基于 RFID 发现服务的一种供应链建模技术

李信鹏[1,3],赵 文[2,3],刘殿兴[1,3],袁崇义[1],张世琨[2,3],王立福[2,3]
(1.北京大学信息科学技术学院,北京 100871; 2.北京大学软件工程国家工程研究中心,北京 100871;
3.北京大学信息科学技术学院软件研究所高可信软件技术教育部重点实验室,北京 100871)

摘 要: RFID 发现服务负责从供应链不同合作伙伴的信息服务中搜集与某个(些)物品相关的动态 RFID 事件,作为数据集.为了便于对发现服务获得的数据集进行可视化分析,本文给出了一种供应链建模技术.规约了发现服务获得的数据集,这种数据集描述了物品的移动、包装/解包装、加工制造;同时,采用 P2P 和并行处理技术,给出了一种新的分布式 RFID 发现服务;然后,基于 Petri 网提出了一种新的建模工具"Supply-Net",并给出了构建 Supply-Net 的算法.算法分析和实验表明这种构建 Supply-Net 的算法具有较高的效率和可用性.

关键词: 无线射频识别 (RFID); 发现服务; 供应链; Petri 网; 建模
中图分类号: TP311 **文献标识码:** A **文章编号:** 0372-2112 (2010) 2A-107-10

A Supply Chain Modeling Technology Based on RFID Discovery Service

LI Xin-peng[1,3], ZHAO Wen[2,3], LIU Dian-xing[1,3], YUAN Chong-yi[1], ZHANG Shi-kun[2,3], WANG Li-fu[2,3]
(1. *School of Electronics Engineering and Computer Science, Peking University, Beijing* 100871, *China*;
2. *National Engineering Research Center for Software Engineering, Peking University, Beijing* 100871, *China*;
3. *Key laboratory of High Confidence Software Technologies (Ministry of Education), School of Electronics Engineering and Computer Science, Peking University, Beijing* 100871, *China*)

Abstract: RFID Discovery Service provides a means to find all dynamic RFID events related to certain objects as the data set, from different enterprises' information services. To facilitate visual analysis on the discovered data set, a modeling technology for supply chain is proposed. . The discovered data set of supply chains is specified, which includes the movement of! objects, aggregating with other objects or disaggregating from an integrity, and processing or transforming materials into new products. As well, an innovative distributed RFID Discovery Service is briefly introduced by combining technologies of P2P and parallel processing. Then, a Petri net based modeling tool "Supply-Net" is proposed, and the algorithm for constructing Supply-Net is given. Finally, analysis and experiments on our algorithm show its high efficiency and utility.

Key words: radio frequency identification(RFID); discovery service; supply chain; petri net; modeling

1 引言

RFID(Radio Frequency Identification)技术是一种非接触的移动目标、多目标自动识别技术,近年来已成为自动识别领域的研究热点.RFID 技术可以广泛应用于供应链管理、食品安全、防伪、产品召回、交通监控等方面.通常,供应链连接着很多企业,从原材料的生产商开始,到使用物品的最终客户结束[1].这些物品可用一个全球唯一的 EPC(Electronic Product Code)[2]标签来标记,EPC 的编码结构包括三个主要编码段:(1) *General Manager Number* 标识一个企业或组织;(2) *Object Class* 在某企业/组织内唯一标识同一种类或类型的物品;(3) *Serial Number* 唯一标识某类物品中的一个单品.一个企业也可能存在于多个供应链中,每个企业维护一个 EPCIS (EPC Information Service)[3],用于识别或检验与 EPC 相关的业务事件的发生,然后作为 EPCIS 事件存储起来,并为本地和远程的应用提供查询接口.每个企业单独决定其 EPCIS 事件的访问权限和控制策略.为了降低成本、加强重点业务和关键环节、寻找增加收入的新机会,在同一供应链内的业务伙伴通常彼此共享其内部的 EPCIS 事件.RFID 发现服务(Discovery Service)[3]是这种信息共享的一种机制,它负责从不同业务伙伴的 EPCIS 上收集物品在生命周期内的过程信息,将分布的物品信息按时间序列整合成完整的物品信息链.然而,如何对

收稿日期:2009-09-15;修回日期:2010-01-10
基金项目:国家高技术研究发展计划(863)(No.2006AA04A119,No.2006AA04A121);国家重点基础研究发展计划(973)(No.2009CB320706);国家自然科学基金(No.60803014);教育部博士点新教师基金(No.200800011017)

RFID 发现服务获得的 EPCIS 事件数据进行分析利用，以改善供应链各环节的效率，目前对这个问题的系统性研究还相对较少.

目前的研究工作往往仅限于对供应链的建模. IBM 提出了一种供应链建模工具 Object Movement Graph (OMG)[4]，可在企业级别和企业内部的子公司(厂房、车间)级别这两种粒度下建立供应链的模型. 另外，Henry H. Bi 又提出了一种能够描述三种粒度级别的供应链建模工具[5]，除了上述两种粒度外，还可进一步描述子公司(厂房、车间)级别下更详细的物理位置级别. 但是，这两种建模工具主要描述物品的移动，而对物品的包装/组装或解包装/拆卸这类事件支持不够，特别是无法描述物品的生产制造这类事件. 还有一类对供应链建模的研究是基于 Petri 网理论的. 比较有代表性的研究是 N. R. Srinivasa Raghavan 提出的经典模型“广义随机 Petri 网”(generalized stochastic Petri nets, GSPN)[6]，并在此基础上做了比较简单的库存-订单关系分析，以达到优化库存的目的. 另外 A. Desrochers 还提出了一种具有复数 token 的 Petri 网模型(jPN)[7]对供应链建模，将 token 分为实部 token 和虚部 token 两类，来解决供应链的同步和协调问题，这是有色网的思想. 但是 jPN 仅限于对两种物品的供应链建模和同步性分析，实际应用性不强. H. Chen 提出了批处理确定性与随机 Petri 网模型(Batch deterministic and stochastic Petri nets, BDSPN)[8]，通过大量的历史数据统计出物品移动时选择不同轨迹的概率，进而建立随机 Petri 网模型进行供应链决策分析. 但是 BDSPN 的表达能力仍然局限于物品的移动.

这些研究都是针对特定的供应链环境下的建模. 本文涉及的供应链环境中，不仅包含单个或批量物品的移动，还包括物品的包装/组装与解包装/拆卸，以及将原材料物品加工处理以后生成新物品. 为了对这种供应链环境进行建模，本文基于 Petri 网理论提出了一种新的建模工具“Supply-Net”，并给出了利用 RFID 发现服务获得的数据集构建 Supply-Net 的算法.

2 基于 RFID 发现服务的数据集

本节对 RFID 发现服务获得的供应链的数据集进行了规约. 首先，因为这种数据集是由 EPCIS 事件组成的，所以我们重新为 EPCIS 事件进行建模，使之能够表达供应链活动中的绝大多数事件；然后，简单介绍了一种新的分布式结构的 RFID 发现服务.

2.1 发现服务获得的数据集

为了增加供应链中物品的可跟踪/追溯性，同时更加全面的描述供应链活动中产生的各种事件，尤其是对那些涉及到将原料加工制成新产品的业务事件，下面首先对每个 EPCIS 中的数据重新进行建模. 基于 EPCglobal EPCIS Specification Ratified Standard[9]中定义的事件类型，我们给出了五种事件类型，其中包括一个基类事件和四个派生类事件，如图 1 所示. 它们可以表达在各个行业的供应链活动中产生的事件.

