

Modelling Web Service Composition Using Reo Coordination Language

Sohail Saifipoor

Member of Young Researchers Club,

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 emerging technology for distributed computing and E-business interactions. A Web service represents a unit of business logic that an organization exposes on the World Wide Web. The next idea in Web service technology is the composition of Web services into complex ones and it has received much interest to support business-to-business applications. The current Web service composition approaches have been introduced by business process modelling communities and often lack a theoretical and formal basis. This fundamental drawback sometimes results in difficulties in long running real world Web service composition. In this paper we are going to propose two models for Web service composition based on 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. We first introduce Reo and then propose two models based on Reo. The first model focuses on Web service orchestration and the second model introduces a novel service oriented Reo middleware. We also discuss pros and cons of the proposed models.

Keywords Reo coordination language, Web service composition, formal methods

1 INTRODUCTION

Web services are defined as self-contained, modular units of application logic which provide business functionality to other application via Internet connection [15]. They enable dynamic connection and automation of business process for enterprise application integration and B2B integration. This reflects the need for explicit modelling of long-running interaction and complex dependencies among different Web services to fulfil the business requirements. The Web service composition serves this need and fulfils the complexity of business processes execution. Several organizations and business process modelling communities have proposed different Web service composition models and specifications. The absence of formal semantic in these models is the source of major problems. Some well-know problems related to Web service composition are how to verify the correctness and how to express the composition in a formal and expressive specification [16]. Recently several formal models have emerged which help designers to verify the correctness of Web services and their compositions [17, 12, 13, 19].

In this paper we are going to use Reo coordination language [2] to compose Web services. Reo is a formal channel-based coordination language and presents composition of software components based on notation 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 behaviors. Each channel has its own precise specification and behaviour. Reo specification can be implemented and executed in a Reo middleware. There are a number of middlewares for executing Reo circuits such as ReoLite [9] and Mocha [4]. Due to its formal basis and visual expressiveness, Reo can be used to define the coordination of Web services.

In this paper we are going to introduce two models for composing Web services using Reo coordination language. The first model is based on a predefined Reo circuit. The coordination patterns are presented by Reo circuit and executed in a central Reo middleware. In this model, referred as *Central Orchestration* model, we present a central Reo coordination middleware for managing and modelling Web service composition. The proposed structure which is derived from [5] consists of a central Reo middleware and a number of adapters. The central Reo middleware executes a Reo circuit which represents a coordination pattern. We also introduce two types of adapters which handle data

transfers between Web services. We use BPEL4WS coordination patterns to define Reo circuits. We can consider this model as a Web service orchestration framework.

In the second model, referred as *Distributed Service Oriented Reo Middleware*, we present a novel Reo service oriented middleware. In this middleware, special Web services manage Reo channels and expose valid operations to external users (Web services). In contrast with the first model, Web services are aware of Reo channels and their operations.

The structure of the paper is organized as follows. Section 2 describes the impact of formal methods on Web service composition. Section 3 outlines Reo coordination language and its properties. Section 4 makes a discussion on using Reo for Web service composition and compares Reo with other formal and informal specification techniques. In section 5 we propose two models for Web service composition based on Reo and in section 6 we discuss and compare the proposed models and finally we present conclusion in section 7.

2 WEB SERVICE COMPOSITION AND FORMAL METHODS

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 appropriate Web service description languages, published and discovered according to predefined protocols [16]. A Web service has a specific task to perform and may depend on the 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 [16]. 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 service [14]. There are several orchestration languages such as BPEL4WS, BPML and some for choreography like WSCDL. 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 [18].

It is for the above reasons that formal methods should be used. Recently a variety of concrete proposals have emerged to formally describe, compose and verify Web services. The majority of these are based on state-action models (e.g. labelled transition systems, timed automata, and Petri nets) or process models (e.g. π -calculus and other calculi) [16]. The formal methods can help to verify whether a Web service matches its requirements and works correctly or not. They also help to find whether two Web services are equal or not [14].

Most of the proposed techniques are used to verify the other informal specifications like BPEL4WS specifications and are not used directly in specification process. 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 specification led us to introduce a number of models for Web service composition using Reo coordination language.

