An Adaptive Service Choreography approach based on Ontology-Driven Policy Refinement

Farhad Mardukhi, Naser NematBaksh and Kamran Zamanifar

Department of Computer Engineering, University of Isfahan, Isfahan
{Mardukhi, Nemat, Zamanifar}@eng.ui.ac.ir

ABSTRACT

Business corporations usually require choreography of services to be dynamic and adaptable. One way for answering this demand is to develop the services having dynamic behaviours. However, it is not enough and their behaviours must be composed dynamically too.
The current model such as WS-CDL has a static structure to specify choreography and is not able to describe the choreography of services in a dynamic fashion. From another view, there are various types of changes that each needs to be handled diversely. This work is going to bring adaptability into service choreography model in response to policy changes. Precisely, this target will be met when choreography model has dynamic structure and also the policy changes can be automatically (semi-automatically) refined and propagated into the elements of model. Our proposal is describing choreography model on the basis policy-enabled UML state machine which the policy can be refined through a ontology based process.

KEYWORDS

Adaptive Systems, Service-Oriented Systems, Choreography, UML Stat Machine, Adaptable Choreography Mode, Behaviour, Policy refinement.

1. INTRODUCTION

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. Adaptation is alternation of a system's behaviour to address arbitrary environment changes. Separating computation components from interactions and coordination mechanisms, and also making both components and interactions adaptable are two main methods in general to cope with the continuing changes in dynamic environments [14].

The current service based collaboration model is not able to tune and configure the services behaviour to being used as well as. Service behaviour explains how a service interacts with others? A service with multiple behaviours are those which can play different behaviours when collaborate with others. For example, a Login service which provides login behaviour both for a user who has user name and password, and for another who is new. In fact, the services with multiple behaviours have adaptive behaviours. But, just having services with adaptive behaviours is not sufficient, but these adaptive behaviours need to be managed dynamically at runtime.

For managing the behaviours of service dynamically, it is necessary to have an appropriate adaptable model for specifying a collaboration of services. In current SOA standard, the service based collaboration is studied through two known concepts: Choreography and Orchestration [3,18]. Therefore, having adaptable models of both are essential to providing the adaptable

DOI : 10.5121/ijwest.2011.2205

collaboration of services. This work targets to follow the choreography model since it has not been regarded in compare with orchestration model so far. Another point is that, for inter-organizational corporations, the policies are usually being changed which consequently impact on choreography model, and enforce it to be changed accordingly.

This work studies on an approach to handle automatically the impacts of policy changes on choreography model. We found that such approach requires at least two essential pre-requites. First a dynamic choreography model of services which is capable of altering its elements dynamically without designing the model by hand [21]. Second an approach to impact the policies on the structure and behaviour of choreography model. Usually the policies is changed by administrator who specify them at high-level of abstraction. Such policies need to be decomposed and refined into enforceable policies at a low level of abstraction through a process namely policy refinement.

The proposed model is policy-enable UML state machine as a dynamic model for describing the choreography of services [9]. This model is dynamics because it is basically follow the event-driven paradigm for modelling. A finite number of distributed agents communicate to establish a choreography model. Each agent executes an UML state machine enriched by policies to control its behaviour correctly.

The suggested policy refinement process considers the policies into two main types, measurable QoS-based polices and Immeasurable policies, then presents two mechanisms for decomposing them. For first type, the aggregated quality function annotated for the choreography model is decomposed to individual quality measures associated to choreography partners. About the second type, the policies are decomposed into lower levels on the basis of ontology concept decomposition.

At the rest of this paper, the second sections overviews the related concepts, then the proposed dynamic model is described at third section. The policy refinement process is discussed in section four and finally the

2. DECENTRALIZED SERVICES CHOREOGRAPHY APPROACH

In a service-oriented environment, a number of services are combined to make a composed service providing more capabilities. There are three main, though overlapping concepts: Choreography, Behavioural Interface and Orchestration [3,4]. Choreography describes how several services interact 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 [18]. Based on definition of W3C***, behavioural Interface*** shows how a service behaves in choreography. In other words, it explains the observable behaviour of a service relate with other parties which declare the dependencies between its interactions. An ***Orchestration*** view describes how a service manages the internal activities to supply its capabilities [6]. The internal actions include data transformation and invocation to internal software modules (e.g., legacy applications). An Orchestration model defines a set of "active rules" are executed to manage the behaviour of a participant which is described in choreography model.

Most known service specifications like WSCI [15] and BPEL4Chor [10] consider these concepts very close to the view of WS-CDL in a choreography specification of W3C. Though, WSCI refer to choreography from view of one participating service. It describes the observable behaviour of a service relative to the message exchanges the service must support. This definition of choreography matches with *Behaviour Interface* term in W3C.

