文章编号: 1001- 2486(2009) 01- 0080- 06

基于迭代 Bargaining 策略优化服务合成执行路径*

任开军, 宋君强, 肖 侬, 张 卫民
(国防科技大学 计算机学院, 湖南 长沙 410073)

摘 要: 服务质量(QoS) 是优化服务合成执行路径的关键研究内容, 当前绝大多数存在的方法很少注意到商业环境下服务商之间存在的隐性竞争压力可能会迫使服务商动态改变他们的 QoS 值以至于合成优化执行路径发生改变。针对此问题, 提出一个基于迭代 Bargaining 策略的约束违背纠正方法。该方法使用本地最优化策略, 在没有考虑用户 QoS 约束的情况下建立一条最优执行路径。对此路径, 全局 QoS 计算模型和全局约束违背检查模型能找出所有发生的约束违背。一个迭代的 Bargaining 策略被递归作用于关键路径执行节点, 使得更好服务提供商被选出替换原有执行节点, 从而一个优化的执行路径能被重新建立以满足用户综合的 QoS 约束需求。

关键词: 服务合成; QoS 约束违背; 迭代 Bargaining 策略
中图分类号: TP311 文献标识码: A

An Iteratively Bargaining-based Strategy for Optimizing Service Composition Execution Path

REN Kai-jun, SONG Jun-qiang, XIAO Nong, ZHANG Wei-min
(College of Computer, National Univ. of Defense Technology, Changsha 410073, China)

Abstract: Quality of service(QoS) is a critical research issue in optimizing service composition execution path. Unfortunately, the currently existing methods ignore the hidden competition pressure between service providers which can push them to dynamically change their initial QoS values in that the initial optimal execution path can be changed. To address this isssue, an iteratively bargaining-based constraint correction strategy is proposed. With this method, an initial execution path for a service composition is firstly built by using the local optimization policy. Based on this path, the global QoS aggregating and checking models are used to determine all occurred QoS constraint violations. For all violations, the proposed bargaining strategy can be recursively used to correct such violations. Finally, an optimized path can be rebuilt to meet the overall end-to-end QoS requirements expressed by a user.

Key words: service composition; QoS constraint violation; iterative bargaining strategy

在服务计算中, 如何有效协同单个服务的 QoS 从多条候选路径中选出一条优化的执行路径并满足用户综合 QoS 约束需求, 已成为服务合成 QoS 研究中的一个重要研究问题。遗憾的是, 该问题是一个 NP 计算难的问题。尽管提出了许多方法来解决这一问题[1-6], 但这些方法主要依靠服务提供商广播的初始非功能属性值, 而很少注意到他们之间存在的隐性竞争关系可能会迫使他们改变初始值。例如, 在商业环境下, 服务提供商为了追求最大化的利益, 一些初始的 QoS 值如价格与费用往往高于实际值。因此这就提供了用户同服务提供商之间可以进行讨价还价的谈判空间。特别, 如果一个用户同时发起对多个功能相似服务提供商的谈判, 并使用特殊的谈判策略让服务商意识到要想赢取同用户的交易其竞争十分激烈, 如果他们在某些关键的质量属性上不作出任何让步, 恐怕会失去同用户交易的机会。进而, 在激烈的竞争压力下, 一些服务商被迫对质量属性作出调整以尽力赢取同用户的交易。经过这样反复的谈判, 用户最终将获取更大的受益和更好的服务质量。遗憾的是, 这一普遍在商业环境中存在的现象却很少被用来考虑解决服务合成的 QoS 路径选择问题, 以至于当前的服务合成 QoS 研究方法没有正

* 收稿日期: 2008- 08- 20
基金项目: 国家自然科学基金资助项目(NSFC60573135, NSFC60736013)
作者简介: 任开军(1975—), 男, 讲师, 博士生。

