure of dynamic RSWSs, they need an internal Reo middleware for constructing and executing the requested Reo circuits. As the Reo connectors and channels are central in these Web services, they can also include verification tools to check the correctness of the requested connector prior to execution. In the following section we will present three coordination examples using the proposed framework.

## 5. Case Studies

In this section we are going to present three case studies for Web service composition using the proposed framework. In Example 1, we review a simple communication between two Web services through a Reo Sync channel. In Example 2, we will outline the dynamic construction of Reo connectors to make a coordination process. Example 3 constructs a complete purchase order process by the proposed framework. The definition of this process is presented in [25] by BPEL4WS specification language.

**Example 1:** In this simple example, we're going to present the construction of a Reo Sync channel between two REWSs. As shown in Fig. 6, Web service1 receives the request of the external user by Web service common operations. According to the coordination process which is defined for handling the request, Web service1 will create a Sync channel. After constructing the channel, it calls the Connect method on Web service2, passes the created channel end Id to Web service2 and asks it to get connected to the channel end. Now the connection is constructed between these two Web services and they can communicate by this channel.

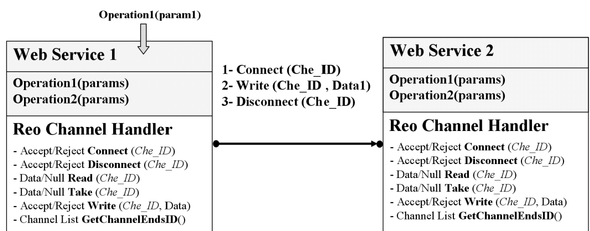

**Figure 6. A Sync channel between two Reo-Enabled Web services**

Whenever the active instance in Web service1 makes a Write operation on the source channel end, the Reo channel handler consequently will call the Write operation on Web service2. The data which is sent by the Write operation can be a simple variable, a method call on Web service2 by its parameters or a coordination process definition. We can see that the

coordination structure which is a simple Sync channel is independent of the exchanged data.

**Example 2:** The following example illustrates an order system in which the customer sends a request to the Order Web service. The Order Web service first asks the Authentication Web service to check the user information. The Authentication Web service will check the user information and response the Order Web service. After confirming the user information the Authentication Web service will return a 'Yes' response to the Order Web service. On receiving the 'Yes' response, the request will be sent to the Logging Web services concurrently to log the order data. The Reo circuit for this coordination process is illustrated in Fig. 7.

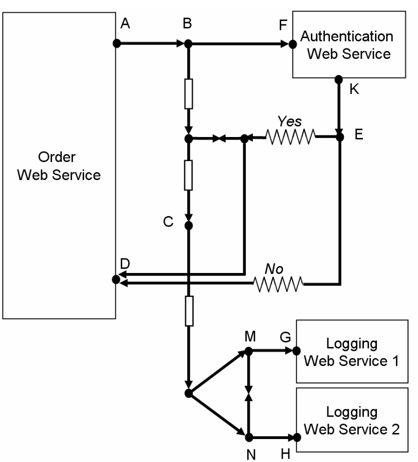


**Figure 7. Reo circuit for Ordering process**

In this example the Reo circuit is going to be constructed by a dynamic RSWS. It will also ask the Authentication Web service and Logging Web services to get connected to their corresponding channel ends. The coordination process and participant Web services are illustrated in Fig. 8.

As Fig. 8 shows, when the external user sends its request to Order Web service, it will call the MakePattern method on a dynamic RSWS and pass the coordination process structure which contains the Reo circuit and information of the participant Web services in this process. The dynamic RSWS can be selected by the Order Web service at runtime. After receiving the coordination process, the dynamic RSWS will ask the target Web services to get connected to their corresponding channel ends. When the dynamic RSWS constructs the Reo circuit and connects the channel ends to target Web services, it will return the channel end Ids to Order Web service. The Order Web service will write the order data on the Sync channel end 'A' to start the process.