Recently a little different view is presented by WSMO. The Web Service Modelling Ontology (WSMO) provides a conceptual framework and a formal language for semantically describing all relevant aspects of Web services in order to facilitate the automation of discovering, composing and invoking electronic services over the Web [16]. Based on WSMO view, choreography describes how one service interacts with its users. While choreography in W3C and SOAS is described respectively by a non-executable languages WS-CDL and WSCI [10, 4], WSMO describes choreography by an executable language based on abstract state machine.
Choreography, in view of this works inspired of [2, 3, 15, 18, and 17] and also regarding to the level of choreography abstraction in W3C specification [8], defines a set of independent roles and their interaction behaviours. The roles are distributed on participating parties and each one provides corresponding interaction behaviours. The choreography requirements will be provided when each role provides its interaction behaviour. The interaction behaviour determines a set of active rules which a role must execute to take part correctly in choreography. According to this view, an executable choreography framework can be defined to specify model, approach, and language and execution engine of choreography. Therefore, conversely to the most research opinions in which choreography is only a description language, in this work, we refer to choreography at two levels of abstractions. In one level, choreography is description language to interactions amongst services at the level of public view. In a lower level, local view, choreography regarded as an executable code to manage interaction behaviours of one service against others at runtime. In the public view, choreography is a centralized abstract process basically using WS-CDL language which describes observable behaviours of services which collaborate to obtain a common goal and from local point of view, choreography is decentralized processes amongst the participants basically using UML state machine describes interaction behaviour of each service required to take part in collaboration. Therefore, a choreography solution developed for service collaboration across participate boundaries while preserving the local autonomy of their own business processes. This solution is a decentralized approach to coordinate orchestration processes through a choreography description which is implemented by a set of observable service behaviours.

2.1. UML STATE MACHINE

Today, the UML state machine [8] along with other elements of UML is leveraged broadly in object-oriented application development. UML state machine is a hierarchical model of system behavior showing how system reacts to the events happened at some states of system. It is composed of nested states, transition between them and actions.

UML state machines preserved the form of traditional FSM and introduce new concepts. The most important concept of UML state machine in compare with FSM is hierarchically nested states. Hierarchy means that the states and machines can be built within other states and machines. This hierarchal processing of elements makes it easier to describe complex conditions and transitions. Another important new concept of UML state machine is orthogonal regions [which refer to two or more independent, complement and concurrent active regions making a large region named composed state.

The activities (actions) of the UML state machine can be described as an action within the state itself or expressed as an action on the state transition. Also, it is possible to mix and match these styles depending on the problem at hand. Also, we found that we needed ways to handle common default conditions and exceptional situations. Hierarchy was a perfect solution as it permits factoring common code, allowing more natural and concise expression. UML state machines are event driven; actions come in the form of events that are presented to the state machine, typically driving transitions.

3 PROPOSED ADAPTIVE CHOREOGRAPHY MODEL

In this section, the adaptive choreography model is proposed on the basis of UML state machine as a dynamic structure to show choreography of model. Firstly, the essential requirements for bring adaptability into choreography model is described, and then we argue that providing dynamic structure of choreography model is the key addressing the adaptability requirements.

3.1. EXPECTED ADAPTATION REQUIREMENTS IN OUR MODEL

Adaptation is a relationship between a system and its environment where change is initiated to facilitate the survival of the system in that environment. Adaptation in our view generally refers to *ontogenetic adaptation which* is the ability of a system to regulate itself and reconfigure its structure as it interacts with the environment [1, 14]. Also, it is important that such system can manage adaptation mechanisms during the life cycle of system. In context of software systems, adaptation above can be phrased as:

Adaptive software architecture = Dynamic structure + Management [5], where management activities monitor the system, decide and command to reconfigure the structure and regulate the behaviour over that structure. Dynamic structure means that the structure of system must be flexible to being reconfigured and regulated dynamically in response to commands of management activities.

In this research, we are going to provide a dynamic structure for choreography system of a service based collaboration and management activities are out of this research. This work uses the specified term “engine’ instead of system.

Therefore, adaptation is ability of the choreography engine to regulate and reconfigure its parts dynamically. Recall from previous section, the model of choreography engine is a set of state machines which each execute the behaviour of one participated roles.

TABLE I. List of Adaptation Requirements

Property	Description
(re)Configuration	
• Reconfiguration possible at runtime.	Can the components be added or removed? Can the connections between components in the structure be changed at runtime?
• Functionality recursive structure.	Do configured composites of components themselves form a unity that can be configured into larger composites?
• Non-functional restructuring supported.	Can multiple components of the same type be created in parallel to serve a single functional output?
• Elements can be substituted.	Can one component be substituted for another component at runtime? Can the components be *safely* substituted?
• Composition based on declarative description possible at runtime.	Is it possible at runtime to create compositions from declarative descriptions, or does the basic structure have to be defined at compile time?
• Formal reasoning about structure possible	Can the structure be formally represented so that it can be reasoned about? For example, can proposed compositions be checked for integrity?
• Elements can be reused or can use other reusable sub elements.	Can the elements be expressed in explicit representation allowing reusability of elements?
Regulation	
• Non-functional regulation possible.	Does the application have the ability to monitor non-functional properties, and adjust its behaivour accordingly?
• Constraints rules and policies regulation possible	Does the application have the ability to change the constraints which governs its behaivour?
• Concepts definition changing possible	Is it possible to change definition of concepts and their properties when system is running?

Each state machine is set of states that machine can pass through, and the transition rules which describes state exchanges as result of occurring the events and existing conditions. The machine performs the actions (operations) when is placed in a state (Moor machine) or during a transition.

The operations of a state machine are limited to a number of operation types which are defined at choreography standards, see section 5. On the basis of this definition, adaptable choreography engine is set of adaptable state machines, each has a dynamic structure. Hence, our attention is to build an adaptable state machine providing a dynamic architecture enabling corresponding participating role to manage its behaviour dynamically at run time of system.

Inspired of work [17, 5], dynamic structure of desirable state machine requires a variety of properties which a number of them are followed. These distinguishing characteristics can be seen as an elaboration of the desirable properties and determining the specified goals for adaptable choreography engine. Also, these characteristics create a framework which can be used to evaluate our work with others.

A multi interface service is a service providing multiple interface of a function (operation). For example, a shipment service has an abstract function namely Post which is implemented by two TNT_POST and USUAL_POST functions. Such service can play Post role in a collaboration using each of this concrete service.

