动态联盟成员间工作流协同中的语义冲突消解①

Semantic Conflicts Handling in Workflow Coordination Among Members of Dynamic Alliance

陶 洁 王恒山 商艳艳 (上海理工大学 管理学院 上海 200093)

摘 要： 动态联盟由于其成员的自主性和异构性必然导致流程的异构性和冲突的产生。如何检测和消解跨组织工作流中的语义冲突也就成为了近年来相关研究的主要课题。在分析了前人相关研究的基础上，分析了动态联盟成员间工作流协同条件下语义冲突的主要类型，并通过基于本体的方法实现了主要对模式层的语义冲突的检测和消解，最后以 C#.Net 和 OWL 为技术平台实现了系统原型。

关键词： 动态联盟 工作流协同 跨组织工作流 语义冲突消解

1 引言

动态联盟是由两个以上的组织成员(单位)组成的一种有时限(暂时、非固定化)的相互依赖、信任、合作的组织，以便以最小的投资、最快的反应速度(最短的反应时间)对市场机遇做出反应。其成员以自身的实力和社会信誉通过竞争被核心公司(企业) 吸收加入。动态联盟企业除具有一般企业的基本特征，还有一些专门的特点：动态联盟企业通常由盟主企业发动组建，根据机遇产品的资源需求设定企业模型、选择联盟伙伴、确定协作关系，并在企业运作过程中根据需要对协作关系做出调整，其组织过程需要灵活可变的动态性；动态联盟企业是以生产经营过程为主线的企业组织，各成员在这一主线的贯穿下，按照一定的协作规则完成各自环节上的任务，其运作过程强调明确可控的协作性；动态联盟企业的各成员在组织上相对独立，地理上较为分散，但他们之间的协作需要快速畅通的信息传递，其信息系统要满足及时有效的分布性。

针对以上特点，国内外很多学者都做过相关的研究。例如储静辉、宋斌恒将流程划分为生产流程和管理流程，并对之进行了建模[1]；杜彦华、范玉顺运用 ESP 规则从语义的角度分析了跨组织工作流系统的模型[2]。但是，现在的研究主要集中于协同的正向方面，因为跨组织的流程牵涉到不同的企业，各个企业间的流程一定存在不同程度的不一致性，如何通过语义方面的分析规避这种语义上的冲突，是本文研究的主要内容。

2 基于语义的工作流协同模型

现有的基于语义的分析一般都是制订一定的规则，无论是协同过程模型(CPM,Coordinated Process Model) 还是事件-状态-过程规则模型(ESP, Event-State-Process)。在 CPM 方法中，生产流程是十分稳定的，但是一旦生产流程出现异常，如何应对这种异常则是十分动态的。所以本文选用 ESP 规则模型作为描述工作流协同模型的工具，从而使其具有良好的敏捷性和协同性。

ESP 规则的定义如下：base-workflow/ event-class:state→meta-workflow。其中，元工作流(meta-workflow) 是一种高层控制流，且与业务工作流程分离。针对不同的事件和业务流程状态，会执行一个元工作流去处理各种功能。在基于 ESP 规则的实现框架中，业务工作流的状态可以通过流程中的活动运行状态,以及相应的业务环境状态(如各种对象)来表

① 基金项目:上海市重点学科——管理科学与工程(s30504) ；上海市研究生创新基金（JWCXSL0902）
收稿时间:2009-03-10

达。活动运行状态分为：结束(Done)、运行中(Active)、就绪(Ready)、终止(Aborted)，以及挂起(Suspended)。因此分别定义集合 Done, Active, Ready，Aborted 和 Suspended，某一活动 A ∈ Done,表明 A 已经执行完成,其他同理。例如某一 ESP 中的状态表达(A∈Done∧{B,C}Active∧cost≤300)表示在该状态下活动 A 执行结束，B 和 C 正在执行，并且全局对象 cost 不大于 300[2]。

例如现有工作流 $wf_1(t_{11}, t_{12}\wedge t_{13})$，$wf_2(t_{21}, (t_{22}, t_{24})\wedge(t_{23},t_{25}))$和 $wf_3(t_{31}, t_{32}\wedge t_{33})$，其结构如下图(图 1)所示。