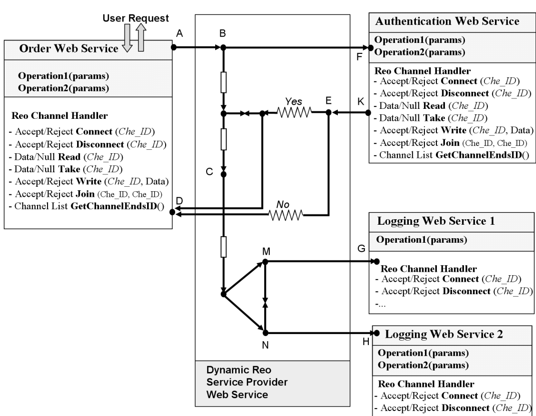


**Figure 8. Participant Web services in Ordering process**

On receiving the order data by dynamic RSWS, it will execute the Reo circuit, write data on 'BF' channel by calling the Write method on Authentication Web service. Assuming that the user information is confirmed, the Authentication Web service will write 'YES' value on 'KE' channel. This way, the dynamic RSWS will continue the execution of the circuit and write concurrently on 'MG' and 'NH' channels. Then the Logging Web services will receive the request and do the logging.

**Example 3:** The following example is an orchestration process which is defined in [25] by BPEL4WS specification language. In this PurchaseOrder process, there are four participant Web services: Invoice Web service, Shipping Web service, Order Web service and Schedule Web service. The Order Web service is the process initiator which on receiving the purchase order from a customer, initiates three paths concurrently; calculating the final price for the order, selecting a shipper, and scheduling the production and shipment for the order [25]. While some of the processing can proceed concurrently, there are control and data dependencies between the three paths. In particular, the shipping price is required to finalize the price calculation, and the shipping date is required for the complete fulfilment schedule. When the three concurrent paths are completed, invoice processing can proceed and the invoice is sent to the customer [25]. Fig. 9 illustrates the whole coordination process which is constructed by Reo circuits.

For executing this process, the Order Web service selects a dynamic RSWS at runtime and passes the Reo circuit to this Web service by calling its MakePattern method. The Web service selection can be done according to the process execution load or QoS properties. The dynamic RSWS will construct the Reo

circuits and pass the channel end Ids to the Order Web service. After that, the Order Web service will write the order information to the Sync channel and pass the order data to the dynamic RSWS. RSWS will communicate with other Web services to fulfill the process. Finally the result will be sent back by a FIFO1 channel to Order Web service and consequently the Order Web service will return the result to the external customer. Fig. 10 shows the structure of the dynamic RSWS and participant Web services in PurchaseOrder process.

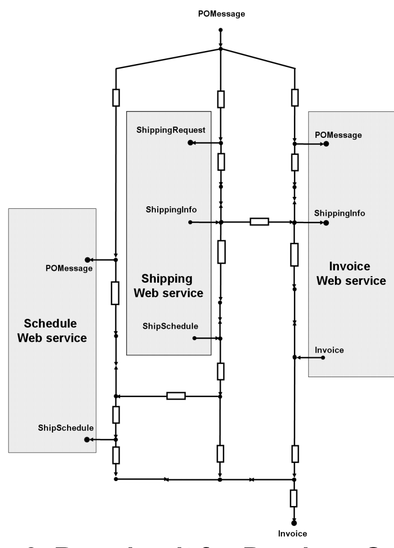


**Figure 9. Reo circuit for PurchaseOrder process**

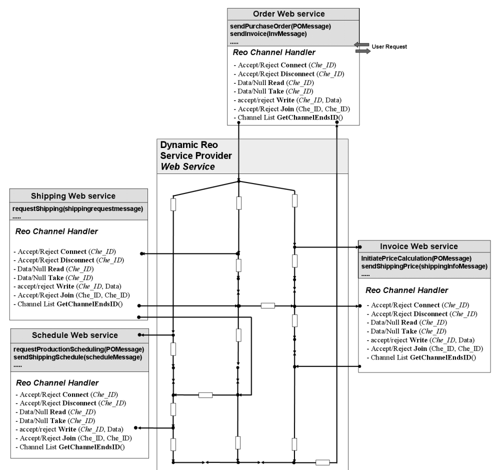


**Figure 10. Participant Web services in PurchaseOrder process**

The following paragraphs will present the pros and cons of the proposed framework according to the case study in Example 2:

**Non-functional properties:** In this example the Order Web service can select a different dynamic RSWS for managing the coordination process. The Web service selection is done dynamically, according to some QoS consideration. For defining the non-functional and QoS properties of dynamic RSWSs and REWSs, we need a Web service QoS properties specification. This specification should also contain Reo channels QoS properties. In this framework, reconfiguration is a mechanism for fulfilling non-functional requirements. The Ordering Web service can change the structure of coordination process to serve non-functional QoS properties, such as performance and cost.

**Composition correctness:** In this example as constructed Reo circuit by dynamic RSWS is central and located in a Web service, we can check its correctness. But in a more general form of this model, where different parts of the coordination process are located on different RSWSs, checking the whole coordination process is a challenging task and requires more consideration.

**Composition scalability:** In this example as the process is defined by Reo connectors, the Order Web service can easily extend the coordination process to add more Logging Web services to log the order data. As Reo connectors are made out of simpler ones, the connector structure can be extended to coordinate more Web services without changing the whole process; hence it is scalable.

**Loose coupling between Web services:** In current Web service composition models such as BPEL4WS, the coordination process, participant Web services and exchanged data are tightly coupled, hence the coordination process is a data dependent one. But in the proposed framework, the coordination process is totally independent of the exchanged data. This property will help designers to easily maintain and extend the coordination process. In this example, the Order Web service can easily select a new dynamic RSWS as a coordinator. It also can extend the coordination process to have more Logging Web services or select a new Authentication Web service according to the user profile category.

**Data independence**: In this example the Order Web service has defined the coordination process without any data consideration for target Web services. This property provides a dynamic mechanism on Web service selection in which the target Web service can be chosen without changing the coordination process definition. In the proposed framework, Web services use Reo channels for communication, hence the coordination and interaction between Web services are considered to be in a data neutral format. But as the

framework is dynamic and reconfigurable, Web services need to communicate with other participant Web services, so the coordination structure, method calls and parameters need to be exchanged to make the coordination process applicable. Thus we need to define a protocol for coordination process, method calls and parameters on channels. The coordination process is needed to be sent to dynamic RSWSs to build the Reo connectors and connecting the channel ends to their corresponding Web services. We can use the proposed structure in [13] to define Reo circuit by XML, but this XML structure should be extended to hold the definition and properties of the participant Web services.

**Tool support for execution and verification:** Any composition approach must be supported by software tools. These software tools usually provide implementation, verification and execution. In this framework we need a local Reo middleware in RSWSs to manage and execute Reo circuits. For these local Reo middlewares we can use ReoLite [6] to run Reo circuits but it should be extended to manage service-oriented mechanism for communication between Web services. For constructing such a service-oriented middleware we can use, the proposed models in [22,23]. As Reo circuits are central in dynamic RSWSs, we can also verify their structure prior to execution by proposed tools and models in [7, 8, 14].

## 6. Related Works

Reo as a channel-based exogenous coordination language supports loose coupling, distribution, dynamic reconfiguration and mobility. Because of these properties, currently it has received much attention as a coordination language for composing Web services. Several researches are available in the context of Web service composition by Reo coordination language [15,16,22,24].

In [16] Arbab et al. defined a coordination model for Web services in which Reo channels are used for communication between Web services. In [16] the Web is transferred from a collection of clients and servers into an object space of distributed objects. This model emphasizes on common operations for Reo channels and do not involve Web services directly in the coordination process. In comparison with [16], the proposed dynamic reconfigurable framework in this paper defines the whole coordination process by Reo circuits which are distributed between RSWSs.

