计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.201100024

基于元模型的协同建模模型组装与更新方法

张子良　庄　毅　叶　彤
南京航空航天大学计算机科学与技术学院　南京 211106
(zhangziliang0106@163.com)

摘　要　随着软件规模日益增大,软件复杂度不断提高,飞机、轮船等大型系统的设计与开发往往是由多个不同专业领域、具有不同职能的团队相互协同完成的。针对协同建模中局部模型之间缺失信息所导致的模型不完整问题和更新操作之间发生冲突所导致的模型不一致问题,文中首先提出了一种基于元模型的协同建模模型组装与更新方法(Model Combination and Update,MCAU),该方法在元模型上定义了协同关系与更新操作,可在协同建模过程中保证模型的完整性与一致性,并通过一个实例对所提方法进行了应用与分析。其次,文中还提出了一种基于模型驱动的软件协同建模框架(Software Collaborative Modeling Framework,SCMF),该框架可有效支持多种建模语言的扩展。最后,基于 Eclipse 框架开发了软件协同建模原型系统 CorModel,并通过相关实验进一步验证了 MCAU 方法的有效性。

关键词:协同建模;元模型;冲突检测;模型完整性;模型一致性
中图法分类号　TP311

Cooperative Modeling Model Combination and Update Method Based on Meta-model

ZHANG Zi-liang,ZHUANG Yi and YE Tong
College of Computer Science and Technology,Nanjing University of Aeronautics and Astronautics,Nanjing 211106,China

Abstract　With the increasing scale of software and the increasing complexity of software,the design and development of large-scale systems such as aircrafts and ships are often completed by teams with different professional fields and functions. Aiming at the problem of incomplete model caused by missing information between local models and model inconsistency caused by conflict between update operations,this paper proposes a method of model combination and update (MCAU) based on the meta-model. This method defines the collaborative relationship and update operation on the meta model,which can ensure the integrity and consistency of the model in the process of collaborative modeling. An example is given to illustrate the application and analysis of the proposed method. Secondly,this paper proposes a model driven software collaborative modeling framework(SCMF),which can effectively support the extension of multiple modeling languages. Finally,this paper develops a software collaborative modeling prototype system(CorModel) based on eclipse framework,and further verifies the effectiveness of MCAU through related experiments.

Keywords　Collaborative modeling,Meta-mode,Conflict detection,Model integrity,Model consistency

1　引言

随着技术的进步与科技的发展,近年来软件不断向复杂化、规模化与开放化的趋势发展。飞机、轮船等大型系统的设计与开发往往是由多个不同专业领域、具有不同职能的团队相互协同完成的。由于各个团队的专业领域不同,以往的软件开发过程基本上是在各自的设备上进行开发和调试,通过人员沟通、现场集成、联调等方式来实现开发过程的协同。结合过程管理方法,通过协议与研发规约、人工部署与协调的方式来实现软件的研发集成和部署。随着软件的不断发展和软件功能需求的不断增多,原有主要依靠人力的协同开发已经越来越无法满足不同领域、不同专业之间相互协同开发的需求。因此,研究动态、适应能力强的软件协同开发技术具有十分重要的意义和应用前景。

软件工程[1](software engineering)中协同方法在很大程度上实现了软件开发效率的提高。然而,当前软件工程中的协同方法大多集中在工程实现的代码层次上,缺少在模型设计阶段对系统建模的协同方法。使用传统的版本控制系统进行模型版本控制可能会丢失模型所包含的许多信息,在模型的合并过程中可能会出现语法和语义不一致的现象。因此,

到稿日期:2020-11-03　返修日期:2021-04-23
基金项目:国家自然科学基金(61572253);航空科学基金(2016ZC52030)
This work was supported by the National Natural Science Foundation of China(61572253) and Aeronautical Science Foundation of China(2016ZC52030).
通信作者:庄毅(zy16@nuaa.edu.cn)

目前需要一种在模型层次上的软件协同方法，以保证在模型设计阶段对系统的协同建模。

模型驱动软件工程[2](Model-Driven Software Engineering，MDSE)指在软件工程的整个生命周期中，系统性地用模型来描述整个软件项目。模型驱动的开发方法将软件开发中的第三代编程语言代码转移到了用建模语言描述的模型上，实现了软件开发从代码层到模型层的转换，提高了系统的抽象程度。MDSE 的目标是通过系统模型，对系统的具体实现细节进行抽象，使得开发人员能够更专注于系统的整体结构与功能，从而提高软件开发的效率。虽然 MDSE 在一定程度上降低了开发过程的复杂度，但是面对日益庞大的软件系统，特别是针对需要多个不同专业领域团队进行协同开发的软件，MDSE 不能给予很好的支持[3]。

协同软件工程[4](Collaborative Software Engineering，COSE)指多个软件开发人员通过协同的方式对软件项目进行开发。其关键要素包括 3 点：协作、沟通和协调[5]。协作指不同专业领域团队协同进行软件开发，沟通指开发人员之间的信息交互，协调指当发生冲突时协同规约所制定的解决方案。COSE 是传统软件工程的扩展，对分布式团队合作中的协同性进行了研究，其广泛的应用推动了大型软件系统的整合和集成，方便了项目管理、工作流管理和人员沟通。但是 COSE 的软件开发核心仍处于代码层面，无法针对模型对象进行协同开发。随着软件复杂度的提高，COSE 的不足之处越来越明显。

