SUITABILITY OF UML STATE MACHINE FOR MODELING CHOREOGRAPHY OF SERVICES

Azadeh Mellat[1],Naser Nematbakhsh[2],Ahmad Farahi[3] and Farhad Mardukhi[4]

[1]Department of Computer Engineering and Information Technology,PayamNoor University,Tehran
A_mellat@yahoo.com
[2]Department of Computer Engineering,University of Isfahan
Nemat@eng.ui.ac.ir
[3]Department of Computer Engineering and Information Technology,PayamNoor University,Tehran
afaraahi@pnu.ac.ir
[4]Department of Computer Engineering,University of Isfahan
Mardukhi@eng.ui.ac.ir

ABSTRACT

Recently, a lot of research works have attempted to model the choreography of services by different languages. Each language models the choreography on the basis of different view and level of abstraction. The features of each language demonstrate how much it is suitable for service choreography modeling. Among all languages, WS-CDL is a popular language which has the rich syntax to model choreography. But it is much limited for modeling the semantic and adaptability of choreography[16,18]. This paper aims to show the suitability of such language to specify service choreography. For that, we define the requirements of a typical choreography language on the basis of a number of interaction patterns. Then, the UML state based language is checked out against those patterns to recognize its suitability for addressing the requirements of service choreography. We also compare this language with other main languages in terms of interaction patterns appeared within a compare ison table.

KEYWORDS

Web Service, Choreography, UML state machine, Choreography modeling language.

1. INTRODUCTION

There is increasingly requirement in service oriented applications to make the applications on the basis of Web services distributed on the Web. Such applications are made when such Web services are choreographed in accordance with the rules of relevant business corporation. In current SOA standard, the service based collaboration is studied through two known concepts: Choreography and Orchestration [14].

DOI : 10.5121/ijwest.2011.2403

Choreography describes the interaction of several services corporate to gain a common goal. Choreography is a description language to express the interaction protocol among participants to show that all things are go accord to plan [8]. The *Orchestration* is a concept to describe how a compound service manages its internal activities to supply its capabilities [6]. An Orchestration model defines a set of "active rules" are executed to manage the behavior of a participant which is described in choreography model [22].

Currently, there are a variety of languages and models to model the choreography and orchestration of services [22]. The most languages regard to choreography as a description language just for defining the rules of corporation which are realized and executed by an orchestration language. Therefore, they distinguish languages for each concept. For example, WS-CDL is used only to describe choreography and BPEL is used for modeling orchestration of services. From another point of view, these concepts are much overlapped. This is why some languages like BPEL4Chor describe both concepts together [8, 22].

Since there are many different languages for expressing service choreography, comparing them is an essential work aiming to know their features and limitations. Up to know, a bit of works have dealt with this issue. They use a typical comparison table to show the features which are (are not) supported by the compared languages. This work aims to look into a language recently has introduced by [Harel] to model choreography of services. It is an approach which uses" UML State machine" as a fundamental mean to model choreography. Thus, "UML State machine" is investigated to find out its capabilities for modeling service choreography. We extend also the comparison aspects usually are leveraged by recent works to compare the choreography languages. Consequently, the other known languages such as WS-CDL are compared based on the new introduced aspects [14].

The rest of this paper is organized as below. Section two firstly overviews the choreography model requirements through a list of essential interaction/workflow patterns and some other features. Section three overviews the position of UML state machine based language and classifies choreography modeling languages into for categories. Section four introduces state based models and specifications that give to choreography.Section5 introduces UML state machine and it's properties as well as the approach of that in supporting work flow and service interaction patterns. Finally section 6 concludes.

2. CHOREOGRAPHY MODEL

Choreography is a word leveraged for describing observable interactions among the participant services of a corporation. The modeling of choreography can be performed from a global perspective where the interaction among services are described or by a set of local perspective where the observable behavior of each service is described through a sequence of activities [8].
As the main requirement supposed to a choreography language, it must captures service interactions requirements. Some of requirements are sequencing the interacting messages, the scenario of sending/receiving messaging and synchronization of message [14]. The requirements of service interactions have been reported by some works during several recent years through several patterns namely service interaction patterns. Use of Service interaction patterns is ideal proposal for evaluate capabilities and limitation of existing languages; also is a basis to design a new choreography language. In addition, patterns can help us to classify the requirements of

choreography into a set of classes [8, 14]. Barros [1] classifies the service interaction patterns into four main classes as shown in table 1.

- ***Bilateral interaction patterns***: Elementary interactions where a party sends (receives) a message, and as a result expects a reply (sends a reply) [1].
- ***Basic multilateral*****:** A party may send or receive multiple messages but as part of different interaction threads dedicated to different parties [1].
- **Multi-transmission:** (non-routed) interactions, where a party sends (receives) more than one message to (from) the same party [1].
- **Routed interactions:** The messages are routed among the participants of an interaction [1].

In addition, the proposed patterns may be composed through structures expressing flow dependencies such as sequence, choice, and synchronization [13]. Therefore, the interaction patterns and essential structures must be handled by a typical choreography language and are two main criteria adopted by this work for evaluating the languages. Table 2 summarizes the essential overall interaction patterns and structures [11].

The second criterion for evaluating language is ability of language to support the choreography at least two level of abstractions: The logical level and implementation level [1]. A promised approach recently was proposed for developing service oriented application is a methodology which begins by defining the choreography of services. In this approach, it is assumes that each participating service is aware of the corporation rules and coordinates itself accord to the protocol of corporation. Therefore, choreography is merely a logical specification of the observable behaviors. In the ideal case, the choreography completely captures the service behaviors; and orchestration must capture the local behavior of each service [1, 8]. It is promised that believe that a clear separation of logical and implementation levels should be fundamental principle that would allow us to separate and localize concerns and make them independent in developing better systems[16 , 17, 20]. One of problem appeared here is that how to propagate the changes of choreography level to orchestration level. This problem is known as conformance problem which deals with consistency between the logical requirements presented by choreography with what is implemented via orchestration. Thus, the clear relationships between chorography and orchestration language is a fundamental requirement [1, 2, 6]. A specification of a target choreography language must be able to define the choreography on both perspectives. In other words, the choreography can be modeled on two different levels of abstractions. At the global level, the requirements of choreography are described, and local level presents the observable behavior which the implementation of a service must provide. Precisely, A *choreography modeling language* must provides means to define choreography models, i.e., choreographies, service implementations, and their semantics including a mechanism to compare global behaviors generated by service implementations with choreography[4].

The third criterion is the potential of choreography modeling language to be dynamic. A dynamic choreography model is capable of altering its elements dynamically without designing the model by hand [16]. The applications and services needed to be dynamic and adaptable, particularly at the organizations taking part in inter-organizational corporations. Consequently, there is increasingly requirement in service oriented applications to make the applications more dynamic. Dynamic structure means that the structure of system must be flexible to being reconfigured and regulated dynamically in response to commands of management activities [16, 20 , 21].