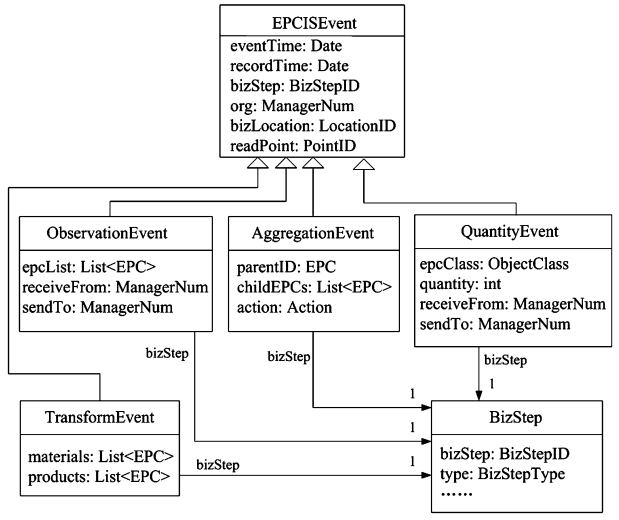

图1 EPCIS的数据模型

(1) EPCISEvent 是所有事件类型的基类，它描述了 EPCIS 所捕获事件的一般特征，包括事件的发生时间，发生地点以及基本的业务背景[9]. *eventTime* 是事件的发生时间；*recordTime* 是事件的记录时间，是一个可选的属性；*bizStep* 是一个 BusinessStepID，它记录了事件所属的业务步骤，通过该属性可以实现业务步骤的跟踪；*org* 记录了事件在哪个企业发生；*bizLocation* 记录了事件发生时所处的位置，例如“warehouse # 1”；*readPoint* 记录了事件在哪个物理位置发生，通常由物理读写器所标识，例如“dock door # 12”.

(2) ObservationEvent 描述了“物品仅被观察到但是没有被处理”这样一类事件. *epcList* 是一个 EPC 编码列表，它记录了事件中所读取到的所有物品标签编码；*receiveFrom* 记录了物品的直接上游(供应方)，它以 *General Manager Number* 的形式唯一标识一个企业；*sendTo* 记录了物品的直接下游(接收方). 在 ObservationEvent 中，*receiveFrom* 和 *sendTo* 两个属性只能有一个有值.

(3) AggregationEvent 表达了物品的包装/组装与解包装/拆卸的发生，例如：“一些零件被装入箱子中”. *parentID* 标识了包装物品的容器，或是由多个部件组装成的新物品；*ChildEPCs* 是一个 EPC 编码列表，它记录了事件所涉及的所有被组装的部件；*action* 描述了事件中所发生的动作，例如“ADD”表示包装/组装，而“DELETE”表示解包装/拆卸.

(4) QuantityEvent 描述的事件只关注同一种类物品的数量，而不关注具体的单品. *epcClass* 描述了物品的种类；*quantity* 记录了该类物品的数量.

(5)TransformEvent 刻画了多个物品经过加工制造以后生成新物品的事件,例如,“一块猪肉和一袋面粉加工成一些香肠”. *materials* 是一个 EPC 编码列表,记录了所有的原材料; *products* 也是一个 EPC 编码列表,记录了利用原材料加工成的所有物品.

上述四个派生类事件都通过一个“*bizStep*”属性关联了一个“业务处理步骤类”(BizStep),它刻画了事件发生时所处的业务背景.

发现服务获得的数据集是指,与给定物品(及其零件、原料)相关的、发生在供应链不同节点的所有 EPCISEvent 集合.下面通过一个典型实例来描述 RFID 发现服务获得的数据集的结构:一件产品 D 由 10 个零件 C 组装而成,每 2 个零件 C 由 3 件原料 A 和 5 件原料 B 加工制成. D 的制造商 M_D 从原料 A 的供应商 S_A 采购 3000 件原料 A,从原料 B 的两个供应商 S_{B1}, S_{B2}分别采购 3000 件和 2000 件原料 B,用原料 A, B 加工制成一批零件 C,然后把每 10 个零件 C 组装成一件产品 D,最后将 100 件 D 发给 D 的分销商 D_D, D_D 又将 50 件 D 发给一个零售商 R_D. 表 1 列出了与产品 D_1 相关的查询结果.

表 1　RFID 发现服务获得的数据集的一个实例

序号	事件类型	发生时间	Org.	收发关系	物品的 EPC 编码
1	Observation	2009/5/1	S_A	to M_D	$A_1 \sim A_{3000}$
2	Observation	2009/5/1	S_{B1}	to M_D	$B_1 \sim B_{3000}$
3	Observation	2009/5/1	S_{B2}	to M_D	$B_{3001} \sim B_{5000}$
4	Observation	2009/5/4	M_D	from S_A	$A_1 \sim A_{3000}$
5	Observation	2009/5/5	M_D	from S_{B1}	$B_1 \sim B_{3000}$
6	Observation	2009/5/6	M_D	from S_{B2}	$B_{3001} \sim B_{5000}$
7	Transform	2009/5/6	M_D	无	$m: A_1 \sim A_3, B_1 \sim B_4, B_{3001}$ $p: C_1\ C_2$
8	Aggregation	2009/5/7	M_D	无	$c: C_1 \sim C_{10} p: D_1$
9	Observation	2009/5/9	M_D	to D_D	$D_1 \sim D_{100}$
10	Observation	2009/5/16	D_D	from M_D	$D_1 \sim D_{100}$
11	Observation	2009/5/17	D_D	to R_D	$D_1 \sim D_{50}$
12	Observation	2009/5/20	R_D	from D_D	$D_1 \sim D_{50}$

注:m-原料,c-零件,p-产品

2.2　一种分布式 RFID 发现服务

下面简单介绍一种采用分布式和并行处理技术的 RFID 发现服务系统,用来获得上述数据集.该系统由安装在各个 EPCIS 节点上的 DS 引擎构成,每个 DS 引擎仅为其所在的供应链伙伴节点提供访问内部 EPCIS 信息的接口,每个 DS 引擎只知道其直接上游和直接下游的节点的访问接口.该系统获得数据集的步骤如下:(1)客户端生成初始查询,根据不同查询编码向相应的 EPCIS 节点并行发起多个查询;(2)每个节点的 DS 引擎接收到查询后首先查询其本地 EPCIS 的相关动态信息,再根据本地查询结果生成与其直接上游和直接下游相关的若干新查询,然后并发地转发给相应上下游节点,同时将本地查询结果直接返回给初始客户端.这个过程沿着供应链上相关的节点递归地进行,直到供应链边界节点为止;(3)客户端把从来自节点的查询结果整合、排序,得到最终查询结果.

3　用 Supply-Net 对供应链建模

利用上一节给出的方法,我们可以发现给定物品及其相关零件、原料等在供应链中各个节点发生的 EPCIS 事件,为了方便直观地对这些 EPCIS 事件数据进行分析利用,有必要建立供应链的模型.为此,本文定义了一种基于 P/T 系统[10]的 Petri 网建模工具“Supply-Net”,用来对供应链进行建模;然后给出了利用 RFID 发现服务获得的数据集构建 Supply-Net 的算法.