To make Reo applicable in software components like Web services, we need a Reo middleware. In [22] Arbab et al. introduced a Reo coordination middleware model for enabling Web services to manage Reo circuits. The proposed middleware is based on Mocha [5] structure. In comparison with [22] where Reo operations are implemented by Java RMI, in this paper Reo operations are defined by Web service operation mechanisms. Another issue on Web service composition is dynamic Web service selection and automatic composition. In [15] Heydarnoori et al. worked on developing an automated deployment planner for the composition of Web services as software components using Reo coordination middleware in a distributed environment. In this work they have introduced an automated composition model for Web services and defined the required specification for target environments, Web services and user defined constraints. In [15], Reo channels are located on different servers in such a way that it will satisfy a number of QoS constraints.

Besides reconfigurability and loose coupling properties, Reo has a formal semantic which is based on constraint automata [2,7,24]. In [24] Arbab et al. investigated the possibility of representing the internal and external behaviours of Web services by constraint automata. They also used Reo circuits for choreography of Web services and illustrated a number of case studies on Web service choreography by constraint automata. In comparison with our proposed framework, [24] has emphasized on formal basis of Reo but we mainly focused on Reo channels and their related operations in Web services environment. The proposed model in [24] can be considered as a formal basis for composing Web services by Reo circuits.

## 7. Conclusion

In this paper we proposed a framework to compose Web services using Reo coordination language. Reo is a channel-based coordination language with a formal basis. The proposed framework benefits from Reo reconfigurability and data independent structure. In this framework, the coordination process is constructed out of Reo circuits and managed by Reo service provider Web services. These Web services hold Reo circuits and manage Reo channel operations; hence we can consider them as orchestrators. Other Web services are Reo-Enabled which will delegate their coordination processes to Reo service provider Web services. Reconfigurability and data independent structure of this framework makes the coordination process dynamic enough to select participant Web services at run time. This property also helps the coordinator to consider QoS properties in target Web service selection and topological changes at runtime.

## 8. References


[1] F. Arbab, “Reo: A channel-based coordination model for component composition”, *Mathematical Structures in Computer Science, 14(3)*, 2004, pp.1–38.

[2] F. Arbab, C. Baier, J. Rutten, and M. Sirjani,”Modeling component connectors in Reo by constraint automata”, *Electronic note in Theoretical Computer Science*, vol.97, No.22, July 2004, pp. 25-46.

[3] F. Arbab, C.T.H. Everaars, D.F. De Oliveira Costa, and N.K. Diakov,”A distributed computational model for Reo”, *Technical Report, REPORT SEN-E0601*, CWI, the Netherlands, February 2006.

[4] F. Arbab, and J. Rutten,“A coinductive calculus of component connectors”, *In Processings of 16th International Workshop on Algebraic Development Techniques (WADT 2002)*, *Lecture Notes in Computer Science 2755, Springer*, 2003, pp.35-56.

[5] F. Arbab, F.S. deBoer, M.M. Bonsangue and J.V. Guillen-Scholten, “MoCha: a middleware based on mobile channels”, *In Proceedings of 26th International Computer Software and Application Conference (COMPSAC02)*, IEEE Computer Society Press, 2002.

[6] D. Clarke, ReoLite: Reo coordination engine, CWI, the Netherlands, http://homepages.cwi.nl/~dave/reolite/.

[7] A. Ghadiri, ”A Tool for Constraint Automata Join”, *BS project, Technical report*, *ECE Department University of Tehran*, Tehran, Iran, 2004.

[8] F. Ghassemi, and S. Tasharofi, “A Tool for Converting Reo Circuit to Constraint Automaton”, *BS project, Technical report*, *ECE Department University of Tehran*, Tehran, Iran, 2005.

