

基于动态建模的低代码检修管理平台实现方案

周宁¹, 张锦斌², 胡川³, 雷晓波³, 李佳蓉², 陈爽²

(1. 重庆邮电大学, 重庆 400065; 2. 国网重庆市电力公司北碚供电分公司, 重庆 400700;

3. 重庆甘润科技有限公司, 重庆 400014)

摘要: 为了大幅度提高检修管理系统的开发效率, 提出一种基于动态建模的低代码平台实现方案。在该方案中, 应用模型作为容器与功能模块对应, 包含对象模型、关联模型、界面方案、工作流模型等要素。对象模型定义现实世界的业务对象及其属性, 关联模型定义对象之间的关系, 界面方案定义对象的展现策略; 模型存储于关系数据库。基于以数据结构复用支撑软件复用的方法, 提出一种解释引擎, 将模型直接解释成可运行的应用系统, 由模型数据在运行期动态决定应用系统的外观和行为。解释引擎可与程序代码融合, 兼顾应用生成高效化与应用功能精细化。该方案可将业务管理系统的代码量减少 40% 至 70%, 开发效率提升 2 至 3 倍。该方案在大量项目中成功实施, 证明了其有效性。

关键词: 动态建模; 数据结构复用; 软件复用; 解释引擎; 低代码

中图分类号: TP311

DOI: 10.19768/j.cnki.dgjs.2022.20.038

Implementation Scheme of Maintenance Management Platform with Less Code Based on Dynamic Modeling

ZHOU Ning¹, ZHANG Jinbin², HU Chuan³, LEI Xiaobo³, LI Jiarong², CHEN Shuang²

(1. Chongqing University of Posts and Telecommunications, Chongqing 400065, China;

2. Beibei Power Supply Branch of State Grid Chongqing Electric Power Company, Chongqing 400700, China;

3. Chongqing Ganrun Technology Co., Ltd., Chongqing 400014, China)

Abstract: To improve the development efficiency of the repair management system, a platform implementation scheme with less code based on dynamic modeling is proposed in this paper. In this scheme, the application model is used as a container corresponding to the function module. Besides, it includes object model, association model, interface scheme, workflow model, etc. The object model defines the real-world business objects and their attributes, the association model defines the relationship between objects, and the interface scheme defines the object presentation strategy. Models are stored in relational databases. Based on the method of supporting software reuse with data structure reuse, an interpretation engine is proposed, which directly interprets the model into a runnable application system, and the appearance and behavior of the application system are dynamically determined by the model data during runtime. The interpretation engine can be integrated with the program code, taking into account the high efficiency of application generation and the refinement of application functions. This scheme can reduce the code amount of the business management system by 40% to 70%, and improve the development efficiency by 2 to 3 times. The program has been successfully implemented in a large number of projects, proving its effectiveness.

Key words: dynamic modeling; data structure reuse; software reuse; interpretation engine; less code

0 引言

随着市场经济的快速发展, 电网企业检修管理面临日益复杂且不确定的形势, 快速变化的生产管理、安全运行及降本增效目标对检修管理系统提出了更高的要求, 而传统的以定制开发为主的模式暴露出开发效率低下、实施风

险高、迭代升级困难等问题, 难以满足新形势的要求。针对传统模式存在的问题, 研究者提出了敏捷开发、多层模型、概念模型、基于人工智能的方法、代码自动生成等思路与方法。这些方法在各自的局部领域取得了一些进展, 然而在检修管理领域, 由于业务的复杂性和多样性, 未有一种方法能够较好地实现预定目标, 其实用性也未达到要求。

本文使用基于关系数据库的模型描述方式, 直接描述业务对象的数据结构、对象间的关系、对象展现策略、工

收稿日期: 2022-01-06

作者简介: 周宁(1972—), 硕士, 高级工程师, 从事企业软件架构设计及应用研究工作。

作流等要素；采用以数据结构复用支撑软件复用等方法，提出一种解释引擎，将模型直接解释成可运行的应用系统；同时给出了定制代码与解释引擎融合及协同运行方案，以期提升效率的同时较好地解决功能精细化问题。

1 总体框架

如图1所示，开发人员或系统管理员使用建模工具创建或维护模型，保存于关系数据库。此时，系统具备定义信息、描述系统的模板，但是没有任何业务数据，可称之为“骨架”。然后解释引擎读取模型，将模型直接解释成可运行的应用系统，由模型数据在运行期动态决定应用系统的外观和行为，此时系统界面已呈现出来，能供业务人员操作，并做出适当的响应。最后，业务人员录入业务数据，保存于关系数据库，此时系统变得丰满起来。