3.1　Supply-Net 的定义

首先,基于 P/T 系统的供应链建模工具 Supply-net,定义如下:

定义 1　一个九元组 $\Sigma = (P, T; F, K, Q, D, Ts, Type, tp)$称为一个 Supply-Net 的充分必要条件是:

(1)$(P, T; F)$是一个有向网, P 表示同类物品处于同一状态, T 表示发生的 EPCIS 事件集合, F 表示有向弧集合,该有向网没有回路;

(2)$K: P \to N \cup \omega$ 是库所的容量函数,其中 N 是自然数集合, ω 表示正无穷;

(3)$Q: F \to N$ 是一个有向弧 F 到自然数 N 的映射函数,该函数值表示同类物品的数量;

(4)$D: P \to R^+ \cup 0$ 是一个从库所 P 到非负实数(其中 R^+表示正实数)的映射函数,该函数值表示一个物品或者一类物品在一个库所的停留时间;

(5)$Ts: T \to Timestamp$ 是 T 到时间戳的映射函数,该函数值表示变迁发生的时刻;

(6) *Type* 是库所和变迁的类型的集合,其中变迁的类型包括 MOVE, LOSE, AGGREGATE, DISAGGREGATE, TRANSFORM 等.库所的类型包括 SHIPPING, LOST, STORING, PREPARED, PROCESSED 等;

(7)$tp: P \cup T \to Type$,函数的值表示库所、变迁的类型.

Supply-Net 的状态由 M 标识(Marking)表示. $\forall x \in P \cup T$, $^\cdot x$ 表示 x 的前集(pre-set), $x^\cdot$表示 x 的后集(post-set),并且 $^\cdot x = {}^\cdot x \cap x^\cdot$另外,这里还要提出 Supply-Net 中的一个重要概念“同步器(Synchronizer)”,定义如下:

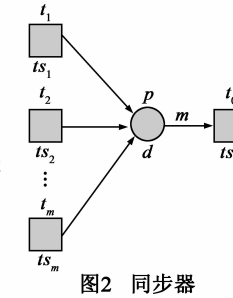

图2　同步器

定义 2　一个库所 p 称为一个同步器,当且仅当:

$$(\forall t \in {}^\cdot p: Q(t, p) = 1) \wedge (\exists t_0: p^\cdot = \{t_0\} \wedge Q(p, t_0) = |{}^\cdot p|)$$

其中,“$|\ \ |$”表示集合的元素数目.图 2 中的 p 是同步器的一个实例,其中 $ts_i (0 \leqslant i \leqslant m)$是 $Ts(t_i)$的函数值, d

是 $D(p)$的函数值.

同步器 p 的变迁规则是:变迁 t_0 在标识 M 下有发生权,当且仅当:

$$(\forall t\in{}^{\cdot}p: t \text{ 已经发生}) \wedge M(p)\geqslant Q(p,t_0)$$

3.2 Supply-Net 的四种模式

在建立 Supply-Net 之前,需要首先定义 Supply-Net 的几种基本模式.本文从供应链中典型的过程中(从原材料到最终用户使用成品的整个过程),总结出以下四种模式.

(1) Shipping 模式:描述单个或批量物品的移动、运输.如图 3 所示,p_1 代表库存状态,d_1 是库存持续时间,t_1 代表出库,p_2 代表运输途中,t_2 代表接收入库.

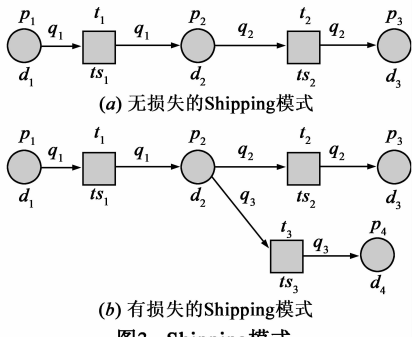

图3 Shipping模式

在无损失 Shipping 模式中(见图 3(a)),$q_1 = q_2$;在有损失的 Shipping 模式中(见图 3(b)),t_3 代表丢失或损坏,且 $q_1 = q_2 + q_3$,p_4 代表损失状态.

该模式的变迁规则与 P/T 系统类似,对模式中任一变迁 t,在 M 有发生权的条件是:

$$\forall p\in{}^{\cdot}t: M(p)\geqslant Q(p,t) \wedge \forall p\in t^{\cdot}: M(p)+Q(t,p)\leqslant K(p)$$

若 t 在 M 有发生权,t 发生后,网的状态记为 M',即 M 的后继为 M',则 M' 变为:

$$\forall p\in{}^{\cdot}t - t^{\cdot}: M'(p) = M(p) - Q(p,t)$$

$$\forall p\in t^{\cdot} - {}^{\cdot}t: M'(p) = M(p) + Q(t,p)$$

$$\forall p\notin {}^{\cdot}t\cup t^{\cdot}: M'(p) = M(p)$$

(2) Choosing Suppliers/Recipients 模式:描述同一种类(具有相同 ObjectClass)的物品来自多个供应商或发给多个分销商.图 4(a)中,对同一种原料或零件,可能有多个供应商,$p_{1i}(1\leqslant i\leqslant m)$代表第 i 个供应商发来的物品处于存储状态,t_i 代表搬运这些物品,p_2 代表这些物品就绪,可以进行下一步处理(如:包装/组装、解包装/拆卸、加工制造).图 4(b)中,对同一种类的物品,也可能有多个分销商,p_2 代表物品库存状态,$t_i(1\leqslant i\leqslant m)$代表向第 i 个分销商发出物品,p_{1i}代表向第 i 个分销商发出的物品处于运输状态.

该模式的变迁规则:对变迁 $t_i(1\leqslant i\leqslant m)$,在 M 有发生权的条件是:

在 Choosing Suppliers 模式中:

$$M(p_{1i})\geqslant Q(p_{1i},t_i) \wedge M(p_2) + Q(t_i,p_2)\leqslant K(p_2)$$

在 Choosing Recipients 模式中:

$$M(p_2)\geqslant Q(p_2,t_i) \wedge M(p_{1i}) + Q(t_i,p_{1i})\leqslant K(p_{1i})$$

若 t_i 在 M 有发生权,则 t_i 发生后,M 的后继 M' 为:

在 Choosing Suppliers 模式中:

$$M'(p_{1i}) = M(p_{1i}) - Q(p_{1i},t_i)$$

$$M'(p_2) = M(p_2) + Q(t_i,p_2)$$

在 Choosing Recipients 模式中:

$$M'(p_2) = M(p_2) - Q(p_2,t_i)$$

$$M'(p_{1i}) = M(p_{1i}) + Q(t_i,p_{1i})$$

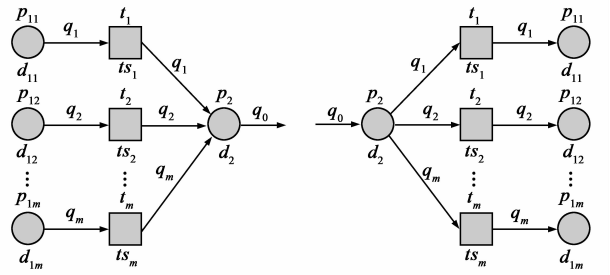

图4 Choosing Suppliers/Recipients模式

(3) Aggregating/Disaggregating 模式:描述将不同种类(ObjectClass 不同)的物品包装/组装为一个整体,或将一个整体解包装/拆卸为多种物品.图 5(a)中,$p_{1i}(1\leqslant i\leqslant m)$代表第 i 种被包装/组装的物品处于准备状态;t_i 代表将第 i 种物品包装/组装到整体中;$p2$ 代表一个同步器,等待被包装/组装的所有种类的物品凑齐,且 $K(p_2) = m$;t_0 代表将所有种类物品进行包装/组装;t_0 只输出一个 token,p_3 代表包装/组装后的状态.图 5(b)是物品的解包装/拆卸过程,p_2 代表被解包装/拆卸的物品处于就绪状态,t_2 代表将一个物品解包装/拆卸为若干种物品,p_{1i}代表第 i 种物品被解包装/拆卸出来后的状态.