3.2. CHOREOGRAPHY=DISTRIBUTED SERVICE OBSERVABLE BEHAVIOURS

Currently in service oriented systems, there is an increasingly need to make the applications more adaptable. Complexities like heterogeneity, dynamism and uncertain conditions of environment and user requirements [17, 18].

Such complexities define key requirements for service-oriented applications. Two much important of such requirements which are now promised are [2, 14]:

(1) Separation of concerns; for example computational behaviours of services should be separated from interaction and coordination behaviours, and their combination should be programmable. (2) Adaptability of concerns; for example the computation and interaction/coordination behaviour should be able to adapt themselves with the changes in the environment and dynamic application requirements. The first requirement is mostly overcome by current industrial application development environment. But, the second requirement is still open research domain and there are only limited techniques to cover it. This research is adapting interaction/coordination behaviour scope. On the basis of architecture of SOA [2] and with regards to the above requirements, we decompose the behaviour of a service which collaborates with others into three main layers: (1) Computational behaviour: shows what the component does. It includes the functionalities of a component which delivered on its interface; for example port definition at WSDL. (2) Composition behaviour: shows how a component controls its interactions with others and accordingly how uses computation behaviours from itself or others to behave correctly in composition. (3) Management behaviour: shows how a component manages its behaviours. It is usually expressed by high level rules derived from human knowledge to manage and configure the runtime component behaviour.

The second layer, composition layer, is divided into two sub layers: internal-action layer and interaction (choreography) layers. The former describes how a service must control its internal actions to play in collaboration. The choreography layer concerns with the observable behaviour of a service must have during its interactions with others relevant to the pre-defined choreography. Though we consider these sub layers separately, most standards like WS-CDL specify them together because of complexity of separation. Of course, these standards conceptually distinguish these sub layers but they are described in a unified language since they are much interrelated.

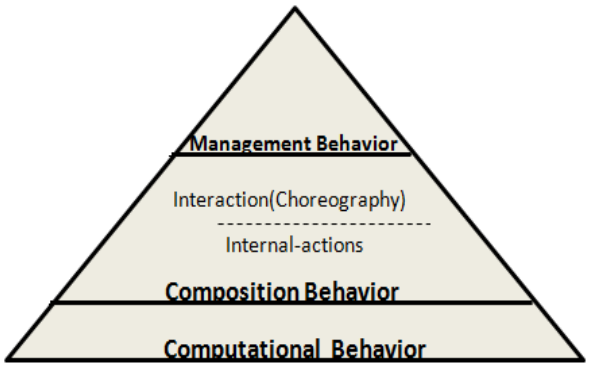

Figure 1. The layers of service behaviours

We point to service interactions through business collaboration as a known term: service choreography. This work focuses on interaction (choreography) sub layer and intends to provide an adaptive approach for this layer (see figure 1). The main goal is realizing adaptive choreography of services especially in response to changes of service collaboration rules and

policies. Adaptability of service choreography means each of them controls its interactions with others adaptively to address changes in requirement and condition of environment.
For developing an appropriate solution we consider the following main strategies:

- It is better to use promised technologies and standard languages as much as possible and extending them if necessary.
- Developers use high-level and simple model to design choreography and transform it automatically to low-level model as possible as.
- The engine should provide a platform able to use of reusable and compostable behavioural elements.

3.3. SEMANTIC MODEL OF CHOREOGRAPHY

In this section, we concentrate on a semantic model describing the proposed service choreography model. To achieve the objectives of this work, the choreography model is constructed on the basis of UML state machine controlled by a set of rules. Accordingly, the choreography model behaves dynamically which is capable of changing its behaviour at runtime.

The choreography model is realized when a set of local choreographies are modelled. Our proposed model is a set of distributed local choreography engine, namely LCE establishes a decentralized approach to choreography of collaborative services. Each LCE is located at the boundary of each participant as shown at figure 2. Based on this figure, choreography domain includes a number of observable behaviours domains. For any domain there is a UML state machine activates under the control of rules defined at choreography domain. The rules enforce the machines to behave according to with the relevant PP, public protocol, and its defined behaviour. Therefore, when all machines work in according to these rules finally the choreography of parties is obtained.

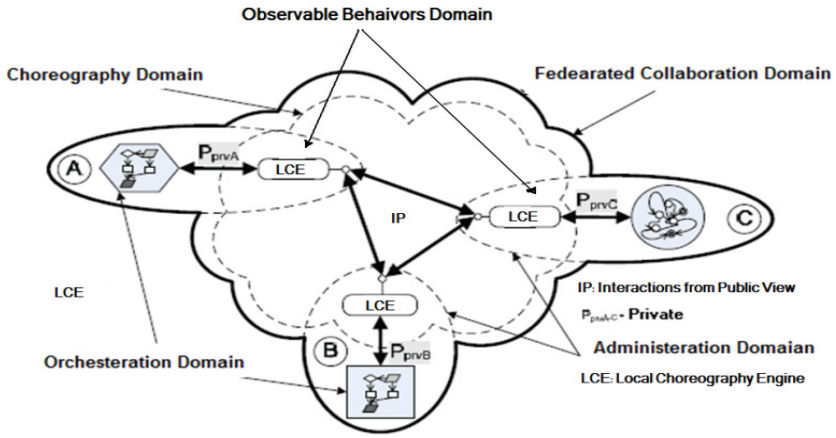

Figure 2. The distributed choreography conceptual model [25]

Each LCE is modelled as a UML state machine which represents a specified situation of a choreography process. A situation of choreography is defined by values of some concept instances which are important at that situation. For example in a typical scenario for build to

order (BTO), for state "offering" the values of *offer properties* and *number of tries* to make agreement on offer declare the situation of the choreography. Thus, to capture the situation/data of choreography we assign some concept instances to each state. Concept instances are instantiated from ontology concepts which are defined at the OWL schema [19] attached to the choreography model. Since the states are nested in UML, a concept instance is accessible through the state where is defined and all of its internal states.