[9] C. Koehler, Eclipse Coordination Tools, CWI, the Netherlands, http://homepages.cwi.nl/~koehler/etc.

[10] B. Srivastava, and J. Koehler, “Web service composition - current solutions and open problems”, *In Proceedings of ICAPS'03 Workshop on Planning for Web Services*, Trento, Italy, June, 2003.

[11] M.H. Ter Beek, A. Bucchiarone, and S. Gnesi, “A Survey on Service Composition Approaches: From Industrial Standards to Formal Methods”, *Technical Report 2006-TR-15, ISTI, Consiglio Nazionale delle Ricerche*, 2006.

[12] J. Koehler, G.Tirenni, and S. Kumaran, ”From business process model to consistent implementation: a case study for formal verification methods”, *In Proceedings of the 6th International Enterprise Distributed Object Computing Conference*, Lausanne, Switzerland, September 2002, IEEE, pp.96-106.

[13] N.K Diakov, “Reo XML schema”, *Internal report, Software Engineering Department,CWI*, the Netherlands, 2004.

[14] S. Kl¨uppelholz, and C. Baier, “Symbolic Model Checking for Channel-based Component Connectors”, *In Proceedings of 5th Int. Workshop on the Foundations of Coordination Languages and Software Architectures (FOCLASA’06)*, 2006, pages 19–36.

[15] A. Heydarnoori, F. Mavaddat, and F. Arbab,”Towards an Automated Deployment Planner for Composition of Web Services as Software Components”, *Electronic Notes in Theoretical Computer Science 160, Elsevier* (2006) 239–253

[16] F. Arbab, T. Lemniotes, and G.A. Papadopoulos, “Coordinating Web services using channel based communication”, *In Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC’04)*, IEEE, 2004.

[17] W3C, Web Service Choreography Description Language (WS-CDL) 1.0. http://www.w3.org/TR/ws-cdl-10/.

[18] K. Liu, D. Roller, D. Smith, F. Curbera, H. Dholakia, T. Andrews, Y. Goland, J. Klein, F. Leymann, S. Thatte, I. Trickovic, and S. Weerawarana, “Business process execution language for web services version 1.1”, 2003, http://www-106.ibm.com/developerworks/webservices/library/ws-bpel.

[19] W3C. Web Service Description Language (WSDL) 1.1. http://www.w3.org/TR/wsdl.

[20] X. Wang, K. Yue, and J.Z.H. Aoying, ” Service selection in dynamic demand-driven Web services”, *In Proceedings of IEEE International Conference on Web Services, 2004*, Volume, Issue, 6-9 July 2004 pp. 376 - 383.

[21] K. Fujii, and T. Suda,”Semantics-based dynamic service composition”, *Selected Areas in Communications, IEEE Journal on Volume 23, Issue 12*, Dec. 2005 pp. 2361 - 2372

[22] F. Arbab, and N.K. Diakov,“Compositional construction of Web services using Reo”, *In Proceedings of International Workshop on Web Services: Modeling, Architecture and Infrastructure (ICEIS 2004)*, Porto, Portugal, April 13-14, 2004.

[23] S. Saifipoor, B. Tork Ladani, and N. Nematbakhsh, ”Modeling Web services composition using Reo coordination language”, *In Proceedings of International Conference on Internet Technology and Secured Transactions 2007 (ICITST2007),*London, United Kingdom, 21-23 May 2007.

[24] F. Arbab, and S. Meng, “Web services choreography and orchestration in Reo and constraint automata”, *In Proceedings of the 2007 ACM symposium on Applied computing, Seoul, Korea*, 2007, pp.346-353.

[25] OASIS, “Web Services Business Process Execution Language Version 2.0”, Committee Draft, 25 January, 2007, available at http://docs.oasis-open.org/wsbpel/2.0/

[26] Web Services Business Process Execution Language Technical Committee. WS BPEL issues list, (2006), available at:http://www.choreology.com/external/WS_BPEL_issues_list.html.