3. THE POSITION OF UML STATE MACHINED-BASED LANGUAGE

Based on the survey we did over choreography modeling languages, we classify them into four categories: workflow-based, state-based, formal-based, and rule-based [16, 21]. The general properties of the languages included at these categories are shown briefly in the table 1.

Workflow based languages usually are defined based on process algebra. The most of related studies specify the choreography differently at logical and implementation levels. The synchronous messaging model is a model used for communication [1, 8, 22].

Table 1. The classes of choreography languages in a comparative view [1, 8, 16, 21, 22]

	Workflow-based	State-based	Petri-Net	Rule-based
Service Interaction Patterns	The most of patterns are supported because of existing rich syntax of such languages.	Have rich syntax Support	strong theoretical foundation for traditional intra-organizational business processes	The interaction behavior determines a set of active rules which a rule must execute to take part correctly in choreography
Flow dependencies	All of such patterns are supported directly by such languages.	directed graphs in which nodes denote states and connectors denote state transitions	Bipartite graphs consisting of places and transitions that are connected via directed edges	set of rules to describe the goals, decisions, actions, or constraints of a system
Messaging	Channels	Events	Tokens	Events
Dynamic modeling	The requirements for dynamic routing are outside the scope of direct support through work flow- based	Have rich modality support	Petri nets do not support the concept of mobility	promised and modem paradigm that dynamically manages the behavior of a system correctly
Examples	WS-CDL-WSCI-BPEL4Chor	Let's dance	Petri nets used in BPM	WSMO

4. STATE BASED MODELS

State based choreography models represent both choreography and service implementation using states and transitions among them. The main advantage is that state machines explicitly can capture the description of a choreography and logic of a service implementation as a set of behaviors. Each behavior can be easily modeled as a sequence of states, each associate with transitions. Each transition may be labeled by a message or an activity [4,6]. State based models are generally adopted for representing the behavior of objects widely at many of science field. For example, at the most of software development methods the behaviors of objects are shown by state based diagram. The behavior of an object shows how it reacts to the occurred events [7,8]. The automata based choreography modeling approach specifies choreography through states and transitions. It is easy to use since this approach is commonly used to specify protocols and policies [14].This group of choreography modeling languages includes conversation protocols and Mealy services [10, 19], UML collaboration diagrams [9], and the Colombo service composition model [1], Let's dance [2], UML state based [3].The use of conversations to specify choreographies was originally proposed in the IBM Conversation Support Project [14]. A formal model based on this idea was developed in [10] under which a conversation protocol is represented as finite state automaton over messages and each service as a Mealy machine over the input/output messages of the service. And in a Mealy service where the leading symbol "!" denotes an action of sending a message and "?" a receiving action. Each service has an associated FIFO queue (of unbounded capacity) for storing unconsumed incoming messages. When services are executing, a virtual global *watcher* records the sequence of messages for all send actions. A *conversation* is the sequence of messages recorded by the watcher in a successful execution. A conversation protocol is *satisfied* if every conversation by the services is a word accepted by the conversation protocol automaton [3, 12]. The language *Let's dance* [2] provides a set of sequencing constraint primitives to allow a choreography to be specified in a graphical language [3]. Finally, another interesting variation is the Colombo model used to study an automated composition problem for semantic web services [1]. Similar to the UML model, the local services are represented as Mealy services (extended to allow OWL-S like semantic descriptions) but the message queues are limited to at most one message (size 1).Choreographies, however, are represented by finite state automata over only activities without messages. The model is an extension of the earlier "Roman" model for composing Interactive web services [2].Abstract State Machine (ASM) is a formal model to describe the choreography of services from user point of view. On the basis of ASM ,the choreography is defined as a set of states which the transition among them are done through several basic formal operation as defined in "ASM" model such as *For Each, If ,Switch*, etc[1].

5. UML-STATED BASED CHOREOGRAPHY MODEL

In this section the choreography model on the basis of" UML state machine" is looked into to know its potential and features for supporting the choreography requirements. The choreography model based on "UML state machine" as [16] reported defined as a set of local choreography model each is depicted by a "UML state machine" enabled by a set of policies. The main activities of choreography are *Send, Receive, Send-Receive, invocation, and assign*. The interactions patterns are designed based on the dependency flows on these basic activities. Both transitions and states are controlled by a set of policies which checked upon occurring associated events [16].

5.1 UML State Machine

UML state machine is known also as UML state chart introduced based on the *Harel* [3] hierarchal state machine. Nowadays "UML State Machine" besides other UML elements has prepotency strength in developing functions. "UML State Machine" overcomes limitation of old finite state machines, it preserves basic advantage of that and gives more benefits. Finite state machine is a model of system behavior that shows how systems react versus similar occurred events. "UML State Machine" includes soma states (finite states), transactions between them and activities. "UML State Machine" has a strong and semantic sign for specifying behavior of a system or component [3, 12].

Besides keeping old form of FSM, "UML State Machine" introduces new nested states concepts. Nesting means states and machines that can be built with other states and machines .describing of complicated states and transitions with and transition with nesting activities is easier. "UML State Machine" can be so good candidate for dynamism in choreography. According various studies, necessary properties for services dynamism are [18 , 21]:

1. Hierarchal structure of "UML state machine" enables us to build the compound state. This is a way to make choreography dynamic since it can be designed based on reusable elements. Clearly, this feature realizes the principle "separation of concern" as well [14, 15, 23].
2. It is possible to assign the states to *QoS* properties. This is an opportunity to adjust a state on runtime by changing the *QoS* properties. For example, we can associate a state to be done during a specified response time [3,5,16].
3. "UML state machine" is inherently event oriented model. Therefore, injecting the events by the same style can help us to change the reaction of model during running time. Also, add/remove the events and corresponding reactions can be taken account [3, 23].

The ability of "UML state machine" to model the behavior of objects and also having inherent properties to support dynamic structure devote us a good chance to use it a mean to model the choreography especially when adaptability is a key requirement.

In addition, the "UML state machine" can be used at both choreography model and implementation level. For choreography model, the states present interactions and transitions show how the interactions are sequenced to provide the choreography. For implementation level, each service is designed through a "UML state machine" where the states present the status of service via the values of corresponding properties, the events trigger the transitions and the actions can be ran when during the transitions or entering/exiting the states[12,17].

5.2 Workflow and Interaction patterns

This section surveys how UML-state machine based model can support both workflow and services interaction patterns. To have a baseline for checking the abilities of UML-state machine based model of choreography, we overview the known patterns introduced by some related researches [7, 8].

5.2.1 Messaging

Both synchronous and asynchronous messages are supposed to be supported directly with "UML State Machine" [3, 12].

5.2.2 Basic Activities

The basic activities are **Send, Receive, Send-Receive, Invocations and Assign.** The model of choreography must provide these primitives basically. Therefore, they are supposed to be implemented by the languages. For modeling the choreography, having these primitives is not sufficient, but they must be composed together through a set of dependency flows. The main dependency flows, known as workflow patterns are earned by "UML state machine" as shown below [14, 15].

Sequence*:* This pattern is realized when states seriated and linked each others with transitions [12]. Figure 1.a shows how such pattern is constructed in "UML State Machines".