The states can be associated to some quality requirements too. This quality requirements define the essentials must be obtained during that state. For example, if payment *state* are assigned to the response time=5ms. It means that the maximum long time that the payment state allowed to be finalized is 5ms.

In addition, there is a variety of policies defined that are applied while the control of machine is passing through a state. Each policy is a rule which describes a decision or a constraint on choreography model. The UML state machine is subject to various business rules that govern the way a choreography participant behaves. A business rule is a conditional statement alike "Provide credit payment only for customers who are golden and build an order with total value larger than 1000".

The dynamic semantic of model basically is based on events. While an event occurs, the model has the capability to react to it. Events can be happened in the form of receiving a message, a new decision and system faults and exceptions. To capture the dynamic semantic of events, it is essential to model the three following main concerns:

- How are the events queued up to be consumed by state machine?
- How does the event dispatcher mechanism work?
- How does the event processor consume the event instances?

The first and second concerns are not provided by standard UML state machine and depend on the application of state machine. More information about them is given in the rest of paper. The third concern is described at the semantic of UML state machine.

The actions are carried out in response to events by state machine. Regarding to requirements of choreography model, the actions are divided into two main classes: operational and policy-driven actions. The first class includes the basic actions which usually are defined in the standard choreography language like WS-CDL. The second class refers to the actions which must be performed in result of firing a rule.

In this proposed model, the events and actions are managed respectively by the event provider and control object entities. These main entities are linked to state machine. For any choreography model there is at least one instance for each of them. The event provider first gathers the events from diverse sources then insert them in the queue and finally inject them into machine. The control object is an object that collects related actions under control of same entity. The state machine concerns the control object as the executor for actions. While the machine is transiting, come into a state or leave it may ask the control object to perform actions by calling them. In such way the whole choreography model is separated into independent concerns.

To sum up, if we want to design a choreography using this proposed model, we have to describe the event providers, the controlled objects and the automata which will react to event by transiting from one state to another along with calling corresponding actions of controlled objects.

4. POLICY REFINEMENT

Policy is a declarative word which is defined based on different views [13]. Generally, policy is set of rules to describe the goals, decisions, actions, or constraints of a system [11].
Policy is a set of rules and constraints which control the behaviour of a system during its life. Policy is as an aspect of the information, which can impact the behaviour of objects in the system. The policy based management system is a promised and modern paradigm that dynamically manages the behaviour of a system correctly in accordance with the business goals [13]. The most of current software development processes encompasses a separate process to model the policies of software. On the basis of principle **"separation of concerns"** which is followed by almost all development process, the policy is a main concept should be separated from other concepts such as functions, interaction protocol, data access and interface.

4.1. POLICY TYPES

Policies are defined at different levels of abstraction ranging from those declaring the goals of a system to enforceable policies for individual resources. The administrators express the goals by a set of high level policies which are not executable unless they decomposed to a set of low level policies; each is enforced on a specified resource. Usually this task is done by designers through a process namely policy refinement. Policy refinement derives low level policies from high level policies specified by stakeholders in abstract style [20, 11]. Process of policy refinement is very complicated because a lot of manual operations are needed to extract low level enforceable declared based on pre defined properties of system. For example, a policy may enforce an entity to perform an action when an event is occurred and the entity has particular state. The entity state is exposed by values of its important attributes. The events, actions and attributes must be defined previously at the system.

Policy refinement for service Choreography: An essential element for enabling SOA application with policy- is the ability to automatically derive individual services policies from the policy of service choreography and vice versa. This is very important because it saves policy editors from having to manually codify policies, which is time consuming and error-prone.

In this wok, the choreography of services is subjected to policies for adding some considerations. A policy from our view is a rule describes how the choreography must do to obtain the ultimate goals. We assumed that the services participating in choreography are multi behavioural. Therefore, they must know how to adjust their behaviours in choreography. For example, suppose that a seller service has behaviour for payment both for *Debit* and *Credit* types. For some choreography instances, the Debit payment is displayed and for some else the Credit. Therefore, the seller service uses a policy to control its payment behaviour.

Clearly, developing a mechanism for deriving the low level policies needs primarily a formal and reasonable language for describing the policies. Also, there must be relationship, directly or indirectly, between both levels of policies. Therefore, we define the levels of policies based on two following trade-off goals:

- It's easy to be defined by administrators.
- It's possible to apply low level policies directly on service behaviours.

As the policies in this work are categorized at four main levels:

High level Policies: The goals and objectives of a chorography are declared by these policies. These types of policies are abstract and usually needs manual operations to be refined. For example, "User satisfaction is the major desire" is a sample for this kind of policies.