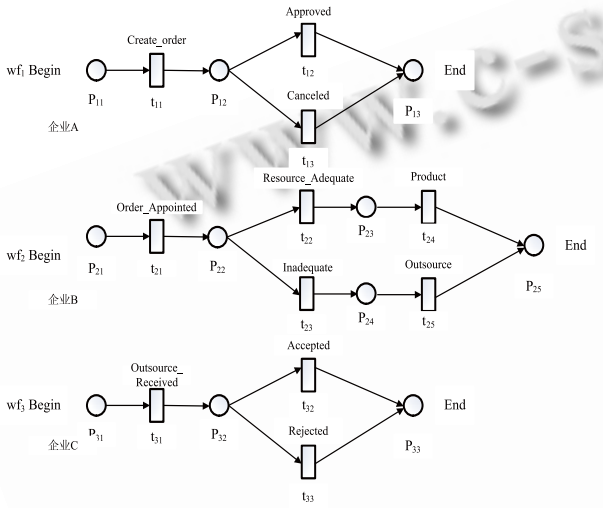

图 1　动态联盟的三个企业的工作流示意图

因为篇幅所限，本文对实际的工作流进行了简约化处理。图 1 中的 wf_1 代表盟主企业——也就是接包企业的工作流程，客户发出订单，盟主企业决定是否接受订单；wf_2 代表生产企业的流程，接到盟主企业分配的订单后首先判断生产所需的资源是否足够，如足够则进行生产，如不够则发出外包动作；而 wf_3 则代表外包企业的工作流程，在接到外包任务的指派后，外包企业同样有接受与否的选择。基于 ESP 规则的概念,我们可以得出以上工作流的 ESP 规则如下：

Events: $t_{12} \subseteq Active; t_{23} \subseteq Active;$

Coordination Workflows: cwf_1; cwf_2

ESP Rules:

$$(wf_1, wf_2)/\ t_{11} \subseteq Active \rightarrow cwf_1;$$
$$(wf_2, wf_3)/\ t_{24} \subseteq Active \rightarrow cwf_2;$$

cwf_1, cwf_2 的结构如图 2 所示。

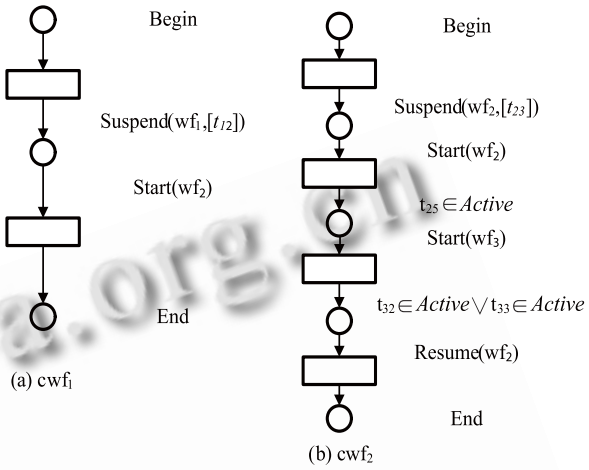

图 2　协同工作流的结构

图 2(a)的含义是例如当盟主企业接标时，则 $t_{12} \subseteq Active$ 的条件为 true，则触发 cwf_1，首先执行 Suspend (wf_1,[t_{12}])，而后转至启动 wf_2；当生产型企业决定外包产品生产过程时，则 $t_{23} \subseteq Active$ 为 true，则触发如图 2(b)所示的协同工作流，首先执行 Suspend(wf_2,[t23])，而后当满足 $t_{25} \subseteq Active$ 为 true 时执行 wf_3,在捕捉到 t_{32}∈Active∨t33∈Active 为 true 时 Resume(wf2)。

3　工作流间的语义冲突及其检测的方法

工作流之间的语义冲突主要是在不同组织的业务信息资源之间存在不一致性，正是这种不一致性导致了不完整和错误信息的产生。不完整和错误的信息导致集成的冗余，干扰组织间信息的发布、处理和交换。对于跨组织工作流之间的冲突，更注重于信息的发布和交换的方面。语义冲突在学术界被广泛认可的定义是：语义冲突是指当描述同一现实世界事物时，两个对象在描述方式、结构上和内容上的不同造成的语义不一致性。

