中图分类号：TP311.5

论 文 编 号 ：10006SY1106514

北京航空航天大學

硕士学位论文

消息中间件设计模型性能分析

作 者 姓 名　　徐 龙

学 科 专 业　　计 算 机 软 件 与 理 论

指 导 教 师　　张 莉　教 授

培 养 院 系　　计 算 机 学 院

Model-based Performance Analysis for Message Oriented Middleware Design

A Dissertation Submitted for the Degree of Master

Candidate：Xu Long

Supervisor：Prof. ZhangLi

School of Computer Science & Engineering

Beihang University, Beijing, China

中图分类号：TP311.5

论 文 编 号 ： 10006SY1106514

硕士学位论文

消息中间件设计模型性能分析

作者姓名	徐龙	申请学位级别	工学硕士
导师姓名	张莉	职　　称	教授
学科专业	计算机软件与理论	研究方向	软件性能分析
学习时间自	年　月　日	起　　至	年　月　日止
论文提交日期	年　月　日	论文答辩日期	年　月　日
学位授予单位	北京航空航天大学	学位授予日期	年　月　日

关于学位论文的独创性声明

本人郑重声明：所呈交的论文是本人在指导教师指导下独立进行研究工作所取得的成果，论文中有关资料和数据是实事求是的。尽我所知，除文中已经加以标注和致谢外，本论文不包含其他人已经发表或撰写的研究成果，也不包含本人或他人为获得北京航空航天大学或其它教育机构的学位或学历证书而使用过的材料。与我一同工作的同志对研究所做的任何贡献均已在论文中作出了明确的说明。

若有不实之处，本人愿意承担相关法律责任。

学位论文作者签名：_____ 日期： 年 月 日

学位论文使用授权书

本人完全同意北京航空航天大学有权使用本学位论文（包括但不限于其印刷版和电子版），使用方式包括但不限于：保留学位论文，按规定向国家有关部门（机构）送交学位论文，以学术交流为目的赠送和交换学位论文，允许学位论文被查阅、借阅和复印，将学位论文的全部或部分内容编入有关数据库进行检索，采用影印、缩印或其他复制手段保存学位论文。

保密学位论文在解密后的使用授权同上。

学位论文作者签名：_____ 日期： 年 月 日

指导教师签名：_____ 日期： 年 月 日

摘　要

随着企业规模的扩张，数据量的迅速增长，消息中间件技术正受到越来越多的关注。消息中间件虽然能够有效的解耦消息发布者与消息接收者，但额外的消息处理逻辑增加了消息的发送延迟。这在时间性能要求不高的企业信息管理系统中，不会产生较大的问题。但是在像船舶控制系统这样的实时领域，过高的消息延迟是无法满足其业务需求的。

设计人员在为实时系统设计、实现消息中间件时，必须考虑其性能需求。如果仅仅在系统实现后对系统进行测试，可能会导致系统的重构。因为系统性能问题一般是由设计缺陷造成的，不是代码优化能够解决的。但是在系统设计阶段，由于缺乏有效的工具，设计人员无法对系统设计的性能进行量化评估。为此本文拟设计一种基于设计模型的性能分析方法，该方法能够对消息中间件的系统设计进行量化分析，为设计人员评估系统性能提供量化依据。本文的主要工作内容如下：

首先，为了分析消息中间件系统性能，需要确定具体的建模内容。本文首先根据消息中间件的系统特性及本文的目标，定义了一组性能度量目标；并由性能度量目标出发，确定了进行性能分析需要建模的具体内容；最后详细介绍了本文定义的消息中间件性能分析建模语言 MePAL。

接下来，根据 MePAL 语言能够描述的系统行为，以及系统执行过程中需要请求的资源，本文设计并实现了性能仿真指令以及具体的性能分析方法。性能仿真指令是对系统动态行为及其资源请求的抽象描述，以简化性能分析方法的设计实现。性能分析方法基于离散事件仿真原理，将系统中的软硬件资源抽象为优先权队列，并通过本文自定义的事件驱动性能分析系统的运行。

最后，本文设计并实现了消息中间件设计模型性能分析系统。为了验证本文提出方法的有效性，以某船舶控制系统中的消息中间件为背景，本文对两种系统设计进行了性能分析，并同实际系统的性能测试结果进行了对比，最后对实验结果进行了总结和分析。实验结果表明本文的方法可以有效的分析系统性能，为系统开发人员评估系统性能，定位性能瓶颈提供数据支持。

关键词：性能分析，消息中间件，设计模型

Abstract

With the expansion of business scale, rapid growth in the amount of data, message oriented middleware is getting more attention. Although messaging oriented middleware can effectively decouple the message publisher and recipient of the message, but the additional message processing logic increases the message transmission delay. In enterprise information management systems, which have no demand for real time message processing, this will not lead to serious problems. However, in domains such as real-time command and control system of ships, high message delay is unable to meet their business needs.

When design and implement the message oriented middleware for real-time systems, performance requirements must be considered. If it is done only after implementation, the system may result in reconstruction as performance problems are generally caused by defects in the system design, which could not be solved by code optimization. However, at system design stage, it's difficult to estimate system design performance quantitatively for the lack of effective tools. This paper intends to propose a model-based method for performance analysis, which could analyze the message oriented middleware system design quantitatively, to provide a quantitative basis for designers to evaluate system performance. The main contents of this paper are as follows:

First, in order to analyses performance of message-oriented middleware, we need to determine the information need to be modeled. On the basis of message-oriented middleware and the goal of this paper, we define a set of performance metrics. From the performance metrics, information need to be modeled are discussed. After that, we give a detailed description of the Message-Oriented Middleware Performance Analysis Language-MePAL.

Next, based on the system behavior and resource requirements modeled by MePAL, we design and implement the performance simulation instructions and the method to analyze system performance. Performance simulation instructions is used to describe the dynamic behavior of the system as well as the resource requirements associated with the behavior, so as to simplify the implementation of performance analysis algorithm. Performance analysis

method is based on discrete event simulation theory. It abstracts the system hardware and software resources into priority queues, and drive the simulation system with self-defined event defined in this paper.

Finally, we design and implementation the performance analysis system for message oriented middleware. In order to verify the effectiveness of the method proposed, we design an experiment based on a message oriented middleware within a ship control and command system. Two different system designs are analyzed with the method proposed in this paper, and then one of them is chosen to compare with its implementation. At last we summarize and analyze the experiment results. Experiment results show that the method proposed in this paper can effectively analyze the system performance, providing a basis for quantitative assessment to system developers.

Keywords： Performance analysis, Message-oriented middleware, Design model

目　录

图目录

表目录

第一章　绪论

本章首先介绍了论文的选题背景和意义。接下来提出本文中需要解决的问题：在设计阶段分析消息中间件系统设计的性能，为设计人员评估系统设计、定位性能瓶颈提供量化依据。然后介绍了本文的研究目标以及研究内容，最后给出了本文的组织安排。

1.1 论文的选题背景与意义

随着企业规模的扩张，数据量的迅速增长，信息管理、数据处理技术的更新换代，企业信息管理系统必须对现有功能进行合并、添加以及扩展，满足不断变更的需求。这可能会将整个企业信息管理系统划分为多个异构的子系统，每个子系统具有各自独立的功能，彼此之间无法通信，成为信息孤岛[1]。

将这些独立的功能子系统集成在一起，最有效的方法不是重新开发一套完整的信息管理系统，而是提供一种允许异构子系统间通信的解决方案。消息中间件能够屏蔽异构子系统之间的不同，允许它们通过统一的接口彼此交互[2]，从而有效的解决上述问题。

消息中间件为外部应用提供了两种消息模型，点到点的消息模型[3]和基于订阅发布的消息模型[4]。二者的不同之处主要是，点到点的消息模型中一个消息只能够发送给一个接收者，而发布订阅模式对消息接收者的数量没有限制。因而基于发布订阅模式的消息中间件得到了较为广泛的应用[5]。

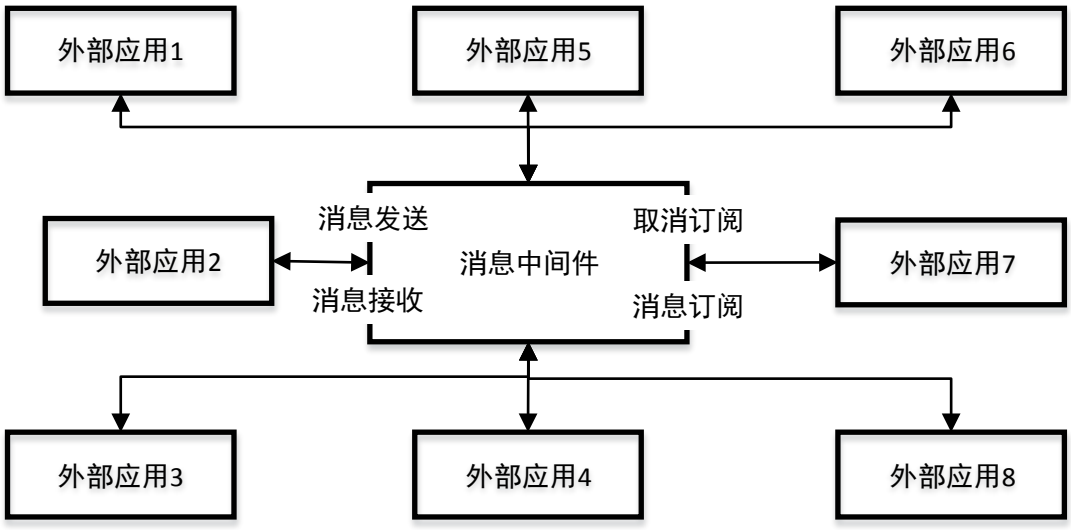

图 1　消息中间件及外部应用[2]

基于发布订阅的消息中间件为外部应用提供了消息发布、消息订阅、消息接收以及取消订阅四个标准接口，如图 1 所示。外部应用可以通过消息订阅接口在消息中间件中注册感兴趣的消息[6]，并通过取消订阅接口取消之前的订阅；还可以通过消息发布接口将产生的消息传给消息中间件，并由它确定该消息的接收者。消息中间件可以部署在一个计算节点上，也可以部署在多个计算节点组成的分布式系统中。

消息中间件系统通过解耦消息发布者与消息订阅者，为外部应用提供了异步通信的能力[7]，避免了同步通信固有的限制。基于消息中间件通信的外部应用不必等待已发送消息的反馈，便可以继续自己的处理工作，而不是一直处于阻塞状态[8]。

消息中间件虽然能够有效的解耦消息发布者与消息接收者，通过异步消息发送有效提高了外部应用的系统效率。但额外的消息处理逻辑，增加了消息的发送延时。这在对时间性能要求不高的企业信息管理系统中，不会产生较大的问题。但是在像船舶控制系统[9]这样的实时领域，较高的消息延迟是无法满足其业务需求的。

设计人员在为实时领域设计、实现消息系统中间件时，必须考虑该领域的性能约束。如果仅在系统实现后对其进行测试，当性能测试结果无法满足要求，可能会导致系统的重构，因为系统性能问题一般是由系统设计缺陷造成的，不是代码优化能够解决的[10]。但是在消息中间件设计阶段，由于缺乏有效的方法和工具，往往无法对现有设计的性能进行量化评估。