确反映业务流程处理的市场规律。

1 基于本地最优化策略建立执行路径

定义 1 n 表示一个合成方案包含抽象服务的数目, 也称为合成长度。

定义 2 $ws_i(i=1, 2, \cdots, n)$ 表示长度为 n 的合成方案的一个抽象服务, 每个抽象服务 ws_i 有 l_i 个候选服务提供商;

定义 3 c_{ij} 表示抽象服务 ws_i 的第 j 个服务提供商, 其中, $j=1, 2, \cdots, l_i$。

定义 4 m 表示总的 QoS 属性个数, $q_{ijr}(r=1, 2, \cdots, m)$ 表示抽象服务 ws_i 的第 j 个服务提供商 c_{ij} 在第 r 个质量属性上的取值。

鉴于以上的定义, 对于一个抽象的服务 ws_i, 如果考虑它的所有候选服务在所有质量属性上的取值, 则导致如下一个关于 ws_i 的质量矩阵 $\boldsymbol{Q}_i(q_{ijr}, 1\leqslant j\leqslant l_i, 1\leqslant r\leqslant m)$, 其中, $\boldsymbol{Q}_i$ 矩阵的每行对应于 ws_i 的一个服务提供商 c_{ij}, 而每列对应于一个质量维。

$$\boldsymbol{Q}_i=\begin{bmatrix} q_{i11} & q_{i12} & \cdots & q_{i1m} \\ q_{i21} & q_{i22} & \cdots & q_{i2m} \\ \cdots & \cdots & \cdots & \cdots \\ q_{il_i1} & q_{il_i2} & \cdots & q_{il_im} \end{bmatrix} \tag{1}$$

对于服务 ws_i, 既然存在多个在不同质量属性有不同取值的服务提供商 c_{ij}, 而在合成方案执行时, 只有一个服务提供商 c_{ij} 被选出来执行 ws_i 的功能, 因此对于 ws_i, 需要评价和排序各个 c_{ij}, 进而可以根据排序选出合适的 c_{ij} 执行 ws_i 的功能。由于 c_{ij} 具有不同的质量属性, 且不同质量属性的度量单位也不尽相同, 如价格的度量单位可能为美元, 而时间的度量单位可能为秒。此外, 有的质量属性取值越大, 对用户越有益, 这类质量维称为正质量维; 而有的质量属性取值越大, 对用户越不利, 这类质量维称为负质量维。因此, 为成功对 ws_i 的候选服务提供商 c_{ij} 按 QoS 进行排序, 则首先应对它们不同质量属性的度量单位进行统一和标准化处理[7]。经过标准化处理后, 质量矩阵 $\boldsymbol{Q}_i$ 转变成另一标准化矩阵 $\boldsymbol{Q}'_i=(q'_{ijr}, 1\leqslant j\leqslant l_i, 1\leqslant r\leqslant m)$。

采用一个权值 $w_r(r=1, 2, \cdots, m)$ 来表示不同质量属性对用户的重要程度, 权值越大, 表示该质量属性对用户重要程度越高, 其中 $\sum_{r=1}^{m} ws_r = 1$。当不同的质量属性分配不同权值后, 对每个抽象服务 ws_i, 可以为它所有的候选服务实例 c_{ij} 计算出一个综合的排队分数, 式(2)用来计算不同 c_{ij} 的排名分数。

$$Score(c_{ij}) = \sum_{r=1}^{m} q'_{ijr} w_r \tag{2}$$

如果不考虑用户 QoS 约束需求, 显然经过排队分数计算后, 分数最高的服务提供商具有最好的综合服务质量, 进而, 对于一个合成方案的所有抽象服务 ws_i, 如果都将排队分数最高的服务提供商 c_{ij} 选出来组成一条合成执行路径, 显然, 该路径就是一条最优的合成执行路径。但该路径不一定满足用户提出的 QoS 约束需求, 因此应对这条路径进行修正, 以使它满足用户综合的 QoS 约束需求。

