CAX系统中智能Undo/Redo操作的研究

刘云华,何 苗,罗年猛
LIU Yunhua, HE Miao, LUO Nianmeng
华中科技大学 国家CAD支撑软件工程技术研究中心,武汉 430074
National CAD Support Software Engineering Research Center, Huazhong University of Science and Technology, Wuhan 430074, China

LIU Yunhua, HE Miao, LUO Nianmeng. Reserch of Undo/Redo operations in CAX systems. Computer Engineering and Applications, 2012, 48(4): 61-64.

Abstract: In CAX system, for complex operation and related operation, existing Undo/Redo mechanism is difficult to ensure the Undo/Redo operation semantic consistency, avoid the destruction of graphic data, thus it leads to realize specific methods of Undo/Redo and further development difficultly. This paper analyzes the characteristics of complex and related operation in CAX systems, discusses the atomicity and consistency of transaction and the reactor mechanism, introduces the transaction-based to ensure the Undo/Redo operation semantic consistency and the reactor mechanism to reduce the relationship and complexity of operations as the resolution method, and describes the concrete implementation process and main algorithms. A case analysis in large-scale CAD software shows the effectiveness of the strategy.

Key words: Undo/Redo; transaction; reactor; consistency maintenance; related operations

摘 要: CAX系统中,对于复杂操作和关联操作,现有Undo/Redo机制很难保证Undo/Redo操作语义一致性,避免破坏图形数据,从而导致实现具体的Undo和Redo方法困难、二次开发难度大的问题。分析了CAX复杂操作和关联操作的特点,讨论了Undo操作中事务的原子性和一致性以及触发器机制,提出了基于事务保证操作语义一致性和基于触发器机制降低操作复杂度和关联度的解决策略,描述了具体的执行流程及主要算法。在大型CAD软件开发中实例分析证明了该方法的有效性。

关键词: Undo/Redo; 事务; 触发器; 一致性维护; 关联操作
DOI:10.3778/j.issn.1002-8331.2012.04.018 **文章编号:** 1002-8331(2012)04-0061-04 **文献标识码:** A **中图分类号:** TP311

1 引言

Undo/Redo操作是CAX系统的一个重要功能,极大地方便了用户操作,提高了绘图的效率。

目前主流的商业CAD软件中,Undo/Redo的实现可归结为两种原理:检查点(Check Point)和命令模式(Command Pattern)[1]。

检查点方法的实现过程是记录每次操作前后的文档内容并作为临时文件保存于内存中,其优点是响应速度快,适合处理小规模数据,所以经常用于一些中、小数据量的CAD软件。

命令模式方式,即将一个请求封装为一个对象,从而可用不同的请求对对象进行参数化;对请求排队或记录请求日志,以及支持可撤销的操作。命令模式的典型结构如图1所示。

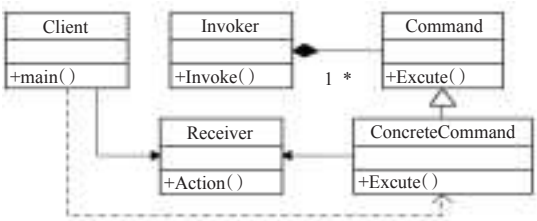


图1 Command模式结构图

图1中Client代表编辑器对象,管理各图形元素;Invoker代表Undo/Redo操作的管理类,管理各种Undo/Redo操作[2]。每一个具体的请求都对应有具体的Command子类实现,完成请求对象的参数化,从而实现操作的撤销和重做。

命令模式实现相对复杂,资源消耗小、运行速度快,适用于各种规模的CAD软件,尤其对海量数据的CAD软件系统,使用该原理的实现非常有效。

2 当前Undo/Redo机制分析

检查点方法采用的是文件备份方式,每次操作,文件就整体备份,造成冗余数据过多,资源消耗过大的问题,命令模式方式中,Command对象又分为两种方式,一种是基于操作[3-4],一种是基于图形对象[5]。

