Contract-Performing Circumstance-Driven Self-Adaptation and Self-Evolution for Service Cooperation

Ji Gao [1,2], Hexin Lv [1]

[1] College of Information Science & Technology
Zhejiang Shureng University, Hangzhou, China, 310015

[2] College of Computer Science & technology
Zhejiang University, Hangzhou, China, 310027

gaoji1@zju.edu.cn hexin10241024@sina.com

***Abstract*—Service cooperation-based Virtual Organizations (VOs) have become the mainstream approach for developing application software systems in Internet computing environments. However, the large-scale deployment of VOs encounters serious difficulty due to the non-autonomy for their organization and maintenance. This paper focuses on VO self-maintenance and proposes the framework for achieving the contract-performing circumstance-driven self-adaptation and self-evolution of service cooperation, in order to maintain effectively the capability for a VO to achieve its objectives in two stages: self-adaptation and self-evolution.**

Keywords*—*contract-performing; circumstance-driven; self-adaptation; self-evolution; virtual organization

I. INTRODUCTION

Constructing Virtual Organizations (VOs) by creating service cooperation (i.e., service-oriented cooperation) has become the mainstream approach for reforming the development of application software systems in Internet computing environments due to the development of Service-Oriented Computing (SOC) [1] and Service-Oriented Architecture (SOA) [2]. However, the current techniques for service cooperation are confronted with severe limitation: service cooperation is non-autonomic, making it unable to agilely adapt to the dynamically changing and unpredictable Internet cooperation environments. The leading cause is the inherent non-controllability of business services across different management domains (i.e., the consumer of a service can't control the process of the service provision). It is the non-controllability that brings on the so-called "trust" crisis that the success and benefit of cooperation cannot be ensured, and therefore makes service cooperation have to depend on a great deal of manual intervention.

Evidently, without the self-organization and self-maintenance of service cooperation, it is difficult to realize the large-scale deployment of VOs. Therefore, we have developed a series of research work for overcoming "trust" crisis and achieving autonomic service cooperation in the support of the National Science Foundation and the National High-Technology research and Development Program (863) of China. We have established a model oriented to multiagent systems, called IGTASC (Institution-Governed Trusted and Autonomic Service Cooperation) [3], to overcome "trust" crisis first. Then, based on IGTASC, we have developed two frameworks to support the self-organization of VOs [4] and the self-maintenance of VOs respectively.

This paper focuses on the framework for achieving the self-maintenance of service cooperation, called CCAE (Contract-performing Circumstance-driven self-Adaptation and self-Evolution for service cooperation). The next section introduces the relative work and our countermeasure, including the foundation created by IGTASC. Then, Section Ⅲ specifies the proposed framework CCAE in general. Section Ⅳ, Ⅴ, and Ⅵ describe main constituents of CCAE: contract-performing circumstance model, Joint Contract-Conforming Mechanism, and VO Self-Adaptation and Self-Evolution Mechanism respectively. After the implementation and application analysis in Section Ⅶ, the conclusions and future work (in Section Ⅷ) are provided.

II. RELATED WORK AND OUR COUNTERMEASURE

How to achieve the self-adaptation and self-evolution in abnormal situations is a difficult problem, worrying MAS (Multi-Agent System) researches for a long time [5][6].

2.1 Related Work

The current research for this problem is focused on the large-scale MASes composed of simple homogeneous agents, such as computing intelligence (evolution computing [7], artificial immunity systems [8], adaptive learning [9], etc.) and swarm intelligence [10]. However, the same research for small-scale MASes dynamically composed of self-interested, often much more complicated, heterogeneous agents (denoted by d-si-h-MASes hereafter) is much less and no systematic research results with practical value have been reported though such MASes are much more valuable and have the potential for large-scale deployment (see Section 2.2).

The main cause is that the methodologies of statistics, randomization, and optimization suiting computing intelligence and collective intelligence cannot be used in d-si-h-MASes, and again, there is no enough motivation and requirement for driving the researches adapting to d-si-h-MASes due to two hindrances. One is the inherent non-controllability mentioned above while the other is that the

MAS technology itself is disjoined with real-life application software systems [5]. Although there have been some self-healing research work (which belongs to self-evolution research category) for statically deployed small-scale MASes [11][12], the research results cannot adapt to open and dynamically configured d-si-h-MASes.

2.2 Our Countermeasure

The model of IGTASC mentioned above and its infrastructure can be used to conquer the two hindrances, and thereby create a substantial basis for researching the self-organization, self-adaptation, and self-evolution of service cooperation.

IGTASC proposes a three-level Virtual Society as the environment where VOs live and work (Figure 1): Agent Community, TAVOs (Trusted and Autonomic VOs), and Rational Agents, and depends on three technologies to make service cooperation both trusted and autonomic: Institution-Governed cooperation, Policy-Driven self-management, and Cooperation Facilitation management [3]. Also, reforming MAS technology by adopting the "service-oriented" concept removes the "disjoined" hindrance.