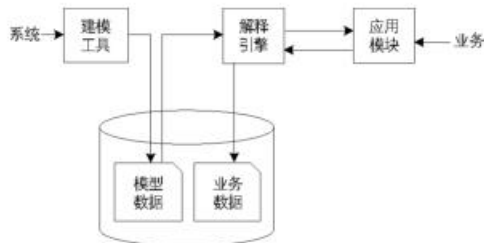


图1 低代码平台总体框架

2 模型

现有的UML建模本质是以图形描述系统实现方案，仍需开发人员以大量代码去实现相应的应用系统，因而效率低下，维护和扩展困难，属于一种静态建模。本文的建模方式抛弃了对实现的描述，而是以数据取代图形，描述系统结构、特性、外观及行为，是一种更加面向结果及面向业务的建模方式。相较于图形，数据的结构化程度更高，规范性和一致性更好，更容易被程序解析，因而更加便于实现应用生成自动化，该建模方式属于一种动态建模。

动态模型包含对象模型、视图模型、关联模型、属性选项、界面方案、工作流模型等要素。应用模型作为容器对上述要素进行打包，统一提供给解释引擎生成应用。

2.1 对象模型

对象模型描述现实世界业务对象，建模工具根据对象模型的描述，在关系数据库中建立相应的表。在图2中，“对象模型”为主表，描述对象的基本信息及在关系数据库的存储表名；“业务属性”为从表，描述主表中各对象的多个属性，包括属性名、数据类型及长度等。模型提供了中立数据类型及其与数据库管理系统数据类型的映射关系表，以支持多种类型的数据库。

2.2 界面方案

界面方案定义对象的展现策略，由主表“界面方案”，

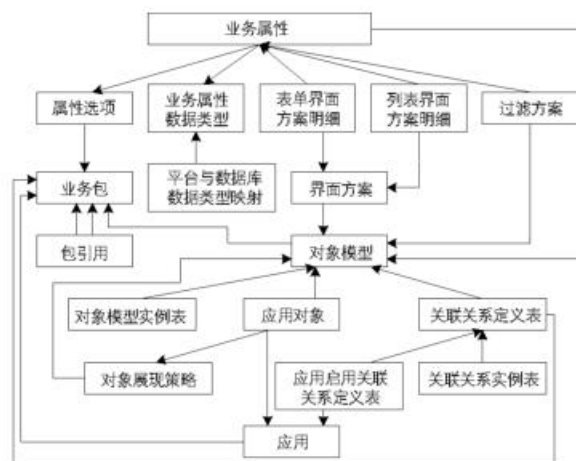


图2 动态模型数据结构

从表“列表界面方案明细”及“表单界面方案明细”组成。“界面方案”描述基本信息，“界面方案明细”定义每个界面方案中对象属性的可见性、展现顺序、可编辑性、是否必填、宽度、高度等特性。“界面方案”是“对象模型”的从表，一个对象模型可以有多个界面方案，支持按需启用。参见图2中的“对象展现策略”表及“应用对象”表，其中“应用对象”表描述特定应用中的对象模型启用的展现策略。

2.3 关联模型

关联模型定义对象模型间的对应关系，参见图2主表“关联关系定义表”及从表“关联关系实例表”，其核心字段包括“源类型ID”“目标类型ID”“源业务数据ID”“目标业务数据ID”。

2.4 属性选项

针对部分属性，需要在事前给定的枚举值中选择录入的情况，以属性选项进行建模，参见图2“属性选项”表，描述了“业务属性”对属性选项的引用。

2.5 应用模型

应用模型是一种容器，描述应用包含的基本模型要素，以“应用”表定义应用的基本信息，以“应用启用关联关系定义表”描述其包含的关联关系，以“应用对象”定义所包含的业务对象模型及其展现策略。需要在应用界面展现一棵树的场景，则以“根节点类型”及“根节点标识”定义树的起点。

3 解释引擎

解释引擎负责将存储于关系数据库中的模型直接解释成可运行的应用系统，其基本思路为以应用模型为入口，查询所包含的基本模型要素，根据基本模型要素，在运行期动态创建列表、表单、树、工具栏及其按钮等界面控件，然后查询数据并按照展现方案的定义填充数据到界面控件，完成应用系统外观的生成。