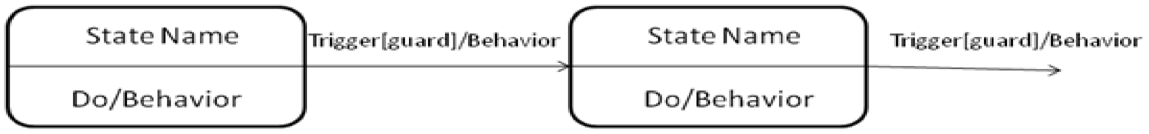

Figure 1.a Sequence pattern [2, 12]

Parallel: Orthogonal region is a novel idea of UML state machine which address the frequent problem of a combinatorial increase in the number of states when the behavior of a system is fragmented into independent, concurrently active parts. The parallel pattern can be modeled by orthogonal region [3, 12]. Related to this issue, join and fork are two concepts adopted in UML state diagram to enter into or exit from orthogonal regions as depicted in figure 2.a.Both Parallel split and synchronization patterns are supported by Parallel pattern [11].

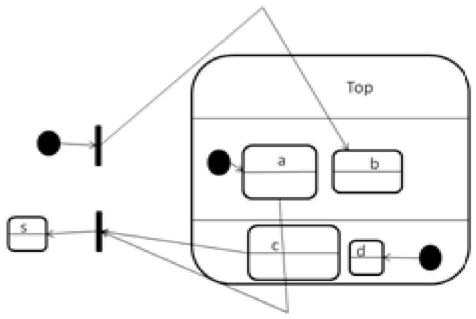

Figure 2.a orthogonal region to present parallel [4.12]

Exclusive Choice: It specifies that only one activity within the structure is selected and other activities are disabled [11]. In "UML State Machine", we can use choice pseudo state [3] .When a transition occurs in which a logical condition should be investigated, it puts guard on transition and depending on that specify which transition will be done. We illustrated this pattern in figure 3.a in "UML State Machine".

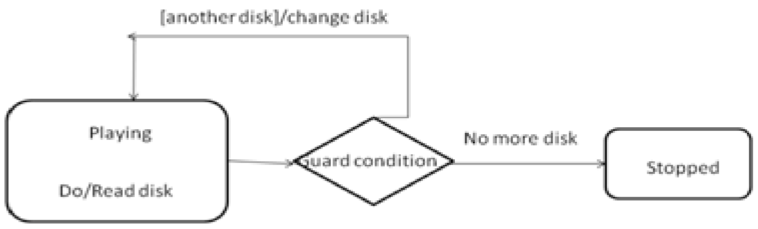

Figure 3.a Exclusive pattern [11,12]

Simple Merge: A Simple Merge is a distinct point in a business process where two or more branches are merged into one single branch. Each incoming branch then activates the subsequent branch.UML state machine supported this pattern by choice pseudo [24].For example the office assistant wants to make an appointment for an interview with an applicant by sending a proposal via e-mail. The applicant can then accept the appointment via phone and/or via e-mail. The office assistant will then define the appointment as fixed. Figure 4.a shows how such pattern is constructed in "UML State Machines".

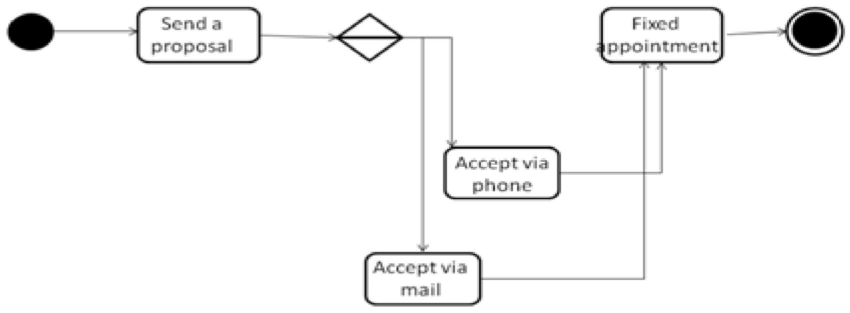

Figure 4.a Simple Merge pattern [11, 12, 24]

5.2.3 Advanced Branching and Synchronization patterns

This group presents several patterns which characterize more complex branching and merging concepts which arise in processes [24].

Synchronizing Merge: In the case of *"Multiple Choice*", a number of branches are chosen. This can be expressed using a parallel structure containing guard conditions. A matching "*Synchronizing Merge*" is also covered by this structure [4,11]. Both patterns are supported by "UML State Machine" parallel interactions. This structure implements branches in the shape of states, transitions and locating them (two or more branches) in a *"Parallel*" structure as mentioned before. Figure 4.a that is shown in below can execute Searching and Pacing simultaneously.

Multiple Merge: A point in a process where two or more branches converged without synchronization. If one more branches get activated, possibly concurrently, the activity following the merge is started for every activation of every incoming branches. In "UML state machine" exists direct support for this pattern. Some times one object can be representative of two or more objects that are in the same type. Fork pseudo state and join pseudo state show synchronic embranchment and then renewed link. Incoming transition is broken by fork and caused pacing and searching occurred simultaneously as shown in figure 5.a.Join pseudo merge two transitions and shows in a form of one out coming transition.[3,5,11,12]

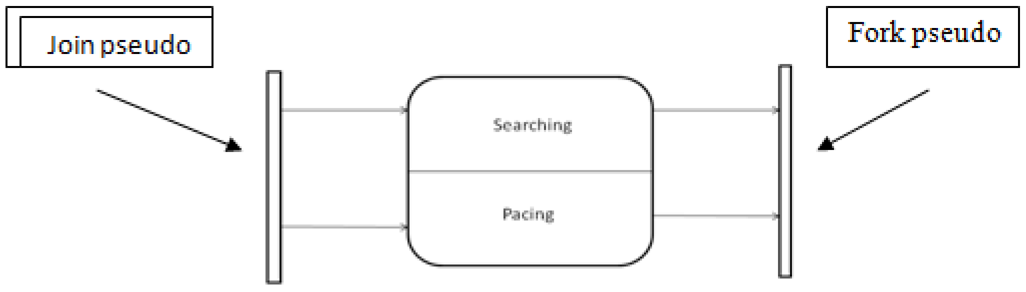

Figure 5.a Multiple merge pattern [7, 11, 12]

Discriminator: A point in a process that describes the merger of two or more branches into one single branch with a corresponding split beforehand somewhere in the business process. The thread of control is passed to the proceeding activity as soon as one incoming branch has been enabled regardless of the progress of the other incoming branches. This pattern occurs in a structured context so it is essential that there is one single Parallel Split construct somewhere earlier in the business process with which the Structured Discriminator is associated. The Structured Discriminator must merge all of the branches coming from the split. For example the marketing department has just one advertising space available on an event. The marketing department then tries to find a customer for this ad space on two different channels - the internet and the daily newspaper. As soon as one customer books the advertising space, it doesn't matter what the other customers in the two advertising channels do [24]. The contract gets signed and the ad space is booked from the customer. There is no clear support for this pattern in "UML State Machine".