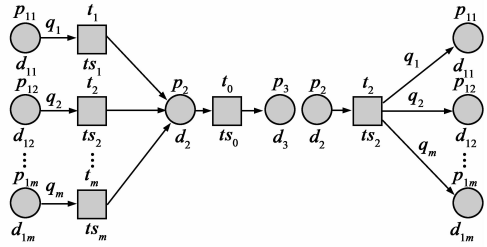

图5 Aggregating/Disaggregating模式

该模式的变迁规则如下:

在 Aggregating 模式中,对变迁 $t_i(1\leqslant i\leqslant m)$,在 M 有发生权的条件是:

$$M(p_{1i})\geqslant Q(p_{1i},t_i) \wedge M(p_2) + 1\leqslant m$$

若 t_i 在 M 有发生权,则 t_i 发生后,M 的后继 M' 为:

$$M'(p_{1i}) = M(p_{1i}) - Q(p_{1i},t_i)$$

$$M'(p_2) = M(p_2) + 1$$

对变迁 t_0,在 M 有发生权的条件是:

$$M(p_2) = m \wedge M(p_3) + 1\leqslant K(p_3)$$

若 t_0 在 M 有发生权,则 t_0 发生后,M 的后继 M' 为:

$$M'(p_2) = M(p_2) - m = 0$$
$$M'(p_3) = M(p_3) + 1$$

在 Disaggregating 中,对变迁 t_2,在 M 有发生权的条件是:

$$M(p_2) \geqslant 1 \wedge \forall p \in t_2^{\cdot}: M(p) + Q(t_2, p) \leqslant K(p)$$

若 t_2 在 M 有发生权,则 t_2 发生后,M 的后继 M' 为:

$$M'(p_2) = M(p_2) - 1$$
$$\forall p \in t_2^{\cdot}: M'(p) = M(p) + Q(t_2, p)$$

(4)Transforming 模式:描述将多种原材料物品加工制造成一种或多种新的物品.图 6 中,$p_{1i}(1 \leqslant i \leqslant m)$ 代表第 i 种原材料处于准备状态,t_i 代表将第 i 种原料进行预处理,p_2 代表一个同步器等待所有种类的原材料物品凑齐且 $K(p_2) = m$,t_0 代表将所有原材料物品进行加工制造,$p_{2j}(1 \leqslant j \leqslant n)$代表第 j 种新物品制成后的状态.

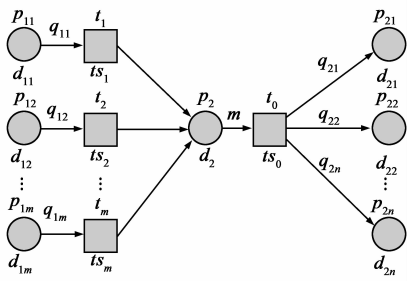

图6　Transforming模式

该模式的变迁规则如下:对变迁 $t_i(1 \leqslant i \leqslant m)$,在 M 有发生权的条件是:

$$M(p_{1i}) \geqslant Q(p_{1i}, t_i) \wedge M(p_2) + 1 \leqslant m$$

若 t_i 在 M 有发生权,则 t_i 发生后,M 的后继 M' 为:

$$M'(p_{1i}) = M(p_{1i}) - Q(p_{1i}, t_i)$$
$$M'(p_2) = M(p_2) + 1$$

对变迁 t_0,在 M 有发生权的条件是:

$$M(p_2) = m \wedge \forall p \in t_0^{\cdot}: M(p) + Q(t_0, p) \leqslant K(p)$$

若 t_0 在 M 有发生权,则 t_0 发生后,M 的后继 M' 为:

$$M'(p_2) = M(p_2) - m = 0$$
$$\forall p \in t_0^{\cdot}: M'(p) = M(p) + Q(t_0, p)$$

3.3　基于发现服务获得的数据集构建 Supply-Net

基于上述四种模式,下面给出 Supply-Net 的构建算法.算法思路是:对已按发生时间(*eventTime*)排序的由 RFID 发现服务获得的数据集中的每一个 EPCIS 事件,根据其事件类型对应的模式,生成不同的库所、变迁、有向弧和 K, Q, D, Ts, tp 等函数的映射.首先,该算法需要定义一个数据类型和一些全局变量:

[数据类型]

PRemark,描述库所的状态信息,如所含 token 代表的物品类别(Object Class)、这些物品所在的位置或组织机构,这些 token 进入该库所的时刻:

PRemark[*epcClass* : *ObjectClass*, *org*: *ManagerNumber*, *startTime*: *Date*]

[全局变量]

snet:所要构建的 Supply-Net;

pRemarkH:哈希表 $P \rightarrow PRemark$,记录每一个库所的状态信息;

epcCtoQuantity:哈希表 $ObjectClass \rightarrow N$(自然数),记录某一 EPCIS 事件中每类物品的数量.