3.1 工作流协同中的语义冲突类型

从语义冲突结构的角度对之进行区分，可将其划

分为数据层冲突和模式层冲突。数据层冲突(Data-level conflicts)是由于对相同概念不同感知，在命名、标识符、精度、表示方式、可信度等方面造成的冲突。数据层冲突又可具体划分为数据值冲突（同一数据值在不同环境下解释不一致造成的冲突）、命名冲突(同一概念被不同的设计人员主观决定造成的冲突)、实体标识符冲突(不同系统间对同一实体采用不同的标识符带来的冲突)、数据表示冲突(不同系统对同一概念采用不同的数据类型或表示方法造成的冲突)和取值范围冲突(不同系统间相同概念具有不同的值域和定义域造成的冲突)。

在工作流协同的环境当中，数据层冲突也是明显存在的。以上文所举的场景为例，任意两个企业工作流间的数据都可能存在不一致性。具体例子如表 1 所示。

表 1 组织间的数据层语义冲突类型及实例

数据层冲突类型	企业A	企业B	实际含义(A/B)
数据值冲突	1	1	整体/可行
命名冲突	Customer	Client	代表“客户”
实体标识符冲突	Customer_id	Reg_Name	“客户”的标识符
数据表示冲突	DD-MM-YYYY	YYYYMM DD	日期型对象表示方式
取值范围冲突	[0.15,0.35]	[0.25,0.5]	对象“discount”取值范围

从上表中可以看出，在 A、B 企业间的工作流进行协同的时候，以上冲突会导致协同的不稳定甚至是彻底失败。表 1 中所举的例子仅仅是诸多情况中的一种。但是，在实际工作之中，数据层的冲突并不是关注的焦点。这是因为：

① 数据层冲突是较容易被发现的。正如表 1 中的例子表示出的，除了数据值冲突因为牵涉到对象内部的逻辑含义较难发现以外，其他的错误都可以通过简单的比对和协同组织业务人员和建模人员的有效沟通得以发现和避免。

② 除了较易发现以外，数据层冲突同时也是比较容易解决的。XML(可扩展标记语言，EXtensible Markup Language)自其于 1998 年 2 月由 W3C 发布 XML1.0 标准以来，就一直致力于提供更中立的方式以让消费端自行决定如何消化、呈现从服务端所提供的信息，其主要功能也就是通过解释元数据(Metadata)的形式消解异构数据的冲突，发展至今，XML 已经取得了极其丰硕的成果，工作流协同语义冲突中的数据层冲突这一部分也可通过引入 XML 得到解决。事实上，这项技术早已进入实践阶段，多家厂商的工作流产品都已经按此实施，由于篇幅所限，本文不做赘述。

除了数据层冲突以外，语义冲突还有另外的一种结构类型就是模式层冲突(Schema-level Conflicts)。模式层冲突是由于相同概念在不同信息源里采用不同的逻辑结构造成的冲突。模式层冲突主要包括异构信息、对概念的不同理解以及由于采用不同的语言或者同一语言的不同版本描述的概念存在导致的结构差异。模式层冲突包括以下几类：

① 模式异构冲突(Schema Isomorphism Conflicts):在不同系统相同概念的属性集合不相似造成的冲突。也可称为属性集冲突。

② 概括冲突(Generalization Conflicts)是指一个系统中的多个概念在另一个系统中可能被某一个概念所涵盖。

③ 聚合冲突(Aggregation Conflicts):一个系统内单个概念的属性或属性值来自于另一个系统一个集合概念的属性或属性值。

④ 关系冲突(Relation Conflicts):在不同系统相同概念与其他相关概念间关系不一致引发的冲突。

⑤ 约束冲突(Restriction Conflicts):不同系统对相同概念或关系采用不同的约束条件造成的冲突。

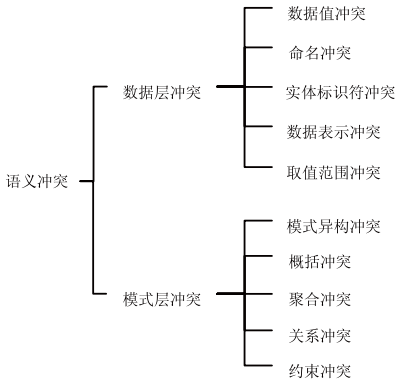

图 3 工作流协同中的语义冲突类型[3]

综上所述，在工作流协同中的语义冲突的类型如上图(图 3)所示。得出以上类型后，语义冲突的处理——特别是模式层冲突的处理——必须被分作两步来考虑：语义冲突的检测和语义冲突的消解。