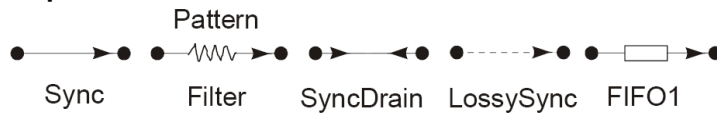
3 REO COORDINATION LANGUAGE

Reo is a channel-based coordination language which has a strong formal basis and promotes loose coupling, distribution, mobility, exogenous coordination and dynamic reconfiguration of coordination pattern [5]. Reo enforces a model that defines how designers can build complex coordinators, called *connectors* out of simpler ones [2]. The simplest connectors in Reo are a set of channels with well defined behaviors supplied by the users. 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 [2].

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 [2], namely Sync, Filter, SyncDrain, LossySync, and FIFO-1. Reo provides a comprehensive visual notation for its channels. Figure 1 shows the visual notation for some primitive Reo channels.

Reo also has a formal semantic, based on coinductive calculus of flow [8] [1] and on constraint automata [3]. It also has a formal operational semantic that defines the rule for computing connectors in a distributed computing environment [6].

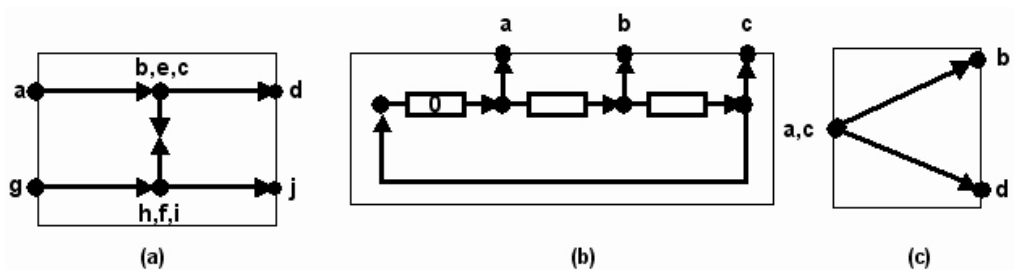
Figure 1. Reo primitive channels



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* [2]. Topological operations are *join* and *split*; because of space limitation we do not explain them here.

Each connector in Reo imposes a special coordination pattern on the entities (e.g., components) that perform I/O operations through that connector without the knowledge of those entities [2]. Some of Reo connectors represent coordination patterns that are usually used in Web service composition specifications, for example Reo Barrier Synchronizer connector which is presented in [2], can define a parallel pattern, and Sequencer connector [2] can be used to model a sequence in a process flow. Figure 2 shows three Reo connectors. These connectors can be trivially extended to handle any number of pairs. As Reo is based on channels, users can define their own connectors out of simple channels to handle any coordination pattern.

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



In Figure 2.a we have illustrated a barrier synchronization 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 [2]. In [2] Arbab has presented several connectors with full description of their mechanisms.

4 WHY WEB SERVICE COMPOSITION USING REO?

As we mentioned in Section 2, some formal methods have been introduced for verification of Web service composition. In contrast with these formal techniques which are used in verifying the composition correctness, Reo satisfies specification and simulation needs besides the verification. In [7] Arbab et al. introduced the specification, simulation and verification of Reo connectors' behaviour. Also recent work on comparing Reo and Petri nets shows that one can relatively easy transform existing Petri net to Reo circuits, while the opposite proves to be difficult [20]. Petri net and other related models can be viewed as specialized channel-based models that incorporate certain specific primitive coordination constructs [20]. These properties of Reo shows its verification and specification ability in defining the coordination of Web services.

In comparison with current Web service composition languages like BPEL4WS and WSCDL, Reo provides dynamic reconfiguration of the coordination pattern at runtime and a comprehensive visual notation. Dynamic reconfiguration of coordination pattern helps designers to fulfil non-functional requirements at runtime. Furthermore visual notation makes the coordination pattern more understandable. Also Reo channel-based structure enables designers to define their own channels and coordination patterns out of simple channels according to their requirements. This property results in extensibility and scalability of the coordination specification. This property is not supported in current composition languages and they are usually restricted to a subset of process flows and coordination patterns and designing new and complex coordination patterns in these languages is a challenging task.