算法"constructSupplyNet"是本文给出的 Supply-Net 构建算法,为了方便理解,我们将其拆成 4 段分别介绍.下面一段是处理 ObservationEvent 和 QuantityEvent 对应的 Shipping 模式(包括对损失的处理):首先找到给定事件的前一个库所,即该事件中涉及的物品所对应的 token 在当前 *snet* 中的最后一个库所,然后增加一个变迁代表该事件,增加一个库所代表变迁发生后的状态,添加有向弧连接新变迁和新库所.然后添加 Q, K, D, Ts, tp 的相应映射,具体步骤参见下面的算法描述.另外该算法还处理有损失的 Shipping 模式:以具有 SHIPPING 类型的当前库所为起点,添加新的变迁、库所和 Q, K, D, Ts, tp 映射.

```
constructSupplyNet(answer: List < Result > )
//初始化一个 Supply-Net 为空
snet (P, T; F, K, Q, D, Ts, Type, tp): Supply-Net: = Ø
//记录每个 p 的状态
pRemarkH: HashTable < p: Place, remark: PRemark > : = Ø
epcCtoQuantity: HashTable < epcClass: ObjectClass, q: int > : = Ø
epcC: ObjectClass: = null//表示单个或一批物品的 ObjectClass
for each e: EPCISEvent in answer//"answer"已经按照 eventTime 排序
  /* 处理 Shipping 模式 */
  if typeof(e) = ObservationEvent | QuantityEvent
    if typeof(e) = ObservationEvent
      epcC : = objectClassOf(e.epcList)//epcList 具有相同的 Object-
          Class
    else epcC: = e.epcClass
    //找到这些具有相同 ObjectClass 的物品当前所在的最后一个
      库所
    p: = findLastPlace(epcC, e)
    P: = P∪p' //添加一个新库所
    T: = T∪t //添加一个新变迁
    F: = F∪(p, t)∪(t, p') //添加有向弧
    //以下 6 行是添加 Q, K, Ts, tp 的相应映射
    if typeof(e) = ObservationEvent
        Q((p, t)): = Q((t, p')): = e.epcList.size()
    else Q((p, t)): = Q((t, p')): = e.quantity
    K(p'): = Q((t, p'))
    Ts(t): = e.eventTime
    tp(t): = "MOVE"
    if e.sendTo≠null //处理发货
      tp(p): = "STORING"
```

```
    tp(p′):="SHIPPING"
    //记录 p′的状态信息
    pRemarkH[p′]:=[epcC,e.sendTo,e.eventTime]
  else if e.reciveFrom≠null //处理收货
    tp(p):="SHIPPING"
    tp(p′):="STORING"
    //记录 p′的状态信息
    pRemarkH[p′]:=[epcC,e.org,e.eventTime]
  /*处理有损失的情况*/
  if tp(p)="SHIPPING" && (∃t′,Q((t′,p))>Q((p,t)))
    P:=P∪p*//添加一个新库所 p*表示损失状态
    T:=T∪t*//添加一个新变迁 t*表示发生损失
    F:=F∪(p,t*)∪(t*,p*) //添加有向弧
    //以下6行是添加 Q,K,D,Ts,tp 的相应映射
    Q((p,t*)):=Q((t*,p*)):=Q((t′,p))-Q((p,t)
    K(p*):=Q((t*,p*))
    D(p*):=0 //0表示"无法确定"
    Ts(t*):=null
    tp(t*):="LOSE"
    tp(p*):="LOST"
```

下面一段是处理 AggregationEvent 对应的 Aggregating/Disaggregating 模式:首先添加事件对应的变迁;对 Aggregating 模式,则要添加同步器库所及其 K,D,tp 映射,然后处理每种被包装/组装的物品,找到该类物品所处的最后一个库所,添加相应的 PRE_AGGREGATE 类型的变迁和相应的有向弧及 Q,Ts,tp 映射;接着添加新库所表示所有种类的物品被包装/组装后的状态,对事件对应的变迁添加其输入/输出的有向弧和 K,Q,tp 映射.对 Disaggregating 模式,则要找到解包装/拆卸出来的物品所处的最后库所,添加有向弧连接到该事件对应的变迁,然后添加每类解包装/拆卸出来的物品对应的库所和有向弧及 K, Q,tp 映射.

```
/*处理 Aggregating/Disaggregating 模式*/
if typeof(e)=AggregationEvent
  T:=T∪t //添加一个新变迁 t 表示该事件
  Ts(t):=e.eventTime
  //获得 parentID 中的 ObjectClass
  epcC:=objectClassOf(e.parentID)
  //获得每类被包装/组装的物品的数量
  epcCtoQuantity:=groupByObjectClass(e.childEPCs)
  if e.action="ADD"//处理 Aggregating 模式
    P:=P∪p*//添加一个库所作为同步器
    tp(p*):="PRE_AGGREGATED"//设置同步器的类型
    D(p*):=0
    //同步器的容量是被包装/组装的物品的种类数
    K(p*):=epcCtoQuantity.size()
    //处理每类被包装/组装的物品
    for each entry in epcCtoQuantity
      //找到具有相同 epcClass 的物品当前所处的最后一个库所
      p:=findLastPlace(entry.epcClass,e)
      T:=T∪t′//添加一个变迁表示该类物品被包装/组装
      F:=F∪(p,t′)∪(t′,p*)//添加 t′的输入/输出有向弧
      Q((p,t′)):=entry.q
      Q((t′,p*)):=1
      Ts(t′):=e.eventTime
      tp(p):="PREPARED"
      tp(t′):="PRE_AGGREGATE"
    tp(t):="AGGREGATE"
    P:=P∪p′ //添加一个库所表示所有种类的物品被包装/
            组装后的状态
    F:=F∪(p*,t)∪(t,p′)//添加 t 的输入/输出有向弧
    Q((p*,t)):=K(p*)
    Q((t,p′)):=1
    tp(p′):="PROCESSED"
    pRemarkH[p′]:=[epcC,e.org,e.eventTime]
  if e.action="DELETE"//处理 Disaggregating 模式
    tp(t):="DISAGGREGATE"
    p:=findLastPlace(epcC,e)
    F:=F∪(p,t)
    Q((p,t)):=1
    tp(p):="PREPARED"
    //以下为每类解包装/拆卸出来的物品添加对应的库所和
      有向弧
    for each entry in epcCtoQuantity
      P:=P∪p′ //添加新库所表示该类解包装/拆卸出来的
              物品状态
      F:=F∪(t,p′)
      Q((t,p′)):=entry.q
      tp(p′):="PROCESSED"
      //记录 p′的状态
      pRemarkH[p′]:=[entry.epcClass,e.org,e.eventTime]
```

下面一段是处理 TransformEvent 对应的 Transforming 模式:该模式可以看做 Aggregating 和 Disaggregating 模式的组合,生成相关变迁和库所的步骤与上一段类似,只是这些变迁和库所的 tp 映射不同,语义不同.

```
/*处理 Transforming 模式*/
if typeof(e)=TransformEvent
  T:=T∪t //添加新变迁 t 表示该事件
  Ts(t):=e.eventTime
  tp(t):="TRANSFORM"
  P:=P∪p*//添加一个库所作为同步器
  tp(p*):="PRE_TRANSFORMED"//设置同步器的类型
  D(p*):=0
  //获得每类被加工制造的原材料物品的数量
  epcCtoQuantity:=groupByObjectClass(e.materials)
  //同步器的容量是被加工制造的原材料物品的种类数
  K(p*):=epcCtoQuantity.size()
  for each entry in epcCtoQuantity //处理每类原材料物品
    //找到具有相同 epcClass 的物品当前所处的最后一个库所
```

```
    p: = findLastPlace(entry.epcClass, e)
    //添加一个变迁表示该类原材料物品被加工制造
    T: = T∪t'
    F: = F∪(p,t')∪(t',p*) //添加 t'的输入/输出有向弧
    Q((p,t')): = entry.q
    Q((t',p*)): = 1
    Ts(t'): = e.eventTime
    tp(p): = "PREPARED"
    tp(t'): = "PRE_TRANSFORM"
  F: = F∪(p*,t) //添加 t 的输入有向弧
  Q((p*,t)): = K(p*)
  //对加工制成的新物品,获得每类物品的数量
  epcCtoQuantity: = groupByObjectClass(e.products)
  for each entry in epcCtoQuantity //处理每类新物品
  添加一个库所表示制造出来的这类新物品的状态
    P: = P∪p' //
    F: = F∪(t,p')
    Q((t,p')): = entry.q
    tp(p'): = "PROCESSED"
    //记录 p'的状态
    pRemarkH[p']: = [entry.epcClass, e.org, e.eventTime]
return snet
```