3.2 基于本体的语义冲突检测方法

本体(Ontology)原是哲学上的概念，在计算机领域作为信息抽象和知识描述的工具。根据 Gruber 的定义，本体是概念模型明确规范的描述。本体作为一个公认的概念集，概念具有公认的语义，并通过关联以实现。

语义冲突的处理分为两个步骤，分别是语义冲突检测和语义冲突消解。语义冲突检测首先需要检测不同系统内概念的语义相关程度，判别不同系统间概念是否存在不一致性，如果存在，则应判断存在语义冲突。本文考虑采用以下结构的系统用以判断语义相关程度的不一致性。

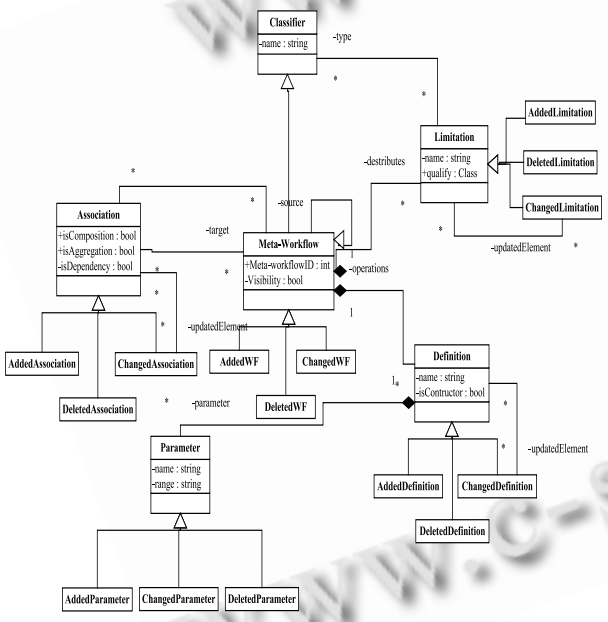

图 4　语义冲突检测系统的 UML 静态结构图

图 4 的结构表示：首先对不同的语义场景进行分类，按照不同的场景建立不同的元模型(Meta-model)，用以解释系统间工作流的结构。

Association 模块的主要功能是判断语义冲突中的关系冲突的问题，该模块将不同的元模型的概念加入系统中，首先判断概念的构成结构以及依赖关系，而后将其对应关系表示出来。isComposition(C)对象用以表征概念间的构成关系；isAggregation(A)对象用于检测概念间的发展关系，即某一概念是另一概念的衍生和拓展；isDependency(D)用以检测依赖关系。假设△1，△2 表示不同系统中的概念，那么其间的冲突可以表示为

$$\text{Conflict}(\triangle 1,\ \triangle 2)=\triangle 1:\triangle 2 \tag{1}$$

而后经过 Association 模块整理后的对应关系可以描述为(其中 T 为△1，△2 的内在逻辑关系)

$$Coordination(\Delta 1,\Delta 2)=T(\Delta 1):T(\Delta 2)\quad [T\in\{C,A,D\}] \tag{2}$$

Operation 模块用以解释聚合冲突或是概括冲突，因为从本质上说聚合冲突和概括冲突仅是方向上的不一致，也即某系统中一个实体对应一个系统中另几个实体的对应关系，Name 对象用以发现概念间的对应，而 isConstructor 用以检测概念之间的组成逻辑关系。假设系统 A、B 间存在如下的概念对应关系，其中 $\lambda_i(i=1,2,3,4)$ 为系统中的基本实体：

$$A.\lambda_1=conflict(B.\lambda_2,B.\lambda_3,B.\lambda_4) \tag{3}$$

式(3)表示的含义是 A 中的 λ_1 对应于 B 中的 $\lambda_2,\lambda_3,\lambda_4$。首先需要判断 $\lambda_2,\lambda_3,\lambda_4$ 中属性的逻辑关系，消除其整合后的冲突性和冗余性。假设 λ_1 包含的属性可以表示为 λ_1 (c_1，c_2，c_3，c_4)，而 $\lambda_2,\lambda_3,\lambda_4$ 中分别包含的属性为(c_5,c_6)，(c_7,c_8,c_9)，(c_{10},c_{11})，其中 c_5,c_7 和 c_3 存在逻辑上的不一致性，而优化过的属性为 $c_5^{'},c_7^{'}$。而 c_6，c_{11} 是冗余的，以取 c6 为例，则结合属性的对应关系如下：

$$\lambda_1(c_1,c_2,c_3,c_4)\leftarrow Coordinate(\lambda_2,\lambda_3,\lambda_4)\quad (c_5^{'},c_6,c_7^{'},c_8,c_9,c_{10}) \tag{4}$$