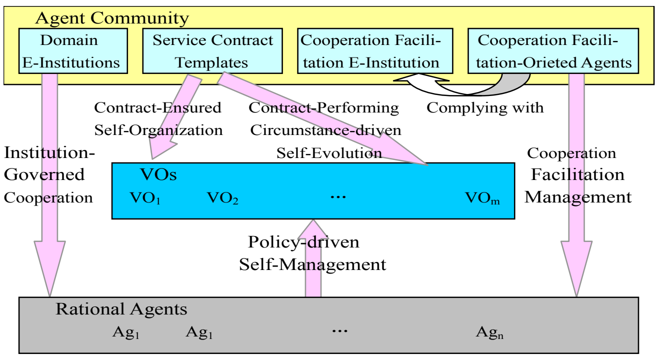
Agent Community
Domain E-Institutions
Service Contract Templates
Cooperation Facilitation E-Institution
Cooperation Facilitation-Orieted Agents
Contract-Ensured Self-Organization
Contract-Performing Circumstance-driven Self-Evolution
Complying with
Institution-Governed Cooperation
Cooperation Facilitation Management
VOs
VO1
VO2
…
VOm
Policy-driven Self-Management
Rational Agents
Ag1
Ag1
…
Agn

Figure 1 Three closely coupled mechanisms constituting IGTASC

IGTASC restricts the organizational form of a VO to the most familiar and widely-used cooperation form in human society: an alliance based on service providing-requiring relations, which is sponsored and created by some physical organization to satisfy a business requirement dynamically occurring (such as making new products, solving complex problems, searching for knowledge, purchasing merchandise, etc.). Such an alliance often concerns multiple binary collaborations which are managed by the sponsor centrally, but there are no interactions between other members (these interactions can be removed by partitioning business activities reasonably and arranging the appropriate messages sent by the alliance manager). Of course, every member of a VO should set up an agent as its broker for providing business services, and this makes the VO a typical d-si-h-MAS.

Because VOs are organized dynamically on user requirements (i.e., the newest objectives and tasks), and such VOs are of short life: the life-period of a VO ends once relevant user requirements are satisfied, this paper does not consider the change of user requirements in a life-period, and only focuses on responding the abnormal change of cooperation circumstances.

The sponsor (and manager) of a VO should create and sign a providing-requiring contract with the provider of each outer service. Since these contracts specify, by contract-performing norms, the detail of bi-cooperation activities, the run of the VO becomes the interaction process for its members to complete cooperation according to contracts.

The social facilitation e-institution for cooperation facilitation management formulates not only the cooperation facilitation-oriented services, but also macro-level cooperation behavior norms, such as the obligations for service providing and requiring parties to comply with micro-level contract-performing norms and report the post-states for executing those norms. It is the execution of those macro-level norms that supports the in-time creation of contract-performing circumstances, which constitutes the foundation for achieving VO self-adaptation and self-evolution.

Based on IGTASC and the above countermeasure, we have proposed the framework of CCAE. By developing technologies of contract-performing circumstance model, joint contract-conforming mechanism, abnormal circumstance-driven VO maintenance, and 3-phase control cycle for transacting abnormal circumstances, CCAE can maintain the capability for a VO to achieve its established objectives effectively.

III. SELF-ADAPTATION AND SELF-EVOLUTION FRAMEWORK CCAE

CCAE supports the maintenance of run-time VOs (which are in d-si-h-MAS form) in two stages: self-adaptation and self-evolution. The former does not change the constituents of a VO, including its members and business process, while the latter requires replacing some members or even the business process. However, both stages depend on the mechanism of joint contract conformity driven by contract-performing circumstances.

The work model of CCAE is illustrated in Figure 2. It consists of joint contract-conforming mechanism, VO self-adaptation and self-evolution mechanism, contract-performing circumstance model, contract-performing circumstances, and service contract set. The joint contract-conforming mechanism manages contract-performing processes and monitors contract-performing circumstances. The management function enables the provider and consumer of a business service to execute in turn protocol entries in a service contract (say *sc1*), and creates in time, according to contract-performing circumstance model, the contract-performing circumstance (say CPC^{sc1}) to make both parties in service cooperation have a whole view of cooperation states. It is the whole view that drives the alternate and compact execution of contract entries and lets the monitoring function discovery in time the occurrence of contract violation events.

Once receiving a contract violation event, the VO self-adaptation and self-evolution mechanism activates the model ACVOM (Abnormal Circumstance-driven VO self-Maintenance), which drives a 3-phase control cycle to transact the abnormal circumstance indicated by the event. The VO self-Maintenance tries the self-adaptation first, and, if it fails, then tries the self-evolution.

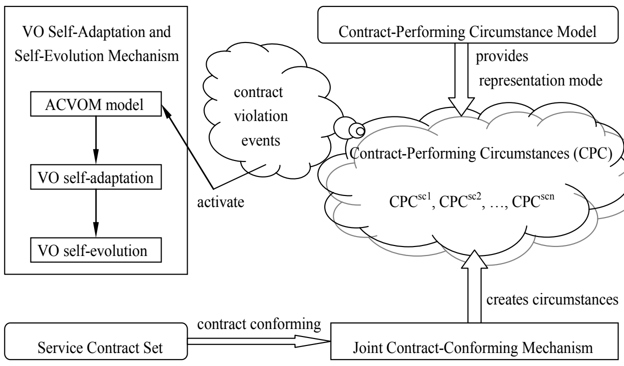
VO Self-Adaptation and Self-Evolution Mechanism
ACVOM model
VO self-adaptation
VO self-evolution
contract violation events
activate
Contract-Performing Circumstance Model
provides representation mode
Contract-Performing Circumstances (CPC)
CPC^sc1, CPC^sc2, …, CPC^scn
creates circumstances
Service Contract Set
contract conforming
Joint Contract-Conforming Mechanism

Figure 2 The work model of CCAE

IV. CONTRACT-PERFORMING CIRCUMSTANCE MODEL

This representation model describes the performing circumstances of service contracts. A typical Service Contract for business service *bs*, denoted by SC^{bs}, can be defined as a 3-tuple:

SC^{bs} = (BI, QoSG, CPP)

• BI: the Basic Information of service cooperation, which is used to specify the identity of both parties, the business transaction roles enacted by both parties, period of validity for this contract, service content (e.g., the operations or product items, price, number, deadline), payment mode, etc.