下面一段是上述三段算法用到的子过程"findLastPlace",它根据 pRemarkH 中各库所的状态信息,找到给定事件的前一个库所(即事件中的某类物品所在的最后库所),判定条件是:若一个库所与给定事件包含的物品类别相同、组织机构(企业)相匹配、库所类型符合且满足时间先后顺序,则该库所就是"最后库所",即该事件的前一个库所.若没有库所满足上述条件,则新建一个库所作为事件的前一个库所.而且,如果遇到 AggregationEvent 和 TransformEvent,需要额外判断有无 Choosing Suppliers 模式存在,并作相应处理.该子过程的详细算法如下:

```
findLastPlace(epcClass:ObjectClass, e:EPCISEvent)
  if typeof(e) = ObservationEvent|QuantityEvent
    if e.sendTo≠null //对"发货"事件,线性查找获得前一个库所
      for each entry<p,remark> in pRemarkH
        //前一个库所的状态必须满足以下条件
        if remark.epcClass = epcClass∧remark.org = e.org
          ∧tp(p) = STORING|PROCESSED∧remark.startTime < e.
                   eventTime
          D(p): = e.eventTime-remark.startTime //计算 D 映射值
          return p      else if e.reveiveFrom≠null
    //对"收货"事件,线性查找获得前一个库所
      for each entry<p,remark> in pRemarkH
        //前一个库所的状态必须满足以下条件
        if remark.epcClass = epcClass∧remark.org = e.reveiveFrom
          ∧tp(p) = "SHIPPING"∧remark.startTime < e.eventTime
          D(p): = e.eventTime-remark.startTime //计算 D 映射值
          return p
  /*若没有满足条件的库所,则新建一个库所作为该事件的前一个库所并初始化 K,D 映射*/
  P: = P∪p //添加新库所 p
  pRemarkH[p]: = [epcClass,e.org,null] //记录 p 的状态信息
  K(p): = ω
  D(p): = 0
  //处理 AggregationEvent 和 TransformEvent 中的 Choosing Suppliers 模式
  if typeof(e) = AggregationEvent|TransformEvent
    //线性查找所有可能的来自不同供应商的同类物品当前所在的库所
    for each entry<p',remark> in pRemarkH
      //这些库所的状态必须满足以下条件
      if remark.epcClass = epcClass∧remark.org = e.org
        ∧tp(p') = "STORING"∧remark.startTime < e.eventTime
        T: = T∪t' /*添加一个额外的变迁 t'表示将物品移
                    动到具有 PREPARED 状态的库所*/
        F: = F∪(p',t')∪(t',p)
        Q((p',t')): = Q((t',p)): = K(p')
        Ts(t'): = null
        tp(t'): = "MOVE"
return p
```

3.4　利用 Supply-Net 建模的实例

下面用一个实例来说明利用 Supply-Net 对供应链建模的结果.这个实例就是 2.1 节给出的实例.我们用表 1 给出的数据集实例作为输入,采用 Supply-Net 构建算法,最后构建 Supply-Net 实例如图 7 所示.在图 7 中,p_{1A}是原料 A 的供应商的库存状态,通过 t_{1A}发出 3000 个 A,p_{2A}是运输状态,t_{2A}表示 D 的制造商接收到 A,A 进入制造商的库存状态 p_{3A},准备生产时将 A 搬运(t_{3A})至车间,进入准备状态 p_{4A};与上述过程相似,路径 $p_{1B}\rightarrow p_{7B}$和 $p_{4B}\rightarrow p_{7B}$分别描述了两个 B 的供应商向制造商发货和制造商接收的过程,在 p_{7B}形成一个 Choosing Suppliers 模式;t_1 表示加工制造,构成一个 Transforming 模式,3000 个 A 和 5000 个 B 按照 3:5 的比例加工制造出 2000 个 C,此时 t_1 发生了 1000 次,进入 C 的已处理状态(P_{1C});t_2 表示包装/组装,构成一个 Aggregating 模式,每 10 个 C 包装/组装成 1 个 D,得到 200 个 D,即 t_2 发生了 200 次,进入 D 的已处理状态(P_{1D});然后 D 经过搬运(t_{1D})至 D 的库存状态 P_{2D};最后,路径 $P_{2D}\rightarrow P_{4D}$和 $P_{2D}\rightarrow P_{6D}$分别描述了向两个 D 的分销商发货和零售商接收的过程,在 P_{2D}形成一个 Choosing Recipients 模式.

在有损失的情况下(例如经过 t_3 损失了 150 个 A),则在 t_{7B}中的 5000 个 B 只有 4750 个被消耗,t_1 只发生了 950 次,生成 1900 个 C,进而 t_2 只能发生 190 次,组装成 190 个 D,然后再分发给两个不同的分销商.

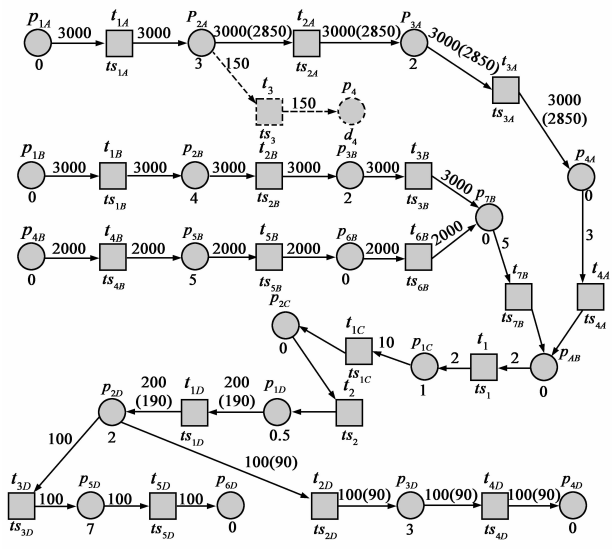

图7 利用Supply-Net建模的实例

注:( )中的数字表示运输过程有损失的情况下,受此影响的弧的权值,虚线箭头和虚线图元表示物品发生损失

3.5 Supply-Net 模型的实际应用能力

首先,该模型可应用于实际供应链的以下情况:(1)在实际产品召回应用中,某零件的生产商需要掌握经销该零件、用该零件生产出成品、经销成品的所有企业.Supply-Net 模型能够根据该零件及其成品的移动记录,刻画出零件的各级经销商、成品制造商和成品的各级经销商,正如 3.4 的建模实例;(2)在优化供应链方面,利用 Supply-Net 在不同时间对同一供应链建模,能够及时发现供应链拓扑结构的动态变化,方便企业进行供应链的优劣、可靠性分析和对出错节点的改良,例如 3.4 中对有损失情况的及时发现;(3)在降低库存成本方面,Supply-Net 模型提供了物流的准确量化信息,企业能够借此优化采购、供应策略,以达到库存成本的最小化;同时这些量化信息提高了预测的准确性,缩短物品的周转时间;(4)另外,Supply-Net 模型可以根据特殊的应用进行扩展,例如在物品质量保证方面,尤其对于易腐物品,可在 Supply-Net 模型中的增加库所对应的温度信息,这样就能够直观刻画哪些流通环节的温度高于给定阈值而发生质量问题,以便及时修正.