基于以数据结构复用支撑软件复用的思路，解释引擎预设了通用操作接口及其实现，首先根据模型描述决定需要操作的对象，然后重定向到相应对象进行操作，例如增删改查、打印、分享、导入导出等，完成应用系统行为的生成。描述模型的数据结构是标准化可复用的，对业务对象的操作由查询模型表之后动态拼接而成的 SQL 语句执行，因而可以在业务对象数据表结构差异化条件下实现常用功能的复用，这样的解释引擎具备较强的通用性。

3.1 列表页面的自动生成

解释引擎自动生成通用列表页面的过程如图 3 所示。对于任何对象模型，该生成过程无需修改，体现了基于软件复用动态决定应用系统外观的方法。

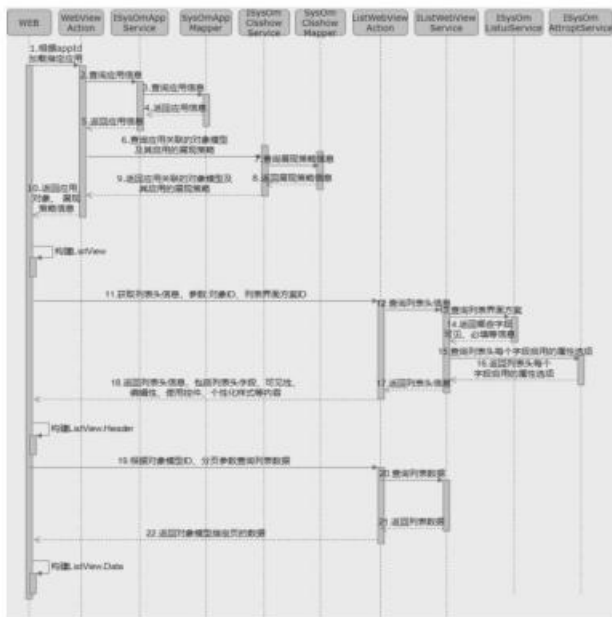


图 3 列表页面自动生成时序图

3.2 列表页面保存数据功能复用

基于解释引擎保存列表数据的过程如图 4 所示。图 5

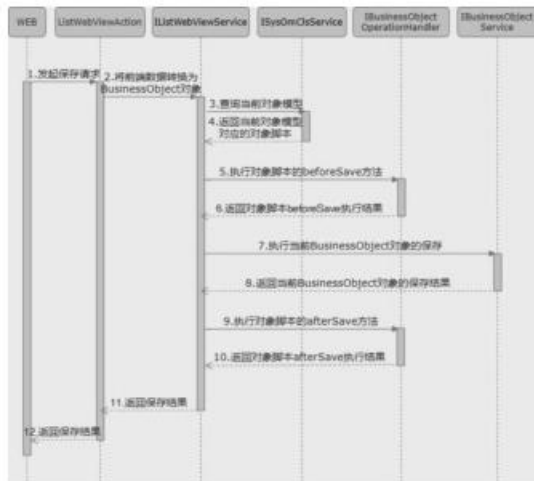


图 4 列表页面数据保存时序图

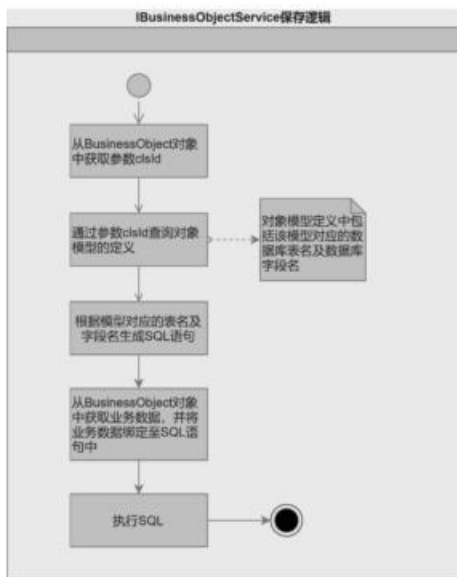


图 5 在不同业务对象之间复用数据保存功能原理图

解释了图 4 中 IBusiness-ObjectService 在不同业务对象之间复用保存数据功能的原理，体现了基于解释引擎动态决定应用系统行为的方法。

3.3 扩展机制

参见图 4 “5. 执行对象脚本的 BeforeSave 方法”，每个对象基于此机制按时序触发事件函数，具体执行类可添加到对象模型里并在事件函数中被调用。这样就将解释引擎的通用功能与定制代码的专用功能结合起来，兼顾了高效率与精细化需求，从而使低代码平台满足复杂场景的需要并达到实用化水平。