Choreography-Level Policy: The policies which manage the flows of control and data though process of choreography. These policies are described depends on ontology concepts which are meaningful between the participants of services from public point of view. This level of policy is regarded as high-level policy for policy refinement process.
the following are cases for this type of policy:

- A "Golden Customer" is a customer who has spent more than $10000 in services. Golden Customers are entitled a 10% discount on new orders.
- The 10% discount for golden customers does not apply for packaging products
- The TNT_Service is used for shipping the orders which are built for packages by golden customer.
- The aggregated price of a choreography should not be more than 30

Behavioural-Level policy: This type of policy controls the observable behaviour of a participant for playing a role of choreography. Since the observable behaviour of a participant is shown by a UML state machine, the policies are associated to the state of machines. For each state, the policies are checked when enter into/exit from that state. Each policy in this level is defined by a rule having ECA template, where E is a set of events, C is a condition expression and A is a list of actions. E and C are or joined for each rule. The following example is an
E: Golden_User
C: Order.Type==Packaged
A: Perform Credit_Payment

Service-Level Policy: This type of policy allows services to specify requirements and capabilities needed for establishing a connection with them. The known language for expressing this type of policies is WS-Policy. The *Web Services Policy Framework* is a standard that supports the specification of various quality properties for Web Services and service systems.
From another aspect, policies are also divided into two main categories:

Measurable quality-driven policy: A policy which express a consideration about a measurable quality property. A quality property specifies a desired quality aspect, for example cost, and availability, which can be declared in terms of numbers. For example, the following are two cases for this type of policy respectively at choreography-level and behavioural-level:

- It is desire to perform the choreography of services whose cost less than 20.
- It is assumed that a specified participant of choreography provides payment service with maximum cost 5.

Business-driven policy: Such policy specifies a business decision about controlling the behaviour of an entity or adjusting its uncountable quality requirements. Two examples are followed:

- For a golden customer which request for a packaged product perform TNT_service for shipping.
- All data exchanged with seller part must be encrypted.

This work focuses on deriving behavioural-level policies from choreography policies. The figure 3 shows this issue.

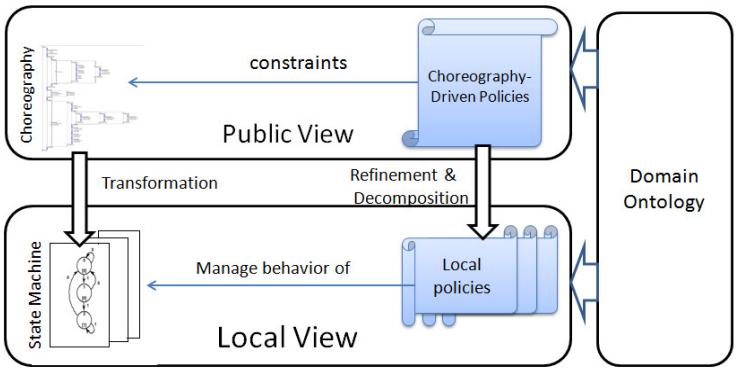

Figure 3. The overall process of policy refinement

4.2. BUSINESS-DRIVEN POLICY REFINEMENT PROCESS

In this section, we are going to discuss on how policy refinement brings adaptability to choreography model. When a new decision is made by managers, the policy refinement transforms it into the low levels of policies. Then, each low level policy must be deployed to one choreography participant and associated to a specified state through a decomposition sub process. Therefore, this work performs the policy refinement through two main sub processes: Transformation and Decomposition. Figure 4 shows this scenario in which both sub process use ontology model to decompose and enforce the high-level policies.

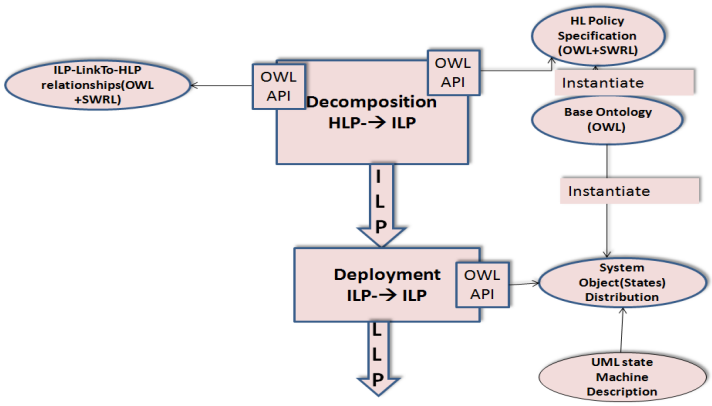

Figure 4. The outlined process of Policy refinement

4.2. 1. Policy Decomposition

This sub process is done on the basis of ontology concept decomposition. The policies on two both level of granularity, choreography and observable behaviour levels are described by adopting ontology concepts. OWL, the Web ontology Language, is proposed as a language for declaring the policies and more of information aspect of choreography. Currently, the OWL is enriched by SWRL [21], as a semantic rule language, to expressing the ontology not only in hierarchal of concepts but also in rule based relation among them.

The process of policy decomposition can be summarized in the following steps:

1. Preparing the OWL file for polices at two levels.
   A very simple of ontology model for our case is followed.

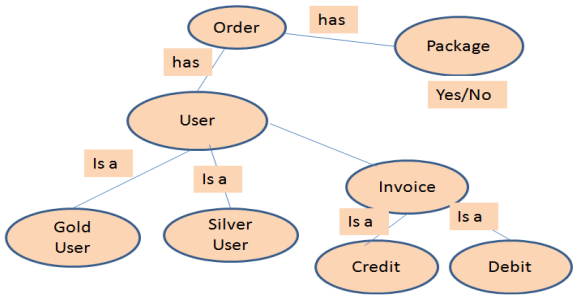

Figure 5. An outline of domain ontology for a case study

Also, the high level policies are describe depends on ontology concepts in a sort of SWRL rules. For example, a sample of policies in high level of abstraction is below:

if package(?order,?pckg) ^ swrl:eq(?pckg,"Yes") and user(?order,?User)^ srwl::Isinstanceof(?User,GoldUser) ➔ Shipement (?order, "TnT service")

This rule declares that for users who are golden and make the orders for packaged products, the TnT service is proposed for shipment their orders. In this rules the concepts which are used are defined for choreography level of services. In other words, these concepts are meaningful for all participants of choreography.