In this way, the inherently dynamic topology of connectors and Reo formal background and very liberal notation of channels make Reo more general and hence more powerful in definition and verification of coordination specification.

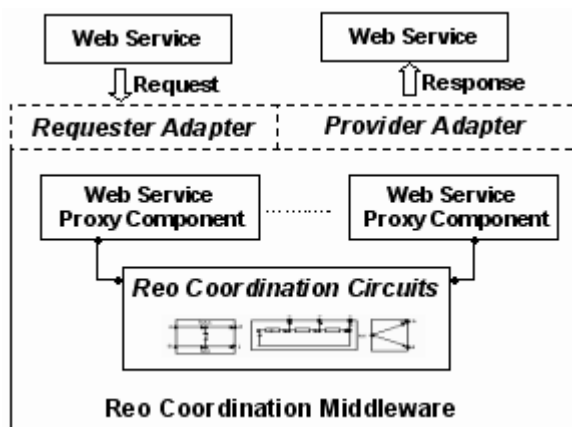
5 MODELLING WEB SERVICE COMPOSITION USING REO

In this section we are going to introduce two models for composing Web services using Reo coordination language. The base idea in the first model, which is called *Central Orchestration* model, is derived from [5]. We have extended the idea and expressed Web service composition processes by Reo circuits. The second model introduces a new service oriented Reo middleware which can be used to compose Web services. In the following subsections we introduce and discuss the models.

5.1 Central Orchestration Model

The Central Orchestration model introduces an orchestration structure for composing Web services using Reo coordination middleware. In this model Reo circuits are used to define a coordination pattern and these circuits are managed by a Reo coordination middleware. Figure 3 shows the structure of the Central Orchestration model.

Figure 3. Central Orchestration Model



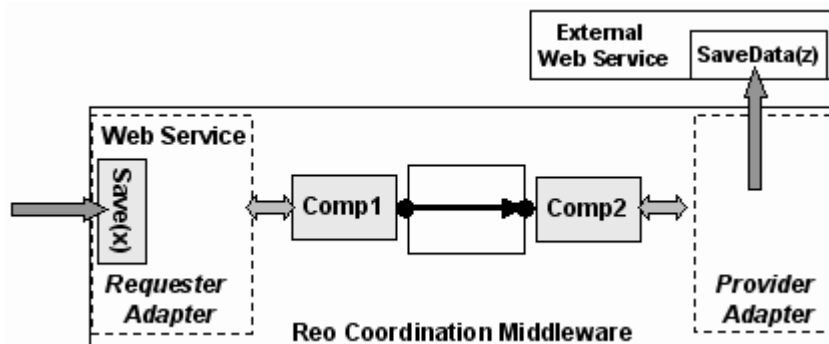
This model consists of a central Reo coordination middleware (not distributed) and two types of adapters. The Reo middleware is a structure with circuits and Web service proxy components and is responsible for managing the circuits' process. Web service proxy components manage the channel end operations and communicate with adapters to send and receive requests through channels. The adapters are responsible for managing external requests and transferring Web service proxy component requests to external Web services.

We have used two types of adapters, Provider Adapter and Requester Adapter. Requester Adapter is a Web service which provides operations for external Web services. The operation calls on Requester Adapter are mapped to Web service proxy components and consequently they do *write* or *take* operations on their channel ends. Provider Adapter sends the Web service proxy components responses to external Web services.

The main advantage of this model is that, Web services are not aware of the Reo circuits and Web services can be coordinated by a predefined circuit. Due to Reo channel-based structure, the circuits can be extended to manage more complex coordination patterns.

Example1) Simple Sync Channel: Figure 4 shows a simple *sync* channel in this model. As illustrated in Figure 4, the Requester Adapter is a Web service which has a "Save(x)" operation and external users (or Web services) can call it. This call is transferred to Web service proxy component Comp1. Comp1 makes a *write* operation call on the source end of the channel and as Comp2 is suspended, any *write* operation by Comp1, lets the data be passed through the channel. At this time Comp2, calls an operation on Provider Adapter and the adapter calls the related operation such as "SaveData(z)" on the external Web service.