5.2.4 Structural pattern

Arbitrary cycle: This pattern is a point in a process where one or more activities can be done repeatedly [11]. In "UML State Machine", we have a state with an event that should perform repeatedly. We should use guard conditions for reaching this goal. Different state can do this pattern for various events simultaneously. Figure 6.a illustrated this pattern in "UML State Machine" as well. While the guard condition is true, this cycle is continuing [7].

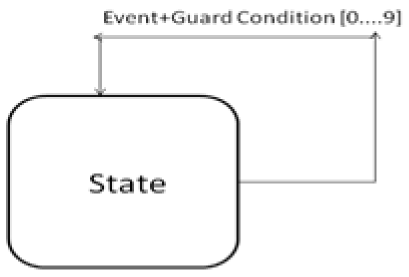

Figure 6.a Arbitrary cycle [11, 12]

Implicit termination: A given sub process should be terminated where there is nothing else to be done [1, 7]. It is like arbitrary cycle and guard condition must be checked. When guard condition become false, "UML State Machine" should stop (e.g. if structure in java).While guard condition is true, cycle is continuing .As soon as condition be false cycle will be stopped. For this Use case we take a look at the hardware ordering process. The inventory department receives a request for a new hardware. An employee of this department then sends the new hardware to the computing centre and a bill to the accounting. The process is finished or terminated as soon as the new hardware has been installed and the bill has been booked [24] .Figure 7.a depicts this use case in UML state machine.

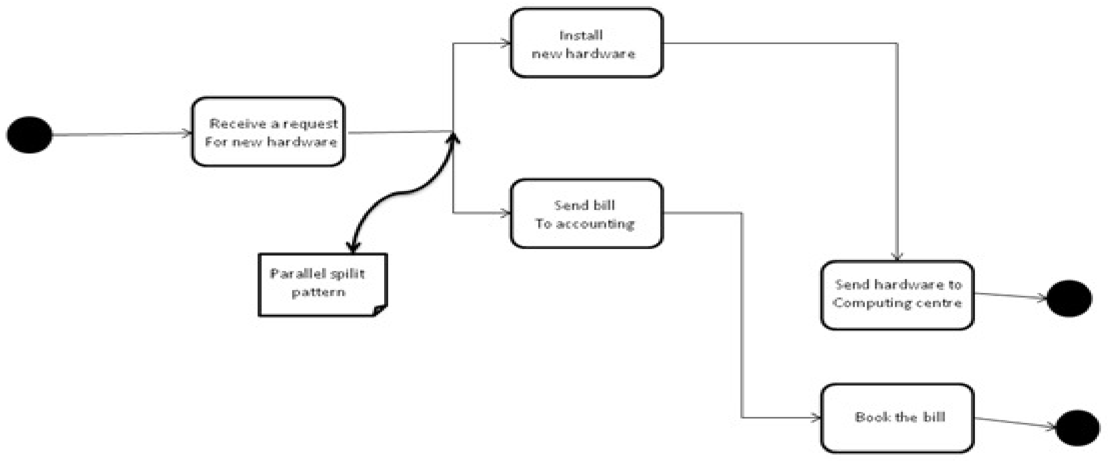

Figure 7.a Implicit termination pattern in hardware ordering process [24]

5.2.5 Patterns involving multiple instances

This pattern describes scenarios where multiple instances of an activity can be created in the context of a single case. Multiple instances can arise through three different reasons [11, 24]:

1. An activity is able to initiate multiple instances of itself.
2. An activity is initiated multiple times as a consequence of receiving several independent triggers for example as a part of a loop.
3. Two or more activities in a process share the same implementation definition.

MI with a priori design time knowledge: If the number of instances is known at design time, the activities could be replicated and placed in a parallel structure [9]. It is supported in "UML State Machine" by Orthogonal region as well as composition state diagrams as mentioned before .For this use case ,there's an event coming soon .The number of participants is known beforehand .The Office sends an invitation to each of the twenty participants. As soon as all of them have replied to the invitation the Office can continue the event preparation [24].

.

MI without synchronization: These structures also indicate synchronization after completion of activities [8]. Alternatively sub choreography can be defined and activated several times using parallel structures like fork and join pseudo or composition state diagram. In this point in "UML State Machine", main structure either could finish but substructures are executing or both of them are executing simultaneously.

MI with a priori runtime knowledge: We have no direct support about this pattern in WSCDL. This pattern is similar to MI with a priori design time knowledge. The only difference is that the amount of instances is not known beforehand but during execution before the task instances must be created. It is necessary to synchronize the different task instances after completion before any subsequent tasks can be enabled [24]. This Use Case is similar to the one presented before for MI with a priori design time knowledge too. The only difference is that the amount of messages sent out by the office is not known beforehand but dynamically during the planning phase. In this case, we should beforehand guess mutual states and defined them as states or sub states that could be executed in parallel[11,13] .One state can be located in the bottom of other state as sub state. When an event receives, main state checks it and realizes which state should handle event. Figure 8.a illustrated nested states.

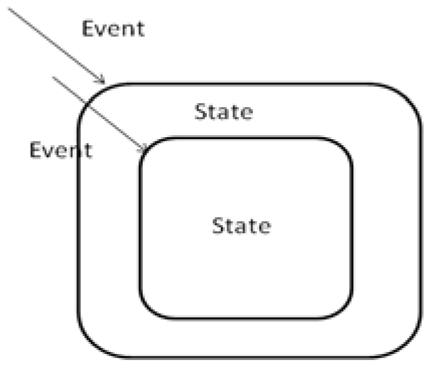

Figure 8.a Multiple Instances pattern [3.11]

MI with no a priori runtime knowledge: This pattern is similar to MI with a priori run time knowledge. The only difference is that the amount of instances is not known until the final instance has completed. At any time, whilst instances are running, it is possible for additional instances to be initiated. It is necessary to synchronize the instances at completion before any subsequent tasks can be triggered [24]. This Use Case is similar to the one presented before for MI with a priori run time knowledge. The only difference is that the Office continues inviting people until the required amount of participants has been reached. in WSCDL a counter is used to record how many instances have already completed. A blocking work unit is activated as soon as the counter has reached a certain value."UML State Machine" uses ontology for defining concepts (e.g. response time) [1, 11]. Response time introduced as a concept in ontology and values between a minimum and maximum. This pattern is not directly supported by" UML State machine". We opt for partial support for this pattern [8].

5.2.6 State based patterns

The group of state-based patterns reflects situations for which solutions are most easily accomplished by the notion of states.

Deferred choice : A point in a process where one of several branches is chosen [11]. In contrast to the *XOR* split, the choice is not made explicitly (e.g. based on data or decision) but several alternatives are offered to the environment. For example an applicant has read the AGB and decides now to become a member. He has now to decide whether he wants a sponsor contract or an ordinary membership application. Doing both is not possible. The decision is manual. In "UML State Machine" depends on which guard condition is true , some subsequences of interactions are executed. Choice pseudo structure as well as guard conditions in internal behavior support this pattern. Figure 9.a and 10.a depict deferred choice pattern in "UML State Machine".