对于前者,每一次操作,在对图形对象构成的数据集合进行操作的同时,还要把相应操作的命令名称和相关数据进行记录和管理[6]。执行的命令被存储在一个历史列表中,可通过向后和向前遍历这一列表并分别调用Undo和Redo来实现步数不限的“取消”和“重做”。这种方式适用于通用操作批量修改图形对象时的情况,具有数据量小的特点,但需要由具体的操作完成具体的Undo和Redo实现,且可扩展性及二次开发性不好。另外,基于操作的Command模式中,每一个操作即一个Command对象,需要为每一个Command编写Redo()方法和一个Undo()方法。而在一个复杂的CAX系统中,拥有很多不同的操作,导致工作量大,二次开发困难,而开发中如果要进行功能的扩充和更改,则要变动相应的Undo()和Redo()方法,带来了很高的维护成本和开发难度。

对于后者,在每一次操作时,把修改的对象作为一个集合进行记录,封装为一个Command对象,由一个专门类来进行

基金项目: 国家自然科学基金(No.50275060,60573178);国家高技术研究发展计划(No.2002AA4Z3370)。
作者简介: 刘云华(1974—),男,博士,副教授,主研领域为三维轻量化、CAD&CG;何苗(1984—),男,硕士研究生;罗年猛(1972—),男,博士,讲师。
收稿日期: 2011-08-29;**修回日期:** 2011-11-22

管理,Command的Undo操作可在实施操作前将状态存储起来,在取消操作时这个状态用来消除该操作的影响。这种方式可实现与具体的操作无关,只与构建的Command类型(分为新增、修改、删除三种类型)有关。但是这种方式在处理多次修改的对象、有关联关系的对象以及嵌套操作时,存在着同一对象被多次添加、封装Command信息不完整等问题。

对于关联操作,无论是基于操作还是基于图形对象构建Undo/Redo的一个Command,都要求保证Undo/Redo操作语义一致性,避免破坏图形数据。通用的做法是将关联操作分解为原子操作。每一个原子操作分别提交一个Undo/Redo对象。在设计Redo()和Undo()方法时,把整个方法封装在事务中,依赖事务保证Undo()的原子操作要么都执行,要么都不执行。

这种处理方式在支持Undo操作时会产生如下问题:

(1)在实际开发中,除非是修改数据库中的数据,否则一般的系统并没有提供事务的支持。因此很难保证Redo()方法的原子性。

(2)对于多目标操作,如果执行一次Undo操作,其意愿是一次性地撤销对多个目标实体的处理效果,而非最后一个被处理的目标实体[1]。但是这样的语义一致性很难得到保证。

(3)如果通过标记Undo/Redo链表中起止位置来维持关联操作的意愿一致,则对于关联实体,又会带来下面的问题:关联的对象在一次Undo/Redo中被分为几步执行,破坏了事务的原子性,导致关联对象之间出现数据不一致现象。

3 基于事务和触发器机制的原理

针对上述问题,现提出基于事务和触发器的Undo/Redo解决策略。

在该策略中,首先引入事务(Transaction)的概念,事务:一次Undo/Redo操作的基本单位,维护Undo/Redo操作的语义一致性,由一个操作中发生变更的所有数据(包括图形对象和操作)组成。

事务的构建则由标记策略来提供支持,标记:由对象所拥有,记录对象在操作过程中的变更过程和状态。在一个操作开始时,采用标记持续记录发生变更的数据,而在操作结束时,根据标记按照一定的规则对变更数据集进行整理,封装为一个事务,提交后由专门的事务管理器负责管理。

拥有共享数据的对象称为关联对象,对于关联对象的操作称为关联操作。关联操作的存在会引发事务的嵌套构建问题,针对此问题的解决,引入了触发器机制,触发器:一种特殊的消息机制,其内部记录了关联对象之间的关联关系。通过事件来触发,从而引发另一事件的执行。利用触发器可以减少关联操作之间的关联度,避免操作的嵌套调用,从而避免了事务的嵌套构建。

基于事务和触发器机制的Undo/Redo机制的模型图如图2所示。

在模型图中,标记策略实时记录着对象的变更状态;而利用触发器则减少了外部操作之间的嵌套,从而保证事务构建时对象之间的平行关系;最后通过提交一个完整的事务,保证事务的一致性和原子性,从而保证Undo/Redo操作的语义一致性。