4 应用效果及分析

建立的对象模型及其生成的列表页面如图 6、图 7 所示。



图 6 创建对象模型



图7 基于对象模型零代码生成的录入页面

基于上述方案研发的低代码平台已在国家电网、中国石油、中小商贸公司等不同业务领域大量应用,由于业务逻辑复杂程度的差异,相较于传统开发方式,低代码平台开发的业务管理系统的代码量减少40%至70%不等,开发效率提升了2至3倍。

5 结语

限于篇幅,本文仅就低代码平台实现方案的核心要点进行了论述,还有大量的技术实现方案,例如树形结构的生成、动态模型与 workflow 模型的整合及协同、系统扩展点的设计与实现、工具栏按钮的自定义等未能一一介绍。当前,业务过程管理系统的开发仍然复杂、缓慢、昂贵,低代码平台实质是软件的再一次分层,以解决业务描述和快速应用生成的问题。

参考文献

- [1]Schölzel C,Blesius V,Ernst G,et al.Characteristics of mathematical modeling languages that facilitate model reuse in systems biology:A software engineering perspective[J].NPJ Systems Biology and Applications,2021,7(1):27.
- [2]Bardini R,Politano G,Benso A,et al.Multi-level and hybrid modelling approaches for systems biology[J].Comput Struct Biotechnol J.,2017,15:396-402.
- [3]Viktoria S,Nils B.Understanding coordination in global software engineering:A mixed-methods study on the use of meetings and Slack[J].The Journal of Systems and Software,2020.
- [4]邵志胜,张国富,苏兆品,等.基于软件体系结构和广义差分进化的测试资源动态分配算法[J].计算机应用,2021(12):3692-3701.

- [5]Graña M,López-Guede J,Muñoz J,et al.Special issue SOCO-CISIS 2018:New trends in soft computing and computational intelligence in security and its application in industrial and environmental problems[J].Neurocomputing,2021,452:414-415.
- [6]YANG Z,QIU Z,ZHOU Y,et al.C2AADL_Reverse:A model-driven reverse engineering approach to development and verification of safety-critical software[J].Journal of Systems Architecture,2021.
- [7]Derakhshandi M,Kolahdouz R,Troya J,et al.A model-driven framework for developing android-based classic multiplayer 2D board games[J].Automated Software Engineering,2021,28(2).<http://10.1007/S10515-021-00282-1>.
- [8]Daun M,Brings J,Obe P,et al.Reliability of self-rated experience and confidence as predictors for students' performance in software engineering[J].Empirical Software Engineering,2021,26(4):75-78.
- [9]Hakan Kekül,Burhan Ergen,Halil Arslan.A new vulnerability reporting framework for software vulnerability databases[J].International Journal of Education and Management Engineering(IJEME),2021,11(3):11-19.
- [10]Isaac Nyabisa O,Matteo M,Stephen K,et al.A survey on mobile applications for smart agriculture[J].SN Computer Science,2021,2(4):1-16.
- [11]Amarjeet Prajapati.Software package restructuring with improved search-based optimization and objective functions[J].Arabian Journal for Science and Engineering,2021:1-21.
- [12]Magomedov I,Bagov A,Elmurzaev A.Comparative analysis of CAD software packages for engineering design[J].IOP Conference Series:Materials Science and Engineering,2021.<http://10.1088/1757-899X/1155/1/012083>.
- [13]Aliakbar Karizani.Can a software engineer be a successful entrepreneur? A conceptual framework model[J].Journal of Hotel and Business Management,2021,10(5):1.
- [14]Damien I,Kate H,Luke J,et al.Research Software Engineering with Python:Building software that makes research possible[M].CRC Press,2021.
- [15]Shrikanth N,Nichols W,Fahid F,et al.Assessing practitioner beliefs about software engineering[J].Empirical Software Engineering,2021,26(4).<http://10.1007/S10664-021-09957-5>.
- [16]Petersen K,Carlson J,Papatheocharous E,et al.Context checklist for industrial software engineering research and practice[J].Computer Standards & Interfaces,2021,103541.<http://10.1016/J.CSI.2021.103541>.