综上所述，消息中间件为异构子系统提供了基础的通讯设施，其本身的性能会对上层应用的总体性能产生重要的影响[11]。除了受实现细节和部署平台的影响外，消息中间件的设计在很大程度上决定了其性能的优劣。但是在消息中间件的设计过程中，设计人员目前缺乏有效的手段在设计阶段量化评估现有设计方案的性能，可能延迟性能问题的发现时间，导致已实现系统的重构，浪费人力和时间。

为了在设计阶段分析消息中间件系统设计的性能，作为设计人员评估系统设计的量化依据，本文拟提出一种方法，实现下面两个目标：

1. 基于消息中间件的设计模型预测系统设计的性能；
2. 基于消息中间件的设计模型为性能瓶颈的定位提供数据支持。

1.2 问题的提出及研究内容

1.2.1 本文问题的提出

UML[12]是一种在设计模型中广泛使用的建模标准，该语言本身提供较为灵活的扩展机制，而且有很好的工具支持。由于 UML 的普适性和灵活性，相较于其它建模语言，研究以 UML 设计模型为基础的性能分析具有更为重要的意义。因此本文的研究以 UML 设计模型为基础。UML 描述的设计模型通常包括静态结构和动态行为两部分。静态结构使用包图、类图描述，动态行为则采用顺序图、状态图或者活动图描述。在设计阶段量化分析系统设计的性能，仅有系统的静态结构和动态行为是不够的[13]，还需要为设计模型添加一些性能相关信息；在对性能相关信息进行充分描述的基础之上，需要一种基于设计模型的性能分析方法，用来获得度量系统设计性能的数据，作为设计人员进行性能评估的量化依据。

综上，本论文主要解决以下两个关键问题：

1. 需要在消息中间件设计模型中添加哪些性能相关信息，以及如何表示这些信息；

2. 如何分析面向性能分析的设计模型，以及需要输出哪些数据，作为设计人员评估系统设计性能的量化依据。

1.2.2 本文的研究内容

本文的研究内容主要由以下几个方面组成：

1. 根据消息中间件的特点，确定性能度量目标、需要为设计模型添加的性能相关信息，以及这些信息在设计模型中的表示方法。

目前在设计阶段分析系统设计性能的方法，大多是针对一般性的系统，没有考虑消息中间件这类系统的特点。本文在相关研究的基础上，根据消息中间件的特性，选取、定义了用于分析消息中间件系统设计性能的性能度量目标。由此出发，确定了需要在设计模型中添加的性能相关信息，为之后的性能分析提供足够的数据。设计模型一般使用 UML 描述，它本身并不具备建模这些性能相关信息的能力，因而需要通过对 UML 进行扩展以支持性能相关信息的建模，在此本文借鉴了 MARTE[14]中的部分概念。

2. 提出一种自动化的设计模型性能分析方法，该方法能够自动分析设计模型，输出所需的量化结果，同时具备良好的效率。

为了减轻开发人员的负担，方便相关人员在设计阶段评估系统设计的性能，本文提出了一种自动化的设计模型性能分析方法，该方法解析面向性能分析的设计模型，将其转换为其它易于处理的格式，并通过多次运行获得性能度量目标的平均运行结果。由于消息本身在消息中间件中扮演着重要角色，还需要该方法能够追踪消息在系统中的传输路径、传输延迟等。

3. 为了验证本文提出方法的可行性，本文设计并实现了消息中间件性能分析系统。

为了验证本文提出方法的可行性，本文设计并实现了消息中间件性能分析系统，然后基于该系统对某船舶控制系统中的消息中间件进行了性能分析实验，最后对分析结果进行讨论和总结。实验结果证明本文提出的方法能够较好的分析消息中间件系统设计的性能。

1.3 主要贡献与组织安排

1.3.1 本文的主要贡献

具体来说，本文的主要贡献可以概括为以下几个方面：

1. 提出了一种面向消息中间件性能分析的建模方法

通过分析消息中间件系统的性能度量目标，本文首先确定了需要在设计模型中添加的性能相关信息，以及建模这些信息的方法；然后通过扩展 UML 语言定义了与性能和消息操作语义相关的建模元素。

2. 提出一种基于消息中间件设计模型的性能分析方法

为了解耦设计模型存储格式与性能分析方法，简化性能分析方法的设计实现，本文设计并实现了性能仿真指令。并根据离散事件仿真的基本原理，设计了基于消息中间件设计模型的性能分析方法。

3. 设计并实现了面向性能分析的建模工具和模型仿真工具

基于面向消息中间件性能分析的建模方法，本文以扩展型的方式为已有的 UML 建模工具提供了性能分析的建模功能。基于带有性能相关参数的设计模型以及本文提出的模型分析方法，本文设计并实现了消息中间件系统设计性能分析工具。该工具能够辅助设计人员在设计阶段量化评估消息中间件系统设计的性能，具有一定的实用价值。

1.3.2 本文的组织安排

围绕着上述研究内容，本文的整体章节安排如下：

第一章 绪论。本章首先介绍了论文的选题背景和意义。接下来提出本文中需要解决的问题：在设计阶段分析消息中间件系统设计的性能，为设计人员评估系统设计提供量化依据。然后简要介绍了问题的研究思路以及本文的研究内容，最后给出了本文的组织安排。

第二章 相关研究。本章首先给出了软件性能工程的概念以及简单介绍，并举例说明了软件性能工程在实际开发过程中的应用；接下来给出了同消息中间件相关的性能分析研究；然后对软件系统非功能属性建模方法的相关研究进行了详细描述；最后给出了在设计阶段分析软件系统设计模型的方法，并对这些性能分析方法进行了比较和分析。

第三章 面向性能分析的设计模型。为了在设计阶段分析消息中间件系统设计的性能，需要为设计模型添加性能相关参数，为第四章介绍的仿真分析方法提供足够的信息。本章首先根据消息中间件的系统特性以及本文的目标，定义了一组性能度量目标，作为性能相关参数的建模依据；接下来给出了性能相关参数包含的内容，作为扩展 UML 的依据；最后简要介绍了 MARTE 语言，通过借鉴 MARTE 语言对性能相关参数的建模要素，给出了利用 UML 的 Profile 扩展机制定义的性能标签和消息操作语义，提出了面向性能分析的消息中间件建模语言 MePAL（Message-Oriented Middleware Performance Analysis Model Language）。

第四章 性能仿真指令与仿真方法的设计。为了解耦消息中间件系统设计模型的存储格式和仿真分析方法，简化性能分析方法的设计实现。基于消息中间件性能分析模型，本文设计并实现了性能仿真指令（Performance Simulation Instruction，PSI）。PSI 从 MePAL 语言描述的系统行为及行为执行所需的资源出发，通过分支、顺序和循环控制结构指令描述系统执行流程；通过具体的指令属性描述资源请求。本章首先根据 MePAL 语言的建模能力，设计并实现了性能仿真指令，用于描述由消息中间件设计模型中提取的性能仿真信息；然后介绍了基于性能仿真指令的仿真分析方法。

第五章 消息中间件设计模型性能分析系统及实验验证。基于前文选取并定义的消息中间件系统建模语言、定义的性能仿真指令和设计的基于离散事件的性能分析方法，本章设计并实现了消息中间件系统设计模型性能分析系统，并基于该系统对某船舶控制

系统的消息中间件系统进行了性能分析实验，以验证本文的有效性。本章首先给出了消息中间件系统设计模型性能分析系统的需求、架构与详细设计，并通过实际使用场景展示了系统的功能；接下来，为了验证基于设计模型进行性能分析的有效性，基于某船舶控制系统消息中间件系统的设计模型，进行了性能分析实验，然后对实验结果和实际执行结果进行了分析和比较，实验结果证明本文的方法可以有效的分析消息中间件系统设计的性能。

第二章　相关研究

本章首先给出了软件性能工程的概念以及简单介绍，并举例说明了软件性能工程在实际开发过程中的应用；接下来给出消息中间件设计阶段或者利用模型分析其性能的相关研究以及发展现状；然后对软件系统非功能属性建模方法的相关研究进行了详细描述；最后给出了在设计阶段分析软件系统设计模型的方法，并对这些性能分析方法进行了比较和分析。

2.1. 软件性能工程简介

软件性能工程[15]（Software Performance Engineer，SPE）由 C.U. Smith 于 1990 年提出，是一种构建符合性能目标的软件系统的系统化、量化方法。软件性能工程是一种工程学方法，避免了性能驱动开发和开发完成后再修补这两种极端的开发方式，允许用户在开发早期发现设计中存在的性能问题，并对不同的设计方案进行量化评估。

软件性能工程主要通过软件系统的用例以及描述用例的场景分析系统性能[16,17]。从开发人员的角度看，用例及其场景提供了对系统需求、体系结构和设计进行理解和文档化的一种方法。从性能角度看，用例有助于确定对性能要求较高的工作负载。图 2 给出了实践软件性能工程的一般过程：设计人员首先确定关键用例和性能度量目标，然后绘制用于性能分析的性能分析模型，并根据评估结果修改性能目标或者现有设计，直到设计能够满足既定的性能需求。

下面通过一个真实的案例说明软件性能工程的实践意义。某个已完成的软件系统由于无法通过扩大规模来支持合同中规定的用户数量，没有达到要求的性能目标，按照合同规定受到了处罚。在发现性能调优无法解决问题后，开发人员为系统的关键部分提出了一种新的设计方案。通过构建原系统的性能分析模型，开发人员在系统的同步策略以及系统的体系结构中发现了两个关键性的问题，通过解决这两个问题排除了原系统的性能问题。而新提出的设计方案通过对其设计模型进行性能预测，发现会导致更差的系统性能表现。