其次,分析模型与所刻画供应链环境的一致性.本文涉及的供应链环境,限定为单个或批量物品的移动,物品的包装/组装与解包装/拆卸,以及将原材料物品加工制成新物品.这是对供应链最常见业务过程的分类抽象,在该抽象层次上的 Supply-Net 模型与对应的供应链应用是一致的,但不排除模型与更微观层面的供应链应用具有不尽一致的情况.比如,将原材料物品加工制成新物品的过程可能包括若干工序,但这不属于 Supply-Net 模型所要刻画的粒度范围,而且如此细节的业务过程信息因涉及企业隐私,通常是无法通过 RFID 发现服务获取的.当然,本文给出的 Supply-Net 的四种模式,只是覆盖到本文限定的供应链环境,它们刻画了实际供应链应用中最主要的业务过程.如果需要描述其它业务过程,那么就要增加新模式,或利用已有模式的组合表示新模式.

最后,介绍模型的用途和效果.模型的主要用途,一是为复杂的供应链过程提供了一种可量化的、信息丰富且带有语义的可视化建模工具.随着生产和物流规模的扩大,供应链将涉及越来越多的属于不同地区、国家甚至大洲的企业,为了跟踪追溯物品、保证质量,企业需要精确定位零件在何处生产、何处组装,产品在何处存储、如何分销.利用 Supply-Net 建模能够快速、及时、可视的掌握这些信息.二是多数企业很难了解到其直接上下游合作伙伴之外的客户和供应商的信息,而通过 Supply-Net 建模就能够把握供应链的全貌,这对供应链管理的研究和实践具有重要价值,企业可以从模型的丰富信息中进一步分析挖掘出商业智能.目前,我们基于 Petri 网建模工具实现了 Supply-Net 建模系统,并与 RFID 发现服务系统进行了集成,用户通过对发现服务提交查询请求,就可以在返回的数据集的基础上自动建立可视化模型.这些数据集的生成规则同 4.2 节,并且随时间而有略微的变化,比如,物品原来是通过节点 A 到达 B,过一段时间会绕开 A 而经过 C 到达 B;另外物品在库所的停留时间、物品数量也会发生变化.Supply-Net 建模系统能够对比不同时间对同一查询建模的拓扑结果变化,根据预先设定的单位收益/成本参数,给出模型变化前后的收益/成本对比,并向用户建议优化的拓扑结构和参数配置.系统还能自动报告发生物品损失的环节.由于篇幅的限制,关于该系统的详细功能和仿真分析将在下一篇论文中详述.

4 算法分析和实验

4.1 算法的时间复杂度分析

为了检验 Supply-Net 构建算法的效率,下面首先对“constructSupplyNet”算法进行时间复杂度分析.该算法的时间复杂度主要取决于事件的数目(设为 n)和子过程 findLastPlace 中的线性查找循环次数(设为 p).现根据不同的事件类型分析如下:

(1)对每个 ObservationEvent 和 QuantityEvent,都要执行 findLastPlace,所以需要时间 pn.

(2)对每个 AggregationEvent,如果是处理“包装/组装”事件,则对每种被包装/组装的物品,findLastPlace 都要执行一次,所以耗时为 Apn,其中 A 为被包装/组装的物品种类数;如果是处理“解包装/拆卸”事件,则 findLastPlace 只需要执行一次,然后分别处理每种解包装/拆卸出来的物品,所以耗时为 $(p+A)n$.

(3) 对每个 TransformEvent,需要对每种原材料物品,分别执行 findLastPlace 一次,然后分别处理每种加工制造出来的新物品,设原材料物品的种类数为 B,新物品的种类数为 C,则耗时为 $(Bp + C)n$.

以上三种情况相比较,其中最大耗时为 Apn 或 $(Bp + C)n$,不妨假定最大耗时 T_{max} 为:

$$T_{max} = (Bp + C)n \quad (1)$$

因为 p 是 *pRemarkH* 中的库所数量,p 是随着算法的执行而不断增大的,所以很难计算.但是我们可以定量分析在最坏情况下 p 与 n 的关系:

(1)对 ObservationEvent 和 QuantityEvent,每处理一个事件,p 就增加 1(在无损失的 Shipping 模式下)或 2(在有损失的 Shipping 模式下),所以在最坏情况下,$p = 2n$.

(2)对 AggregationEvent,每处理一个"包装/组装"事件,需要先对每种被包装/组装的物品检查并处理 Choosing Suppliers 模式,会分别增加 1 个库所,还要增加一个同步器库所,完成包装/组装后,又产生 1 个库所,故 $p = (A + 2)n$;每处理一个"解包装/拆卸"事件,同样的分析可得,$p = (1 + A)n$.

(3)对 TransformEvent,每处理这样一个事件,可看作一个"包装/组装"事件和一个"解包装/拆卸"事件的连接组合,故 $p = (B + 1 + C)n$.

比较以上三种情况,我们假定最坏情况下,p 和 n 的关系为:

$$p = (B + 1 + C)n \quad (2)$$

将式(2)代入式(1)可得,整个算法的时间复杂度为:

$$O((B(B + 1 + C)n + C)n) = O(B(B + C + 1)n^2 + Cn) \quad (3)$$

在实际意义上,把 B, C 视为常量或远远小于 n(即 $B, C \ll n$)是合理的,则时间复杂度变为 $O(n^2)$.

4.2 实验一:时间复杂度

下面用实验来验证上述算法时间复杂度分析的结论.我们给定 A, B, C 的 3 种取值($A = 1, B = 2, C = 1$ 或 $A = 5, B = 5, C = 1$ 或 $A = 5, B = 5, C = 3$),在每种取值下,n 依次取 1000,2000,3000,…,19000,20000 个 EPCIS 事件,在 n 的每种取值下,四种类型的事件数量各占 25%.我们的实验环境配置包括 Pentium(R) Dual-Core 2.5GHz 处理器、3GB RAM.因为 n 有 20 种取值,这样我们总共进行了 3×20 组实验,每组实验的执行时间是经 5 次测试获得的平均值,实验结果数据如图 8 所示.该实验表明,随着 n(即事件数量规模)的增大,算法的平均执行时间呈现出多项式级别的增长态势;随着 A, B, C 取值的增大,所需执行时间也逐渐增加,这是因为 A, B, C 取值越大,说明 EPCIS 事件中的 AggregationEvent 和 TransformEvent 涉及的物品种类越多,供应链越复杂,导致构造算法的时间增加.我们还利用"最小二乘拟合"方法来近似计算 n 的指数 x,x 的调整步长精度为 0.01,对图 8 中的 3 组数据点依次经过曲线拟合得到 3 条曲线,按自底向上顺序,x 分别为 1.82,1.89,1.93 是最合适的取值,则 3 条曲线所示的时间复杂度为均小于 $O(n^2)$.可见 Supply-Net 的构建算法的时间复杂度小于 $O(n^2)$,由此证明该算法具有较高的时间效率,可实际应用.