本文提出了一种基于元模型的协同建模模型组装与更新方法(MCAU)，该方法在元模型上定义了协同关系与更新操作，可在协同建模过程中保证模型的完整性与一致性。本文还提出了一种基于模型驱动的软件协同建模框架(SCMF)，包括建模层、协同模型组装与更新层和数据层，该框架可有效支持多种建模语言的扩展。本文基于 Eclipse 插件框架开发了软件协同建模原型系统 CorModel，并通过相关实验进一步验证了 MCAU 的有效性。

2 相关工作

传统的软件团队开发通常使用集成开发环境(Integrated Development Environment，IDE)和软件配置管理系统(Software Configuration Management，SCM)。集成开发环境集成了代码编辑器、语言编译器、调试工具和图形界面等工具，用于为开发人员提供开发环境。配置管理系统用于对项目进行过程管理，提供版本控制工具、配置管理工具和文件存储与备份功能等，以保证项目的完整性与可跟踪性。传统的软件工程中的协同方法大多集中在工程实现的代码层次，常用的协同开发与版本控制工具(如 Subversion[6] 和 Git[7])都是面向文本的，并且基本上在面向行的级别上进行管理与更改。由于模型比纯文本更复杂，其互连元素的数据结构更丰富，使用传统的版本控制系统进行模型版本控制可能会丢失模型所包含的许多信息，在模型的合并过程中可能会出现语法和语义不一致的现象。

目前，已有不少学者从多方面对软件协同建模方法开展了研究，并取得了一定研究和应用成果。模型驱动软件工程中的协同建模技术在学术界和工业界中正受到越来越多的关注。根据协同工作方式的不同，可将协同建模分为同步(synchronous)和异步(asynchronous)两种协同建模方式[8]。

同步协同建模方式指不同的建模人员可以实时进行协同操作，通过对共享模型的编辑来进行协同建模。一般使用锁的方式实现多人同时对一个模型进行修改，建模人员获取锁资源以防止其他人员进行修改，在修改完成后释放锁，解除资源锁定。这种协同建模方式通过共享的工作空间和同步编辑工具实现，能够有效避免模型出现不一致的问题。建模工具 Morsa[9] 和 Eclipse CDO[10](Eclipse Connected Data Objects)采用悲观锁的方法支持协同建模，用户锁定他们想要编辑的元素，并阻止其他人访问这些元素。Wuest 等开发了 FlexiSketch 工具，用于支持在移动设备上进行模型草图的协同绘制[11]，该工具使用锁的方式实现同步操作，一次只有一个用户可以对访问元素进行操作。Debreceni 等采用基于属性的锁定方法[12]，提出使用建模语言的语义来避免用户引入可能导致不一致性的模型更新。Gao 等提出了一种实时协同环境下的语义冲突解决方法 ACAS[13]，其结合了 CAS 乐观锁的并发控制思想，维护实时编程语义一致性。虽然以上提到的同步协同建模方法采用锁机制来保证模型的一致性，可避免因多人同时操作一个模型而产生冲突。但是，种锁定的方法存在建模人员长时间独占模型的情况，为软件开发带来了额外的负担，造成了资源与时间不必要的浪费。

异步协同建模方式指不同的建模人员在不同时间进行协同操作。该方式通常为集中式，即采用一个服务器和多个分布式协同建模节点的网络架构。考虑到异步协同建模中，各个局部模型是不同开发人员分别在本地建立的，为获得完整的软件全局模型，需将各局部模型上传到协同服务器进行组装与更新。其中模型的完整性与一致性是异步协同建模中需要解决的关键问题。Koshima 等提出了一个并发协同建模框架 DiCoMEF[14]，该框架针对模型冲突问题，通过令牌技术分配模型的修改权限，并使用本地存储库存储模型修改记录。Elaasar 等提出了支持整个生命周期的协同框架 DM[15]，该框架基于 IBM Jazz 平台为 Web 分布式协同建模节点，并提供中央数据库，通过数据库中模型的比较与合并来进行冲突消解。Nantes 大学的 Sunyé 提出了基于对等体系结构的同步协同框架[16]，并使用一致性算法进行模型的同步更新。Wang 等提出了一种在移动云平台下基于局部复制的结构性文档协同编辑冲突消解算法 MCPS[17]，使用副本监听与活跃度来判断冲突产生时操作执行的优先级。Zhang 等设计了一种面向问题的软件开发协同建模工具[18]，使用对象约束语言实现了自动化校验问题图的正确性和完整性。虽然以上提到的异步协同建模方法都提供了冲突检测与解决的方法，但是这些方法并没有考虑在模型组装过程中各个局部模型之间的关系以及在模型更新过程中各个更新操作之间的关系。这就导致无法对模型的完整性与一致性进行准确的校验，且其复杂性会带来额外的系统开销。

综上所述，为了解决传统版本控制工具在协同建模过程中导致模型信息丢失的问题、同步协同建模方法锁机制造成共享模型长时间被独占的问题，以及异步协同建模方法在模

型组装与更新过程中缺少协同语义的问题，本文提出了一种基于元模型的协同建模模型组装与更新方法和一种基于模型驱动的软件协同建模框架。

3 软件协同建模框架

基于模型驱动的协同软件工程[19]（Collaborative Model-driven Software Engineering，CMSE）属于模型驱动软件工程与协同软件工程的交集。在 CMSE 中多个建模人员可以在本地或远程共享工作空间中同步或异步地进行建模，并支持模型管理与消息通信等机制。CMSE 将模型视为关键要素，同时支持模型驱动软件工程和协同软件工程，属于软件工程发展的重要方向。