其中 $Coordinate(\lambda_2,\lambda_3,\lambda_4)$ 为调和了属性和概念间的逻辑关系后的对应关系。

Attribute 模块用以解决约束冲突的问题。Name 对象用以储存同名概念间的关系，而 visibility 对象用以检测约束条件集的逻辑冲突。假设系统 A 中存在一个实体关系 θ_1，其约束条件集为 R_1，R_2，R_3，而相应的 B 中也存在 θ_1，但其约束关系为 R_4，R_5，其中 R_1 和 R_4 之间存在逻辑冲突，经过消除逻辑冲突的约束条

件为 R ’，则协同后的θ_1(R ’,R_2,R_3,R_5)为原本实体关系θ_1的对应关系。

而 Parameter 模块用于检测系统(组织)间的参数冲突，基于上文所述，name 对象用于储存参数的名称，range 对象用于储存参数的取值范围，cata 对象用于储存参数的类型，例如整型(int)、浮点型(float)和日期型(date)等等。

4　基于本体的语义冲突消解方法

OWL(Web Ontology Language)是 W3C 在 2005 年推荐成为标准的基于语义的互联网本体描述语言。它建立在资源描述框架(RDF, Resource Description Framework)的基础上，但具有较之更丰富的逻辑描述功能。针对不同的需要，OWL 可划分为三个层次：OWL lite，OWL DL，OWL full[4]。

根据上文所述，本文采用本体的方法以检测和消解语义冲突。首先，需要对本体进行定义和推理，其目的是为了提取出隐含在显式的定义和规则中的信息。只有建立起有效的定义和推理机制，才可以准确的检测和明确的消解冲突。下面结合前文所述的三个企业的实例具体讨论。这三个企业的工作流中牵涉到的主要本体类及其关系抽象如下图所示。

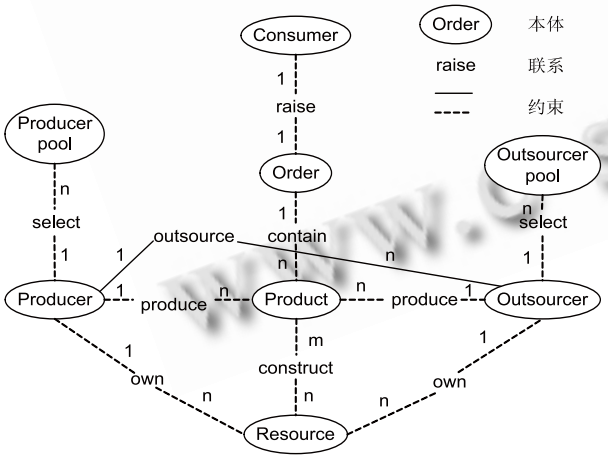

图 5　主要本体类的实体-联系图

根据图 5 所示，上述三个企业工作流中牵涉到的主要本体有以下八个：Consumer(客户)、Order(订单)、Resource(资源)、Product(产品)、Producer(制造商)、Outsourcer(外包商)、Producer pool(制造商池)和 Outsourcer pool(外包商池)。其中 Producer 和 Outsourcer 分别为 Producer pool 和 Outsourcer pool 的实例。

冲突消解方法的核心过程可以描述如下：

输入：冲突检测结果、对象(属性)；

处理：

STEP1. 从接口读取冲突检测结果和对象；

STEP2. 根据检测结果和对象类型调用适合的本体模型，如果模型不存在则以对象为原型建立新本体；

STEP3. 将本体按照具体场景实例化，同时将实例对象按本体的结构规范化；

STEP4. 将对象规范化的规则存档，并作为检测语义冲突的标准；

输出：规范化的对象(属性)、规范化规则.