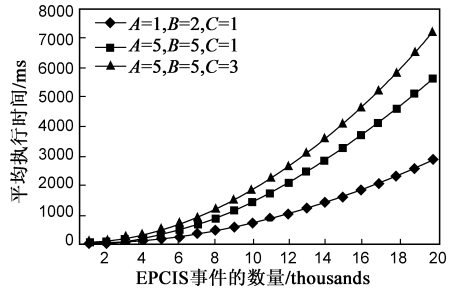

图8 Supply-Net构建算法的平均执行时间

4.3 算法的空间复杂度分析

算法的空间复杂度主要取决于 *snet* 和 *pRemarkH* 两个全局变量占用的空间.很明显 *pRemarkH* 的空间随着库所总数量的增加而线性增长. *snet* 是一个九元组 $(P, T; F, K, Q, D, Ts, Type, tp)$,其中 K, D 的空间与 P 线性相关,Q 的空间与 F 线性相关,Ts 的空间与 T 线性相关,$Type$ 的空间大小为常数,tp 的空间与 P、T 空间总和线性相关,因此只要分析 P, T, F 与 EPCIS 事件规模 n 之间的关系即可,而 P 与 n 的关系在 4.1 中已经讨论过,现只讨论 T, F 与 n 的关系:

(1)对 ObservationEvent 和 QuantityEvent,每处理一个事件,T 就增加 1(在无损失的 Shipping 模式下)或 2(在有损失的 Shipping 模式下),所以 $T = n$ 或 $2n$.对于 F,每处理一个事件,F 的增量是 T 的增量的 2 倍,故 $F = 2n$ 或 $4n$.

(2)对 AggregationEvent,每处理一个"包装/组装"事件,需要先对每种被包装/组装的物品检查并处理 Choosing Suppliers 模式,会分别增加 1 个变迁,还要增加 1 个表示包装/组装的变迁,故 $T = (A + 1)n$,A 的含义与 4.1 相同;F 的增量是 T 的增量的 2 倍,故 $F = 2(A + 1)n$.每处理一个"解包装/拆卸"事件,仅增加 1 个变迁,故 $T = n$;而 F 的增量是 $1 + A$,故 $F = (1 + A)n$.

(3)对 TransformEvent,每处理这样一个事件,可看作一个"包装/组装"事件和一个"解包装/拆卸"事件的连接组合,故 T 的增量是 $B + 1$,F 的增量是 $B + 1 + C$(B, C 的含义与 4.1 相同),故 $T = (B + 1)n$,$F = (B + 1 + C)n$.

综上所述,在 A, B, C 视为常量或远远小于 n 的情况下 T, F 与 n 分别呈线性关系,因为 P 与 n 也呈线性

关系,所以 K, Q, D, Ts, tp 也分别与 n 呈线性关系.

所以,整个算法的空间复杂度为 $O(n)$.

4.4 实验二:空间复杂度

在本实验中,我们给定 A, B, C 的 3 种取值、n 的 20 种取值均与 4.2 相同,四种类型的事件数量也是各占 25%.总共进行了 3×20 组实验,每组实验的空间占用量是经 5 次测试获得的平均值,实验结果数据如图 9 所示.该实验表明,随着 n(即事件数量规模)的增大,算法的空间占用量基本上线性增长,这与空间复杂度分析的结果是一致的.

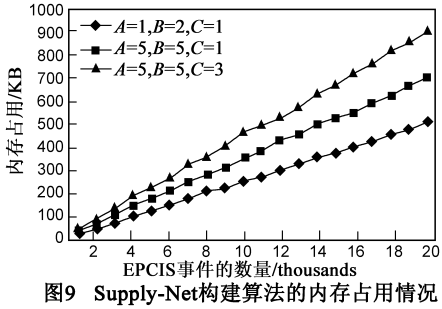

图9 Supply-Net构建算法的内存占用情况

5 结论

在采用 RFID 应用的供应链环境中,为了便于对 RFID 发现服务获得的数据集进行可视化分析,本文提出了一种供应链建模技术,包括:(1)规约了由 RFID 发现服务获得的供应链的数据集,该数据集覆盖了供应链 RFID 业务中最典型的四种事件数据类型,除了对单品或批量物品的位置移动的描述,还包括对物品进行包装/组装、解包装/拆卸的描述,尤其是增加了 TransformEvent 类型来描述由原材料加工制造出新产品这类事件;(2)基于 P/T 系统定义了一种供应链建模工具 Supply-Net,增强了建模工具的表达能力,如变迁发生时刻、库所停留时间、库所和变迁的类型等,其表达能力能够满足供应链中 RFID 的业务需求;(3)给出利用 Supply-Net 对供应链的数据集建模的过程(算法),通过数学分析和实验验证,算法的最大时间复杂度为 $O(n^2)$,空间复杂度为 $O(n)$,复杂度较低,适合实际应用.

这种供应链建模技术目前已应用于酒类供应链中,对原材料的加工,对酒类产品从生产商到不同级别的分销商再到零售商的流动,进行建模.该应用的效果表明,这种供应链建模技术对供应链中 RFID 发现服务获得的数据集具有足够的表达能力和较高的建模效率,以进行可视化分析.下一步的工作主要包括建模工具的完善和分析方法的研究.首先,进一步完善 Supply-Net,使之能够对更加复杂、更加精细的实际供应链进行建模.其次,供应链建模的最终目的是让供应链上的各企业分析和充分利用发现服务获得的数据集,从而改进自己的关键业务环节甚至供应链结构,因此将来还要研究分析、验证和优化 Supply-Net 的方法.这些工作将作为本文的重要补充,并将单独成文.

参考文献:

[1] K Vitasek. Logistics Terms and Glossary[S]. Supply Chain Visions, Bellevue, WA, Oct 2003.

[2] EPCglobal. EPCglobal Tag Data Standards Version 1.4[S/OL]. http://www.epcglobalinc.org/standards/tds/tds_1_4-standard-20080611.pdf, 2008-06.

[3] EPCglobal. The EPCglobal architecture framework[S/OL]. http://www.epcglobalinc.org/standards/architecture/architecture_1_2-framework-20070910.pdf, 2007-09.

[4] R Agrawal, A Cheung, K Kailing, S Schönauer. Towards traceability across sovereign, distributed RFID databases[A]. Desai B C. Proc. of the 10th Int. Database Engineering & Applications Symposium[C]. Delhi, India, 2006. 174 – 184.

[5] H H Bi, D K J Lin. RFID-enabled discovery of supply networks [J]. IEEE Transactions on Engineering Management, 2009, 56 (1): 129 – 141.

[6] N R Srinivasa Raghavan, N Vswanadham. Performance analysis of supply chain networks using petri nets[A]. Proc. of the 38th Conf. on Decision & Control[C]. Phoenix, USA, 1999. 57 – 62.

[7] A Desrochers, M P Fanti. A supply chain model using complex-valued token petri nets[A]. IEEE Int. Conf. on Robotics & Automation[C]. Barcelona, Spain, 2005. 3709 – 3714.

[8] H Chen, K Labadi, L Amodeo. Batch deterministic and stochastic petri nets: a tool for modeling and performance evaluation of supply chain[A]. IEEE Int. Conf. on Robotics & Automation [C]. Washington D.C., USA, 2002. 78 – 83.

[9] EPCglobal. EPC information services (EPCIS) version 1.0.1 specification[S/OL]. http://www.epcglobalinc.org/standards/ epcis/epcis_1_0_1-standard-20070921.pdf, 2007-09.

[10] Yuan Chong-yi. Principals and Application of Petri Nets[M]. Beijing: Publishing House of Electronics Industry, 2005. (in Chinese)

作者简介:

李信鹏 男,1982 年出生,北京大学博士研究生,主要研究方向为软件工程和 RFID 相关技术. E-mail: lixp05@sei.pku.edu.cn