如图 1 所示，CMSE 包括 3 个关键要素：协作、管理、沟通[20]。“协作”指开发人员协作处理模型的协同方式，包括版本控制系统、可视化系统、模型合并机制、冲突管理机制等。“管理”指用于管理模型生命周期的模型管理方式，包括存储模型和元模型的数据库，用于创建、编辑和删除模型的建模工具和项目之间模型的交换格式等。“沟通”指开发人员之间进行信息交流的沟通方式，包括消息记录板、消息通知机制和问题跟踪机制等。

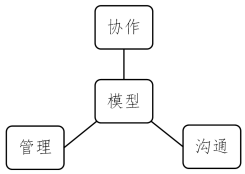


图 1 基于模型驱动的协同软件工程的 3 要素

Fig. 1 Three elements of model-driven collaborative software engineering

为有效支持基于模型驱动的软件协同建模开发，本文提出了一种协同建模框架 SCMF。该框架给出了协同建模步骤，可用于指导实际软件协同设计，并能有效支持软件协同过程中的模型组装与更新，提高软件对协同功能的适应能力。

如图 2 所示，本文提出的 SCMF 框架包括建模层、协同模型组装与更新层、数据层。

（1）建模层对应 CMSE 中的“沟通”要素，首先对软件功能进行分解，将建模任务下发到协同建模节点，各个节点分别在本地建模环境中建立局部模型，节点的建模人员通过消息通信创建局部模型之间的协同关系，并将建立好的局部模型与协同关系上传到协同服务器。

（2）协同模型组装与更新层对应 CMSE 中的“协作”要素，包括模型组装和模型更新两个模块。在模型组装模块中，协同服务器创建关系队列来接收协同关系，对其中的每个关系进行完整性校验。将符合完整性要求的协同关系组装到组装序列中，协同服务器根据组装序列进行模型组装；对于不符合完整性要求的协同关系，将其错误信息发送给对应的协同建模节点。协同服务器根据组装序列进行模型组装。在模型更新模块中，协同服务器创建消息队列来接收更新消息，并对其中的每个操作进行一致性校验，将符合一致性要求的更新操作添加到更新序列中，对于不符合一致性要求的更新操作，将其错误信息发送给对应的建模节点，协同服务器根据更新序列进行模型更新。

（3）数据层对应 CMSE 中的“管理”要素，协同服务器将组装和更新后的模型存储到模型数据库中，以保证在出现错误时可以进行版本回退。

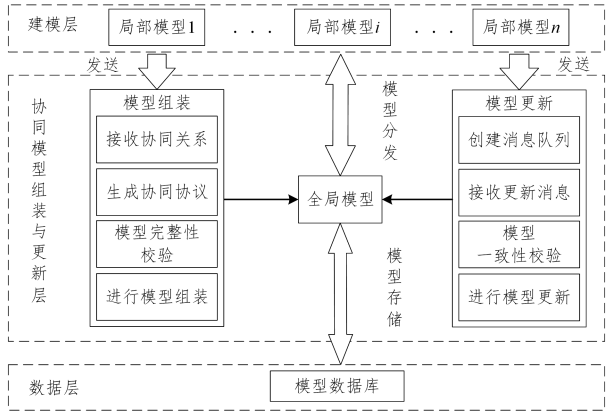


图 2 基于模型驱动的软件协同建模框架

Fig. 2 Model-driven software collaborative modeling framework

4 模型组装与合并方法

4.1 元模型与协同关系

（1）本文使用元模型对模型元素进行描述。元模型属于模型的“模型”，用于定义模型的概念和创建该领域中模型的构建元素。可以将元模型类比为模型的模板，将模型类比为元模型的实例。通过元模型提供的模型模板的定义和概念，可以将模型进行进一步的抽象，以便更好地描述模型之间的关系与交互。元模型和模型相比具有更好的扩展性与灵活性，可以扩展并适应多种模型描述语言。本文将模型中的元素描述为：1）每一个节点都有唯一的标识符，用 NID 表示；2）每一个模型都有唯一的标识符，用 MID 表示；3）每一个模型元素都有唯一的标识符，用 EID 表示；4）模型元素中包括若干属性，每个属性的类型和特征都有唯一的标识符，分别用 TID 和 FID 表示，一对（EID，FID）表示元素 EID 上特征 FID 的值。

每个协同建模节点包含若干个模型，每个模型包含若干个模型元素，每个模型元素包含若干个属性，而每个属性都有相应的类型和特征。模型的定义如下。

定义 1 模型可以表示为式（1）所示的四元组，其中 MID 是模型的唯一标识符，NID 是模型所属节点的标识符，$Element$ 是模型的一组模型元素，$Attribute$ 是模型元素的一组属性。

$$Model=(MID,NID,Element,Attribute) \tag{1}$$

对多个模型进行模型组装时，需要建立不同模型的模型关系。标准模型关系不具备协同语义，无法表示关系的源模型元素和目标模型元素所属模型的信息，导致在形成的全局模型中将无法区分各个模型的边界，也就无法在对模型进行完整性校验时将问题定位到对应的模型元素。为解决以上问题，本文采用定义协同关系的方法为模型元素之间的关系添加协同语义，协同关系的定义如下。