2 全局 QoS 计算模型与约束违背计算

2.1 合成执行路径全局 QoS 计算模型

合成执行路径全局 QoS 的计算目的是解决如何通过单个服务质量计算一条路径上各个质量维的总的服务质量。不同的质量维以及不同的合成结构其全局 QoS 计算方法都不尽相同。文献[2-3, 7-9]提供了一些具体的综合 QoS 计算方法。文献[7]讨论了如何根据历史信息和概念策略将一个循环结构转化为一个顺序结构。本文的全局 QoS 计算模型也做了简化, 假设循环合成结构可以转变为顺序合成结构。此外, 对于条件分支, 合成路径的选取主要取决于当前服务的运行结果, 因此, 在正式决定服务合

成执行路径前, 一般根据历史信息用概率做出分支预测。而对于条件分支的 QoS 计算, 可以先对每个概率分支进行独立 QoS 计算, 而后采用概率进行综合计算, 实际上, 条件分支的 QoS 计算同样可以演变为顺序结构的计算。而对于并行分支的 QoS 计算, 情况则稍有不同, 不同的质量属性可能有不同的计算方法。由于参与并行结构的所有服务都要被执行, 因此当进行价格计算时, 所有服务的费用应该被计算在内, 而如果考虑的质量属性为时间时, 则并行结构的 QoS 时间只以并行分支最大的时间计入总的合成时间。表 1 总结了各个常用质量属性在合成路径中的计算方法。其中, p 表示一条由 n 个抽象服务构成的合成方案的执行路径, $\sum_{j=1}^{l_i} x_{ij} = 1, x_{ij} \in \{0,1\}$, 保证 ws_i 的 l_i 个候选服务提供商 c_{ij} 中只有一个被选出来提供 ws_i 的功能。

表 1 全局 QoS 计算模型

Tab. 1 Global QoS computing models

Execution price	$q_{pri}^{total}(p) = \sum_{i=1}^{n}\sum_{j=1}^{l_i} q_{pri}(c_{ij})x_{ij}, \quad \sum_{j=1}^{l_i} x_{ij} = 1, \quad x_{ij} \in \{0,1\}$
Execution time	$q_{tim}^{total}(p) = \max_{sp_m \in p} \sum_{c_{ij} \in sp_m} q_{tim}(c_{ij})$
Reputation	$q_{rep}^{total}(p) = \prod_{i=1}^{n}\left[\sum_{j=1}^{l_i} q_{rep}(c_{ij})x_{ij}\right]$
Reliability	$q_{rel}^{total}(p) = \prod_{i=1}^{n}\left[\sum_{j=1}^{l_i} q_{rel}(c_{ij})x_{ij}\right]$
Availability	$q_{ava}^{total}(p) = \prod_{i=1}^{n}\left[\sum_{j=1}^{l_i} q_{ava}(c_{ij})x_{ij}\right]$
Data quality	$q_{daq}^{total}(p) = \min_{i}\left[\sum_{j=1}^{l_i} q_{daq}(c_{ij})x_{ij}\right]$
Compensation rate	$q_{cop}^{total}(p) = \sum_{i=1}^{n}\left\{\sum_{j=1}^{l_i}[q_{cop}(c_{ij})q_{pri}(c_{ij})x_{ij}]\right\} / q_{pri}^{total}(p)$
Penalty rate	$q_{pen}^{total}(p) = \sum_{i=1}^{n}\left\{\sum_{j=1}^{l_i}[q_{pen}(c_{ij})q_{pri}(c_{ij})x_{ij}]\right\} / q_{pri}^{total}(p)$

2.2 全局约束违背检查模型

使用式(3) ~ (10)进行 QoS 约束检查。方程的左边表示通过全局 QoS 计算模型算出的相应质量维的综合 QoS 值, 右边表示用户表达的质量约束值。通过这些模型检查, 能够判断在一条执行路径上究竟哪些质量维发生了与用户要求不一致的约束违背。特别地, 约束检查式(3) ~ (10)可以根据不同的应用场景和用户需求进行灵活组合和扩展。式(5) ~ (7)可以组合在一起进行费用、时间、名声的约束检查。