综上，软件性能工程为控制软件系统的性能提供了指导性的方法和意见，具有很高的实用价值。本文借鉴了软件性能工程实践的一般步骤，用于指导设计人员进行消息中

间件系统设计模型的性能分析。

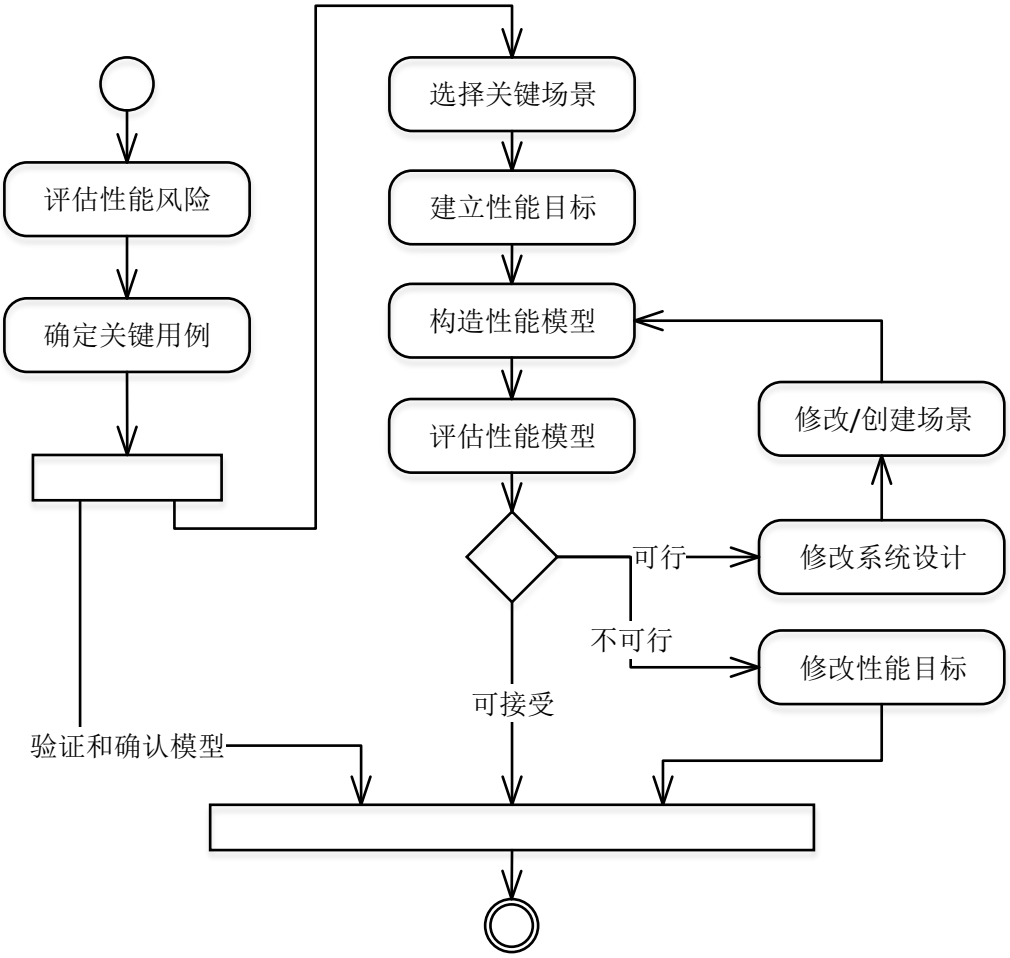

图 2 软件性能工程基本步骤

2.2. 消息中间件相关的性能分析研究

Baldoni 和 Contenti[18] 在 2003 提出的方法中，将基于发布订阅模式的消息中间件建模为经典的分布式系统，将分布式系统看作进程集合∏={p1, …, pn}，并定义了一个全局的离散时钟模拟系统运行时间。之后通过定义进程能够进行的操作，并以概率的方式描述每种操作的行为，来分析整个系统行为。这种方式较为简单，通过定义消息中间件外部接口的概率行为来模拟整个系统的行为，但是无法描述较为复杂的系统行为，也无法分析资源的利用率。

Rathfelder 和 Klatt[19] 在 2013 年提出了离散事件系统的性能分析方法，将离散系统建模为通过消息中间件通信的独立组件。该方法将消息中间件简单的抽象为信道，无法

分析路由策略对系统性能的影响，而且在仿真过程中无法动态的变更发布订阅关系。

Gero 和 Arnd[20]为了评估路由策略以及系统部署对消息中间件性能的影响，于 2009 提出了基于概率的性能分析模型，该模型基于 Petri 网，针对基于发布订阅消息模式，并具有层级网络拓扑结构的消息中间件。他们利用概率描述消息中间件系统中各组成部分的行为：如消息产生的速率，消息传输路径。然而，该方法缺乏对消息中间件系统行为的建模，无法给出较为详细的数据用于性能分析。

目前针对消息中间件设计模型进行分析的研究相对较少，而且已有研究仅针对消息中间件的部分特性进行分析，没有考虑消息中间件的整体性能。因而需要一种简单易用，能够较为准确的分析消息中间件系统性能的方法，辅助开发人员进行系统设计。

2.3. 非功能属性建模

软件系统的设计模型通常定义了系统的静态结构和动态行为，这对性能分析是不够的，还需要在设计模型中描述软件系统的某些非功能属性，为性能分析提供足够的信息。本小节首先介绍了目前常用的建模非功能属性方法，并对不同的方法进行比较；最后选取了适合本文内容的非功能属性建模方法。

2.3.1 基于 UML 的扩展

如果我们需要一种新的语言来建模软件系统，并且这种语言的语义是在已有 UML 建模元素的基础上添加数量或者语义限制，我们并不需要使用元对象机制[21]（Meta-Object Facility，MOF）重新开发一套建模语言，而是可以利用 UML 自身提供的扩展机制[22]。UML2.0 中的 Profile 包定义了一种可以在 UML 建模元素基础上构建领域特定的概念、图符的扩展机制。

Balsamo[23]首先利用 UML 类图定义了如图 3 所示的概念、属性，以及它们之间的关系，之后再利用 UML 的 Profile 机制对 UML 进行性能相关参数建模的扩展，从而允许设计人员将其集成到已有的建模工具中。

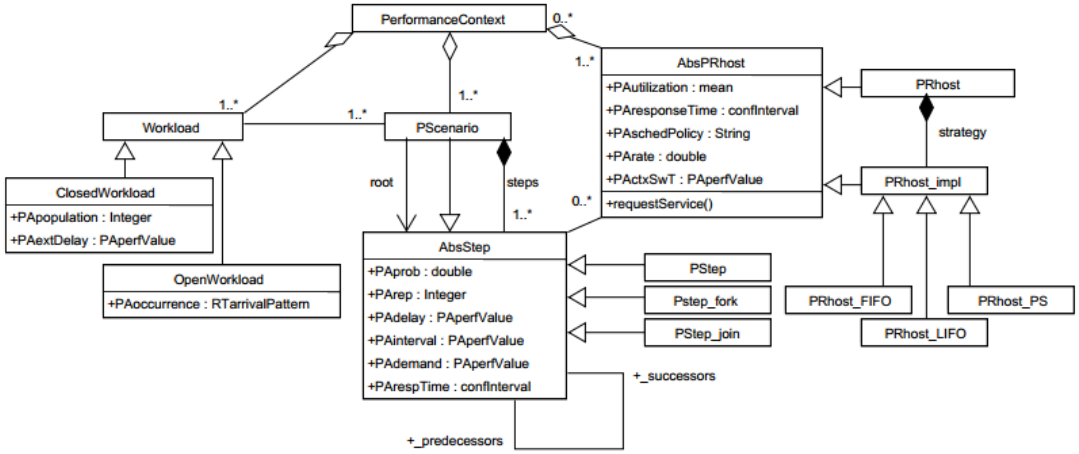

图 3 自定义非功能属性建模元素

2.3.2 基于组件的性能分析

Palladio 组件模型[24]（Palladio Component Model，PCM）是一种基于组件的性能分析模型，它允许设计人员以组件为基本单位设计系统架构，并利用活动图详细描述每个组件的内部行为。由于 Palladio 组件模型较为复杂，而且同本文的相关性较低，所以不再详细描述。

2.3.3 UML-SPT

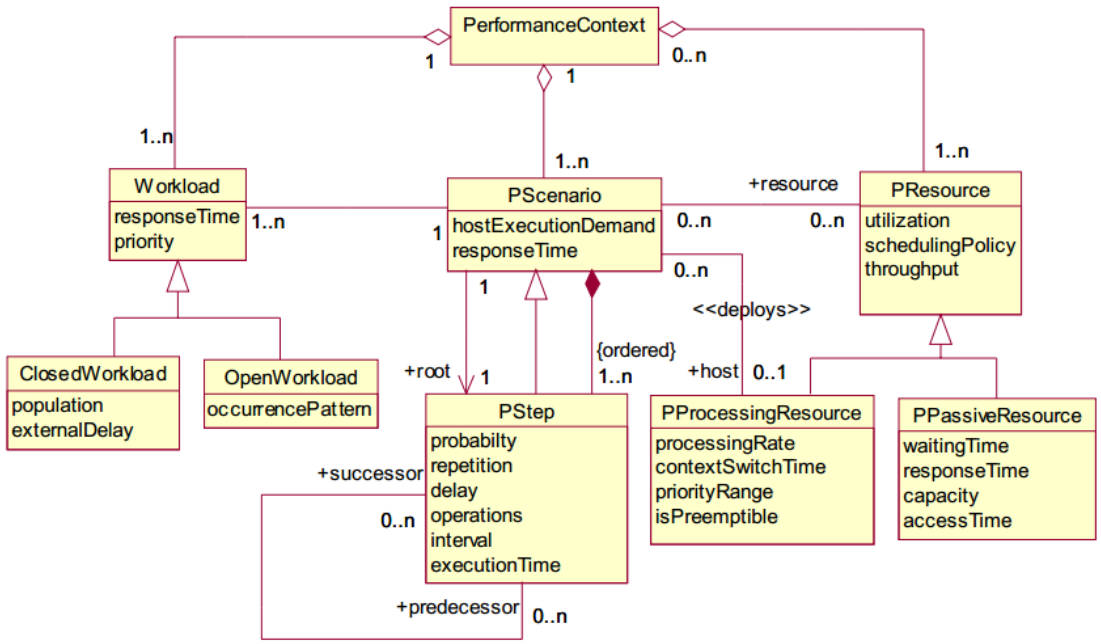

图 4 UML-SPT 性能分析相关建模元素

UML-SPT[25]（UML profile for Schedulability，Performance and Time）是 OMG 组织

提出的建模调度、性能以及时间的标准语言。其中同性能建模相关概念的元模型如图 4 所示。PerformanceContext 表示性能分析上下文，由外部负载（Workload）、系统行为（PScenario）以及系统资源（PResource）三部分组成。外部负载描述同外部环境的交互，分为封闭负载（ClosedWorload）和开放负载（OpenWorkload）两类；系统行为描述软件系统自身的动态行为；系统资源描述软件系统用到的各类软硬件资源。

2.3.4 MARTE 语言

随着 UML2.0 的提出，原有的非功能属性建模标准 UML-SPT 需要更新，而且 UML-SPT 在灵活性以及表达能力上有一定的欠缺。OMG 组织为了更新现有标准，通过 Profile 对 UML2.0 进行扩展，提出了实时和嵌入式系统的建模与分析（Modeling and Analysis of Real-Time and Embedded Systems，MARTE）标准，为实时嵌入式系统的时间性能、调度性能分析提供了标准化的建模元素。

MARTE 并不提供分析设计模型的方法，而是为分析方法提供建模支持。由于 MARTE 是对 UML2.0 的标准扩展，可以很方便的同 UML 描述的设计模型相结合，表示性能相关参数。而且 MARTE 自身具有良好的层次结构，可以通过扩展已有的建模元素，定义领域概念模型。另一方面，MARTE 是 OMG 组织提出的标准扩展，在工业界和学术界得到了广泛的应用，其实用性和良好的扩展性已经得到了较为充分的证明[26]。

2.3.5 建模语言的选择

本文的目标之一是选取或者自定义一种建模消息中间件系统设计中性能相关信息的形式化方法，允许开发人员方便快速在已有设计模型中添加性能分析需要的信息。Palladio 组件模型同 UML 设计模型具有较大的差距，而且不适合用来描述消息中间件系统设计；MARTE、UML-SPT 以及自定义的扩展都是基于 UML 的 Profile 扩展机制。其中 MARTE 是对 UML-SPT 的改进，具有丰富的表达能力以及扩展机制，而且是 OMG 组织提出的建模非功能属性的标准语言，因而本文选取 MARTE 作为消息中间件系统非功能属性的建模语言。

2.4. 设计阶段性能分析方法

设计人员给出系统设计的形式化描述—设计模型，并添加了性能分析需要的非功能属性之后，还需要对该设计模型进行量化分析，才能够评估当前设计的性能。本小节首

先介绍了目前常用的基于设计模型的性能分析方法，并对不同分析方法的优劣进行了对比和分析。

2.4.1 基于马尔科夫模型的分析方法

马尔科夫模型[27]（Markov Models）是一种形式化的概率模型，其核心思想是未来系统状态的条件概率分布仅仅取决于最近一个系统状态，同之前任何的系统状态都无关。马尔科夫模型由一组随机变量组成 X = {X(t): t ∈T}，其中 X(t)表示为 T * Ω -> S，其中 Ω 表示概率空间，T 为索引集合（通常指当前时间），S 为系统的状态空间。图 5 给出了马尔科夫模型的一个简单示例，其中状态空间 S 包括 1、2 和 3 几个系统状态；概率空间 Ω 为每个状态转移上的数字。