定义 2 协同关系 $CorRelation$ 可以表示为式（2）所示的

五元组。其中，NID 表示该模型所属的节点，EID_s 表示源模型元素，EID_d 表示目标模型元素，MID_s 表示源模型元素所属模型，MID_d 表示目标建模元素所属模型。

$$CorRelation=(NID,EID_s,EID_d,MID_s,MID_d) \quad (2)$$

在软件协同建模框架 SCMF 中，模型组装是通过建立协同关系进行的，通过在协同关系上定义协同信息，可以对不同模型进行协同建模与管理，并对全局模型进行完整性校验，以方便错误定位与重新组装。

4.2 模型组装与完整性校验

模型组装是软件协同建模框架 SCMF 的核心，通过模型组装过程可将多个协同建模节点上传的局部模型根据协同关系进行组装，并进行完整性校验以建立全局模型。通过协同关系中包含的协同信息，可以明确各个节点所属模型的边界，从而降低完整性校验的复杂度，使模型组装和模型校验具有较快的响应速度。

设有 n 个协同建模节点 $C_1,C_2,\cdots,C_n$ 和一个协同服务器 S。建模人员在节点 $C_i(1\leqslant i\leqslant n)$ 处完成本地建模任务，得到该节点的局部模型 M_i 和协同关系 $CorRelation$，并将其上传到协同服务器 S。协同服务器 S 维护一个关系队列 Q_R 和组装序列 L_R，关系队列 Q_R 用来接收节点发送过来的协同关系，组装序列 L_R 是符合完整性校验的协同关系序列。

本文提出的模型组装与完整性校验算法如算法 1 所示，它的输入是各个协同建模节点建立的局部模型 M_i、关系队列 Q_R 和组装序列 L_R，输出是组装后的全局模型 M。该算法首先将各个局部模型 M_i 合并成全局模型 M。然后，对关系队列 Q_R 中的每个协同关系逐一进行完整性校验，判断源模型 MID_s 中是否存在源模型元素 EID_s 以及目标模型 MID_d 中是否存在目标模型元素 EID_d。如果找不到该模型元素，则不符合完整性，将错误信息反馈到相应的节点；否则，将协同关系添加到组装序列 L_R 中等待执行。最后，协同服务器 S 根据组装序列 L_R 生成协同关系模型元素，并将其组装到全局模型 M 中，得到组装后的全局模型 M。

算法 1 模型组装与完整性校验算法

```
Input:Mi,QR,LR
Output:M
1.  for each Mi do
2.      M=M∪{Mi};
3.  for each Ri in QR do
4.      if EIDS∈MIDs and EIDD∈MIDd
5.          LR=LR∪Ri;
6.      else send error message;
7.  for each Rj in LR do
8.      M=M∪{⟨EIDS,EIDD⟩};
9.  retrun M.
```

为了保证模型在组装过程中的完整性，模型组装应当遵循以下步骤：

步骤 1 协同建模节点 C_i 完成建模任务并上传至协同服务器 S，协同服务器 S 接收节点 C_i 上传的局部模型 M_i 和协同关系 $CorRelation$。

步骤 2 协同服务器 S 将协同建模节点 C_i 上传的局部模型 M_i 合并为全局模型 M，并创建关系队列 Q_R 以接收节点上传的协同关系 $CorRelation$。

步骤 3 执行模型组装与完整性校验算法。协同服务器 S 对关系队列 Q_R 中的每个协同关系进行完整性校验，将校验合格的协同关系加入到组装序列 L_R 中等待执行。协同服务器 S 根据组装序列 L_R 在全局模型 M 上执行模型组装操作，得到组装后的全局模型 M。

步骤 4 协同服务器 S 将全局模型 M 保存到数据库，并将全局模型 M 分发至各个协同建模节点 C_i。

5 模型更新与同步方法

5.1 更新操作与更新消息

本文在元模型的基础上定义更新操作，模型的更新一般包括模型元素和特征值的添加、删除和修改等操作。通过对模型更新操作的统一定义，可以将协同建模节点对模型的更新描述为统一的更新操作。模型的多个更新操作可以组成一个更新序列，该序列包含了节点对模型所有的修改信息。本文定义的更新操作如下。

(1) $attach(MID,EID)$：向模型 MID 中添加元素 EID。

(2) $detach(MID,EID)$：从模型 MID 中删除元素 EID。

(3) $set(EID,FID,v)$：设置 EID 中特征 FID 的值为 v。

(4) $unset(EID,FID)$：删除元素 EID 中的特征 FID。

(5) $add(EID,FID,v)$：在 FID 的后面添加特征值 v。

(6) $remove(MID,EID,i)$：删除 MID 中 EID 索引 i 处的特征值。

(7) $move(MID,EID,s,t)$：将元素 EID 的所有特征值从源索引 s 移动到目标索引 t。

(8) $clear(MID,EID)$：清除 MID 中 EID 的所有特征值。

更新操作可以用更新序列表示。表达式 $M'=O*M$ 表示在模型 M 进行更新操作 O 后得到模型 M'，运算符“$*$”表示对模型执行更新操作。更新序列的定义如下。