Figure 4.A sync channel in Central Orchestration model

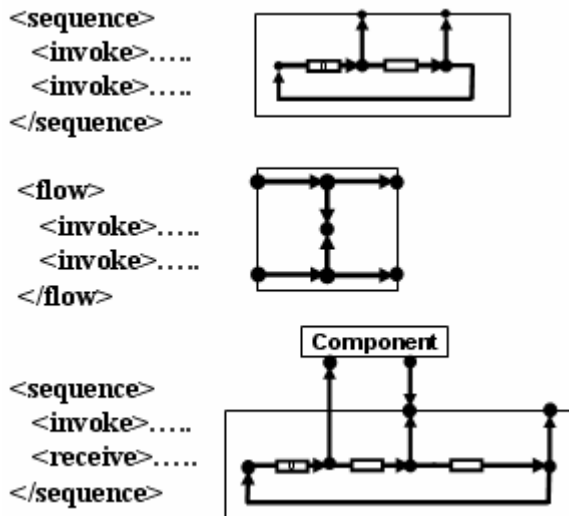


Now we try to define the composition process by Reo circuits and map operation calls and their parameters to channels. The steps in this model are as the followings: (1) Defining the coordination process by Reo circuits (2) Defining the interactions between Web service proxy components and Web services.

Because of simplicity, we have chosen BPEL4WS coordination patterns to construct Reo circuits. The basic idea behind this process is to use the concept of each BPEL4WS pattern and define its corresponding Reo connector. Then subsequent receives and invokes in BPEL4WS process are mapped to a simple sync channel or a syncdrain channel. This approach is a modular one in which we can construct a complex coordination pattern out of simpler ones. Figure 5 shows a number of patterns in BPEL4WS and their corresponding Reo circuits. The selected BPEL4WS process patterns are *sequence*, *parallel* and *invoke-receive* flow. The patterns are mapped to Reo circuits and will be used by Reo coordination middleware to orchestrate Web services.

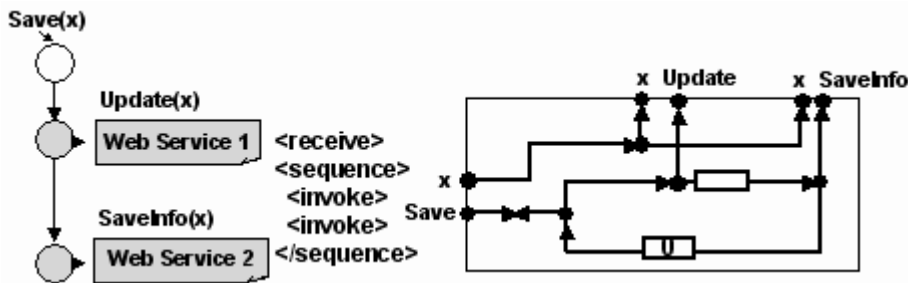
Reo is a coordination model and has very little to say about the computational entities and the data in coordination process. Due to this property of Reo, there is no data in circuits of Figure 5. We propose a data passing mechanism based on channels and map the operations and their parameters to channel end operations. By considering the orchestration process as a sequence of operations, we can define a *sequencer* circuit to handle the process flow. Mapping operations and their messages is illustrated through the following example.

Figure 5. BPEL4WS flow patterns and their corresponding Reo circuits



Example2) BPEL4WS Sequencer Circuit: Assume a composition process in which a request should be sent to two Web services and the data should be saved in those services sequentially. Figure 6 shows the process in BPEL4WS and its corresponding Reo circuit. In this figure, the sequential operation calls are mapped to a Reo *sequencer* circuit and the parameters are passed through a separate *replicator* connector. Using a separate channel for transferring parameters is feasible but when multiple parameters with different formats are used, we need to define a comprehensive protocol for transferring parameters on channels.