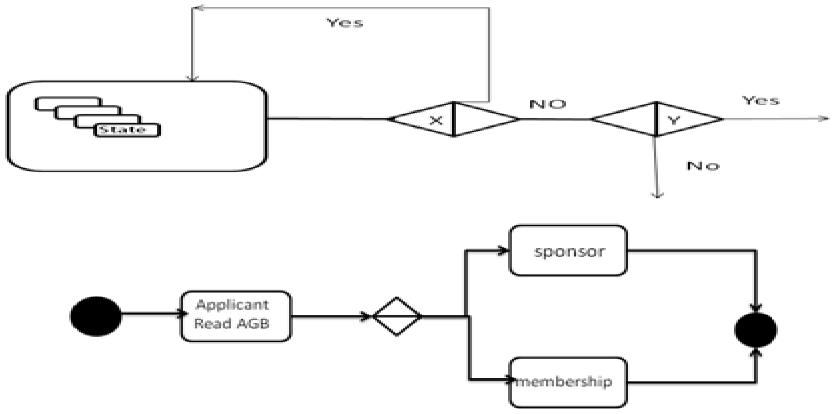

Figure 9.a,10.a Deferred choice pattern [11, 12]

Interleaved parallel routing: A set of activities is executed in an arbitrary order. Each activity in the set is executed. The order is decided at run time and no two activities are executed at the same time. For example the manager of the education department informs the employee that he or she should do three different phases during the aptitude test. A psychological test, an intelligence test and a round of introduction. The order in which the three phases are accomplished is relevant. The intelligence test must be revised before doing the psychological test. The round of introduction can be accomplished as first phase, as last phase or also in the middle of the other two phases. However you are not allowed to do two phases at the same time. One of the most important properties of "UML State Machine" is ability to support nested states .In "UML State Machine" could be defined states in the bottom of other states. As soon as receiving an event, one of state or one of sub states will be activated. Parallel patterns are supported by UML state machine, as mentioned before . Figure 11.a illustrated this pattern [3, 11].

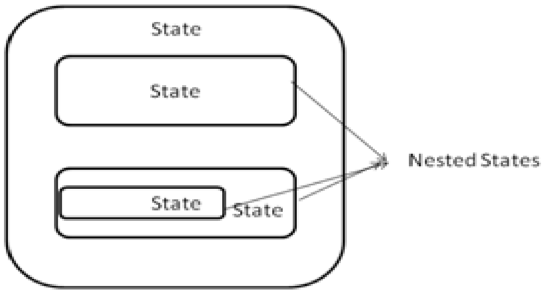

Figure 11.a Interleaved Parallel Routing pattern

Milestone: The enabling of an activity depends on the case being in a specified state [3,5]. The activity is only enabled if a certain milestone has been reached which didn't expire yet. If a state be in special position and a special condition happens, it transfers another state. Figure 12.a is an example of Milestone pattern [3].

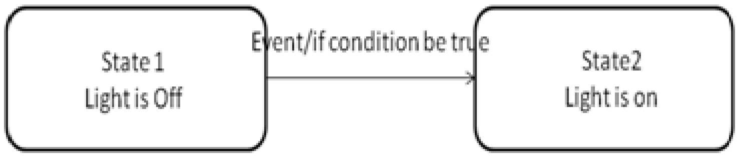

Figure 12.a Milestone pattern [7, 12]

5.2.7 Cancellation pattern

The patters in this group utilize the concept of cancelling or withdrawing activities [24].

Cancel activity: In the case of cancel activity, an enabled activity is disabled. In "UML State Machine" may Close () event cause disabling activities [11, 12]. FOR example, the Office Assistant creates a PowerPoint presentation for a manager. If the manager sends a message that the presentation is no longer required the assistant stops working on the presentation continuing with the normal work .Figure 13.a illustrated an example of Cancel activity pattern

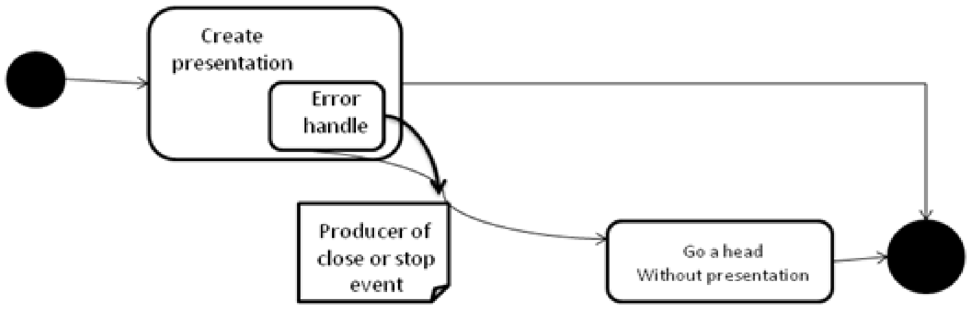

Figure 13.a An example of Cancel activity pattern

Cancel case: When a state is removed, all of its substrates are removed too. As soon as a state receive an event like close(), could finish either only itself or all of it's sub states .When a state and all of sub states need to be stopped ,main state receive an event(e.g. close()).Before finishing itself ,it sends close event to it's sub states and all of them receive close event. Main state waiting for responses from sub states (sync messages).If responses are ok, main states and all of sub states are stopped. if no (even if one sub state),main state and sub states can't be stop[10,11]. For example an applicant has read the AGB and decides to become a member. He or she can do this by signing a sponsor contract and/or a membership application. After submitting his date the applicant might change his or her mind and cancel his or her application. Figure 14.a illustrated an example of cancel case pattern.

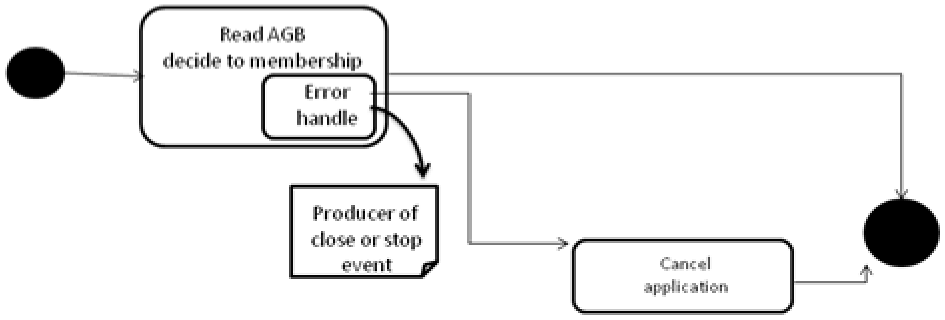

Figure 14.a An example of cancel case pattern

5.3 Service Interaction Pattern Support

The Service Interactions Patterns were introduced by Barros et al. in [13]. They present common interaction scenarios between two or more parties and can be used to assess choreography languages. Although [13] also contains hints about how different languages implement individual patterns no complete assessment of a standard has been carried out so far using this set of patterns.