对于事务的构建,一个最主要的原则便是忽略对象变更的中间状态,确定对象最终的变更情况。采用标记的策略,记录对象被操作的过程,则可以达到此目的。

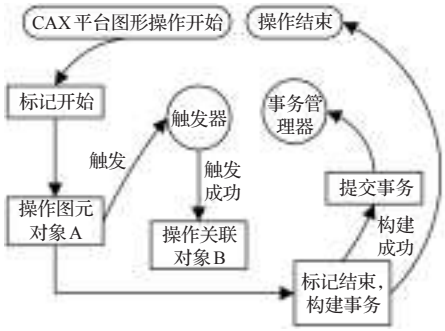


图2 基于事务与触发器机制的操作模型图

3.1 对象变更状态的标记

3.1.1 标记策略规则制定

在此机制中,为标记策略作制定如下规则:

规则1 标记为图形对象所拥有,标记不参与对象之间的数据传输,对一个对象所作的标记只对该对象有效。

规则2 当一个操作从用户界面发出时,标记开始;该操作执行完毕时,标记结束。

规则3 在完整提交一个事务后,清除对变更对象所做的所有标记。

对于标记的内容以原子操作为单位,作如下定义:

定义1 O_X表示对一个图形对象做的某种操作。针对对象的操作(O_X)最终分为三个基本操作:新增(O_A)、修改(O_M)、删除(O_D)。

定义2 $\sim O_X$表示执行该操作的逆操作。对某个对象所作操作的集合记为$S_O=\{O_X|X\in A,D,M\}$。

用户执行一个操作过程中,关联操作所包含的原子操作都将记录到所操作图形对象的S_O中,加入规则如下:

(1)$O_X==O_A$,则$S_O=S_O\cup O_A$,即为对象添加新增标记。

(2)$O_X==O_D$,则$S_O=S_O\cup O_D$,即为对象添加删除标记。

(3)$O_X==O_M$,且$O_M\notin S_O$,则$S_O=S_O\cup O_M$,当对象第一次被修改时,将原始数据提前备份。

(4)如果$O_X==O_M$,且$O_M\in S_O$,则$S_O=S_O\cup O_O$,如果对象已经被修改过,则对于以后的修改不进行记录。

对于新增和删除,本来就是一对互逆操作($O_A=\sim O_D$),因此Undo/Redo时,处理比较方便,规则中将对象分别增加"O_A"和"O_D"标记。对于修改操作,保证修改的是对象本身,而且在进行第一次修改时就将原始数据进行备份,中间的修改过程不予考虑。

3.1.2 数据的备份

数据对象的备份目的是为了记录原始的状态,如果直接使用对象的拷贝,则会与现有对象并存,必须由两个链表通过栈的方式进行对象的对应管理,两个对象在各自链表中要保证同时压栈或出栈。不仅增大了管理难度,同时也增加了异常出现的概率。

另外,对于图形的整体操作,直接使用克隆的方式,备份的将会是整张图形数据,会带来大量的冗余数据,浪费存储空间。

因此,采用下面架构避免上述问题:

(1)利用堆的方式,通过调用对象的序列化接口,将原始数据保存到内存中,内存地址由对象自己负责管理。通过对象与所管理内存地址上数据进行深度拷贝完成Undo/Redo操

作，这样就避免了存在两个管理者的弊病。

(2)对图形进行整体变动的操作(例如：整体缩放、移动等)，将操作的前后状态进行记录，作为一个事务，归入Command进行管理。通过有目的性地交换发生变化的特性，完成Undo/Redo操作。从而避免了冗余数据的产生。

3.1.3　数据的恢复

数据的恢复存在两种方式：内存地址交换及内存数据交换。执行了标记策略后，如果采用前者进行数据交换，则违背了标记策略的规则1，在数据交换后，标记也就不再有意义。因此，在进行Undo/Redo时，进行的是内存数据的恢复，而图形对象在图纸数据库中表现为同一块内存地址。