马尔科夫模型在软件系统的量化分析中具有重要作用，它可以很好的描述软件系统的动态行为，而且包括队列网络、Petri 网络在内很多其它性能分析模型都是在马尔科夫模型的基础上构建的。

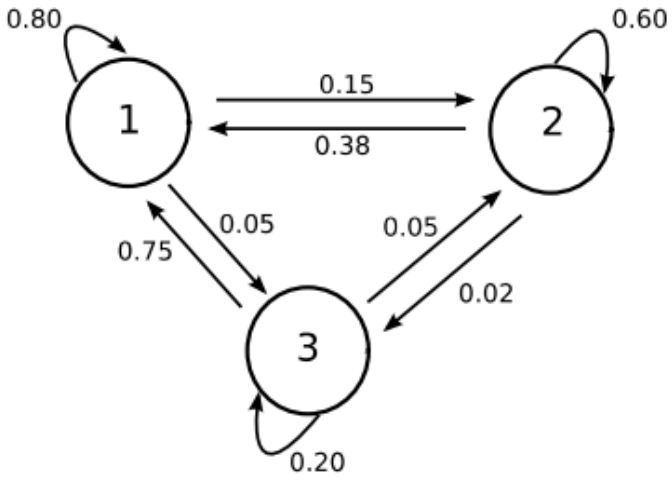

图 5　马尔科夫模型示例

目前已经有很多成熟的经典算法对马尔科夫模型进行分析，还有更为复杂的算法用来解决其空间爆炸问题。但是设计人员一般不熟悉马尔科夫模型，使用该模型分析设计模型的性能会加重设计人员的工作负担。所以一般先将设计人员利用常用的建模语言（一般为 UML）构建的设计模型，自动转换为马尔科夫模型[28]，然后再利用分析或者仿真算法对其进行性能分析，得到期望的性能分析结果。而且将 UML 模型转换为马尔科夫模型的方法已经得到广泛的研究和论证，具有较强的实用性。

受限于马尔科夫模型的表达能力[29]，基于该模型的性能分析方法主要用来建模软件

系统的状态，发现该系统的稳定状态，并通过计算系统在不同状态中的逗留时间，评估其时间性能，因而无法详细描述软硬件资源以及资源竞争对系统性能产生的影响。

2.4.2 基于队列网络的分析方法

队列网络[30]（Queueing Networks，QN）是一种形式化的数学模型，能够有效的描述软件系统中的资源竞争行为。队列网络在软件系统性能分析领域得到广泛应用，主要源于两方面的原因，一是它能够给出较为精确的性能分析结果，二是其具有高效的模型分析性能。

队列网络通常由两部分组成，一组表示系统资源的节点，这些节点之间可以交互；以及一组共享这些资源的用户。可以将队列网络表示为一个有向图，其中节点为系统资源，有向边表示可能的用户请求路径。队列网络中可以同时存在多种不同类型的用户，每类用户具有类似的系统行为（即请求路径和资源请求数量相似）。

队列网络构建性能分析模型的一般步骤为：首先定义该软件系统中包含的软硬件资源，这些资源的初始化数量，以及这些资源之间的互联结构；第二步是定义系统执行过程中用到的各类参数，包括资源请求的需求到达过程、资源请求者的类型、资源请求服务的延迟等信息。

队列网络能够有效的分析软件系统内部的资源竞争，但是需要设计人员在进行性能仿真时，为每类用户定义资源请求路径。这在系统资源不是很复杂的时候是可行的，如果系统过于复杂，就会极大的加重设计人员的负担。因而一般通过模型转换的方式，将UML 图描述的设计模型转换为队列网络，进行性能分析。

队列网络在软件争夺、异步处理方面的描述能力较弱，即便通过扩展（如扩展队列网络和分层队列网络），增加其对软件争夺和同步描述能力，依然无法准确的描述复杂的系统行为，限制了其在性能分析领域的应用[23]。

2.4.3 基于概率 Petri 网络的分析方法

概率 Petri 网络[31]是 Petri 网络的扩展，适合用来分析并发系统中并发行为的正确性。Petri 网主要由表示系统装填的节点（places），表示系统状态变化的状态转移（transitions），以及一组表示待处理任务的令牌（tokens）组成。

Petri 网能够有效的描述复杂的系统行为，即便是对分布式并发系统，都具有很好的效果。但是 Petri 网在建模资源竞争行为方面具有较大的缺陷，无法对软硬件资源竞争

对系统性能产生的影响进行有效分析。为了消除这方面的不足，有人提出了队列 Petri 网[32]，结合了两种模型的优点，建模软件行为和硬件资源。但这种队列 Petri 网络模型可能会产生空间爆炸问题，导致无法通过数学方法分析系统性能，只能通过仿真的方法来解决。

2.4.4 基于仿真模型的分析方法

设计模型的仿真分析可以分为以下三个步骤[29]：

1. 构建仿真模型；
2. 设计系统的执行场景；
3. 运行仿真程序，分析执行结果。

在转化为可执行代码之前，仿真模型是待分析的软件系统的抽象描述。仿真模型的构建通常分为三步：定义系统中可能出现事件的类型；定义不同类型之间的逻辑关系；定义产生、处理不同类型事件的仿真组件。

完成以上三步之后，就完成了仿真模型的构建工作，进而可以将其转换为可执行的仿真程序。仿真程序一般包括仿真引擎、系统数据结构、事件行为三部分，其中仿真引擎是仿真程序的核心，负责驱动整个仿真过程，包括时间驱动引擎和事件驱动引擎两大类。

仿真模型同设计模型具有相同的表达能力，能够描述复杂的系统行为，但是需要将设计模型转换为仿真模型之后，才能得到可执行的仿真代码。同时，仿真模型能够定义各种类型的事件，具有很好的扩展能力。但是仿真分析通常只能给出近似的分析结果，而且对于大型复杂系统，其运行时间也较长。

2.4.5 性能分析方法的比较

本文的目标是在设计阶段分析消息中间件系统设计的性能，为设计人员评估系统设计、定位性能瓶颈提供量化依据。根据绪论中的介绍，消息中间件系统一般部署在多台机器上，属于分布式系统；而且需要追踪消息在系统中的传递，获取消息的发送、接收时间，因而对性能分析方法的表达、扩展能力具有较高的要求。

仿真分析的方法虽然需要运行较长的时间，仿真模型的构建也具有较大的工作量，但是能够分析复杂的系统行为，特别是多用户场景下的并发行为。通过模型转换工具，可以将设计模型自动转换为仿真模型，最终转化为可执行代码，从而减轻了设计人员的

工作量。虽然某些场景下仿真分析结果的精度同其它方法有一定的差距，但是通过多次运行获得统计数据，可以弥补这方面的不足，因而本文采用基于仿真的方法分析软件系统设计模型的性能。

2.5. 本章小结

本章首先介绍了软件性能工程的概念及其应用；接下来给出了基于模型分析消息中间件系统性能的相关研究；然后介绍了在设计模型中表示非功能属性的方法，并对基于 UML 模型的自定义扩展，标准 UML-SPT 扩展和 MARTE 标准进行了比较和分析，选择了 MARTE 语言描述设计模型中的性能相关参数；最后通过比较各种模型分析方法，选取其中符合本文需求的一种，作为本文分析消息中间件系统设计性能的方法。在下一章中，本文将根据性能度量目标确定需要在设计模型中描述的性能相关参数，并通过扩展 UML 定义了消息中间件性能分析建模语言。

第三章　面向性能分析的设计模型

为了在设计阶段分析消息中间件系统设计的性能，需要在设计模型中添加性能相关参数，为第四章介绍的仿真分析方法提供足够的信息。本章首先根据消息中间件的系统特性以及本文的目标，定义了一组性能度量目标，作为性能相关参数的建模依据；接下来给出了性能相关参数包含的内容，作为扩展 UML 的依据；最后简要介绍了 MARTE 语言，通过借鉴 MARTE 语言对性能相关参数的建模要素，给出了利用 UML 的 Profile 扩展机制定义的性能标签和消息操作语义，提出了面向性能分析的消息中间件建模语言 MePAL。

3.1. 本文研究思路

根据第二章中的相关研究，本文设计了如图 6 所示的基于消息中间件设计模型的性能分析流程。设计人员首先按照系统需求，利用本文自定义的消息中间件性能分析建模语言（Message-Oriented Middleware Performance Analysis Model Language, MePAL），构建用于性能分析的设计模型；随后按照本文设计的转换规则，将面向性能分析的设计模型转换为本文定义的性能仿真指令及配置文件，交由性能仿真引擎解释处理；最终得到用于度量系统性能、定位性能瓶颈的性能度量目标，反馈给设计人员，作为修改系统设计的量化依据。

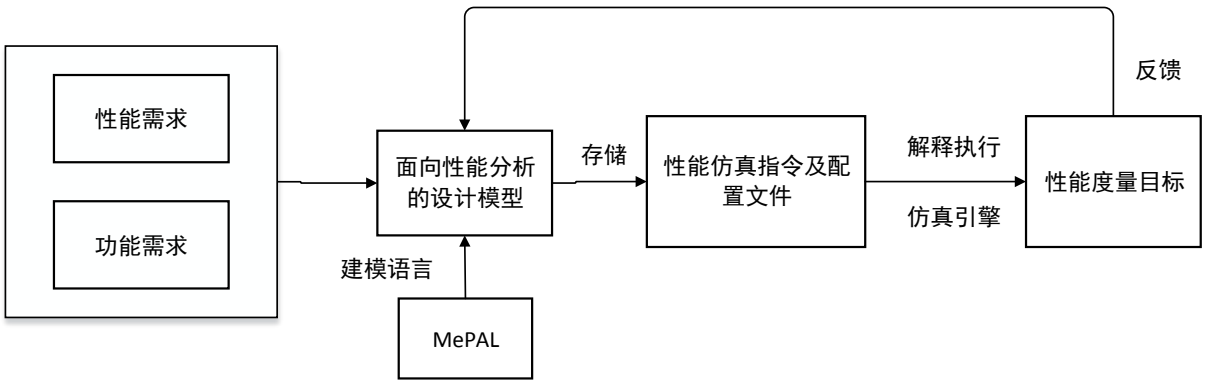

图 6　设计模型性能的分析过程

本文按照图 7 所示的研究思路，实现了图 6 中描述的性能分析过程。本文首先根据消息中间件的系统特性及本文的目标，定义了一组性能度量目标，为度量系统设计性能的量化依据，定位性能瓶颈提供数据支持；然后通过分析影响性能度量目标的因素，

确定了面向消息中间件性能分析的建模内容，并通过选取 UML 建模元素、借鉴 MARTE 语言扩展 UML，设计了消息中间件性能分析建模语言 MePAL；接下来根据 MePAL 建模语言，本文设计并实现了描述系统行为及资源需求的性能仿真指令；最后设计并实现了性能分析算法，分析利用 MePAL 语言建模的消息中间件系统设计。