5.3.1 Single - transmission bilateral interaction patterns

Send/Receive/send\receive: All three patterns are directly supported in "UML state machine". In this view a trigger is shown by a receive signal icon and transitional behavior is shown by a send signal icon. Figure 15.a can illustrate this pattern [2, 3, 6, 12].

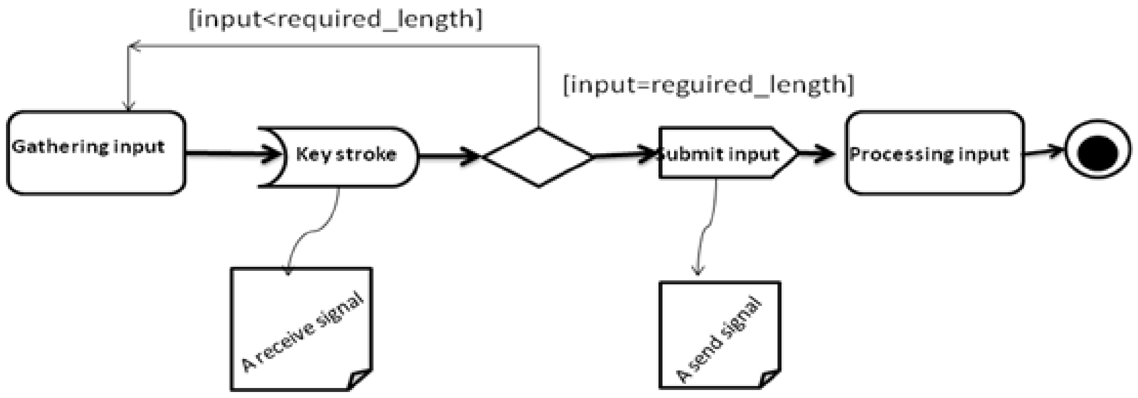

Figure 15.a send and receive patterns [11,12]

For modeling both send /receive pattern simultaneously, "UML state machine" uses orthogonal region in a composition state diagram. It gives possibility to execute both patterns in parallel. Figure 16.a shows a composition state diagram with Send/receive illustration [3, 12].

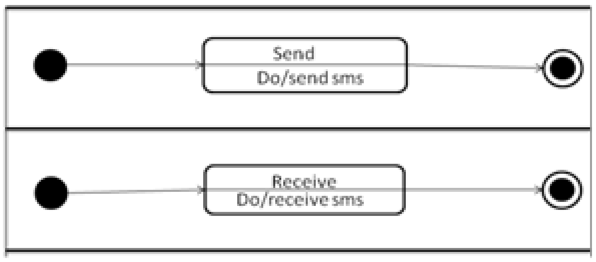

Figure 16.a Send/Receive pattern [1, 3]

5.3.2 Single-transmission multilateral interaction patterns

Racing in Coming Messages: This pattern is similar to the workflow *Deferred* choice pattern. A party expects to receive one among a set of messages [11].This messages may be structurally different (i.e. different types) and may be come from different categories of partners. In "UML state machine" a service received different messages by an event (trigger) like receive event. Input state in a service can recognize which message is suitable for service on that time. Message, in the form of events, enter a service [3, 12].

One to many send: A party sends messages to several parties if the number of recipients is known at design time. Interactions can be placed in a parallel structure like orthogonal region or nested states. If a number of recipients unknown at runtime and messages should be sent to them, choreography is responsible for this subject. Each service that added at runtime must be registered. When a service specified, it was known for other services and choreography [1, 8]. With repeating execution of choreography, previous message will be sent to all services as well as new services. For this subject in "UML State Machine" we located choreography instructions in a state and repeat it for some times. Since "UML State Machine" more directly supports the case where the number of recipients is known at design-time, we opt for partial support for this pattern. Figure 17.a and 18.a show One to many send pattern and choreography updating obviously [3, 11].

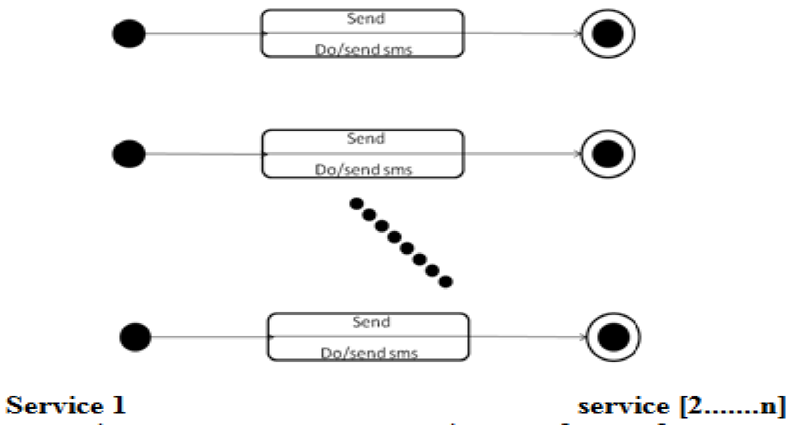

Figure 17.a One to many send pattern [2, 3,12]

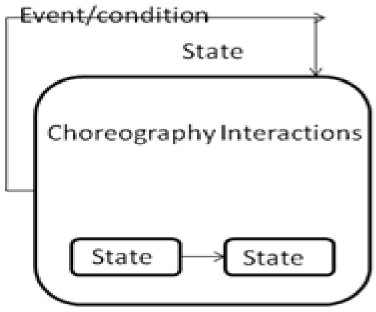

Figure 18.a Arbitrary cycle for realizing new services at run time [12]

One from many receives: This pattern describes that a party receive a number of logically related message that arise from autonomous events occurring at deferent parties. The arrival of message need to be timely so that they can be correlated as a signal logical request [11] . In "UML State Machine" each messages is an event that can activate recipient state. For keeping all of these messages in recipient state, "UML State machine" uses queue data structure [12].

One to many send/receive: This pattern is very similar to one to many send. A party sends a request to several other parties. Responses should turn back within a given time frame. The interaction may complete successfully or not, depending on the set of responses gathered [1]."UML State Machine" can solve deadline problem with timing constraint state. Exception mechanisms in "UML State Machine" can support where interactions are successful or unsuccessful. A bout the problem with an unknown number of participants at design time like it was the case for One –to- many send, we do like before and we opt for partial support for this pattern [7, 8].

5.3.3 Multi transmission interaction patterns:

Multi responses: In the case of multi responses, a party X sends a request to another party Y, subsequently, X receives any number of responses from Y until no further responses are required [4] . "UML State Machine" does this pattern by a state with a cycle that it repeating while existing no answer from Y as shown in figure 19.a.There is direct support for this pattern.