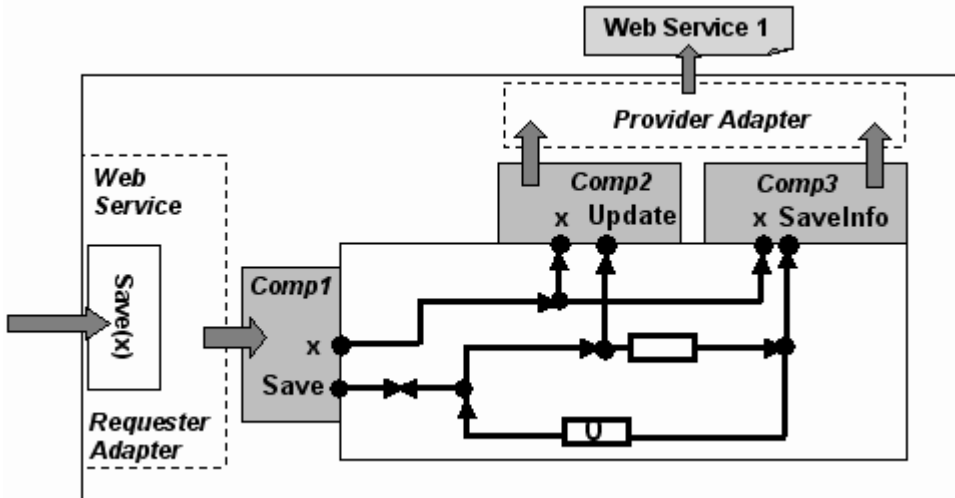
Figure 6. A simple sequence flow in BPEL4WS and its corresponding Reo circuit



After defining the circuit, now we are going to connect the channel ends to components and make connections between Web service proxy components and Web services. We use the proposed middleware structure to complete this phase. As illustrated in Figure 7, the Requester Adapter is a Web service which provides operations for external Web services and maps requests to components. Components are responsible for managing channel ends and communicating with adapters. They also do channel end operations for receiving and sending data through channels.

This model is scalable enough to handle more Web services and we can extend the circuits to manage more Web services. Besides these advantages, defining components responsibility to manage parameters and operation calls needs a flexible approach which must be taken into consideration.

Figure 7. Sequence circuit and Web service proxy components in Central Orchestration model



5.2 Distributed Service Oriented Reo Middleware

In previous model, the Reo coordination middleware was considered to be central and responsible for managing the Reo circuits. In this section we introduce another model for composing Web services by Reo which is based on a service oriented distributed Reo middleware. Reo middlewares such as ReoLite [9] are based on local components and channels. On the other hand, Mocha [4] as a distributed Reo middleware does not support service oriented structure. A service oriented model helps to coordinate any software components and can be used to compose Web services too.

In this model, channels ends and their operations are managed by Web services and channels are not centralized in a local site. This structure can use the potentials of all available distributed sites' resources and hence manages the composition efficiently. In this model Web services provide channels and related operations on channel ends and communicate directly through channels to manage the coordination process.

We have two different Web services in this model: (1) Web services which contain single channel ends (Channel End Web service) (2) Web services which contain Reo Mixed nodes (Mixed Node Web service). Figure 8 shows the proposed Web services. The Channel End Web service in Figure 8 provides a simple Reo *sync* channel and the Mixed Node Web service provides a Reo *replicator* connector. Each Channel End Web service provides Reo channel operations and exposes a WSDL interface for operations on its channel ends, so the external components can call them and request for operations on channel ends. A Mixed node Web service manages a Reo mixed node. Reo mixed nodes consist of different channel ends. A Mixed Node Web service plays two roles: handling external Web service requests on channel ends and communicating with Web services which hold the other end of channels.

In this middleware, Web services expose their operations through WSDL interfaces and they need to be aware of Reo channel operations. In Figure 9 a Channel End Web service and its WSDL interface is shown.

Figure 8. Two types of Web services in service oriented Reo middleware (a) Channel End Web service (b) Mixed Node Web service

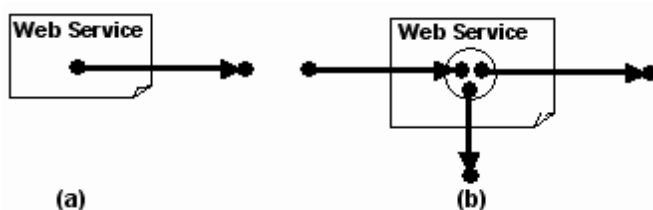
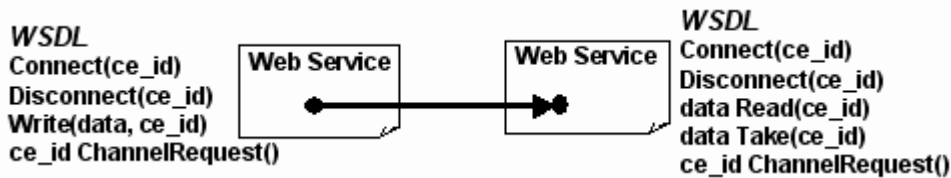