• QoSG: the QoS (Quality of Service) Guarantee, which defines quality parameters and metrics, and stipulates service level objectives (SLOs) based on those definition.

• CPP: the Contract Performing Protocol, which is designed as a partial-order set composed of protocol entries represented as contract-performing norms and uses BI and QoSG as the content referenced when executing those norms.

Definition 1 (contract-performing norm, *cpn*): define, with extended Deontic Logic [13], $cpn = OB_a^{sc}\ (\rho \le \delta \mid \sigma) \mid FB_a^{sc}\ (\rho \le \delta \mid \sigma) \mid PB_a^{sc}\ (\rho \le \delta \mid \sigma)$, indicating respectively that, when σ holds true, party *a* (the agent signing SC^{bs}) is obligated to, forbidden to, or authorized to make ρ true before deadline δ (here ρ, δ, and σ are all the propositions describing contract-performing circumstances). Note, σ and ρ is often as the pre-condition and post-condition for executing *cpn* respectively.

Definition 2 (post-state of an executed *cpn*, *ps*): define *ps* = (cpn-number, status-type, status-description), where cpn-number indicates the serial number of *cpn* in CPP of SC^{bs}, status-type indicates the type of *ps* (success, fail, or exception), and status-description gives the description of *ps*.

Next, an example of *cpn* is given, which comes from a supposed application of data mining.

```
(eio:Norm                        //A simplified norm for ensuring operator quality
    NormNo: 21;                   //Norm 21 in contract performing protocol
    Performer:“RespondingRole” ; //The norm should be executed by provider
    Trigger: (@eio:OperationCall   Operator:“Mining”   CallTime:?x);
                //Triggered by an operator invacation event “(@eio:OperationCall
                 Operator:“Mining ” CallTime <...>)”
    Deadline: (@eio:dateTimePeriod   BeginTime:?x   Period:?nego_03);
      //The deadline completing the norm execution is ?nego_03 which begins
      //from ?x. Herer, the value of ?x come from unification examination
      //when this norm is triggered , and the value of ?nego_03 depends on the
      //nigotiation between providing and requring parties.
    Postcondition:(@eio:SLOSatisfactionStatus   SLOName:“SLOOfMining”
      Operator:“Mining”   Status:“True”);    )
```

This *cpn* specifies: when the operator “Mining” provided by business service “DataMining” is invoked, an event indicating the invocation should occur and activate the *cpn*; and, as the post-condition (i.e., *cpn*.ρ), the service level objective (SLO), named as “SLOOfMining”, should be satisfied (i.e., the SLOSatisfactionStatus is “True”) after *cpn* is executed. Thus, if this *cpn* is executed successfully, *cpn*.*ps*.status-description (the status-description in *ps* of the *cpn*) must be of the same pattern as *cpn*.ρ in order to match with *cpn*.ρ. That is, the status-description should be: (@eio:SLOSatisfactionStatus SLOName:“SLOOfMining” Operator:“Mining” Status:“True”).

Based on the two definitions above, the Contract-Performing Circumstance for SC^{bs}, denoted by CPC^{sc}, is described with the sequence of *ps*es:

$$CPC^{sc} = \{ps_1,\ ps_2,\ \ldots,\ ps_m\},\ ps_i = \text{post-state}\ (Æcpn_j),\ cpn_j\ (\in \text{CPP of } SC^{bs}),$$

where the relevant *cpn*s are executed on the order specified by CPP of SC^{bs}, ith post-state is denoted by ps_i, and $Æcpn_j$ indicates jth norm in CPP is executed.

CPP divides *cpn*s into two types: backbone and compensation. While backbone *cpn*s indicate the main activities that must be executed for achieving the objective stipulated when the VO is sponsored, compensation *cpn*s are only used to recover the normal execution of contracts from abnormal *ps*es of executed *cpn*s.

Definition 3 (abnormal *ps* of a executed *cpn*, *aps*): a *ps* is the *aps* iff *cpn*.*ps*.status-description does not match with *cpn*.ρ.

Definition 4 (activation event for a *cpn*, *ae*): define *ae* as the part of *ps* of a executed *cpn*: *ae* = (cpn-number, status-type) | status-description, where status-type = ‘success’ | ‘fail’ | ‘exception’ and status-description = ‘(@<prefix>: <concept-name> {<parameter-value>}*)’. (Note, @ indicates the instance of a concept defined in the domain ontology denoted by <prefix>.)

It is *ae*s that drive the ordered execution of service contracts (i.e., *cpn*s included in them). While an *ae* coming from the *ps* of a successfully executed *cpn* activates the backbone *cpn* executed next, an *ae* coming from the *aps* activates one or more compensation *cpn*s. Also, the abnormal change of contract-performing circumstances indicated by *aps*es drives the self-adaptation and self-evolution of a VO.