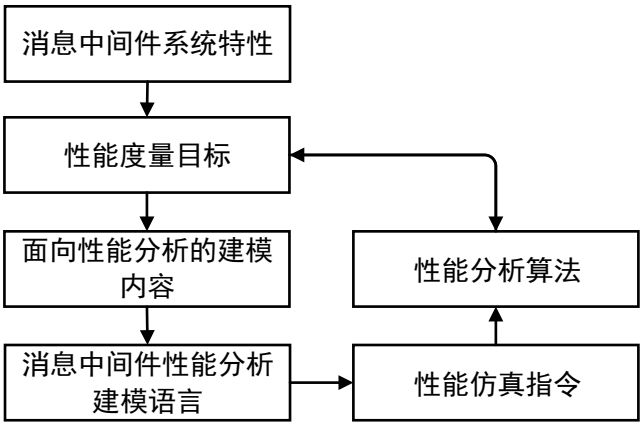

图 7　解决方案结构图

本章首先定义了一组的性能度量目标；然后确定了面向性能分析的建模内容；接下来给出了消息中间件性能分析建模语言，最后说明了建模面向性能分析设计模型的指导原则。性能仿真指令的定义以及性能分析算法的设计将在第四章中详细介绍。

3.2. 性能度量目标

软件系统的性能度量目标是指一类系统运行时产生的数据，为系统开发、测试以及维护人员在软件生命周期的不同阶段分析、评估软件系统的性能提供可靠的依据。相关人员还可以通过分析性能度量目标，找到影响软件系统性能的瓶颈，并对该部分进行优化、重构，达到提高软件系统性能，满足系统性能需求的目标。

本文的目标是在设计阶段定量的分析消息中间件系统设计的性能，并为性能瓶颈的定位提供数据支持。因而本节首先根据消息中间件的系统特性以及本文的目标，定义了一组性能度量目标；然后介绍了性能度量目标的具体内容。

3.2.1 性能度量目标的定义

消息中间件有订阅发布和点到点两种消息模型，其中点到点消息模型将每个消息接收者的数量限制为一，而发布订阅消息模型对每个消息接收者的数量没有限制。

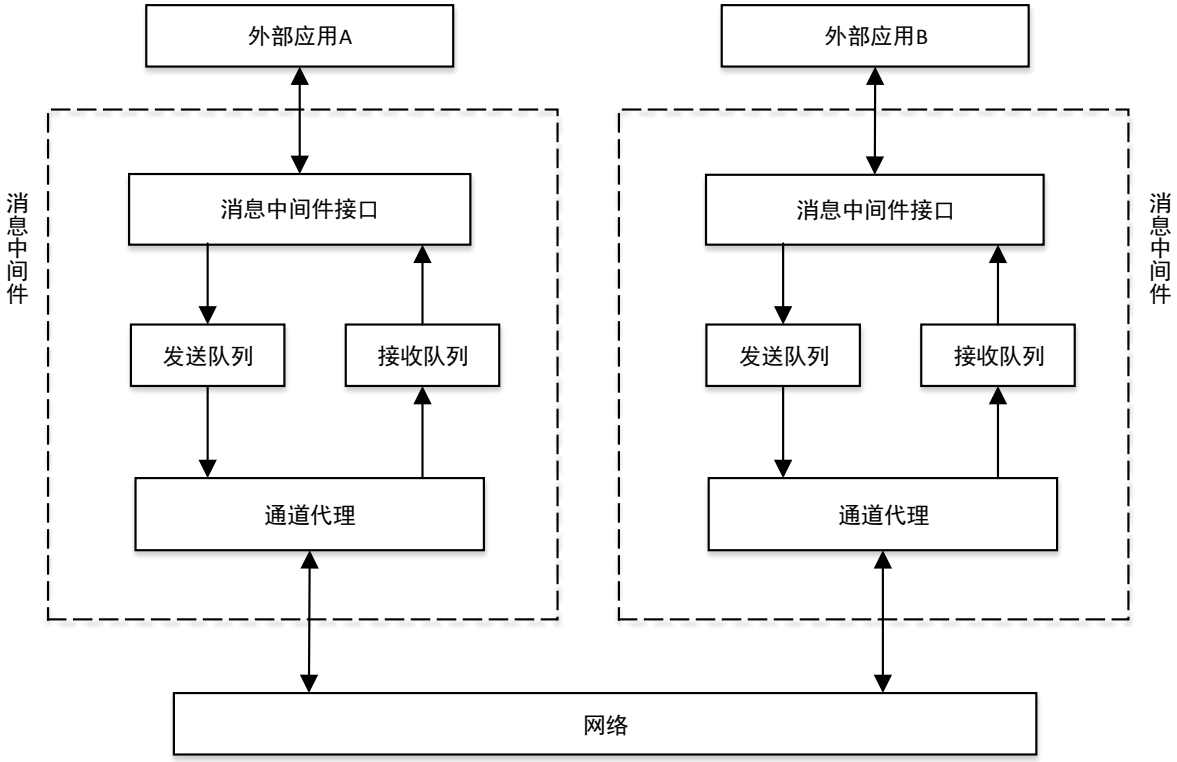

图 8 消息中间件系统的总体架构和工作流程

两种模型都可以用图 8 给出的消息中间件系统的总体架构和工作流程来描述。在该架构中，外部应用通过消息中间件接口将消息传送至发送队列；通道代理从发送队列中获取待发送的数据，确定消息接收者，然后通过网络传输至目标应用所在的计算节点；该计算节点上的消息中间件通过接收队列将消息发布给外部应用。二者的不同之处在于点到点模式下消息中间件只会确定一个接收者，而发布订阅模式下消息中间件会确定多个接收者。

通过该消息传送流程可以看出，消息中间件的外部应用主要关心消息发送时间以及系统响应时间。消息发送时间是指从开始发送消息，直到消息被外部应用接收经过的时间；系统响应时间指外部应用调用消息中间件提供的外部接口时，该接口的返回时间，可以看出系统响应时间是包含在消息发送时间之内的。因而本文选取消息发送时间以及系统响应时间来度量系统设计的性能。此外，图 8 中的发送队列[33]用于保存待发送的消息，其长度表明消息中间件当前阻塞的消息数量，可以从另一个侧面反映消息中间件系统性能，因而将其作为度量消息中间件系统设计性能的度量目标之一。

根据图 8 所示的消息处理流程，消息发送时间包括消息发送处理时间，网络传输时间和消息接收处理时间三部分。消息发送处理时间和消息接收处理时间主要由消息

中间件系统设计决定，而系统设计的性能同系统行为以及系统执行时需要的系统资源（包括计算资源、共享资源以及存储资源）有关。为了定位系统设计中可能存在的性能问题，本文引入了系统执行时请求的资源的占用率、进程队列长度以及系统方法执行时间作为性能度量目标。

网络传输时间不仅受消息中间件路由策略[7]的影响，还和执行环境的网络结构、网络设备相关。为了定位该部分可能存在的性能瓶颈，本文引入了网络资源占用率以及网络资源进程队列长度作为性能度量目标。

综上所述，本文定义了如表 1 所示的性能度量目标，为度量系统性能、定位性能瓶颈提供数据支持。

表 1　性能度量目标

性能度量目标	描述	功能
响应时间	系统对用户请求的响应时间；	度量系统性能；
系统方法执行时间	系统中某方法的执行时间；	定位系统瓶颈；
CPU 利用率	计算资源工作时间占系统执行时间的比率；	定位系统瓶颈；
CPU 请求队列长度	计算资源进程请求队列长度；	定位性能瓶颈；
存储资源利用率	存储资源工作时间占系统执行时间的比率；	定位性能瓶颈；
存储资源请求队列长度	存储资源进程请求队列长度；	定位性能瓶颈；
网络资源利用率	网络资源工作时间占系统执行时间的比率；	定位性能瓶颈；
网络资源请求队列长度	网络资源进程请求队列长度；	定位性能瓶颈；
共享资源请求队列长度	共享资源进程请求队列长度；	定位性能瓶颈；
消息发布时间	特定消息的发送时间；	度量系统性能；
消息接收时间	特定消息的接收时间；	度量系统性能；
消息队列长度	消息中间件中待发送消息的数量；	定位性能瓶颈；

3.2.2 性能度量目标的内容

为了更清晰、准确的理解性能度量目标，本文从以下四个角度对本文选取和定义的度量目标进行介绍：

1. 目标名称：该性能度量目标的名称；
2. 目标描述：对该性能度量目标的简要描述；

3. 影响因素：系统设计中影响该性能度量目标的因素；

4. 统计数据：获得该项度量目标需要统计的系统运行时数据；

5. 评估方法：如何利用该项度量目标评估系统性能（不止这一种评估方式，这里仅仅是给出一种可能的应用场景，以便于理解）。

下面本小节就从上面五个方面，介绍本文用到的性能度量目标：

1. 系统响应时间

目标描述：典型场景下整个系统的执行时间（典型场景指软件系统运行的主要场景，一般在分析系统需求是通过主要用例描述）；

影响因素：系统外部负载（负载指系统外部的输入，包括用户以及环境），软件或硬件资源的需求量，硬件资源性能，多线程中软件资源的竞争，系统自身行为的不确定性，操作系统进程调度策略，外部服务的执行时间（外部服务指系统调用的外部方法），系统的部署方式；