Figure 9. Channel End Web services in distributed service oriented Reo middleware model



This model needs a structure for channel ends and a protocol for Web services to manage the channel ends operations. The main advantage of this model is its independence from requesting components and its ability for supporting reconfiguration at runtime but designing more complicated connectors in this model raises a number of challenges like any distributed model. In this model by providing reconfiguration operations for Reo middleware Web services, the model can be extended to support topological reconfiguration. By dynamic reconfiguration we can change the coordination pattern to fulfil non-functional requirements such as cost and performance at runtime.

6 DISCUSSION AND COMPARISON

In this section we discuss and compare the properties of the proposed models according to some generally accepted Web service composition requirements:

1. **Non-functional properties:** In the proposed models dynamic reconfiguration can be considered as a mechanism to fulfill non-functional requirements. Changing the topology of connectors at runtime in distributed service oriented Reo middleware, helps designers to serve non-functional Quality of Services (QoS) properties, such as performance and cost, but in the first model, due to its central circuit structures, this functionality does not provide remarkable benefits.
2. **Composition correctness:** In the first model, as Reo circuits are located in a central middleware and defined prior to execution, verifying its correctness is possible. In contrast with the first model, the distributed service oriented Reo middleware model consists of a set of channels which are all distributed among Web services, hence checking the correctness of the circuit is a challenging task and requires more consideration.
3. **Composite scalability:** This composition requirement is supported in Reo by its channel-based structure. In the first model as the Reo circuits are central; they can be extended to handle more services. For example a simple Barrier Synchronizer connector in Reo which represents the parallel coordination pattern can be extended to handle any number of pairs.
4. **Tool support for execution and verification:** Any composition approach must be supported by software tools. These software tools usually provide implementation, verification and execution. For the first model, ReoLite [9] can be used as a central middleware to run Reo circuits but there is not any implemented middleware available for the distributed service oriented Reo middleware model. We also can verify the Reo circuits in the first model by proposed tools in [10] and [11] which are based on constraint automata.
5. **Process modeling construct support:** Reo channels provide a comprehensive mechanism to design any coordination pattern out of simpler ones. In the first model as the Reo circuits are central, designing any coordination pattern is possible but in the distributed service oriented Reo middleware model, because of its distributed structure, providing complex patterns is a challenging task.
6. **Graphical notation:** Composition languages are expected to have a visual notation for expressing the coordination process. The visual notation should be independent from software components and the type of data which is used in the coordination process. It also must be expressive enough to be mapped to an

executable code. Reo visual notation serves these requirements and provides an understandable notation which can be mapped to an executable code.

In Table 1, we have compared the proposed models with respect to the above requirements.

Table 1. Comparing Web service composition requirements

	Central Orchestration Model	Service Oriented Distributed Reo Middleware Model
Non-functional Properties		√
Dynamic Reconfiguration		√
Composition Correctness	√	
Composition Scalability	√	
Tool Support (Verification)	√	
Tool Support (Execution)	√	
Formal Semantic & Expressiveness	√	√
Process modeling construct support	√	√
Graphical Notation	√	√

7 CONCLUSION

In this paper we proposed two models for composing Web services based on Reo coordination language. Reo as a channel-based coordination language not only provides a formal specification but also has a visual notation and implementation. The formal basis of Reo, guarantees the possibility of checking and verifying the coordination process and its channel-based structure lets designers to define complex coordination patterns. In the first model, referred as Central Orchestration model, we defined Web service composition processes by Reo circuits. The second model was a distributed service oriented Reo middleware which could be used to compose Web services. The main properties of the first model are scalability and its central circuits which represent an orchestration framework; on the other hand, the second model is a service oriented Reo middleware which can fulfill non-functional requirements through reconfiguration operations. In future work, we are going to define more process patterns by Reo circuits, especially complex ones and those which can not be modeled by available coordination specifications.