V. Joint Contract-Conforming Mechanism

This mechanism is driven by contract-performing circumstances. The two macro-level cooperation behavior norms of obligation formulated in the social facilitation e-institution become the basis for implementing this mechanism: both service providing and requiring parties must comply with *cpn*s and report *ps*es of executed *cpn*s to each other. Also, the cooperation-facilitating service "ContractExecutionReport" for performing "reporting" obligation should be defined in this e-institution and configured to both parties.

The Joint Contract-Conforming Mechanism (JCCM) is represented as a multi-tuple:

JCCM = {vm-set, contract-set, cpn-set, self-executing, self-examining, inter-reporting, inter-examining}, where

• vm-set: the set of members in a VO;

• contract-set: the set of service contracts in the VO;

• cpn-set: the union of *cpn* sets; cpn-set = cpn-set^{sc1} $\cup$ cpn-set^{sc2}, …, $\cup$ cpn-set^{scn}, where cpn-set^{sci} indicates the set of *cpn*s formulated in CPP of contract sc_i ($\in$ contract-set);

• self-executing: vm-set × contract-set $\nrightarrow$ $\mathbb{P}$cpn-set; here, every *vm* ($\in$ vm-set), according to the service contract *sc* ($\in$ contract-set) signed by it, executes the *cpn*s relevant to its obligations and rights; (hereafter, $\mathbb{P}$ denotes power set and $\nrightarrow$ denotes partial function.)

• self-examining: vm-set × contract-set $\nrightarrow$ $\mathbb{P}$cpn-set, here, every *vm* ($\in$ vm-set), according to the *sc*, examines the *ps*es of *cpn*s executed by itself, and self-examining (*vm*, *sc*) = self-executing (*vm*, *sc*);

• inter-reporting: vm-set × contract-set $\nrightarrow$ $\mathbb{P}$cpn-set, here, every *vm* ($\in$ vm-set), according to the *sc*, reports the *ps*es of *cpn*s executed by itself to the opposing party of cooperation, and inter-reporting (*vm*, *sc*) = self-executing (*vm*, *sc*);

• inter-examining: vm-set × contract-set $\nrightarrow$ $\mathbb{P}$cpn-set, here, every *vm* ($\in$ vm-set), according to the *sc*, examines the *ps*es of *cpn*s executed by the opposing party of cooperation, and inter-examining (*vm*, *sc*) $\cup$ self-examining (*vm*, *sc*) = cpn-set^{sc}, inter-examining (*vm*, *sc*) $\cap$ self-examining (*vm*, *sc*) = $\varnothing$.

JCCM is installed into every business operation-oriented agent to implement two main functions: managing contract-performing processes and monitoring contract-performing circumstances. CPP of each contract *sc* ($\in$ contract-set) becomes the basis for agent to manage the execution of *sc*. The contract-performing circumstance for *sc* changes continually along with the execution of *cpn*s ($\subset$ cpn-set^{sc}). If cpn_i ($\in$ cpn-set^{sc}) needs to be executed by the agent itself, this agent should invoke the local operator relevant to cpn_i before deadline δ, create *ps* according to the operation result, and report *ps* to the opposing party of cooperation.

Monitoring contract-performing circumstances includes the self-examining and inter-examining for *ps*es of executed *cpn*s. The examinations are focused on whether or not a *ps* can be generated before the deadline and satisfy $cpn.\rho$.

It is the mutual reporting of *ps*es that enables both parties of each contract *sc* ($\in$ contract-set) to observe and examine the whole contract-performing circumstance in time and thus to drive the alternate and compact execution of *cpn*s.

Reporting actively *aps*es (abnormal *ps*es) occurring in one's own side can facilitate the discovery and transaction of abnormity. Because the compensation *cpn*s can be executed as soon as possible, this enhances the reliability and robustness of service cooperation.

VI. VO Self-Adaptation and Self-Evolution Mechanism

This mechanism uses the model of ACVOM (Abnormal Circumstance-driven VO self-Maintenance) as the basis for implementing VO self-maintenance, and adopts a 3-phase control cycle as the framework.

6.1 Model ACVOM

ACVOM enables the VO sponsor, depending on its management policies, to centrally manage and control abnormal circumstance-driven VO maintenance. The maintenance activities are flexible and scalable: from the small ones such as modifying a service contract to the large such as replacing a service provider or even a business process.

The model of ACVOM is defined as a multi-tuple:

ACVOM = (CPC, CMP, cv-events, cm-principles, cm-plans, cm-actions, analysing, planning, executing), where

• CPC: the set of service Contract-Performing Circumstances; here, CPC = {CPC^{sc1}, CPC^{sc2}, …, CPC^{scn}}, and CPC^{sci} indicates the circumstance of sc_i (ith service contract);

• CMP: the set of management Policies which the VO manager (sponsor) possesses for supporting Cooperation Maintenance; here, CMP = an-policies $\cup$ pl-policies $\cup$ ex-policies, where an-policies, pl-policies, and ex-policies indicate the subset of analysis, planning, and execution policies respectively;

• cv-events: the set of contract violation events (*ae*s from *aps*es) reflecting CPC abnormity;