3.2　触发器机制

关联操作存在以下两种情况：

(1)顺序关联操作：O_{X1}执行过程完后，紧接着执行O_{X2}。

(2)嵌套关联操作：O_{X1}执行过程中执行O_{X2}，当O_{X2}执行完后，接着执行O_{X1}。

为了避免通过操作的嵌套来实现对象之间的协同更新，在标记策略基础之上，构造对应的触发器，来实现对象之间的协同更新。

应用程序中，触发器是个特殊的消息机制，它的执行不是由程序调用，也不是手工启动，而是由事件来触发。当应用程序启动，或者特定的模块被加载时，触发器被注入到CAX系统之中，响应特定的消息事件。当这些特定的消息发生后(如图3所示，对象A被添加、修改或者删除)，CAX系统便会根据所注入的触发器，调用相应的接口来实现一些特定的操作(修改关联对象B)。系统应当在这些操作进行之时，通过触发实现相关对象的更新。

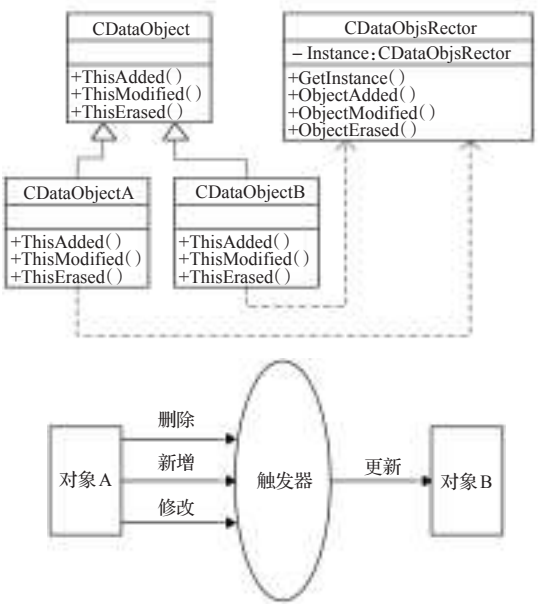


图3　触发器UML图及模型图

通过触发器的实现原理，可知对象的协同修改不是由操作实现。触发器作为原子操作的一种延伸，可避免将关联操作具体化为一个操作类型，从而被嵌套调用，进一步避免了操作提交的Undo/Redo事务的嵌套封装。降低操作复杂度和关联度。

触发器类型具体包括：

(1)对象触发器：通过触发器捕捉到特定的图元对象的添加、删除或者被修改消息。

(2)命令触发器：通过触发器捕捉到特定命令的开始、结束以及取消的消息(如删除、移动等)。

在具体的实现中，触发器采用Singleton模式。由触发器负责处理对象协同更新。

通过触发器降低操作的关联度后，可实现对象在变更集中的平行放置。在操作完成后，就应该根据变更集中对象的标记情况，按照一定的规则来构建事务。

3.3　事务构建

事务由事务开始(begin transaction)和事务结束(end transaction)之间执行的全体操作组成，因此一个操作开始后，即意味着事务的开始，对于所有发生变化的数据(包括新增加的对象)都要以可写的方式进行操作。在操作结束时，就意味着事务的结束，本次操作变动的所有对象都将被放入到一个集合中，作为一个事务进行提交。事务作为变更数据的集合，其结构为：

$$T=\begin{cases}S_{\mathrm{Add}}=\{Item_i|i\in N\}\\ S_{\mathrm{Del}}=\{Item_j|j\in N\}\\ S_{\mathrm{Mdf}}=\{Item_k|k\in N\}\end{cases}$$

事务依据图形对象所记录的操作集合进行构建。设定O_{E}表示对该对象最终所应该做的操作，O_{E}运算具体原则如下：

(1)$O_{\mathrm{A}}\in S_O$且$O_{\mathrm{D}}\in S_O$，则$O_{\mathrm{E}}=O_{\mathrm{D}}$。

表示新增一个对象，又对此对象进行了删除操作，则直接删除该对象。

(2)$O_{\mathrm{A}}\in S_O$且$O_{\mathrm{M}}\in S_O$且$O_{\mathrm{D}}\notin S_O$，则$O_{\mathrm{E}}=O_{\mathrm{A}}$。