2. Relating the concepts of ontology at high and low levels by means of meaningful OWL relationships between HL and LL classes.
   These relations are derived from a relation template to make the associations among the ontology concepts. A relation template can be expressed at the style as:
   LowProperty(?LClassx) ^ HasRelation(?LClassx, ?HClassy)➔ HProperty(?HClassy)
   For exemplify this template, the following shows two cases:

   - **Items(?Order, ?Items) ^ Swrl::Subset(?items, ['HDD','CPU','MB']) ➔ Package(?Order,"Yes").**
   - **Type(?User, ?kind) ^ Swrl::Subset(?kind,'A') ^ amount(?order,?toral)^ Swrl::Largethan(?total,100) V Type(?User, ?kind) ^ Swrl::Subset(?kind,'B') ^ amount(?order,?toral)^ Swrl::Largethan(?total,1000))➔ srwl::Isinstanceof(?User,GoldUser)**

3. Ultimately, the necessary information is extracted from policies at high level by inference on SWRL rules which are prepared by developers. The necessary information is specified by low level policies which defined based on concepts which are meaningful for some participants of choreography. Actually, in this step the low level policies will be generated accord to information model of choreography. For example, a low level policy is shown as below:

   Items(?Order, ?Items) ^ Swrl::Subset(?items, [‘HDD’,’CPU’,’MB’]) ^ (Type(?User, ?kind) ^ Swrl::Subset(?kind,’A’) ^ amount(?order,?toral)^ Swrl::Largethan(?total,100)) V If (Type(?User, ?kind) ^ Swrl::Subset(?kind,’B’) ^ amount(?order,?toral)^ Swrl::Largethan(?total,1000))➔ Shipement (?order, “TnT service”)

We assumed that products of a package include ‘HDD’, ‘CPU’ and ‘MB’. This low level of policy is can be enforced because all the elements and information which are used are indentified for choreography, but not for all participants. For example, the seller is aware of user type but the shipment is not. Therefore, it is necessary to deploy the rules to according participants. In this regard, firstly the policy must be divided into some parts and each is located at a participant engine which access to essential information.

4.2. 2. Policy Deployment

This sub process targets to locate and deploy the decomposed policies. For that, it is necessary to place each policy or a part of it into a participant domain where the information needed for that policy is presented. For example, if a policy is defined on order properties must be enforced on a participant which access to order data type. We performed this activity through a locating process of entities by use of information model of concepts. All instances of ontology which are adopted in choreography are associated with a place they are defined. In our choreography model, each ontology instance is assigned to a state of machine. This feature enables us to associate a definite scope for any ontology concept. Therefore, each concept instance is owned by a state of machine. Since overall choreography is spawn of a distributed UML state machine which in turn each machine is spread its logics overall some nested states.

In this sub process, the information model of concept instances is used to locate the policies. The information is supposed to be declared through a tree of hierarchal concepts which are assigned to the states of machines, as figure 6 shows.

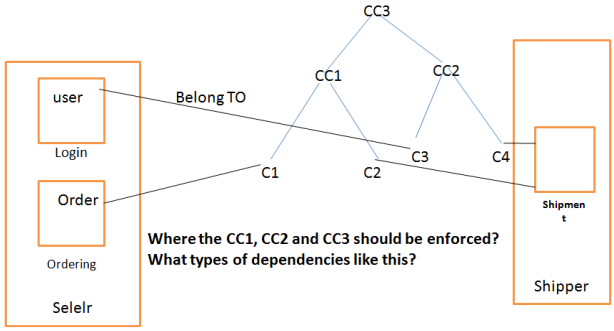

Figure 6 a snapshot of a decomposing a rule to set of sub rules

Regard to figure 6, an overall low level policy as CC3 must be enforced on state machines. To do that, the CC is decomposed into CC1 and CC2. Each of them respectively is divided into C1& C2, and C3 & C3. C1 and C2 in this example are associated to User state. User state itself is an internal state of a state machine belonged to seller partner. Finally, each low level policy must be deployed on corresponding state. In our choreography model, we annotate a state with such deployed policy. Clearly, when a policy is decomposed as some relevant policies which each one may be deployed in independent place, it is needed to making them connected in way that they act totally as a unified policy. This requirement is addressed through even distribution and exchange. For example, when the policies which are declared by the seller are fired, then *Seller* must send an appropriate event to the ***Shipper***.

4.3. MEASURABLE QUALITY-DRIVEN POLICY REFINEMENT PROCESS

In this section we present a novel method to decompose and refinement the quality-driven policies into low level policies. Response time, price and availability are three measurable quality criteria in which each consideration about them for choreography can be regarded as a quality-driven policy. For a choreography which is subjected to a number of measurable qualities, there must be a way to find an optimal definite set of candidate services which finally provide the best aggregated quality. This problem is fundamentals in domain of service composition domain [12]. Almost works leverage optimization techniques to solve this problem. Often, the related works attempt to find the best composition among all candidate composition of services. In their approach, the composition logic is presented by a process of abstract services. Such service combines through a set of control structures like sequence, parallel, switch and so on. For any abstract service there are a lot of concrete services which in make the composition to be created in huge different combinations.