• cm-principles: the set of cooperation modification principles which are the result of contract violation analysis;

• cm-plans: the set of cooperation modification plans which are the planning result;

• cm-actions: the set of cooperation modification actions specified by cooperation modification plans;

• analysing: an-policies × cv-events × CPC $\nrightarrow$ cm-principles; here, the analysis activities denoted by this function adopt a domain-specific circumstance abnormity analysis policy *anp* (∈an-policies), activated by *cve* (∈cv-events), to analyse CPC^{sc} (∈CPC) creating *cve* and propose an analysis result (a cooperation modification principle) *cmpr* (∈cm-principles);

• planning: pl-policies × cm-principles ⇸ cm-plans; here, the planning activities denoted by this function adopt a domain-specific cooperation modification planning policy *plp* (∈pl-policies), activated by *cmpr*, to drive planning and generate a cooperation modification plan *cmpl* (∈cm-plans);

• executing: ex-policies × cm-plans ⇸ ℙcm-actions; here, execution activities denoted by this function adopt a domain-specific plan execution policy *exp* (∈ex-policies), activated by *cmpl*, to start cooperation modification actions (specified by *cmpl*) *cmas* (⊂cm-actions).

The above mapping functions of analysing, planning, and executing constitute jointly the 3-phase control cycle for transacting abnormal circumstances, and the activities in those phases are driven by CMP (Figure 3). Next, we only explain the transaction made by the VO sponsor. In fact, the transaction made by other VO members is similar and simpler.

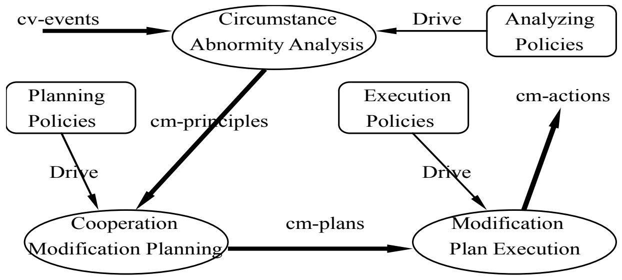

Figure 3 The policy-driven 3-phase control cycle for abnormal circumstance-driven VO maintenance

1) Circumstance abnormity analysis

The analysis work is driven by analysis policies. Once a *cve* (∈cv-events) occurs, the relevant analysis policy is triggered, which is used to analyse the cause, property, and effect of *cve* and select one from multiple activated compensation *cpn*s. It is the multiple compensation *cpn*s that enable the benefit-losing party to have multiple choices for maintaining contract execution. The selection depends on the integrated analysis of multiple factors, such as the respective results expected when executing these *cpn*s, the work situation of current service contract-performing protocol, the whole execution situation of service cooperation in the VO, the business objectives of the VO sponsor, application domain knowledge, etc.

2) Cooperation modification planning

This phase aims at using planning policies to generate relevant cooperation modification plans in order to eliminate the minus affect of abnormity or to decrease the affect to the least degree. The extent of cooperation modification depends on the effect of execution of compensation *cpn*s, whether there are spare service providers or not, the structure of current business process, etc. and therefore can be partitioned into the next types (from the small to the large):

• replace the provider of a service (because the contract for this service is violated);

• defer in turn the time for providing following services in the current business process;

• replace the current business process with new one, including to cancel the contracts for all services not occurring in the new and to create the contracts for new services occurring in the new one;

• dismiss the VO and cancel the contracts for all services in the current business process.

Based on planning policies, extent-different modification plans can be created, and hence make the adaptation and evolution of cooperation display the better flexibility and scalability.

3) Modification plan execution

This phase aims at applying execution policies to detail modification plans and execute modification actions. For example, a modification plan only indicates to replace the provider of a service while the activities for determining the new provider of this service, making negotiation, signing the contract with this new provider, etc. should be driven by execution policies.

Evidently, it is the proper transaction of abnormal circumstances that supports the self-adaptation and self-evolution effectively.

6.2 VO self-adaptation and self-evolution

We view both self-adaptation and self-evolution of a VO as the means to maintain the capability for the VO to achieve its own objectives (Figure 4). The main difference is the extent of VO change: the former does not change VO constituents while the latter requires replacing some VO members or even the business process. In fact, the former can be viewed as the first stage for responding abnormality, and only when it does not bear fruit, the VO maintenance enters into the second stage: self-evolution.

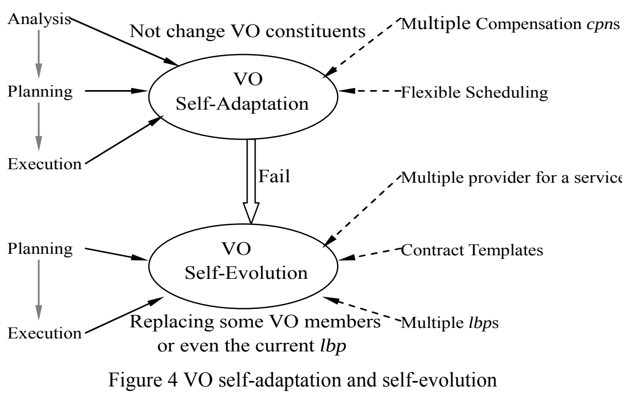
Analysis
Not change VO constituents
Multiple Compensation cpns
Planning
VO Self-Adaptation
Flexible Scheduling
Execution
Fail
Multiple provider for a servic
Planning
VO Self-Evolution
Contract Templates
Execution
Replacing some VO members or even the current lbp
Multiple lbps

Figure 4 VO self-adaptation and self-evolution

1) VO self-adaptation

This stage depends on two key technologies: the flexible scheduling of the local business processes (denoted by *lbp*s hereafter) and the configuration of compensation *cpn*s in CPP (contract-performing protocol) of SC^{bs}. Here, *lbp*s are formulated as the activity-scheduling plans for an agent to achieve local business objectives.