$$q_{pri}^{total}(p) \leqslant price_{user} \tag{3}$$

$$q_{tim}^{total}(p) \leqslant time_{user} \tag{4}$$

$$q_{rep}^{total}(p) \geqslant reputation_{user} \tag{5}$$

$$q_{cop}^{total}(p) \geqslant compensationrate_{user} \tag{6}$$

$$q_{ava}^{total}(p) \geqslant availability_{user} \tag{7}$$

$$q_{daq}^{total}(p) \geqslant data\text{-}quality_{user} \tag{8}$$

$$q_{rel}^{total}(p) \geqslant reliability_{user} \tag{9}$$

$$q_{pen}^{total}(p) \leqslant penalty\text{-}rate_{user} \tag{10}$$

3 基于迭代 Bargaining 策略的约束违背纠正算法

3.1 算法设计

图 1 给出了基于迭代 Bargaining 策略的约束违背纠正算法的关键处理步骤。步骤 1 在没有考虑用户 QoS 约束的情况下对一个抽象的服务合成方案使用本地最优化策略产生一条最优的执行路径。步骤

```
Algorithm 1: BargainingConstraintViolationCorrection
Input: abstractCompositionPlan , userQoS-requirements
QutPut: GOIEPath // Global Optimal Instance Execution Path
1  LOIEpath ← LocalOptimalExecutionPathSelection (abstractCompositionPlan) ;
   //Getting an initial execution path by local optimization policy
2  GOIEPath ← LOIEpath;
3  QoSVector [ ] ← globalQoSComputing (GOIEPath);
   //Calculating the global QoS values according to global QoS computing formulas
4  VioVector [ ] ← constraint Check (QoSVector [ ], userQos-requirements);
   //Cakcukating the occurred constraint violations according to global constrain checking model
5  FOR i=1 to VioVector [ ]. size () DO
     //For each constraint violatioin
6      negoSpaceVector [i] ← globalQoSvalue-userconstraintValue;
       //Calculating the negotiation space for the corresponding constraint violation
7      keyNodeVector [ ] ← keyNodesRanking (VioVector[i]);
       //Selecting and ranking critical nodes which have a big influence on the constraint violation
8      FOR j =1 to keyNodeVector[ ].size( ) DO
9        equCandiateVector [ ] ← equCandidateSearch (keyNodeVector [j]);
         //Selecting all functionally-equivalent candidates which have the same associated abstractservice
10       QoSBargaining (equCandiateVector [ ]);
         //Launching the bargaining process among all candidate service providers
11       QoSUpdating ( );   //Updating QoS values after each round bargaining
12       WHILE (!(deadline ( ) || violationRemoved ( ) )) DO
13         rebargaining ( );   //Launching new round bargaining
14         QoSUpdating ( ); //Bargaining can be done recursively until no better QoS values can be achieved
15       END WHILE
16       supplierRanking ( );  //Ranking all candidates according to the new bargaining results
17       supplierSelecting ( ); //Reselecting the suitable candidate
18       updateNegotiationSpace (negoSpaceVector [i]);
         //Updating the negotiation space for next critical node
19       GOIEPath ← executionNodeAdjusting (GOIEPath);
         //Modifying the execution path with new better service provider
20     END FOR
21     updatingGlobalQoSValues (GOIEPath);
       //Recalculating the changed global QoS values on the new path
22 END FOR
23 RETURN GOIEPath;
```