定义 3 更新序列可以表示为式(3)所示的表达式，其以顺序的方式对模型 M 执行 n 个操作 $O_i(1\leqslant i\leqslant n)$ 后得到模型 M'，操作指上述定义的更新操作规则。

$$M'=(O_1,O_2,\cdots,O_{n-1},O_n)*M \quad (3)$$

由于更新操作可能会存在相互影响的情况，这种相互影响可以表示为操作之间的关系。通过操作的关系可以推断出一个更新序列是否合法，当更新序列不合法时即发生了冲突。若更新序列未发生冲突可以直接执行更新操作，否则协同服务器会将错误信息反馈给协同建模节点。更新操作之间的关系的定义如下。

定义 4(独立关系) 给定任意模型 M 和任意两个操作 O_a 和 O_b，如果 O_a 和 O_b 是可交换的，则称 O_a 和 O_b 为独立关系，如果 $(O_a,O_b)*M=(O_b,O_a)*M$，则 O_a 和 O_b 是相互独立的。

定义 5(依赖关系) 给定任意两个操作 O_a 和 O_b，以及操作 O_a 的前置操作集合 C_{Oa}，如果 O_b 包含在 C_{Oa} 中，则称 O_a 和 O_b 为依赖关系，即 O_a 依赖于 O_b，当且仅当 $O_b\in C_{Oa}$。

定义 6(冲突关系) 给定任意模型 M 和任意两个操作 O_a 和 O_b，以及其前置操作集合 C_{Oa} 和 C_{Ob}，则 O_a 和 O_b 为冲突关系，当且仅当 $C_{Oa}=C_{Ob}$ 和 $(O_a*O_b)*M\neq(O_b*O_a)*M$。

定义 7(等价关系) 考虑到任意模型 M 和任意两个操作

O_a 和 O_b，以及其前置操作集合 C_{Oa} 和 C_{Ob}，O_a 和 O_b 为等价关系，当且仅当 $C_{Oa}=C_{Ob}$ 和 $O_a*M=O_b*M$。

当协同服务器完成模型组装与合并之后，各个协同建模节点拥有全局模型的本地副本，节点的建模人员可以对模型执行更新操作。在执行完更新操作之后，需要将本地更新操作作为更新消息传到协同服务器。更新消息的定义如下。

定义 8 更新消息 *CorMessage* 可以表示为式(4)所示的四元组，其中 *MID* 是模型的唯一标识符，*NID* 是模型所属节点的标识符，*O* 是执行的操作，C_O 是操作的前置操作集合。

$$CorMessage=(NID,MID,O,C_O) \tag{4}$$

在基于模型驱动的软件协同建模框架中，模型更新是通过建立更新来进行的，通过在更新上定义的更新操作，可以根据操作之间的关系来检测冲突，并对全局模型进行一致性校验，方便冲突定位与冲突解决。

5.2 模型更新与一致性校验

软件协同建模的模型更新即模型的版本更新会导致全局模型的版本更新。此时，不同版本的模型可能会因为具有不同的抽象层次而出现模型间的一致性问题。一致性算法校验的目的是保证所有与更新部分相关的模型都处于同一抽象层次，从而保证全局模型的一致性。

设参与协同建模节点的总数为 n，建模人员在节点 C_i ($1\leqslant i\leqslant n$)执行本地更新操作，并将更新消息发送到协同服务器 S。协同服务器 S 维护一个消息队列 Q_M 和更新序列 L_M。消息队列 Q_M 用于接收节点 Ci 发送过来的更新消息，更新序列 L_R 是符合一致性校验的更新操作序列。

本文提出的模型更新与一致性校验算法如算法 2 所示，它的输入是更新前的全局模型 M、消息队列 Q_M 和更新序列 L_M，输出是更新后的全局模型 M'。该算法首先对消息队列 Q_M 中的每一个消息对应的操作 O_i 进行关系判断，若更新序列 L_M 中的所有操作与当前操作均为独立关系，则将当前操作加入到执行队列的末尾；若更新序列 L_M 中存在依赖当前操作的操作，则将当前操作加入到目标操作的前面；若更新序列 L_M 中存在与当前操作为冲突关系的操作，则停止校验并将错误信息反馈给对应节点；若更新序列 L_M 中存在与当前操作为等效关系的操作，则忽略当前操作，继续进行下一个操作的校验。最后协同服务器 S 根据更新序列 L_M 在全局模型 M 上执行模型更新操作，得到更新后的模型 M'。

算法 2 模型更新与一致性校验

```
Input:M,QM,LM
Output:M'
1.   for each Oi in QM do
2.      if ∀Oj∈L,M*Oi*Oj=M*Oj*Oi
3.         LM=LM∪{Oi};
4.      if∃Oj∈L,Oi∈COj
5.         LM={Oi}∪L{Oj};
6.      if∃Oj∈L,COi≠COj,M*Oi*Oj≠M*Oj*Oi
7.         send error message;
8.      if ∃Oj∈L,COi≠COj,M*Oi=M*Oj
9.         continue;
10. for each Oi in LM do
11.    M'=M*Oi;
12. retrun M'.
```