Although there are a variety of domain-specific abnormal circumstances, the mode for transacting them is the same: create a requirement for selecting suitable one from the compensation *cpn*s and use the requirement as an event to activate the policy starting a 3-phase control cycle.

Next, we explain, by the supposed application of data mining, the policy-driven self-adaptation based on a flexible scheduling method. Figure 5 illustrates two *lbp*s for achieving a data mining task. Each *lbp* displays only the activities, indicated by circles, performed by invoking the outer business services. Suppose the execution of activity 2 (in *lbp* 1 of Figure 5) generates an abnormal circumstance because the outer service *bs* for performing this activity is unavailable before the deadline. In order to transact the violation, two compensation *cpn*s have been configured in CPP of SC^{bs}: number 01 and 02, which can be activated at the same time. The former informs *bs* provider to make *bs* available before a later deadline while the latter cancels the contract for *bs*.

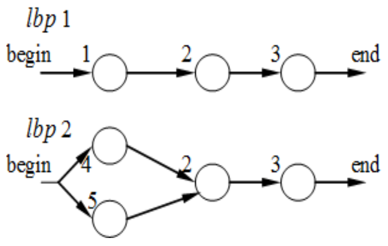

Figure 5 The *lbps* for a data-mining task

The policy for selecting from the two *cpns* is formulated as following:

```
Policy                //The policy for selecting from cpn 01 and 02
      Name: "CircumstanceAbnormityAnalysisForServiceAvailability";
      PolicyType: "Obligation" ;
      Processing: (ruleGroup    ATFSA); //The processing work after this
      //policy are activated: make a choice by executing rulegroup ATFSA
      Target: (@Service "Monitoring"    "GS");
      Trigger: (@ContractNormConflictOccur ActivatedNormNo1:?x
          ActivatedNormNo2:?y) ($=    ?x    01) ($=    ?y    02);      //The
          //trigger condition: cpn 01 and 02 are activated at the same time
End    Policy
      ruleGroup    ATFSA
          mode:           p;      // Denotes the production reasoning
          select:         first;  //Execute the first activated rule
          ruleList:
(→    ($ServiceAvailablePeriodAnalysis)
       ($BBStore    ($CreateConceptInstance    "SelectedNorm"    01)));
(→    ($ExtendedAvailablePeriodAnalysis)
       ($BBStore    ($CreateConceptInstance    "SelectedNorm"    01)));
($SendMessage    "Monitor"    ($CreateConceptInstance
      "PlanningDeferringOfFollowingServices"    01    02));
      End    ATFSA
```

This policy is activated by the *cpn* selection requirement, and then it starts, by executing rule-group ATFSA, the analysis phase of a 3-phase control cycle to analyse the availability of *cpn* 01. The analysis work includes:

• Check whether a later deadline for *bs* can be assigned by executing the Boolean function as the condition part of rule 1 "ServiceAvailablePeriodAnalysis". If it can, the function returns 'true', and further results in the execution of *cpn* 01.

• Or else, by executing the Boolean function as the condition part of rule 2 "ExtendedAvailablePeriodAnalysis", calculate whether a extended later deadline for *bs* can be assigned. If it can, the function returns 'true' and drives the execution of *cpn* 01.

• Or else, send an internal message "(@PlanningDeferring OfFollowingServices 01 02)" to the agent (VO manager) itself by executing rule 3 (the unconditional rule).

This message activates the policy for driving the planning of service deferring. Then this policy starts the planning phase to determine which in the following services *bs*es to be deferred (e.g., *bs* for performing activity 3 in Figure 5) and the deterred time for them. If the deferring plan is generated, a policy for driving the execution of this plan is activated to start the deferring negotiation with the providers of *bs*es (the details are omitted).

In summary, it is the agent management policies that drive effectively the self-adaptation of service cooperation and VOs. Especially, the flexible scheduling of *lbp*s and the configuration of compensation *cpn*s not only enhance the possibility for removing abnormal circumstances but also facilitate the survival of current *lbp*s in abnormal circumstances. Therefore, this enables VOs to agilely respond abnormity due to not changing VO members and *lbp*s..

2) VO self-evolution

If the contract for *bs* must be cancelled (by executing *cpn* 02), the effort for maintaining VO capability enters into the second stage: self-evolution. Because the circumstance abnormity analysis has already been done in self-adaptation stage, only the latter two phases of a control cycle: Cooperation modification planning and modification plan execution, need to be performed.

The work for VO evolution planning is as following:

• Determine whether a new provider of *bs* can start *bs* in the available or extended period of *bs*. If can, send an internal message "(@ExecutingReplacementPlan "ReplacingServiceProvider" <*bs*> <new provider>)" to the agent (VO manager) itself in order to drive the replacement of *bs* provider.

• Or else, evaluate a later deadline and drive the replacement of *bs* provider.