统计数据：系统响应用户服务的开始时间和结束时间（一般开始时间默认为 0）；

评估方法：同系统的性能需求进行比较，查看是否能够满足要求。

2. 系统方法执行时间

目标描述：典型场景下系统中某个方法的执行时间；

影响因素：方法输入参数，软件或者硬件资源的需求量，硬件资源性能，软件资源的竞争，操作系统的调度策略，调用的外部服务的执行时间；

统计数据：该方法的开始执行时间和结束时间；

评估方法：同系统总的执行时间进行比较，查看占用的时间比例，用来判断该方法是否是系统的性能瓶颈。

3. CPU 利用率

目标描述：CPU 的使用时间同系统执行时间的比率；

影响因素：计算资源需求量，操作系统进程调度策略，软件系统的部署；

统计数据：CPU 总的执行时间以及系统响应时间；

评估方法：通过 CPU 使用负载可以判断该节点的计算负载是否过重，如果过重，可以考虑将部分软件系统功能部署到其他计算节点。

4. CPU 请求队列长度

目标描述：计算资源请求队列的长度；

影响因素：处理器执行速度，请求的时间长度；

统计数据：系统执行期间 CPU 请求队列的长度变化间；

评估方法：同 CPU 负载，用于判断处理器的工作负载是否过重。

5. 硬盘利用率

目标描述：硬盘忙碌时间同系统执行时间的比率；

影响因素：系统硬盘读/写请求频率，读/写请求数据量大小；

统计数据：硬盘工作时长以及系统响应时间；

评估方法：通过硬盘负载可以判断该节点的数据读写负载是否过重，如果过重，可以考虑将部分软件系统功能部署到其他存储节点。

6. 硬盘请求队列长度

目标描述：硬盘读取（写入）请求队列的长度；

影响因素：系统硬盘读/写请求频率，读/写请求数据量大小；

统计数据：系统执行期间硬盘请求队列的长度变化；

评估方法：同硬盘负载，用于判断存储节点的数据读/写负载是否过重。

7. 网络资源利用率

目标描述：网络系统的忙碌时间同系统执行时间的比率；

影响因素：网络通信请求频率，以及通信数据量大小；

统计数据：网络系统工作时长以及系统响应时间；

评估方法：类似于硬盘负载，判断该计算节点网络请求负载是否过重。

8. 网络请求队列长度

目标描述：网络通信请求队列的长度；

影响因素：网络通信请求的频率，每次请求的数据量大小；

统计数据：系统执行期间网络请求队列的长度变化；

评估方法：同网络负载，用于判断计算节点网络请求负载是否过重。

9. 共享资源请求队列长度

目标描述：共享资源进程请求队列的长度；

影响因素：共享资源请求的频率，每次请求的数据量大小；

统计数据：系统执行期间共享资源请求队列的长度变化；

评估方法：用于判断共享资源竞争是否影响了系统性能。

10. 消息发布时间

目标描述：特定消息发布的时间；

影响因素：网络队列负载，消息的发送频率，消息包含的数据量大小；

统计数据：消息编号，消息发布的系统时间，发送消息的节点 ID；

评估方法：将消息产生操作的开始时间同消息发布时间进行比较，查看消息从产生到发送的时间间隔是否满足时间性能需求。

11. 消息接收时间

目标描述：接收到特定消息的时间；

影响因素：网络负载，消息包含的数据量大小；

统计数据：消息编号，消息发送的系统时间，接收消息的节点 ID；

评估方法：将消息发布操作的开始时间同消息接收时间进行比较，查看消息从产生到接收到的时间间隔是否满足时间性能需求。

12. 消息队列长度

目标描述：消息中间件中保存消息实体的队列长度；

影响因素：网络负载，消息产生频率，消息包含的数据量大小；

统计数据：系统执行期间消息队列的长度变化；

评估方法：类似于网络负载，用于判断该计算节点消息请求负载是否过重。

3.3. 面向性能分析的建模

系统性能度量目标的影响因素涉及各个不同的方面，这些影响因素对性能分析的结果具有不同程度的影响。本小节首先通过分析性能度量目标及其影响因素，给出了分析消息中间件系统设计性能需要建模的内容；随后介绍了建模这些内容的方法。

3.3.1 面向性能分析的建模要素分析

通过 3.2.1 节的具体分析，本文在 3.2.2 节给出了消息中间件的性能度量目标。这些度量目标可以归纳为三类，一是时间（响应时间、系统方法执行时间、消息发送时间和消息接收时间）；二是资源利用率（计算资源（CPU）、存储资源、网络资源和共享资源）；

三是队列长度（计算资源 CPU 请求队列长度、存储资源请求队列长度、网络资源请求队列长度、共享资源进程队列长队以及消息队列长度）。其中队列长度这一度量目标可以通过仿真分析时对队列模型的仿真来获得，不需要在模型中对队列显式建模。而为了度量时间和资源利用率这两类指标，需要在设计模型中添加性能相关信息为第四章中的性能分析提供建模支持。

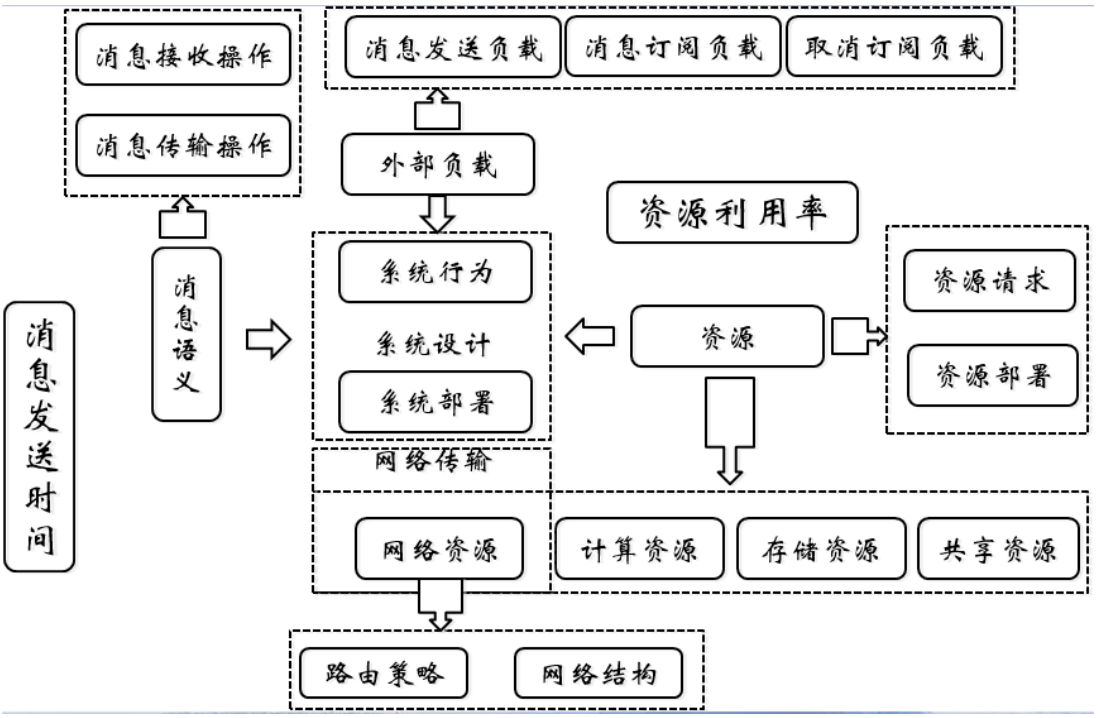

图 9　面向性能分析的建模要素

消息中间件系统时间性能的差异主要来自两个方面：一是取决于系统设计对于有限软硬件资源的使用、争夺和调度，具有良好设计的系统必定是具有良好的资源使用或调度策略，从而保证系统能在资源有限的前提下以最快的速度响应外部请求；二是取决于外部负载，在不同的外部负载下，系统的响应时间往往差异很大，这是因为不同的外部负载所引起的系统内部的资源争夺程度不同，这也是为什么压力测试是对性能测试的一个重要方面。此外，为了能够度量消息发送时间、消息接收时间以及响应时间，需要在系统设计中明确消息在各个节点的语义，以划分消息在系统中的各个阶段，从而对各个阶段的时间进行度量，并获得最终的时间指标，因此还需要对消息的操作语义进行建模。

因此，本文需要从三个方面对系统设计模型进行扩展，分别是资源，外部负载以及消息操作语义，如图 9 所示。系统的设计模型通常包含三方面内容，系统静态结构，系统动态行为以及系统部署。由于性能分析是通过分析系统动态行为来预测系统运行时性

能，系统静态结构对系统性能预测的意义不大，所以不在本文的考虑范围之内。

为了度量消息中间件的时间指标，同时度量资源的利用率，需要对系统行为的资源请求以及资源部署进行建模。在消息中间件系统中，涉及到的软硬件资源主要包括计算资源、存储资源、网络资源和共享资源，其中前三者属于硬件资源，最后一个属于软件资源，比如信号量等。此外由 3.2.1 节的分析可知, 在消息中间件的这几个时间指标中, 响应时间和系统方法执行时间主要系统设计决定；而消息发送时间和消息接收时间相对较难分析，因为它们除了受系统设计的影响，还受网络传输的影响。除了网络的部署情况会影响传输时间之外，在消息中间件中，路由策略也是决定网络传输时间的另一个重要因素。因为路由策略决定了消息传输的目的节点，从而影响消息在网络传输中的路径。

对于消息中间件来说，外部负载来自于外部应用对消息中间件接口的调用，更确切的说是来自外部应用的发布、订阅请求。因此本文引入消息发布负载、消息订阅负载以及取消订阅负载三种。

最后，为了描述消息在系统中的产生、传递以及接收，本文引入了消息操作语义。消息操作语义指操作消息的系统行为，包括消息发布、消息订阅、取消订阅、消息传递以及消息接收五种行为[34]。其中消息发布、消息订阅和取消消息属于外部应用行为；消息接收在有消息到达时触发；消息传递属于系统行为，负责将消息发送到网络。

3.3.2 面向性能分析的建模内容

在上一小节，本文阐述了为了度量 3.2.2 节提出的性能度量目标，需要在系统设计模型（包括系统行为模型和系统部署模型）上扩展的模型元素，主要包含三类，资源、外部负载以及消息操作语义。

本文为了便于描述，结合已有的模型以及扩展的模型，根据性能相关参数建模的内容，将其分为外部负载、系统行为以及执行平台三部分。外部负载主要指外部负载，包括影响性能度量目标的外界因素，例如使用系统的用户数量，或者外部环境刺激的产生类型；系统行为指设计模型中描述的动态行为、资源请求以及消息操作语义，例如分支选择、循环结构的执行次数；执行平台，用来描述系统部署的硬件环境，包含系统部署和资源部署，例如 CPU 执行调度策略、硬盘读写速度。

性能相关参数通常可以表示为系统各个组成部分的属性，例如分支选择、循环次数以及资源请求类型都属于系统行为的属性。因而本小节首先给出系统的组成实体，然后

以实体属性的形式给出本文涉及到的性能相关参数。

1. 外部负载

消息中间件作为中间件系统，其主要功能是为其它软件系统提供消息传输服务，因而其用户仅仅包括调用系统，并且可以将行为抽象为消息订阅、取消订阅和消息发布三类。消息发布用来描述消息生产者的行为，它通过消息中间件提供的消息发布系统调用，将消息发送给特定的消息节点，其性能相关参数如下表所示。

表 2　性能相关参数—消息发布负载

性能相关参数	描述	取值
消息产生的间隔	两个消息产生的时间间隔；	以毫秒为单位，含有数值或者概率分布的表达式；
消息大小	消息实体的大小；	以字节为单位，含有数值或者概率分布的表达式；
消息字段	消息内包含的属性名以及属性取值；	属性名为字符串，属性取值为数值或者字符串；
仿真执行时间	此次仿真的时长；	数值；

外部环境及系统自身具有一定的不确定性，为了描述这种不确定性，本文支持含有概率分布的数学表达式。本文支持的数学操作，概率函数及其输入参数，请参考附录 1。

消息订阅指外部应用通过消息中间件提供的消息订阅系统调用，注册某一类型的消息，使其能够被发送到的订阅者所在的消息节点，其性能相关参数如表 3 所示。其中订阅表达式用来描述需要订阅的消息，用包含 message.attribute 变量的布尔表达式描述。

表 3　性能相关参数—消息订阅负载

性能相关参数	描述	取值
订阅名称	标识该次订阅行为的字符串；	字符串；
订阅时间	该消息的产生时间；	数值，单位为毫秒；
订阅节点	产生该消息的计算节点；	字符串，表示计算节点的名称；
订阅表达式	描述订阅何种类型的消息；	布尔表达式，可用变量 message.attribute 表示，其中 attribute 为属性名，该属性必须在表 2 中的消息字段中定义；

取消订阅行为指外部应用通过消息中间件提供的取消订阅系统调用，取消某一类型消息的注册，使其不再发送到取消订阅者所在的消息节点，其性能相关参数如表 4 所示。其中订阅名称同表 3 中的订阅名称相对应。

表 4 性能相关参数—取消订阅负载

性能相关参数	描述	取值
订阅名称	标识该次订阅行为的字符串；	字符串；
订阅时间	该消息的产生时间；	数值，单位为毫秒；

由于消息中间件中消息订阅以及订阅取消行为较少，所以不以概率的形式描述订阅或取消订阅消息的产生时间，而是通过直接指定时间的方式表达。

2. 用户行为

设计模型的动态行为一般通过方法描述，并通过不同的执行结构（顺序、循环以及分支）组织方法。此外，软件资源（主要指共享资源）在并发环境下会对系统行为、性能产生一定的影响。可以利用系统方法、软件资源两个概念包含系统行为相关的性能相关参数，如表 5 所示。