图 1　基于迭代 Bargaining 策略的约束违背纠正算法

Fig. 1　Bargaining-based constraint violation correcting algorithm

2 将 LOIEpath 初始化成全局最优执行路径 GOIEpath。步骤 3 针对初始化的执行路径 GOIEpath 计算各个质量维的综合 QoS 值, 这些值被记入向量 QoSVector[ ] 。步骤 4 利用全局约束违背检查模型探测所有约束违背的质量维, 记为 VioVector[ ] 。特别地, 对于 VioVector[ ] 的各个质量维, 同样按照用户的偏好对它们进行排序, 重要的质量约束违背被优先得到纠正, 而次等级的质量约束违背纠正不能破坏先期已经得到纠正的质量维。步骤 5~ 22 开始对执行路径 GOIEpath 的约束违背质量维采取迭代 Bargaining 策略进行关键节点的谈判和替换, 并最终建立优化的且满足用户 QoS 约束要求的执行路径。步骤 6 进行相应质量维约束违背空间的计算, 该违背空间被当作初始化的谈判空间, 记为 negoSpaceVector[i] , 其中 i 代表对应的质量维向量下标, 该谈判空间的计算方法为路径在该质量维上的全局 QoS 值减去用户要求的约束值, 这里假设质量维为负, 即 QoS 值越大越不好, 对于正的质量维, 计算的方向相反。步骤 7 对全局执行路径 GOIEpath 的关键节点进行排序, 对全局 QoS 值影响越大的节点进行优先谈判。步骤 8~ 20 开始对选取的关键节点进行迭代谈判。步骤 9 对选取的关键节点定位其他功能相等的候选服务商, 并把这些服务商当作谈判的组对象。步骤 10 采取具体的迭代 Bargaining 策略同候选服务商同时进行谈判。步骤 11 对谈判后改变的 QoS 属性值进行升级。在谈判设置的最后期限内只要该质量维约束违背还没有被删除, 步骤 12~ 15 对候选服务商就可以发起多轮谈判。在对关键节点的所有候选服务商结束谈判后, 按照它们改变的属性值, 步骤 16 按照排位公式(2) 重新对所有候选服务商进行排序。步骤 17 选取

排位较高且能降低约束违背空间的服务商替换原来关键执行节点。步骤 18 根据新的执行节点提供的 QoS 值更新谈判空间。步骤 19 根据改变的执行节点升级执行路径 GOIEpath。只要谈判空间 negoSpaceVector[i] 还大于零, 步骤 8~ 20 就循环, 选取关键执行节点进行类似谈判。当循环 8~ 20 退出后, 步骤 21 根据执行路径 GOIEpath 被改变的执行节点升级全局综合 QoS 值, 并计算下一个权重稍低的约束违背质量维。步骤 5~ 22 循环处理所有约束违背质量维, 直到所有 QoS 约束违背删除。步骤 23 返回新的执行路径。

3.2 迭代 **Bargaining** 策略

在商业经济领域, Bargaining 策略在买卖双方有着重要的作用, 它们往往用来平衡消费者和提供商之间的利益分配[10~12]。单纯来看, Bargaining 策略实际上是买卖双方在经济领域中各自追求利益最大化的一个博弈过程。本文借用 Bargaining 策略在经济领域的谈判技巧来优化服务合成方案的执行路径, 同时尽可能为用户提供更好的服务。在图 2 的算法里初步提到使用 Bargaining 策略来降低约束违背空间。

对选取的关键节点, 图 1 算法步骤 9 能够定位它的所有其他功能相似的候选服务商。实际上, 在商业竞争环境下, 这些功能相似的服务提供商一起构成了针对用户的一个隐性竞争群体, 他们只能通过相互激烈的竞争才能确定谁为用户提供真正的服务。通常, 为了尽可能使自己利益最大化, 服务提供商服务属性的广播初值要高于他们实际的提供值。因此当用户发起对服务提供商的 Bargaining 谈判时, 服务提供商往往根据当前形势实时作出让步以减少他们最初想得到的利润。图 2 展示了一个用户同时与多个功能相似的服务提供商展开 Bargaining 的 Agent 模型。