具体实施方案采用 C#.Net 作为宿主语言，而以 OWL 作为寄生语言。OWL 的主要功能是构建本体和描述本体的结构，而采用 C#.Net 来实现读接口、对本体进行增删改操作、存档输出等外部功能。OWL 和 C#.Net 都具有与平台无关的特性，从而可以满足跨组织工作流跨平台的需要。

用 OWL 描述的本体结构如下[5]，以 Consumer 和 Order 本体为例，由于篇幅所限，本体的属性经过简约化处理：

```
<owl:Class rdf:ID= ”Consumer ”/> //定义 Consumer 类
<owl:Restriction>  //声明限制
<owl:onProperty>  //声明属性
<owl:ObjectProperty rdf ID=”Consumer ID”/>
// 定义对象属性 ConsumerID
<owl:ObjectProperty rdf ID=”...”>
// 定义对象的其他属性
</owl:onProperty>
</owl:Restriction>
<owl:Class rdf:ID= ”Order ”/> //定义 Order 类
<owl:Restriction>  //声明限制
```

<owl:onProperty>　//声明属性

<owl:ObjectProperty rdf ID="OrderID"/>
//定义对象属性 OrderID

<owl:ObjectProperty rdf ID="...">
//定义对象其他属性

</owl:onProperty>

</owl:Restriction>

上述代码声明了两个类，Consumer 和 Order，并声明了它们的主要属性。同时 OWL 也可以定义类之间的联系以及如何按照场景规则将类实例化。通过系统原型验证，5 种主要的语义模式层冲突都得到了良好的消解。

5 结论

动态联盟的内在特点表明它是我国企业做大做强的必经之路，同时它的特性也说明其成员间工作流协同的语义冲突是不可避免的。本文在进行定义和分析的基础上，提出了发现和消解冲突的方案，并以 C#.Net 和 OWL 为技术平台实现了系统的原型。

下一步研究的主要方向是如何将系统的原型和 ASP(Application Service Provider，应用程序服务提供商)相结合，提出可以实际应用的系统，并结合可以消解数据层冲突的 XML，开发出真正可以商业化的产品，更好的为我国产业化集群的工作流整合服务。

参考文献

1 储静辉,宋斌恒.工作流协同过程建模.计算机应用研究, 2007,24(4):121－124.

2 杜彦华,范玉顺.基于事件-状态-过程规则的跨组织工作流协同方法.计算机集成制造系统，2008,14(7):1343－1348.

3 Ram S, Jinsoo Park. Semantic conflict resolution ontology (SCROL): An ontology for detecting and resolving data and schema-level semantic conflicts. Knowledge and Data Engineering. IEEE Transactions, 2005,16(2):189－202.

4 Horrocks I, Patel-Schneider PF, van Harmelen F. From SHIQ and RDF to OWL: The making of a Web ontology language. 2003. http://www.cs.man.ac.uk/- horrocks /T eaching/cs646/

5 W3C.OWL Web Ontology Language Overview. 2004. http://www.w3.org/TR/2004/REC-ow1-features-20040210 /

(上接第 69 页)

和鲁棒性的矛盾，并且使图像信息无损，水印隐藏效果好。本文提出的一种基于最高位面构造零水印的算法，有效的实现了这一技术。通过对比文献[4]的实验表明，该算法不仅在抵抗各种攻击的性能方面优于文献[4]算法，而且在区别不同的、内容相近的无水印图像方面也优于文献[4]算法，具有更好的鲁棒性。

参考文献

1 温泉,孙锬锋,王树勋.基于零水印的数字水印技术研究.全国第三届信息隐藏学术研讨会论文集(CIHW 2001).西安:西安电子科技大学出版社, 2001.

2 谢贤智,归奕红. 零水印技术对数字图像版权保护的应用研究.广西工学院学报, 2007,18(3):113－116.

3 温泉,孙锬锋,王树勋.零水印的概念与应用.电子学报, 2003,31(2):214－216.

4 高青山,罗向阳,等.一种基于混沌阵列的鲁棒零水印算法.计算机科学, 2005,32(9):76－81.

5 Arnold VI, Avez A. Ergodic problems of classical mechanics. Mathematical Physics Monograph Series. New York: W A Benjamin, Inc., 1968.

6 Cox IJ, Joe Kilian, Fthomson. Secure spread spectrum watermarking for multimedia. IEEE Trans. on Image Processing, 1997,12(6):1673－1687.

7 陈灏.空域数字图像水印算法研究与实现.现代电子技术, 2007,249(10):149－165.

8 Gonzalez R,著;阮秋琦,等,译.北京:电子工业出版社, 2005.