模型更新应当遵循以下步骤：

步骤 1 协同建模节点 *Ci* 完成模型更新任务，根据每一个更新操作生成更新消息 *CorMessage*，并将更新消息发送到协同服务器 *S*。

步骤 2 协同服务器 *S* 创建消息队列 Q_M，接收协同建模节点 C_i 发送的更新消息 *CorMessage*，并将更新消息存储到消息队列 Q_M 中。

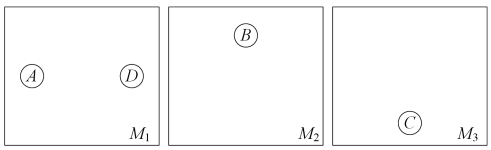
步骤 3 执行模型更新与一致性校验算法。协同服务器 *S* 对消息队列 Q_M 中的每个操作进行校验，将校验合格的操作加入到更新序列 L_M 中等待执行。协同服务器 *S* 根据更新序列 L_M 在全局模型 *M* 上执行模型更新操作，得到更新后的模型 M'。

步骤 4 协同服务器 *S* 将更新后的模型 M' 保存到数据库，并将 M' 分发至各个协同建模节点 C_i。

6 实例分析

6.1 模型组装与合并

假设有 3 个协同建模节点 *Node*1，*Node*2 和 *Node*3，分别用 *n*1，*n*2 和 *n*3 表示。如图 3(a)所示，节点 *Node*1 包含 1 个模型 *M*1(*m*1)，模型 *M*1 包含的元素有顶点 *A*(a)和顶点 *D*(d)。如图 3(b)所示，节点 *Node*2 包含 1 个模型 *M*2(*m*2)，模型 *M*2 包含的元素有顶点 *B*(b)。如图 3(c)所示，节点 *Node*3 包含 1 个模型 *M*3(*m*3)，模型 *M*3 包含的元素有顶点 *C*(c)。



(a)节点 *Node*1 (b)节点 *Node*2 (c)节点 *Node*3

图 3 节点 *Node*1，*Node*2 和 *Node*3 的局部模型

Fig. 3 Partial model of *Node*1, *Node*2 and *Node*3

假设现在节点 *Node*1，*Node*2 和 *Node*3 的建模人员通过消息通信后需要建立模型之间的协同关系。如表 1 所列，模型 *M*1 中的顶点 *A* 与模型 *M*2 中的顶点 *B*、模型 *M*3 中的顶点 *C* 分别建立协同关系，模型 *M*2 中的顶点 *B* 与模型 *M*1 中的顶点 *D* 建立协同关系，模型 *M*3 中的顶点 *C* 与模型 *M*1 中的顶点 *D* 和顶点 *E*(不存在)建立协同关系。

表 1 节点 *Node*1，*Node*2 和 *Node*3 的协同关系

Table 1 CorRelation of *Node*1, *Node*2 and *Node*3

CorRelation	*NID*	MID_s	EID_s	MID_d	EID_d
R_1^1	n_1	m_1	*a*	m_2	*b*
R_1^2	n_1	m_1	*a*	m_3	*c*
R_2^1	n_2	m_2	*b*	m_1	*d*
R_3^1	n_3	m_3	*c*	m_1	*d*
R_3^2	n_3	m_3	*c*	m_1	*e*

根据模型组装与完整性校验算法，协同服务器 *Server* 将模型 m_1，m_2 和 m_3 合并为全局模型 *M*，将协同关系 R_1^1，R_1^2，R_2^1，R_3^1 和 R_3^2 存储在关系队列 Q_R 中，并对其中的每个协同关系进行完整性校验，判断源模型 MID_s 中是否存在源模型元素 EID_s 以及目标模型 MID_d 中是否存在目标模型元素 EID_d。其中协

同关系R_1^1,R_1^2,R_2^1和R_3^1符合完整性检查,将其添加到组装序列L_R中;协同关系R_3^2中的顶点e在m_1中不存在,因此其不符合完整性检查,将相关错误信息返回给节点n_3。最后,L_R可以表示为:

$$L_R=\{R_1^1,R_1^2,R_2^1,R_3^1\}$$

*Server*根据L_R对模型M进行组装,得到图4所示的组装后的模型M,*Server*将组装后的模型保存到数据库,并将其分发至各个节点。

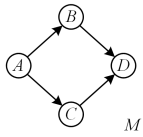


图4 组装后的模型M

Fig. 4 Model M after assembly

6.2 模型更新与同步

协同建模节点*Node*1,*Node*2和*Node*3本地保存全局模型M的本地副本,节点的建模人员在本地副本上进行模型的更新操作。节点*Node*1将顶点A重命名为"$A1$",节点*Node*2将顶点A重命名为"$A2$",并删除顶点D,节点*Node*3首先创建一个新顶点$E(e)$,然后在顶点A和顶点E之间创建一条边$AE(ae)$,最后删除顶点D。表2列出了节点*Node*1,*Node*2和*Node*3更新操作的集合。

表2 节点*Node*1,*Node*2和*Node*3的操作集合

Table 2 Operation set of *Node*1, *Node*2 and *Node*3

CorMessage	Node	Operation
O_1^1	n_1	$set(a,name,"A1")$
O_2^1	n_2	$set(a,name,"A2")$
O_2^2	n_2	$remove(m,vertice,d)$
O_2^3	n_2	$detach(d)$
O_3^1	n_3	$attach(e)$
O_3^2	n_3	$set(e,name,"E")$
O_3^3	n_3	$add(m,vertice,e)$
O_3^4	n_3	$attach(ae)$
O_3^5	n_3	$add(m,edge,ae)$
O_3^6	n_3	$set(ae,source,a)$
O_3^7	n_3	$set(ae,target,e)$
O_3^8	n_3	$remove(m,vertice,d)$
O_3^9	n_3	$detach(d)$