表 5 性能相关参数—系统行为

所属实体	性能相关参数	描述	取值
系统方法	行为类型	该行为的语义	循环、分支、异步、复合、同步、消息传递；
	资源请求类型	执行该行为需要的资源	CPU、硬盘、网络、请求软件资源，释放软件资源；
	资源请求数量	执行该方法需要的资源数量	含有数值或者概率分布的表达式；
	分支选择	选择类型的方法选择每条分支的概率；	含有数值或者概率分布的表达式；
	循环次数	循环类型的方法执行循环体的次数；	含有数值或者概率分布的表达式；
软件资源	资源名称	用于标示该资源；	字符串；

	资源数量	该资源初始化时的数量；	同循环次数；

不同类型的系统方法用来描述不同的行为语义，如循环类型描述具有循环结构的方法，其内部行为需要执行给定的次数；分支类型描述具有分支选择结构的方法，其会选择内部某条分支执行；复合结构指该方法内部有子方法对其行为进行细化描述。

3. 执行平台

典型的执行平台即 PC 机，由 CPU、存储设备和网络三部分组成，通过描述这三类实体的特性，可以抽象出一个完整的执行平台。系统的部署以及不同计算节点的连接无法以属性的方式描述，而是以实体间的关系描述。如表 6 所示。

表 6　性能相关参数—执行平台

所属实体	性能相关参数	描述	取值
CPU	调度策略	该 CPU 的调度策略；	时间片共享；先入先出；
	进程切换时间	切换进程上下文需要的时间；	数值，单位为毫秒；
存储设备	head time	实际读写之前需要的时间；	数值，单位为毫秒；
	读写速度	完成读写请求的速度；	数值，单位为字节数每毫秒；
网络	head time	实际传输之间需要的时间；	数值，单位为毫秒；
	传输速度	数据的传输速度；	数值，单位为毫秒；

3.4. 面向性能分析的消息中间件建模语言

UML 作为一种标准的建模语言，已经广泛应用并熟知于工业界和学术界。鉴于其广泛的影响力以及灵活的扩展机制，本文以 UML 为基础，并通过扩展 UML 定义了面向性能分析的消息中间件建模语言（MePAL： Message-Oriented Middleware Performance Analysis Model Language）。相关研究中已经给出了各种不同建模方法的优劣，同时说明了 MARTE 语言在对时间和资源相关概念建模时的优势，因此本文在扩展 UML 的性能部分建模元素（主要是跟时间和资源相关的概念）借鉴了 MARTE 中的相关概念,构成了 MePAL 的性能标签。本小节首先介绍 MARTE 语言，然后详细阐述了通过借鉴 MARTE 语言并扩展 UML 而定义的面向性能分析的消息中间件建模语言 MePAL。

3.4.1 MARTE 语言简介

MARTE 是 OMG 组织于 2007 年发布的基于 UML2 的标准 Profile 扩展，它被设计用来为实时嵌入式系统的模型驱动开发过程提供建模支持，以取代 UML-SPT。MARTE 增加了 UML2 在非功能属性上的建模能力，是 OMG 组织在嵌入式实时系统建模领域提出的标准规范。

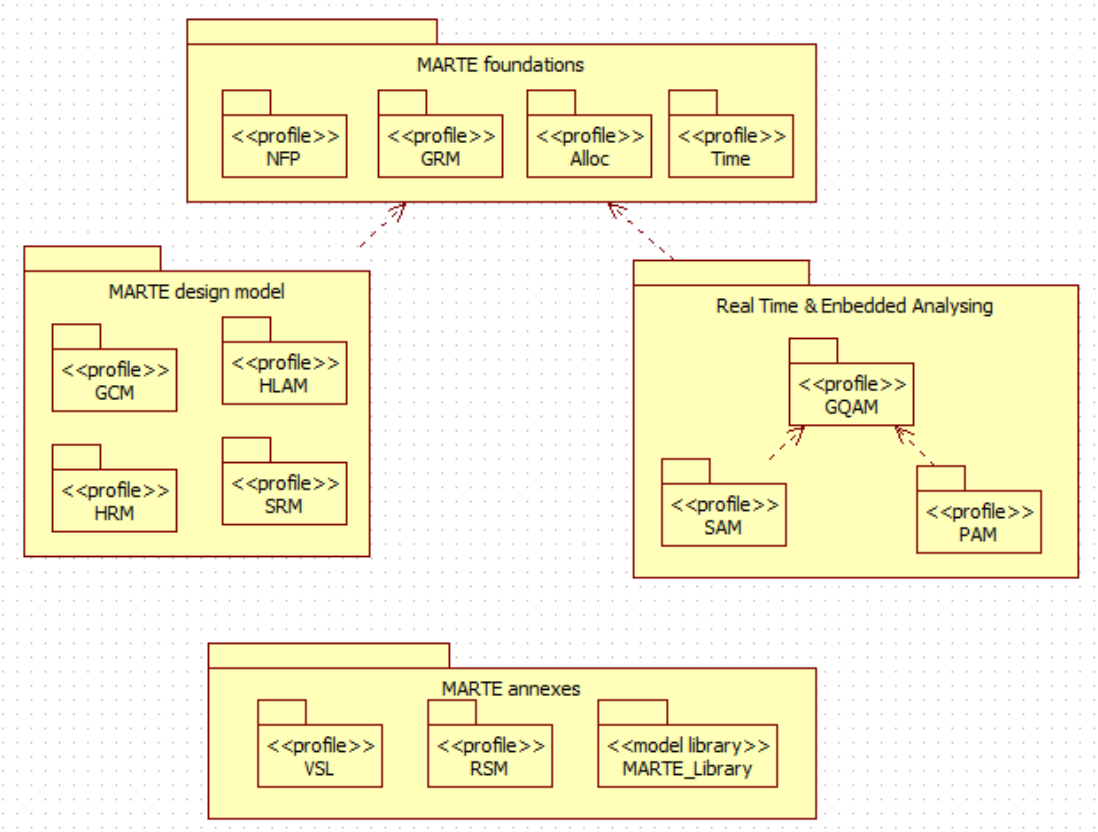

图 10 MARTE 包结构

MARTE 语言标准主要包括四部分：一个定义了实时嵌入式领域基本概念的核心框架；对核心框架的第一次扩展用来支持设计模型中非功能属性的建模；对核心框架的第二次扩展用来支持对设计模型的性能分析，特别是调度和时间性能分析；最后一部分包含了 MARTE 规范所有的附录，例如定义了一种可以在 UML2 中使用的值描述文本语言，以及囊括了所有 MARTE 规范的标准模型库。它们之间的关系如图 10 所示。

1. MARTE 核心包

MARTE 核心包由非功能属性建模（Non-functional Properties Modeling，NFPs）、时间建模（Time Modeling，Time）、通用资源建模（Generic Resource Modeling，GRM）以及部署建模（Allocation Modeling，Alloc）四部分组成。

1) 非功能属性包描述了 NFPs 的表达形式，NFPs 同 UML 模型的关系，NFPs 之间关系的定义等同 NFPs 相关的内容；

2) 时间包给出了一个描述时间及时间相关概念的通用框架，为 MARTE 语言后面的内容提供了基本的时间概念。MARTE 抽象出了三类时间概念来满足不同的时间建模需求，包括：

a) Causal/temporal：该类时间模型仅仅关心指令的先后顺序；

b) Clocked/synchronous：该类时间增加了并发的概念，并将时间分割为离散而连续的时间节点；

c) Physical/real-time：该类时间要求对时间的精确建模。

3) 通用资源包的目标是为建模通用的执行平台提供基本概念，它支持建模平台不同粒度的建模需求，并且能够支持硬件平台和软件平台的建模需求；

4) 部署包描述软件系统功能如何分配到可用硬件平台之上。

2. MARTE 建模包

MARTE 建模包由通用组件模型（Generic Component Model，GCM）、高层应用建模（High-Level Application Modeling，HLAM）、软件资源模型（Software Resource Model）以及硬件资源模型（Hardware Resource Model）四部分组成。

1) 通用组件模型为基于组件的架构建模提供了支持，同时为组件间的数据交互提供了建模支持；

2) 高层应用建模为嵌入式实时系统建模提供了高层的建模概念。同一般的软件建模相比较，实时系统除了建模同行为、通信以及并发相关的定性特性外，还需要建模同执行期限、执行时间相关的定量特性；

3) 软件资源模型用来建模实时系统运行的软件平台，特别是支持多任务的嵌入式操作系统 API，并为不同类型的软件资源定义了相应的图符；

4) 硬件资源模型为建模嵌入式系统运行的硬件平台提供了支持，其中预定义了包括 CPU、硬盘在内的多种硬件资源，并为不同类型的硬件资源定义了相应的图符。

3. MARTE 分析包

MARTE 分析包由通用量化分析模型（Generic Quantitative Analysis Modeling，GQAM）、调度分析模型（Scheduling Analysis Modeling，SAM）以及性能分析模型

（Performance Analysis Modeling，PAM）三部分组成。

通用量化分析模型定义了不同类型量化分析需要的通用概念，调度分析模型和性能分析模型则根据调度分析和性能分析的特点，定义了与其量化分析类型相关的特定概念。MARTE 并没有提出对实时系统设计模型进行性能分析的方法，而是通过定义相关的概念，为各种不同的分析方法提供了模型层面的支持。

3.4.2 面向性能分析的消息中间件建模语言 MePAL

按照 3.3.2 节中对建模内容的分析，MePAL 语言主要包含三个方面的内容，即外部负载，系统行为和执行平台，本节分别给出其元模型（MePAL 建模语言完整的元模型请参考附录 2）。对 UML 扩展的部分主要是性能标签以及消息操作语义。其中性能标签部分的建模借鉴自 MARTE 中的相关概念，在外部负载，系统行为和执行平台上都有体现，消息操作语义是自定义的扩展，主要是对系统行为的扩展。

1 外部负载

描述用户特性的建模元素负载（Workload），分为开放负载（Open Workload）和封闭负载（Close Workload）两部分。开放负载描述用户数量不确定的场景，而封闭负载则指用户数量确定的场景。消息中间件系统仅为应用系统提供消息传递服务，其同外部应用的交互只包括外部系统对消息传递服务的调用，较为简单，不需要复杂方法对交互行为建模。

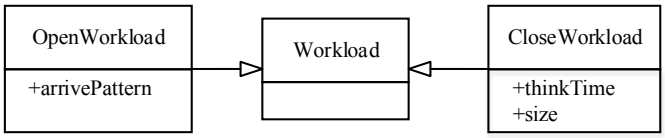

图 11　外部负载元模型

2 系统行为

性能分析包定义了行为场景（Behavior Scenario）和单步（Step）等概念描述软件系统的行为，并在单步概念的基础上定义了某些具有特定语义的行为。GaAcqStep 和 GaRelStep 分别表示软件资源的获取和释放，PaRequestService 描述对外部服务的请求行为，而 PaCommStep 则表示网络通信行为。如图 12 所示。