In this work, we are going to bind each abstract service to a set of quality boundaries so that the aggregate quality is obtained from choreography of services is supposed to become maximize as possible as. In other words, we derive some quality criteria limitations from whole choreography limitations defining some quality levels for each quality criteria. These levels increases the speed of algorithms aim to find the best composition of services because most of inappropriate services will be disregard.

Suppose that Q is aggregate quality value can be obtained from whole choreography, Q can be calculated as a result of linear programming problem as below:

$$\text{Maximize } Q = \sum_{k=1}^{m} \sum_{l=1}^{ML(k)} \sum_{i=1}^{n} P_{ki}^{l} * x_{k}^{l} * \left(\frac{q_{ki}^{l} - q_{ki}^{min}}{q_{ki}^{max} - q_{ki}^{min}}\right)$$

Subject to:

$$\sum_{i=1}^{n} q_{ki}^{l} \leq C_k \,,\ 1 \leq k \leq m$$

$$\sum_{l=1}^{ML(k)} x_{k}^{l} = 1,\ 1 \leq k \leq m$$

In this equation, ***k*** is an index for quality criteria and ***m*** is the number of quality criteria this work regards to. For example, price, response time, availability, reputation and reliability are some cases where some of them are interested in maximizing and other like to be minimized. This issue must be considered by programmer, for example by negating the quality value when calculated at all aggregated quality.

For showing levels particular quality criteria, $\boldsymbol{l}$ is used for this purpose. ML(k) is used for returning the maximum of levels can be considered for a quality. For example, if minimum price is able to obtain for an abstract service is 10 and maximum price is 100, then the levels can be exemplified at 9 levels. It is adjusted by developed when starting the linear programming. To point to services taking part in choreography, $\boldsymbol{i}$ is used to identified them and $\boldsymbol{n}$ is number of them. $\boldsymbol{P_{ki}^{l}}$ is a probability shows how much of candidate service for an abstract service, $\boldsymbol{i}$, can provide a specified level, $\boldsymbol{l}$, of particular quality, $\boldsymbol{k}$,.

q_{ki}^{max} and q_{ki}^{min} are respectively the maximum and minimum quality values are obtainable for a quality as price about a specified abstract service. To normalize the values of quality criteria and map them on [0,1] space, we use $(\frac{q_{ki}^{l} - q_{ki}^{min}}{q_{ki}^{max} - q_{ki}^{min}})$.

This linear programming equation can be solved by standard methods. Also, the problem definition declares some limitations which the solution must be subjected to. For example, the aggregate price of all choreography must not exceed that what customer defined as his/her desires.

When the above program is solved, the results are a set of quality levels for any abstract service. These levels are bind to each abstract service of choreography which is used as local search to disregard those concrete services are out of these levels of requirements. We can say that, the aggregate quality value of choreography is decomposed and refined to a set of independent quality values, each distinguish a required level of quality.

5. RELATED WORK

There are a variety of works follow the goals are close to those we attempt to achieve. Most of them try to bring adaptability into orchestration model. However, the works which are tightly close to this work are few.

WS-CDL [3,4] and WSCI [15] are two main standards have many constructs to describe the choreography of services. These languages do not address the model-driven relation between the abstract process of corporation and executable process which is defined by orchestration. Also, these standards are defined based on workflow view and cannot support adaptability as well because all the constructs must be defined during design time.

Some works add several extra structures to standard language supporting adaptability. For example, some structures are added to BPEL controlling exceptions at runtime. ECF, Executable Choreography Framework, proposed by [17] aiming to perform the choreography on peer-to-peer model which supporting adapting. ECF is an answer to make on-demand collaboration. In ECF, the choreography is modeled based on some active rules. It uses a distributed aspect platform to enable dynamic superimposition of collaboration activities. One of the main properties of ECF is managing control of choreography process through the partners so that the data and flow of any work are remaining under corresponding partner. ECF uses aspect-oriented programming to develop some aspects which can be executed in a chain way after getting a message by a partner.

Another known work which uses a same model to show choreography is WSMO [16]. Choreography in WSMO is described as abstract state machine. WSMO like our work uses the

ontology and rules to describe choreography logic, but despite our work it doesn't support nested states and also the choreography is defined from view of one partner to access the capability of a service, not as global view to a corporation.

Xiaoqiang and et al [1] proposes a model for choreography of services showing collaboration across organization boundaries. This model defines an approach containing three steps: Firstly, a central choreography process is developed showing the business collaboration, then at the second step, this process transformed to a decentralized process which span on the participating. The third step is mediating the decentralized choreography process with the inner orchestration process. Adaptability is gained on the use of this median. This mediator is capable of separating the changes of inner process and choreography contract rules, each one from another.

6. CONCLUSION

Adaptive systems are being requested increasingly especially for distributed Web based systems. Achieving the adapting system in general and particularly in choreography of services is main focuses of this research. We propose a new service choreography model which provides dynamic structures able to support the adaptability. It is a very useful model in dynamic corporations where the model of choreography (contract) and especially its constraints have ongoing changes.

We presented that UML state machine is a good model to specify choreography of services. One of the main advantage that was achieved is that state machine can be regards as an independent component of model which is independent of the actions are performed during corporation. This means that the state machine is separated from control objects which cover the actions. Attention to the primary goals of our work, the adaptability is obtained both in reconfiguration of model and regulation of it. Substitution of stats is possible which means the reconfiguration and the changes of transition rules are possible too which means the regulation of model.