根据模型更新与一致性校验算法,协同服务器*Server*接收节点*Node*1的操作O_1^1,将其添加到更新序列L_M中。*Server*接收节点*Node*2的操作O_2^1,O_2^2和O_2^3,其中操作O_2^1与L_M中的操作O_1^1发生了冲突,将错误信息发送给节点*Node*1和*Node*2,并抛弃操作O_1^1和O_2^1;由于操作O_2^2和O_2^3没有冲突,将其直接加入到L_M中。*Server*接收节点*Node*3的操作O_3^1,…,O_3^9,操作O_3^1,O_3^2和O_3^3涉及添加一个新顶点,与L_M中其他操作为独立关系,因此将其加入L_M中;操作O_3^4,…,O_3^7涉及添加一条新边,与L_M中其他操作为独立关系,因此将其加入L_M中;由于操作O_3^8和O_3^9与操作O_2^2和O_2^3都是对顶点D的删除操作,其中操作O_3^8等价于O_2^2,操作O_3^9等价于O_2^3,因此将O_3^8和O_3^9抛弃,只保留一次删除操作。最后,L_M可以表示为:

$$L_M=\{O_2^2,O_2^3,O_3^1,O_3^2,O_3^3,O_3^4,O_3^5,O_3^6,O_3^7\}$$

*Server*根据L_M分别对模型M进行更新操作,得到如图5所示的更新模型M'。*Server*将更新后的模型M'保存到数据库,并将M'分发至各个节点。

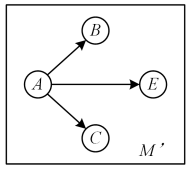


图5 更新后的模型M'

Fig. 5 Updated model M'

7 实验对比

为了验证本文提出的模型组装与合并方法、模型更新与同步方法的可行性与正确性,本文在Windows平台下使用Eclipse,Java语言,编码开发了软件协同建模原型系统CorModel。CorModel系统的前端功能模块基于Eclipse插件框架开发,模型部分基于Papyrus插件实现可视化建模;后台服务器端基于Netty异步通信框架实现相关业务功能,并对客户端的通信事件进行监听。CorModel系统主要包含两部分:1)协同建模模型组装与合并,根据协同关系对各个局部模型进行组装,并进行完整性校验;2)协同建模模型更新与同步,根据更新消息对全局模型进行更新,并进行一致性校验。通过模型组装与更新,可以得到完整的系统模型。CorModel系统模型组装与模型更新界面如图6所示。

图6 CorModel系统模型组装与更新界面

Fig. 6 CorModel system model assembly and updated interface

复杂软件一般需要较长的开发周期,需要较多的协同开发人员,且软件模型文件的数量较多。本实验不考虑软件开发周期因素,仅从协同广度与协同深度两个方面进行实验对比分析。协同广度用协同建模节点的数量表示,协同深度用模型文件的数量表示,其分别在横向和纵向两个方面体现了软件模型的复杂度。

本文将MCAU方法与DiCoMEF方法[14]应用于协同建

模环境，进行效率对比分析，对两种方法在处理模型组合与更新时的完整性与一致性处理时间进行对比，针对处理时间与协同建模节点数量做了相关测试。在仿真协同建模环境下进行对比实验，从协同深度与协同广度两个维度验证 MCAU 方法的有效性。

在模型组装阶段，在各个协同建模节点上建立相应的局部模型，并建立局部模型之间的协同关系，协同服务器接收局部模型与协同关系进行模型组合，得到组合后的全局模型。在模型更新阶段，在各个协同建模节点上进行更新操作，并建立对应的更新消息，协同服务器接收更新消息，根据软件全局模型对模型进行更新，得到更新后的全局模型。

如图 7 所示，从协同广度的角度来看，MCAU 方法和 DiCoMEF 方法在模型数量均为 30 的情况下，随着节点数量的增加，处理时间都会显著延长，这是因为局部模型间的协同关系数量会大大增加，但是 MCAU 方法模型组装与模更新的处理时间整体短于 DiCoMEF 方法。相比 DiCoMEF 方法，MCAU 方法采用了版本回退的方式，因此增加了空间复杂度。

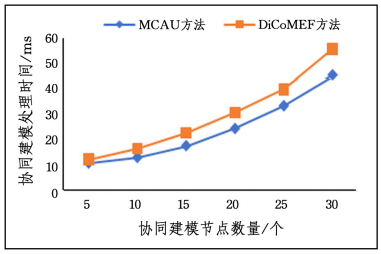


图 7 协同广度的对比结果

Fig. 7 Results of collaborative breadth comparison

如图 8 所示，从协同深度的角度来看，对不同模型数量在节点数量固定为 20 的情况下验证 MCAU 方法和 DiCoMEF 方法的处理效率。在节点数量相同时，模型文件数量越多，处理时间就越长，但是由于模型间的协同关系数量并没有显著增加，因此处理时间增长速度较慢。从整体来看，MCAU 方法的处理时间比 DiCoMEF 方法略短，但是两者之间的差距并不大。