• If the replacement fails, send an internal message "(@Executing ReplacementPlan "RplacingBusiness Process")" to the agent itself in order to drive the replacement of the current *lbp* with a new *lbp*.

• Or else (a new provider is found), send a message to activate the policy for driving the planning of deferring (similar to the planning phase of VO self-adaptation).

If a new provider is found and the deferring plan is generated, the policy for plan execution phase is activated.

The plan execution activities include to negotiate the service provision with new provider and to negotiate the service deferring with the providers of following services. Once all of these negotiations are completed successfully, *lbp* 1 of Figure 5 can be resumed.

When the above planning or plan execution fails, replacing this *lbp* is necessary if there are spare *lbp*s. Therefore, the planning and execution work for replacing the current *lbp* with a new *lbp* should be driven by relevant policies, including: find an available *lbp* (e.g., *lbp* 2 in Figure 5), cancel the contracts for *bs* relevant to activity 1, negotiate, create and sign the contracts for business services relevant to activity 4 and 5, and finally perform *lbp* 2.

If no new *lbp* is available, or, for any one from *bs*es relevant to activity 4 and 5, no applicable provider is found, the VO must be disbanded, and all the contracts for other *bs*es in *lbp* 1 must also be cancelled. Of course some compensation work must be done by executing compensation *cpn*s formulated in those contracts.

VII. Implementation and Application Analysis

We have already created the self-adaptation and self-evolution framework CCAE by intensifying the agent platform included in the infrastructure for IGTASC.

First, a monitor module is nested into the platform to perform CPP of SC^{bs} depending on the contract-performing circumstance model and joint contract-conforming mechanism and to implement the activities for monitoring abnormal circumstances and the ones in the 3-phase control cycle. Because the platform has provided the policy engine, it is easy to make the work of monitor module become policy-driven as long as configuring a set of domain-specific policies and the operators and functions driven by these policies.

Second, the policy-driven self-management enables business operation-oriented agents to rationally conform to macro-level cooperation behavior norms formulated in the social facilitation e-institution, especially the obligations for complying with micro-level *cpn*s in service contracts and reporting the post-states of executed *cpn*s. Besides, the uniform facilitation service "ContractExecutionReport" configured to those agents creates the basis for implementing the joint contract-conforming mechanism and monitoring abnormal circumstances.

Third, formulating one or more service contract templates for each business service simplifies the creation, negotiation, and conformation of contracts. It is those templates that enable application domains to formulate statically parameterized *cpn*s, and thereby enable business operation-oriented agents to possess, by statically configuring operators specified by *cpn*s and management policies, the capability for executing *cpn*s, achieve flexible scheduling of *lbp*s, and implement policy-driven self-adaptation and self-evolution.

We have established several experimental service cooperation-based VOs, such as small meeting arrangement, knowledge provision, data mining, multi-part device cooperation production, and multi-department crisis cooperation transaction, and used those VOs to test and validate the self-adaptation and self-evolution of service cooperation and VOs. The experimental results indicate that CCAE can support the run and maintenance of VOs effectively in very different application domains.

Next, we adopt the supposed application of data mining (see figure 5) to make an analysis. The current VO dynamically created wants to complete a data-mining task by invoking the three sequential business services provided by different partners (see *lbp* 1 in Figure 5). Because the VO sponsor and these partners all are the rational agents supported by CCAE and registering in the agent community, this VO can implement the joint contract conformation, transact abnormity, and maintain VO capability effectively.

In order to validate the compact and fluent execution of the data-mining task in normal circumstances, we let the three services for performing the three activities in *lbp* 1 to be assigned equal-length available periods first, and then make those periods overlap with each other. The process for executing this task indicates that the three services can be provided one after another with no time interval between them.

In another test, the service 1 for performing activity 1 is not available before the deadline. This contract violation event activates compensation *cpns*: 01 and 02, and further activates policy "CircumstanceAbnormityAnalysisForService Availability" (see Section 6.2). Because the provider can make service 1 available in a later deadline (which does not affect the provision of sequential services), this policy drives the execution of *cpn* 01: specify the new deadline, and inform the provider.

Again, if the provider can not make service 1 available in the later deadline, the message "(@PlanningDeferringOf-FollowingServices 01 02)" is sent to activate the policy for driving the planning of service deferring. The produced deferring plan indicates that the provision of service 1 and 2 should be deferred in order to enable service 1 to be provided before the later deadline. And then the policy for driving the execution of deferring plan is activated to drive the deferring negotiation with the providers of service 1 and 2. Due to the success of negotiation, this policy drives the execution of *cpn* 01 before the later deadline.

Evidently, It is the flexible scheduling of *lbp*s and the configuration of multiple compensation *cpn*s that enable the VO to achieve self-adaption without changing its constituents.

We also validate the self-evolution of this VO by testing two instances. One is the fail of deferring negotiation while the other is the QoS violation in providing service 1. Both instances bring on the cancel of the contract for service 1, and thereby this drives the process for finding new provider of service 1. We examine two situations: finding one or no new provider. The former results in the update of single VO member while the latter brings on the replacement of *lbp* 1 with *lbp* 2 (see Figure 5), and the bigger change of VO constituents: removing the provider of service 1 and increasing the providers of service 4 and 5.