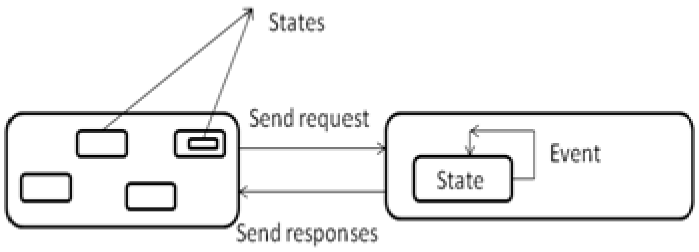

Figure 19.a Multi responses pattern [3,4,11]

Contingent request: A party X makes a request to another party Y, if X doesn't receive a response within a certain time frame; X alternatively sends a request to another party Z and so on. Responses from requests might be still considered or discarded. The limited time frame can be specified. Partner X makes a request to another party Y (in the form of event) and at the same time sends an event to timer state and activates it also [6]. When time is over ,an event from timer state send to state X and state X check buffer of it's receive state for every interaction (for every interaction a new buffer instance is assigned to the recipient state). The case where responses exists, state use them, if no state X activates cancel activity for Y partner and sends request to another party like Z. Arbitrary cycle in party X activated several times until a response arrives before the timeout occurs or the list of potential recipients has been reached. The selection of the next participant would be hidden how ever this only covers the cases where responses of previous request are discarded. The case where other responses are still considered requires a parallel execution of the request –response interactions. In "UML State Machine" this only works for case where the number of recipients is known at design time .As we shown in figure 20.a there is partial support for this pattern. [5, 7]

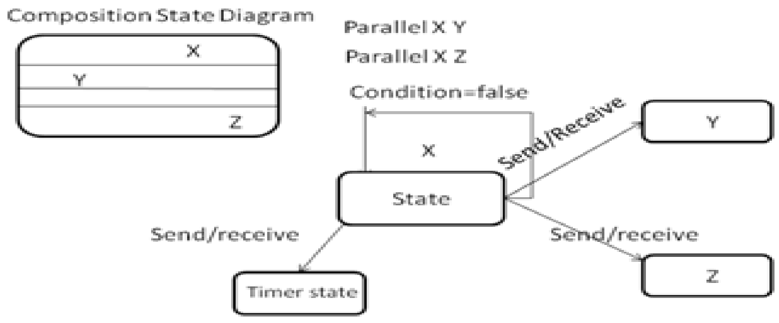

Figure 20.a Contingent Request pattern [5,7]

Atomic Multicast notification: A party sends notification to several parties such that certain a number of parties are required to accept notification within a certain time frame. For example all parties or just one party is required to accept notification. In general the constraint for successful notification applies over a range between a minimum and maximum number [9]. The limited timeframe can be specified through time state structure that has a counter which keeps time and can send an event to each of partners when time is finished. If partners don't receive messages in the certain time, they will send another message in the form of an event to sender state and request to send previous message again. In" UML State Machine" all of constraint about successful range can be implemented in ontology and be attached to "UML State Machine"[3].

5.3.4 Routing patterns

Request with Referral: Party A send request to party B indicating that any follow up responses should be sent to a number of other parties ($p_1, p_2, \dots, p_{n-1}, p_n$), depending on the evaluation of certain conditions."UML State Machine" can support this pattern by transitions and events like send and receive through states in choreography [9, 11].

Relayed Request: Party A makes a request to party B which delegates the request to other parties ($p_1, p_2, \dots, p_{n-1}, p_n$), then continue interactions with party A while party B observes a view of interactions including fault (when an error occurs, this error will be reported to A not B) [8, 11]."UML State Machine" links states by transitions, synchronizing or without synchronizing messages or events (with or without guard conditions). If a partner requires the response of other executing parties, "UML state machine" uses parallel patterns appeared with synchronizing messages or orthogonal region in composition state diagrams [12].

Dynamic Routing: A request is required to be routed to several parties based on a routing condition. The routing order is flexible and more than one party can be activated to receive a request .When the parties that were issued the request have completed, the next set of parties is passed the request. Routing can be subject to dynamic conditions based on a data contained in the original request or obtained in one of the intermediate steps. Since the pattern description is very uppercase, it is hard to judge whether or not this pattern is supported static routing orders can easily be expressed using sequence structure (states with transitions that link them together and specify sequence of interactions)and parallel structure that appears with synchronizing and without synchronizing messaging and composition state diagram. Dynamic routing orders in the sense that a participant can delete interactions or insert new interactions into the choreography at runtime may support partially because of separation of concepts. When new interaction inserts, UML state machine can add and defined events transition and can link between them by transition. We skip this pattern in the assessment [8, 13].

Table 2 Pattern Support in UML State Machine [1, 2, 4, 5, 7, 11]

Workflow patterns	**WS-CDL**	**WSCI**	**BPEL**	**UMLSTATEMACHINE**
1.Sequence	+	+	+	+
2.Parallel Split	+	+	+	+
3.Synchronization	+	+	+	+
4.Exclusive Choice	+	+	+	+
5.Simple Merge	+	+	+	+
6.Multiple Choice	+	-	+	+
7.Synchronizing Merge	+	-	+	+
8.Multiple Merge	+/-	+/-	-	+
9.Discriminator	-	-	-	-
10.Arbitrary Cycle	-	-	-	+
11.Implicit Termination	+	+	+	+
12.MI without synchronization	+	+	+	+
13.MI with a priori design time knowledge	+	+	+	+
14.MI with a priori run time knowledge	-	-	-	+
15.MI with no a priori run time knowledge	-	-	-	+/-
16.Deferred Choice	+	+	+	+
17.Interleaved Parallel Routing	-	-	+/-	+
18.Milestone	+	-	-	+
19.Cancel Activity	+	+	+	+
20.Cancel Case	+	+	+	+
Service Interaction Patterns	**WS-CDL**			**UML State Machine**
1.Send	+			+
2.Receive	+			+
3.Send/Receive	+			+

4.Racing incoming messages	+			+
5.One-to-many send	+/-			+/-
6.One-from –many receive	+			+
7.One –to-many send/receive	+/-			+/-
8.Multi-responses	+			+
9.Contingent requests	+/-			+/-
10.Atomic multicast notification	-			+/-
11.Request with referral	+			+
12.Relayed request	+			+

6. CONCLUSION AND PLAN FOR FUTURE WORK

This paper has discussed about supporting of "UML State Machine" to service interactions and workflow patterns. Table 1 summarizes which workflow and service interactions patterns are supported by UML State Machine. In analogy to the mentioned assessments of other process modeling language we assign a "+" for direct support of patterns,"+/-" for partial support and "-" for lack of support. The table shows that "UML State Machine" have enough potential for supporting choreography. In spite of WSCDL that only describes global view and unable to describing local view, "UML State Machine" can describe both of views. Because of dynamic modalities of "UML State Machine" ,expect it can support dynamic specifications of choreography .Before all of that, this paper should prove ability of "UML State Machine" for supporting patterns(work flow, service interaction) in choreography at least about WSCDL. In future work, could be discussed a bout dynamic specifications of "UML State Machine"(e.g., state-based, nested state-based and capable of seperating of concerns) and their impressions in dynamic aspects of service interaction patterns that previous choreography modeling language can't support that for lack of dynamic properties.