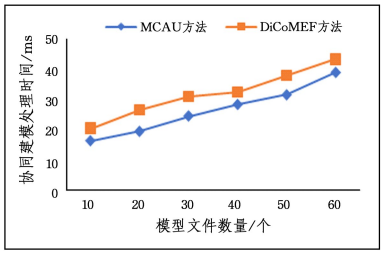


图 8 协同深度的对比结果

Fig. 8 Results of collaborative depth comparison

结束语 针对协同建模中不同局部模型之间信息缺失导致模型不完整的问题和更新操作之间的冲突导致模型不一致的问题，本文提出了一种基于元模型的协同建模模型组装与更新方法 MCAU，该方法在元模型上定义了协同关系与更新操作，可在协同建模过程中保证模型的完整性与一致性。本文提出了一种基于模型驱动的软件协同建模框架 SCMF，包括建模层、协同模型组装与更新层和数据层，该框架可有效支持多种建模语言的扩展。本文基于 Eclipse 插件框架开发了软件协同建模原型系统 CorModel，并通过相关实验进一步验证了 MCAU 方法的有效性。未来我们将研究模型组装发生元素缺失时自动对缺失信息进行填充以及在更新操作发生冲突时自动解决冲突，以进一步提升协同建模的自动化。

参 考 文 献

[1] ROGER S. Software engineering: a practitioner's approach (7th ed)[M]. McGraw-Hil, 2009.

[2] BEZIVIN J. On the unification power of models[J]. Software & Systems Modeling, 2005, 4(2): 171-188.

[3] ROCCO J D, RUSCIO D D, IOVINO L, et al. Collaborative Repositories in Model-Driven Engineering [J]. IEEE Software, 2015, 32(3): 28-34.

[4] BOSCH J, BOSCHS P. Collaborative Software Engineering [C]// International Conference on Software Engineering IEEE Computer Society. 2010.

[5] MISTRIK I, GRUNDY J, DER HOEK A V, et al. Collaborative Software Engineering: Challenges and Prospects[C]// Computational Science and Engineering. 2010: 389-403.

[6] PILATO C M, COLLINS-SUSSMAN B, FITZPATRICK B W. Version Control with Subversion: NextGeneration Open Source Version Control[M]. O'Reilly Media, Inc. , 2008.

[7] LOELIGER J, MCCULLOUGH M. Version Control with Git: Powerful tools and techniques for collaborative software development[M]. O'Reilly Media, Inc. , 2012.

[8] SUN Z, YE T, KONG X, et al. A Model-driven Collaborative Modeling Method for Software[C]// ICMSS 2020: 2020 4th International Conference on Management Engineering, Software Engineering and Service Sciences. 2020.

[9] PAGAN J E, CUADRADO J S, MOLINA J G, et al. Morsa: a scalable approach for persisting and accessing large models [C]// Model Driven Engineering Languages and Systems. 2011: 77-92.

[10] STEPPERE. Connected Data Objects—The EMF Model Repository [OL]. http://www. eclipse. org/cdo/documentation/presentations/EclipseCon_2008/CDO-Presentation. pdf.

[11] WUEST D, SEYFF N, GLINZ M, et al. Sketching and notation creation with FlexiSketch Team: Evaluating a new means for collaborative requirements elicitation[C]// IEEE International Conference on Requirements Engineering. 2015: 186-195.

[12] DEBRECENI C, BERGMANN G, RÁTH I, et al. Property-based locking in collaborative modeling[C]// 2017 ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems(MODELS). ACM, 2017: 199-209.

[13] GAO L P, YOU S W. Research on Semantic Conflict Resolution in Real-time Collaborative Programming Environment[J]. Journal of Chinese Computer Systems, 2019, 40(4): 9-17.

[14] KOSHIMA A A, ENGLEBERT V. Collaborative Editing of EMF/Ecore Meta-models and Models—Conflict Detection, Reconciliation, and Merging in DiCoMEF[C]// International Conference on Model-driven Engineering & Software Development. IEEE, 2015.

[15] ELAASAR M, CONALLEN J. Design management: a collaborative design solution[C]// European Conference on Modelling Foundations and Applications. 2013: 165-178.
[16] SUNYÉ G. Model consistency for distributed collaborative modeling[C]// European Conference on Modelling Foundations and Applications. Springer, Cham, 2017: 197-212.
[17] WANG D, ZHU S Z, GAO L P. Research on Structured Document Collaborative Editing Based on the Partial Replication Architecture in Cloud Platform[J]. Journal of Chinese Computer Systems, 2018, 39(10): 114-121.
[18] ZHANG X, LI Z, ZHAO Z Y, et al. Research and Implementation of Collaborated Modeling Approach for Problem-oriented Software Development[J]. Computer Science, 2018, 45(9): 119-122, 134.
[19] RUSCIO D D, FRANZAGO M, MALAVOLTA I, et al. Envisioning the future of collaborative model-driven software engineering[C]// International Conference on Software Engineering Companion. IEEE Press, 2017.
[20] FRANZAGO M, RUSCIO D D, MALAVOLTA I, et al. Collaborative Model-Driven Software Engineering: A Classification Framework and a Research Map[J]. IEEE Transactions on Software Engineering, 2018, 44(12): 1146-1175.

ZHANG Zi-liang, born in 1995, master. His research interests include software modeling and model verification.

ZHUANG Yi, born in 1956, Ph.D supervisor. Her research interests include trusted computing and formal methods.