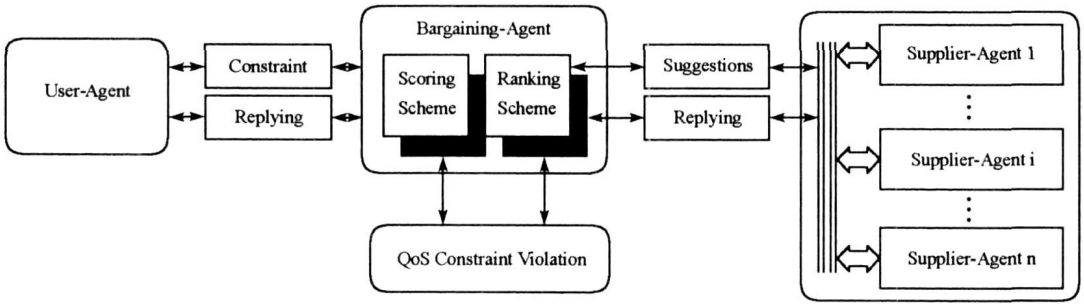

图 2 Bargaining-Agent 模型

Fig. 2 Bargaining-Agent model

Bargaining-Agent 为总的协控中心, 它接受来自相关质量维的约束谈判空间, 以及协调用户 Agent 同多服务提供商 Agent 之间的谈判。基本过程为: 首先, 用户 Agent 发起谈判请求, Bargaining-Agent 收到请求后, 同时向多个服务商 Agent 发出谈判请求, 并让服务提供商意识到存在机会向用户提供服务, 但当前的竞争非常激烈, 如果他们能对当前提供的服务质量作出一些改善, 则赢取同用户交易的成功率将大大增加; 反之, 可能失去同用户交易的机会。同时, 不同的服务提供商可能收到来自 Bargaining-Agent 不同的消息, 这些消息能够让服务提供商意识到当前的竞争形势和自己的综合排名地位, 并进而调整关键服务质量属性以提升自己的综合竞争力。这样的谈判过程能被多次迭代进行, 直到在规定的时间内再没有让步发生或约束违背已经被删除。因此, 总体上看, 用户同时对多个提供商之间的谈判能让用户获取质量更好的服务, 进而通过关键节点的替换可以降低用户 QoS 约束空间违背。但是, 也可能存在用户某些质量维的 QoS 约束需求过于苛刻以至于即使经过多个关键节点的多轮谈判和替换仍无法消除用户的 QoS 约束违背, 则此时 Bargaining-Agent 应设法通知用户修改相应的 QoS 约束条件。

图 3 给出 Bargaining-Agent 的具体处理步骤。Bargaining-Agent 接受具体的约束违背质量维和相应的约束违背空间(谈判空间) (步骤 1) ; 选出执行路径上一个具体的执行节点以及定位它的所有候选服务提供商(步骤 2, 此步骤相当于图 1 算法步骤 8 和 9 相结合) ; 在步骤 3, 利用本地最优化排名公式 2 对候选服务商进行综合排名; 在步骤 4, 根据不同服务提供商的排名情况, 找出可以提高这些服务商综合排名的提高因子, 并作出具体的建议, 以及将这些不同的建议分别发送给不同的候选服务提供商, 鼓励他

们改善相应服务质量以提高自己的综合排名。在激烈的竞争环境中为了赢取同用户交易的可能性, 一些服务商将作出让步并提高他们相应的服务质量。Bargaining-Agent 在规定时间内得到服务商的反馈后, 对所有服务商重新进行排位和再次发起 Bargaining 协商过程(步骤 5) ; 步骤 4 和 5 被重复执行直到在规定的时间期限内没有更好的 QoS 被提供; 当跳出步骤 4 和 5 的循环后, Bargaining-Agent 选出综合排名较高同时又保证约束违背空间降低的服务提供商代替原来执行节点(步骤 6)。在步骤 7, Bargaining-Agent 重新计算新的约束违背空间, 并将它作为谈判空间发送给下一个关键节点继续进行降低约束违背空间的处理。步骤 2~7 被迭代处理直到所有选择的关键节点进行了协商谈判或者相关质量维的约束违背空间已被清除。最后得到两种结果, 要么相关质量维的约束违背被删除, 要么用户在该质量维上的约束条件过于苛刻而没有执行路径满足, 因此需要通知用户重设约束条件。