Here, the basis for implementing VO self-evolution is the standards for business services and business operation-oriented roles and the coupling cooperation behavior norms formulated in e-institutions, and the formulation of multiple compensation *cpn*s while the policy-driven cooperation management enables the general agent platform to be specialized into adapting to application domain requirement by configuring domain-specific policies and policy-driven operations and functions.

In summary, the advantage of CCAE is induced as follows:

1) Realizing the self-maintenance of VOs in d-si-h-MAS form; this enables those VOs not only to be composed dynamically and on requirement depending on the model IGTASC and its infrastructure, but also to maintain their capability for achieving objectives effectively, and thereby creates the solid foundation for the large-scale deployment of VOs

2) Since the cooperation between VO members focuses, by using contracts as tie, the monitoring of cooperation circumstance on the execution states of service contract-performing protocols, this makes the discovery of cooperation exceptions and the self-maintenance for eliminating exception impact have a reliable and accurate basis.

3) Configuring to every business operation-oriented agent the uniform facilitation service "ContractExecution Report" and the obligation for reporting the post-states of executed *cpn*s enables both provider and consumer of a service to acquire in time the whole service contract-performing circumstance, and accordingly facilitates the compact execution of contract-performing protocols and the discovery of contract violation exceptions.

4) The policy-driven self-adaptation and self-evolution not only enables business operation-oriented agents to rationally conform to macro-level cooperation behavior norms formulated in the social facilitation e-institution, but also makes, by formulating domain-specific policies, the general model CCAE specialized easily into adapting to different application domains.

5) Dividing the maintenance of run-time VOs into two stages (self-adaptation and self-evolution) enables service cooperation not only to gain compact and fluent execution by self-adaptation (due to no change of the constituents of a VO), but also to adapt to, by self-evolution, the complex situation requiring replacing some members or even the business process.

VIII. Conclusion and Future Work

This paper focuses on the self-maintenance of VOs in d-si-h-MAS form, which are much more valuable and have the potential for large-scale deployment, and has created the framework CCAE to achieve the contract-performing circumstance-driven self-adaptation and self-evolution for service cooperation. CCAE can maintain effectively the capability for a VO to achieve its dynamically established objectives in two stages: self-adaptation and self-evolution, and thereby enhance the survival of VOs and current business processes for scheduling service cooperation.

The future work will be the formalization of CCAE and the development of real-life application systems based on CCAE.

Acknowledgment

We gratefully acknowledge the support of the National Science Foundation of China (Grant 60775029), the National High-Technology research and Development Program (863) of China (Grant 2007AA01Z187), the Priority Theme Emphases Project of Zhejiang Province, China (Grant 2010C11045), and the Natural Science Finds of Zhejiang Province, China (Grant Y107446).

References

[1] M. P. Papazoglou, P. Traverso, S. Dustdar, et el. Service-oriented computing: state of the art and research challenges. IEEE Computer, 40 (11): 38-45, 2007.

[2] M. Stal. Using architectural patterns and blueprints for service-oriented architecture. IEEE Software, 23(2): 54-61, 2006.

[3] J. Gao, H. Lü, H. Guo, et al. Trusted autonomic service cooperation model and application development framework. Science in China Series F: Information Sciences, 52 (9): 1550-1577, 2009.

[4] J. Gao and S. Ye. ACMFS: an abnormal circumstance-driven self-maintenance mechanism based on flexible scheduling. Proceedings of the International Conference on 3rd Information Sciences and Interaction Sciences (ICIS), 318-323, 2010.

[5] M. Pěchouček and V. Mařík. Industrial deployment of multi-agent technologies: review and selected case studies. Auton Agent Multi-Agent Syst (2008) 17:397–431, 2008.

[6] L. Paulo, V. Paul, and A. Emmanuel. Self-adaptation for robustness and cooperation in holonic multi-agent systems. In Hameurlain A, et al. (Eds.): Trans. on Large-Scale Data- & Knowl.-Cent. Syst. I, LNCS 5740, 267–288, 2009.

[7] E. Di Nitto, C. Ghezzi, A. Metzger, et al. A journey to highly dynamic, self-adaptive service-based applications. Automated Software Engineering, 15 (15): 313-341, 2008.

[8] T. Liu, L. Zhang, B. B. Shi. Adaptive immune response network model. Emerging Intelligent Computing Technology and Applications: With Aspects of Artificial Intelligence, 890-898, 2009.

[9] A. K. Qin, V. L. Huang, P. N. Suganthan. Differential evolution algorithm with strategy adaptation for global numerical optimization. IEEE Transactions on Evolutionary Computation, 13 (2): 398-417, 2009.

[10] R. Charrier, C. Bourjot, and F. Charpillet. Study of self-adaptation mechanisms in a swarm of logistic agents; Third IEEE International Conference on Self-Adaptive and Self-Organizing Systems, SASO '09, 82-91, 2009.

[11] D. Weyns and M. Georgeff. Self-adaptation using multiagent systems. IEEE Software, 27(1): 86-91, 2010.

[12] D. Weyns and T. Holvoet. An architectural strategy for self-adapting systems. International Workshop on Software Engineering for Adaptive and Self-Managing Systems, ICSE Workshops SEAMS '07, 3-12, 2007.

[13] T. Huaglory and U. Rainer. Towards autonomic computing systems. Engineering Applications of Artificial Intelligence, 17 (7): 689-699, 2004.