表示新增加一个对象，然后对该对象进行修改操作，则应该删除备份数据，取消新添加标志，将该对象加入到S_{Add}中。

(3)$O_{\mathrm{D}}\in S_O$且$O_{\mathrm{M}}\in S_O$，则$O_{\mathrm{E}}=O_{\mathrm{D}}\cup(\sim O_{\mathrm{M}})$。

表明在对一个对象进行删除操作之前，对该对象进行了修改操作，则应该利用备份数据将对象回退到修改前状态，然后将该对象放入S_{Del}中。

(4)$O_{\mathrm{D}}\in S_O$且$O_{\mathrm{M}}\notin S_O$，则$O_{\mathrm{E}}=O_{\mathrm{D}}$。

表示仅对该对象仅进行了删除操作，则直接将该对象放入S_{Del}中。

(5)$O_{\mathrm{M}}\in S_O$且$O_{\mathrm{A}}\notin S_O$且$O_{\mathrm{D}}\notin S_O$，则$O_{\mathrm{E}}=O_{\mathrm{M}}$。

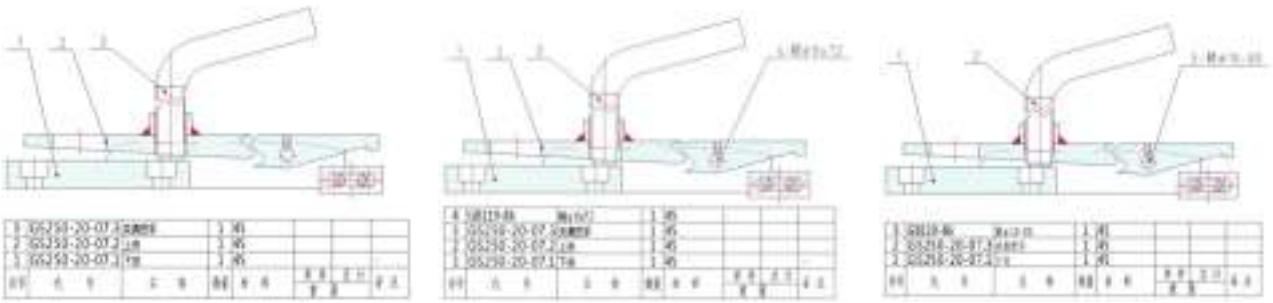
表示仅对该对象进行了修改操作，则将该对象放入S_{Mdf}中。

按照上述规则对所有变更数据进行整理后，一个完整的事务就构建完毕，操作结束后，将此Transaction进行提交。此时，应该遵照标记策略的规则3，清除所有标记。

事务构建的UML模型如图4所示，事务(Transaction)由一次操作中所变更的图形对象(Transaction_Obj)和通用操作对应的数据管理对象(Transaction_Tool)组成。每一次操作的正常结束，都会提交一个事务(Transaction)，由事务管理器(TransManager)负责管理。

4　应用实例

在TYCAD系统中，实现了明细表和标签的协同更新，采用了基于事务和触发器机制的Undo/Redo操作后，语言一致性问题得到了很好的解决，下面通过一个实例的两次操作来分析基于机制对语义一致性的支持。



(a)原始图形 (b)新增标签4 (c)修改明细表:修改标签,删除标签

图5 应用实例图

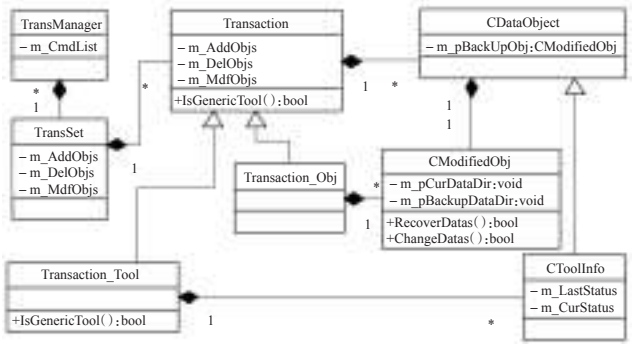


图4 事务构建UML图

操作1 用户执行新增标签4操作。如图5(a)→(b)。

分析 新增标签时,将会协同更新明细表。因此会顺序关联执行更新明细表操作。

在此实例中,产生变更的图形对象集合为:

$S_{Items}=\{Item_{label4},Item_{pratlist}\}$。

对$Item_{label4}$进行的操作集合为$S_O=\{O_A\}$,因此最终的操作$O_E=O_A$。

对$Item_{pratlist}$进行的操作集合为$S_O=\{O_M\}$,因此最终的操作$O_E=O_M$。

本次操作提交的事务为:

$$T=\begin{cases}S_{Add}=\{Item_{label4}\}\\S_{Del}=\varnothing\\S_{Mdf}=\{Item_{pratlist}\}\end{cases}$$

对上述对象执行Undo操作后(图5中(b)→(a)),能准确地删除标签,恢复明细表中的数据,Redo时(图5中(a)→(b))亦然。

操作2 在明细表中,删除标签2,同时修改销的规格为ϕ10~80。如图5(b)→(c)。

分析 删除标签的同时,后续的标签会自动向前更新序号。因此,本次操作既包含顺序关联操作,又包含嵌套关联操作。可分解为更新明细表、更新标签4、删除标签2、依次更新标签3、4的序号。

在此实例中,产生变更的图形对象集合为:

$S_{Items}=\{Item_{label2},Item_{label3},Item_{label4},Item_{pratlist}\}$。

对$Item_{label2}$进行的操作集合为$S_O=\{O_D\}$,因此最终的操作$O_E=O_D$。

对$Item_{label3}$进行的操作集合为$S_O=\{O_M\}$,因此最终的操作$O_E=O_M$。

对$Item_{label4}$进行的操作集合为$S_O=\{O_{M1},O_{M2}\}$,因此最终的操作$O_E=O_M$。

对$Item_{pratlist}$进行的操作集合为$S_O=\{O_M\}$,因此最终的操作$O_E=O_M$。

本次操作提交的事务为:

$$T=\begin{cases}S_{Add}=\varnothing\\S_{Del}=\{Item_{label2}\}\\S_{Mdf}=\{Item_{label3},Item_{label4},Item_{pratlist}\}\end{cases}$$

执行Undo操作后(图5中(c)→(b)),能准确地恢复删除的标签2,恢复明细表中的数据,同时更新标签3、标签4。Redo时(图5中(b)→(c))亦然。

通过上面的实例,对比通用的两种Undo/Redo实现原理,本算法成功解决了下面几个问题:

(1)相对于检查点方式,数据的备份从文件单元变为事务单元。数据的冗余量大大减少。

(2)对于多目标操作,一次Undo操作能正确撤销发生变动的对目标,保证了原子性。

(3)存在关联关系的对象,在Undo/Redo中也能准确保证语义一致性,保证数据之间的关联关系的正确性。

5 总结

Undo/Redo操作采用此种机制后,Undo/Redo中事务的构建由底层完成,二次开发人员不用再关心Undo/Redo的构建。另外,多个对象之间存在的协同关系,通过标记策略和触发器建立,很好地保留在事务中,在进行Undo/Redo时,能够很好地维护操作的语意一致性,保证图形数据的正确性。

对于基于事务和触发器的Undo/Redo机制,仍有可改进之处。本文只在图形的整体操作时,采用了减少数据冗余的算法,从文件单元降为事务单元,其事务单元中的粒度为对象,可进行进一步的算法研究,将事务中粒度细化为对象的属性,这样就可以只对修改的属性进行事务构建,进一步减少数据冗余。

参考文献:

[1] 黄逸民,袁繁,王树青.利用设计模式实现电力图形编辑系统iSee3.0中的Undo/Redo功能[J].计算机工程与应用,2003,39(11):126-128.

[2] Huang Z Y,He F Z,Cai X T,et al.A group undo/redo mechanism to preserve user intention in replicated collaborative modeling systems[C]//Proceedings of the ASME 2009 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference(IDETC/CIE 2009),2009:647-653.

(下转169页)