REFERENCE

[1] M. Barros, M. Dumas & P. Oaks. (2005),"A Critical Overview of the Web Services Choreography Description Language (WS-CDL)", BPTrends [Online accessed: Jan, 2011]. Available:http://www.bptrends.com

[2] G. Decker, O. Kopp, F. Leymann & M. Weske, (2007), "BPEL4Chor: Extending BPEL for Modeling Choreographies". Proceedings of the IEEE 2007 International Conference on Web Services, IEEE Computer Society, pp: 296-303.

[3] G. Booch, J. Rumbaug & I. Jacobson, The Unified Modeling Language User Guide, Addison - Wesley,1999

[4] PetiaWohed,WilM.PvanderAalst, MarlonDumas, ArthurterHofstede, and N.Russell .On the Suitability of BPMNfor Business Proces sModelling. In Proceedings 4th International Conferenceon Business Process Management (BPM 2006), LNCS, Vienna, Austria, 2006. Springer Verlag.

[5] N. Russell, Wil M.P. van der Aalst, Arthur ter Hofstede, and Petia Wohed. On theSuitability of UML 2.0 Activity Diagrams for Business Process Modelling. In Proceedings 3rd Asia-Pacific Conference on Conceptual Modelling (APCCM 2006), volume 53 of CRPIT, pages 95–104, Hobart, Australia, 2006.

[6] Assaf Arkin et al. Web Service Choreography Interface (WSCI) 1.0. Technical report, Aug 2002. http://www.w3.org/TR/2002/NOTE-wsci-20020808.

[7] Petia Wohed,Wil M.P. van der Aalst, Marlon Dumas, and Arthur ter Hofstede.Analysis of Web Services Composition Languages: The Case of BPEL4WS. InProceedings 22nd International Conference on Conceptual Modeling (ER 2003),volume 2813 of LNCS, pages 200–215. Springer Verlag, 2003.

[8] Qiao Xiaoqiang, Wei Jun, "A Decentralized Services Choreography Approach for Business Collaboration, IEEE International Conference on Services Computing (SCC'06), 2006.

[9] M.P Papazoglou , P. Traverso, Schabram Dustdar & F. Leymann, (2007) ," Service-Oriented Computing: State of the Art and Research Challenges", Journal of Innovative Technology for Computer Professionals, IEEE Computer Society, November 2007

[10] W3C, (2005) “WS-CDL XSD schema”, URI=http://www.w3.org/TR/2005/CR-ws-cdl-10-20051109/#WS-CDL-XSD-Schemas (online accessed = Feb 2011).

[11] Gero Decker, Hagen Overdick,Johannes Maria Zaha.(2007),"On the Suitability of WS-CDL for Choreography Modeling" SAP Research Centre Brisbane.

[12] Kim hamilton&RussMiles,(2006)LearningUML2.0, 978-964-2971-18-3,Paarseh publication.

[13] Alistair Barros, Marlon Dumas, and Arthur ter Hofstede. Service Interaction Patterns.In Proceedings 3rd International Conference on Business Process Management(BPM 2005), pages 302–318, Nancy, France, 2005. Springer Verlag.

[14] Barros, A.P., Dumas, M., ter Hofstede, A.H.M.: Service interaction patterns. In:van der Aalst,W.M.P., Benatallah, B., Casati, F., Curbera, F., eds.: Business Process Management. Volume 3649. (2005)302–318

[15] F. Mardukhi, N. Nematbakhsh, K. Zamanifar , (2011), An Adaptive Service Choreography approach based on Ontology-Driven Policy Refinement, International Journal of Web & Semantic Technology (IJWesT) Vol.2, No.2, April 2011

[16] A. Bandara, E. Lupu, J. Moffett & A. Russo, (2004), A goal-based approach to policy refinement, Policies for Distribute SystemsandNetworks,2004.POLICY2004.Proceedings.FifthIEEEInternational Workshop on 7-9 June 2004 Page(s):229 239.

[17] Gustavo Ansaldi Oliva, Fernando Hattori, Leonardo Alexandre Ferreira Leite, Marco AurélioGerosa," Web Services Choreographies Adaptation:A Systematic Review", Technical Report No: RT-MAC-2011-02, January 2011URL: http://www.stattools.net/CohenKappa_Exp.php

[18] Hohpe, G. & Woolf, B. (2004). Enterprise integration patterns: Designing, building, and deploying messaging solutions. Addison-Wesley.

[19] Ezenwoye, O.; Bin Tang; , "Monitoring Decentralized Interacting Services with a Global State Choreography Model, " Web Services (ICWS), 2010 IEEE International Conference on , vol., no., pp.671-672,5-10 July2010, doi: 10.1109/ICWS.2010.108, URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5552718&isnumber=5552704

[20] Pahl, C. (2010). Dynamic Adaptive Service Architecture – Towards Coordinated Service Composition. Software Architecture. M. Babar I. Gorton, Springer Berlin / Heidelberg. 6285: 472-475, http://www.springerlink.com/content/v57u642746022218/

[21] N. Lohmann, O. Kopp, F. Leymann, and W. Reisig, "Analyzing BPEL4Chor: Verification and Participant Synthesis," Proc. Fourth Int'l Workshop Web Services and Formal Methods (WS-FM '08), M.DumasandR.Heckel,eds.,pp.46-60,2008.

[22] Miro Samek , Ph.D , Hierarchical State Machines – a Fundamentally Important Way of Design , Association of C&C++ users, March 11 , 2003.

[23] Prof . Dr . Detlef Seese , Prof . Dr . Rudi Studer , Modeling Workflow patterns through a control flow perspective using BPMN and the BPM Modeler BizAgi : BizAgi . http://www.bizagi.com

Authors

A. Mellat received her B.Sc degree in Computer Engineering from Azad University of Najafabad, Iran in 2003.Currently she is pursuing her master degree in the Software Engineering in PayamNoor University, Tehran. Her interests include Web Service technology and coordination problem. She is working is on dynamic choreography models for Web services in B2B corporation.

Dr.N.Nematbakhsh received his B.S degree of Science in Mathematics from Isfahan University, Iran in 1973 and Master of Science in Computer Science in1978 from Worcester Polytechnic Institute, USA. He received the PhD in Computer Engineering from University of Bradford, England in 1989.working as Assistant Professor in the Department of Computer Engineering, Isfahan University. His experienced areas include Software Engineering Methods, Service Oriented Computing and Software Reliability.

Dr.A. Farahi received the PhD degree in Computer Sciences from Bradford University. He has been working as a full-time faculty member, Assistant Professor and Head of Computer and Information Technology Department in the PayameNoor University. His research interests are programming especially that of educational systems.

F.Mardukhi received his B.Sc degree in Computer Engineering from Sharif University of Technology, Iran in 1996 and Master of Software Engineering from University of Isfahan, Iran in 2002. Currently he is pursuing his PhD degree in the Engineering Faculty of Engineering on Web Service technology, Coordination problem, and adaptive software systems. He is working on dynamic and adaptive choreography models for Web services inB2Bcoporation