The types of changes in a system are diverse which policy change is a key one. When changing a policy its effects will propagated its effects on the most elements of system. Related to choreography model, the changes of policy must be refined to propagate through the elements of choreography, this paper introduces a mechanism for refinement policies into distributed elements of choreography.

One of the main challenges we faces and not done yet is the verification of dynamic choreography model. We are going to do it as next work using the checking model like SPIN. Also transformation WS-CDL, as global view, to UML stat based for any partners of corporation is another main requirement which has been left already and can be regarded in the future.

REFERENCES

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

[2] 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.

[3] 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.

[4] [4] 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).

[5] Z. Li & M. Parashar,(2006), "Enabling Dynamic Composition and Coordination for Autonomic Grid Applications using the Rudder Agent Framework", *The Knowledge Engineering Review*, Vol. 00:0, pp: 1-15

[6] 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.

[7] D. Roman, J. Scicluna, C. Feier, M. Stollberg & D. Fensel, "Ontology-based Choreography and Orchestration of WSMO Services", http://www.wsmo.org/TR/d14/v0.1/, (online accessed= Jan 2011)

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

[9] D Harel and M Politi, Modeling Reactive Systems with Statecharts, McGraw Hill, 1998

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

[11] Kevin Carey, Vincent Wade," Realizing Adaptive Web Services through Automated Policy Refinement", IEEE, 2007

[12] A. Barker, D. Robertson, "Choreographing Web Services", *J. IEEE Transaction on Service Computing*, 2(2): 14, 2009, [doi:10.1109/TSC.2009.8]

[13] Erradi A., "Policy-Driven Framework for Manageable And Adaptive Service-Oriented Processes", PhD thesis, The school of Computer Science and Engineering(CSE), University of New South Wales(UNSW), June 2008.

[14] Li Z. and Parashar M., "Enabling Dynamic Composition and Coordination for Autonomic Grid Applications using the Rudder Agent Framework", *The Knowledge Engineering Review*, Vol. 00:0, 1-15, 2006

[15] Arkin, A., Askary, S., Fordin, S., Jekeli, W., Kawaguchi, K., Orchard, D., Pogliani, S., Riemer, K., Struble, S., Takacsi-Nagy, P., Trickovic, I., Zimek, S., "Web Service Choreography Interface (WSCI) 1.0", Available from "href=" http://www.w3.org/TR/wsci/".

[16] Roman, D., Scicluna, J., Feier, C., (eds.) Stollberg, M and Fensel, D.: D14v0.1. "Ontology-based Choreography and Orchestration of WSMO Services.", Available from http://www.wsmo.org/TR/d14/v0.1/

[17] Thomas Cottenier, Tzilla Elrad, Executable Choreography Processes with Aspect-Sensitive Services, Computer Science Department, Illinois Institute of Technology, 3300 S. Federal Street 60616 Chicago, Illinois, USA {cotttho, elrad}@iit.edu

[18] Svirskas A., Wilson M., and Roberts B.," Adaptive Support of Inter-Domain Collaborative Protocols using Web Services and Software Agents".

[19] Martin, D., et al.,(Nov 2004), OWL-S: Semantic Markup for Web Services. DAML White Paper Release.

[20] A. Bandara, E. Lupu, J. Moffett & A. Russo, (2004), A goal-based approach to policy refinement, Policies for Distributed Systems and Networks, 2004. POLICY 2004. Proceedings. Fifth IEEE International Workshop on 7-9 June 2004 Page(s):229 - 239.

[21] A. Guerrero, V.A Villagr´a, J.E.L de Vergara, A. S´anchez-Maci´an, & J. Berrocal, (2006), Ontology based Policy Refinement Using SWRL Rules for Management Information Definitions in OWL. In: Proc. 17th IFIP/IEEE InternationalWorkshop on Distributed Systems, Operations and Management (DSOM), Dublin, Ireland 227–232.

[22] C. Pahl,"Dynamic Adaptive Service Architecture .Towards Coordinated Service Composition", Lecture Notes in Computer Science, Software Architecture. M. Babar I.Gorton, Springer Berlin / Heidelberg. 6285:472-475, 2010, [doi: 10.1007/978-3-642-15114-9_43]

Authors

F. Mardukhi received his B.Sc degree in Computer Engineering from Sharif University of Technology, Iran in 1996 and Master of Sofware Engineering from University of Isfahan, Iran in 2002. Currently he is pursuing his Ph.D degree in the Engineering Faculty og Engineering in Isfahan University. His interests include Web Service technology, Coordination problem, and adaptive software systems. He is working on dyamic and adaptive choreography models for Web services in B2B coporation.

Dr. N. Nematbakhsh received his B.Sc degree of Science in Mathamatics from Isfahan University, Iran in 1973 and Master of Science in Computer Science in 1978 from Worcester Polytechnic Institute, USA. He received the Ph.D in Computer Engineering from University of Bradford, England in 1989. He is working as Assistant Professor in the Department of Computer Engineering, Isfahan University. His expericed areas include Software Engineering Methods, Service Oriented Computig and Software Realibility.

Dr. K. Zamanifar received his B.Sc and M.Sc degree in Electrical and Electronic Engeering from University of Tehran, Iran (1976-1985). He also received the Ph.D in Computer Science from School of Computer Studies, University of Leeds, England in 1996. He is working as Associative Professor in the Department of Computer Engineering, Isfahan University. He is member of Management Committee of Computer Society of Iran and member of Iranian Association of Electrical and Electronic Enigeering. He activated at the most conferences as member of Executive Committee. His research interests are Parallel and Distributed Systems, Distributed Operating System, Concurrent Systems and Computer Supported Cooperative Work.