REFERENCES

1. Arbab F. (2003), 'Abstract Behavior Types: A foundation model for components and their composition'. In *Proceedings of the First International Symposium on Formal Models for Components and Objects (FMCO 2002)*, LNCS 2852, 2003, pp.33-70.
2. Arbab F. (2004), 'Reo: A channel-based coordination model for component composition'. *Mathematical Structures in Computer Science*, 14(3), 2004, pp.1-38.
3. Arbab F., Baier C., Rutten J.J.M.M. and Sirjani M. (2004), 'Modeling component connectors in Reo by constraint automata'. *Electronic note in Theoretical Computer Science*, vol.97, No.22, July, 2004, pp. 25-46.
4. Arbab F., deBoer F.S., Bonsangue M.M. and Guillen-Scholten J.V. (2002), 'MoCha: a middleware based on mobile channels'. In *Proceedings of 26th International Computer Software and Application Conference (COMPSAC02)* IEEE Computer Society Press, 2002.
5. Arbab F. and Diakov N. (2004), 'Compositional construction of Web services using Reo'. In *Proceedings International Workshop on Web Services: Modeling, Architecture and Infrastructure (ICEIS 2004)*, Porto, Portugal, April 13-14, 2004.
6. Arbab F., Everaars C.T.H., De Oliveira Costa D.F., Diakov N.K. (2006), 'A distributed computational model for Reo'. *Technical Report, REPORT SEN-E0601*, CWI, the Netherlands, February 2006.
7. Arbab F., Mousavi M. R. and Sirjani M. (2004), 'Specification and verification of component connectors'. *Technical Report CSR-04-15*, Eindhoven University of Technology, 2004.
8. Arbab F. and Rutten J.J.M.M. (2003), 'A coinductive calculus of component connectors'. In *Processing of 16th International Workshop on Algebraic Development Techniques (WADT 2002)*, Lecture Notes in Computer Science 2755, Springer, 2003, pp.35-56.
9. Clarke D., ReoLite: Reo coordination engine, CWI, the Netherlands; available at: <http://homepages.cwi.nl/~dave/reolite/> (Jan 2006).
10. Ghadiri A. (2004), 'A Tool for Constraint Automata Join'. *BS project, Technical report*, ECE Department University of Tehran, Iran, 2004.
11. Ghassemi F. and Tasharofi S. (2005), 'A Tool for Converting Reo Circuit to Constraint Automaton'. *BS project, Technical report*, ECE Department University of Tehran, Iran, 2005.
12. Hamadi R. and Benatallah B. (2003), 'A Petri net-based model for web service composition'. In *Proceedings of the 14th Australasian Database Conference*, volume 17 of CRPITS, pages 191-200.
13. Hinz S., Schmidt K., and Stahl C. (2005), 'Transforming BPEL to Petri nets'. In *Proceedings of the 3rd International Conference on Business Process Management*, volume 2649 of Lecture Notes in Computer Science, pages 220-235, Nancy, France, September 2005.
14. Koehler J., Tirenni G. and Kumaran S. (2002), '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.
15. Srivastava B. and Koehler J. (2003), 'Web service composition - current solutions and open problems', In *Proceedings of ICAPS'03 Workshop on Planning for Web Services*, Trento, Italy, June, 2003.
16. Ter Beek M.H., Bucchiarone A. and Gnesi S. (2006), 'A Survey on Service Composition Approaches: From Industrial Standards to Formal Methods'. *Technical Report 2006-TR-15*, ISTI, Consiglio Nazionale delle Ricerche, 2006.
17. Van der Aalst W.M.P.(2003), 'Challenges in business process management: Verification of business processes using Petri nets', *Bulletin of the EATCS*, 80:174-199, June 2003.
18. 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 (Jan 2006).
19. Yang Y., Tan Q., and Xiao Y. (2005), 'verifying web services composition'. In *Proceedings of the ER 2005 Workshops*, volume 3770 of Lecture Notes in Computer Science, pages 354-363, Klagenfurt, Austria, October 2005.
20. Zlatev Z., Diakov N. and Pokraev S. (2004), 'Construction of negotiation protocols for e-commerce applications', *ACM, SIGecom Exchanges*, 5(2), pp.12-22.