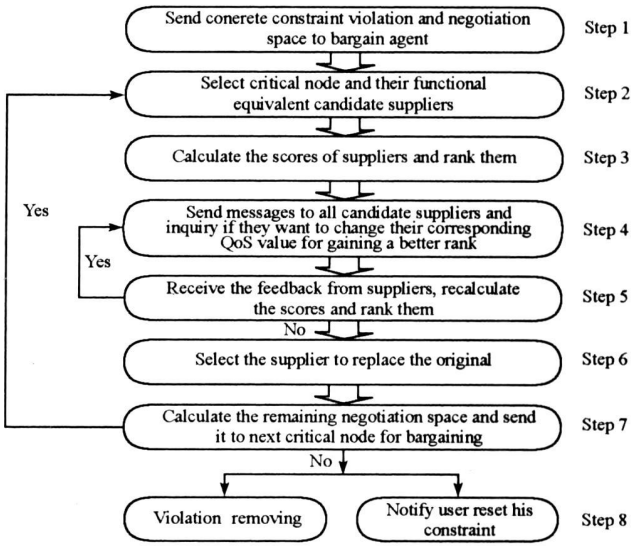

图 3 Bargaining 关键处理步骤

Fig. 3 Key steps for Bargaining

4 结 论

给出了一个在市场竞争环境下基于迭代 Bargaining 策略建立优化执行路径的方法。迭代 Bargaining 策略主要借鉴了商业环境下同一功能服务提供商之间的隐性竞争关系, 并通过发送不同的消息对话策略让服务商相互形成一种竞争压力, 这种压力能够迫使他们在相应服务属性上做出让步, 最后用户能够获取质量更好的服务提供商, 并进一步删除用户 QoS 约束违背以建立优化的执行路径。因此, 该方法更加反映了市场业务现实, 具有可行性。

参 考 文 献:

[1] 代任, 杨雷, 张斌, 等. 支持组合服务选取的 QoS 模型及优化求解[J]. 计算机学报, 2006(7): 1167- 1178.

[2] Yu T, Zhang Y, Lin K J. Efficient Algorithms for Web Services Selection with End-to-end QoS Constraints[J]. ACM Transactions on the Web, 2007, 6(1): 1- 26.

[3] Ardagna D, Pernici B. Adaptive Service Composition in Flexible Processes[J]. IEEE Transactions on Software Engineering, 2007(6): 369- 383.

[4] Kalasapur S, Kumar M, Shirazi B A. Dynamic Service Composition in Pervasive Computing[J]. IEEE Transactions on Parallel and Distributed Systems, 2007(7): 907- 918.

[5] Sun Y, He S Y, Leu J Y. Syndicating Web Services: A QoS and User-driven Approach[J]. Decision Support Systems, 2007(1): 243- 255.

[6] 杨胜文, 史美林. 一种支持 QoS 约束的 Web 服务发现模型[J]. 计算机学报, 2005(4): 589- 594.

[7] Zeng L Z, Benatallah B, Ngu A H, et al. QoS-aware Middleware for Web Services Composition[J]. IEEE Transactions on Software Engineering, 2004(5): 311- 327.

[8] Cardoso J, Sheth A, Miller J, et al. Quality of Service for Workflows and Web Service Processes[J]. Journal of Web Semantics, 2004(3): 281- 308.

[9] Jaeger M C, Rojee-Goldmann G, MUhl G. QoS Aggregation for Web Service Composition Using Workflow Patterns[C]// Proceedings of the 8th IEEE International Enterprise Distributed Object Computing Conference, California, USA, IEEE Computer Society, 2004.

[10] Debenham J. Multi-issue Bargaining in an Information-rich Context[J]. Knowledge-based Systems, 2004(2- 4): 147- 155.

[11] D vila J, Eeckhout J. Competitive Bargaining Equilibrium[J]. Journal of Economic Theory, 2008(1): 269- 294.

[12] Chen D N, Jeng B, Lee W P, et al. An Agent-based Model for Consumer-to-business Electronic Commerce[J]. Expert Systems with Applications, 2008(1): 469- 481.