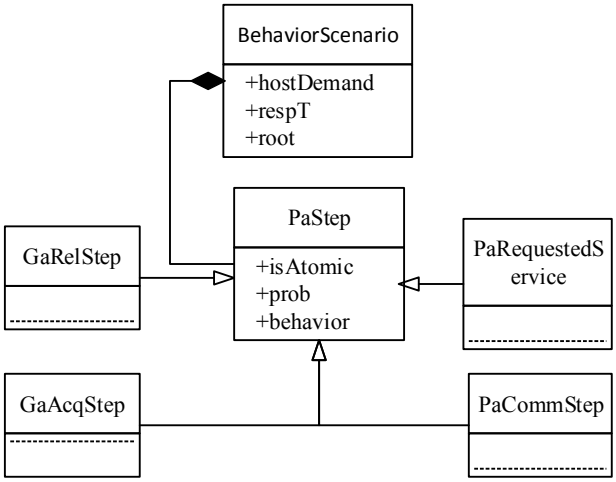

图 12　系统行为元模型

本文采用 UML 中的顺序图、活动图描述软件系统的动态行为，并以属性的形式为设计模型添加性能相关信息，性能相关参数概念和 UML 顺序图建模元素的对应关系如下表所示：

表 7　性能标签与顺序图对应关系

性能标签	顺序图建模元素	描述
行为场景（Behavior Scenario）	顺序图（Sequence Diagram）	单个顺序图描述一个行为场景，由不同的单步组成。
单步（Step）	消息（Message）	顺序图中的同步消息、异步消息和对象创建消息都描述一个单步。
资源获取单步（GaAcqStep）	消息（Message）	目标是获取同步资源的消息。
资源释放单步（GaRelStep）	消息（Message）	目标是释放同步资源的消息。
通信单步（PaCommStep）	消息（Message）	目标是进行网络通信的消息。
外部服务调用单步（PaRequestedService）	消息（Message）	目标是调用外部服务的消息。

系统行为中除了描述在 UML 顺序图(或活动图)的基础上通过属性建模资源请求，还需要通过消息操作语义标签（简称消息语义）来明确指明 UML 顺序图（或活动图）中的消息操作类型。因而需要在现有建模语言的基础上，扩展定义消息语义，从而建模消息在系统中的产生、订阅、传递，为消息相关的性能度量目标提供所需的数据。本文

仅仅需要描述消息操作语义，不需要复杂的表达能力；本文利用 UML 的 Profile 扩展机制，以扩展的 OpenWorkload 和 PaStep 概念为基础，定义消息操作语义，达到描述消息中间件中消息操作语义的目标。

1） 消息发布

外部负载通过调用消息发布操作将产生的消息发布到消息中间件，消息发布操作是消息中间件为外部应用提供的标准接口。其元模型如下图所示：

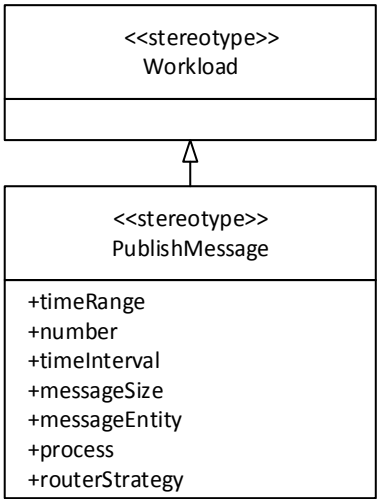

图 13 消息发送元模型

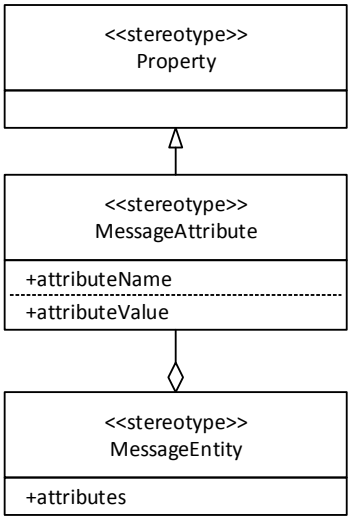

图 14 消息实体元模型

其中 timeRange 表示时间区间，即消息发布行为总的时间长度；timeInterval 描述两个时间相邻消息产生的时间间隔；number 说明该负载的实例数量；messageSize 表示要发送的消息的大小；messageEntity 定义了 SendMessage 操作发送的消息实体，由 MessageAttribute 定义的消息属性组成，而消息属性则借鉴自 MARTE 基础包中的属性；process 表示负责发送消息的系统调用，可以用顺序图或活动图描述；routerStrategy 表示该消息中间件需要采用的路由策略。

2） 消息订阅

外部负载通过调用消息订阅操作订阅发送到消息中间件中的消息，消息订阅操作是消息中间件为外部应用提供的标准接口。其元模型如图 15 所示：

其中 filters 是 MessageSubscribe 实体数组，描述该外部负载能够发起的所有订阅行为；process 表示同订阅行为相关的系统调用，可以用 UML 顺序图或者活动图描述；name 用于标识该次订阅行为；expression 定义了消息的过滤条件；time 描述该次订阅行为发生的具体时间。

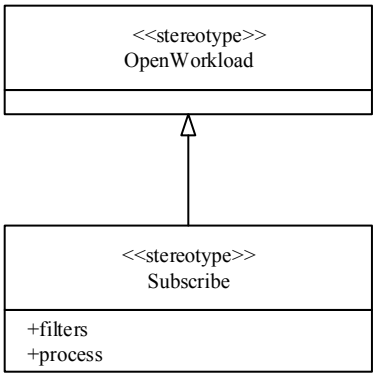

图 15　消息订阅元模型

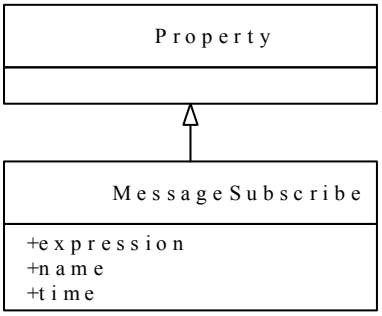

图 16　消息订阅实体元模型

3） 取消订阅

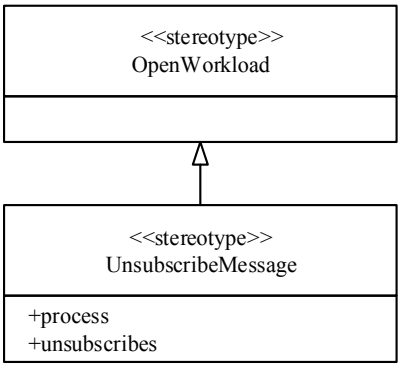

图 17　取消订阅元模型

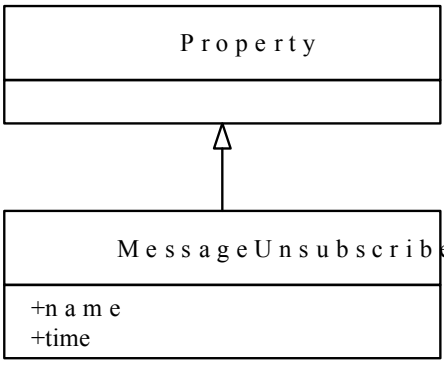

图 18　取消订阅实体元模型

外部负载通过调用取消订阅操作取消 3.4.2 中描述的订阅行为，取消订阅操作是消息中间件为外部应用提供的标准接口。其元模型图 17 所示。

其中 unsubscribes 是 MessageUnsubscribe 实体数组，描述该负载产生的取消订阅行为；process 表示取消订阅操作行为，可以用顺序图或者活动图描述；subName 用于标识要取消的订阅；time 描述该取消行为发生的时间。

4） 消息传输

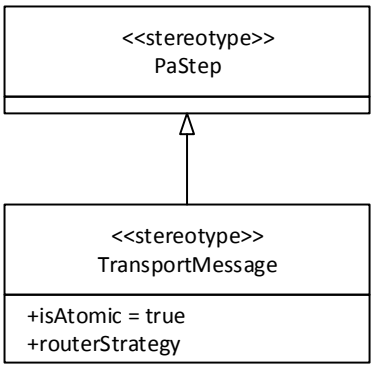

图 19　消息传输元模型

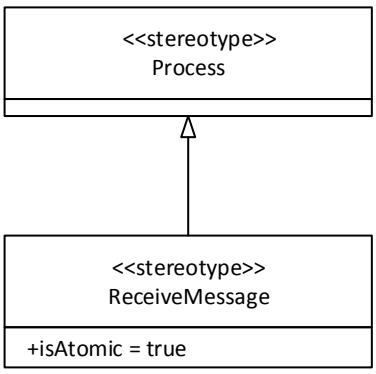

图 20　消息接收元模型

消息传输操作并不属于消息中间件为外部应用提供的标准接口，而是系统内部调用将消息发送到网络中。其元模型如图 19 所示，isAtomic 继承自 PaStep，其值恒为 true，说明该行为是原子操作；routerStrategy 为该消息中间件采用的消息路由策略，需要同发布负载中的策略保持一致。

5） 消息接收

消息接收操作是消息中间件为外部应用提供的标准接口，当消息到达时由消息中间件自动调用。其元模型如图 20 所示，各属性的含义同上。

3 执行平台

1） 软件资源

并发系统中共享资源的竞争是影响软件系统性能的重要因素之一，本文中定义的软件资源，分为共享资源、缓冲区几类，图 21 给出了软件资源的元模型。信号量（Semaphore）和缓冲区（Buffer）属于被动资源（PassiveResource），数量是有限的，通过信号量的初始化大小（initSize）属性或者缓冲区的缓冲池大小（poolSize）属性来描述。

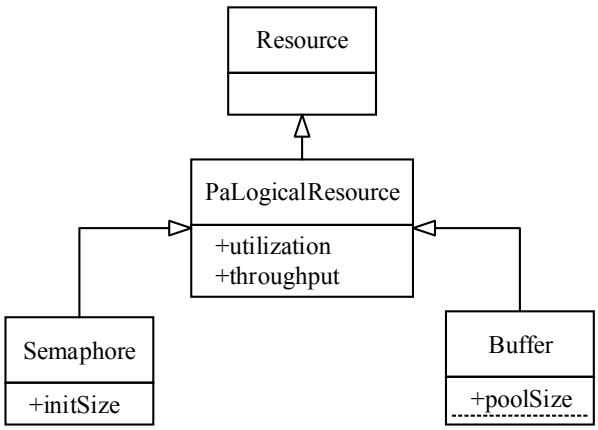

图 21 软件资源元模型

2） 硬件资源

图 22 给出了执行平台中硬件资源的元模型，包括可执行资源（GaExecHost），通信资源（GaCommHost）以及存储资源（StorageResource），每类资源都有自己特定的属性，描述对软件系统性能的影响因素。UML 部署图具有足够的表达能力来描述软件系统各部分同执行平台的映射关系，因而不必进行扩展。

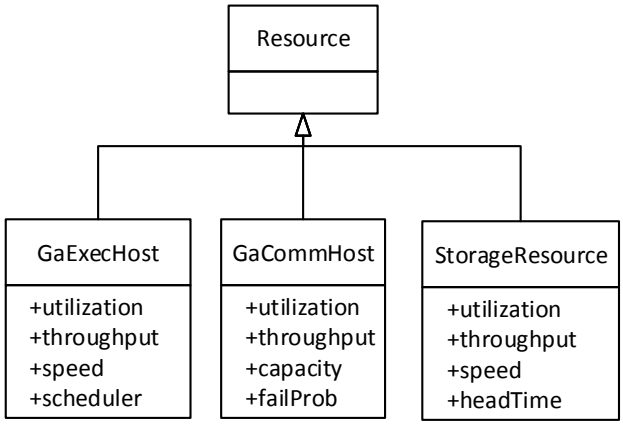

图 22　硬件资源元模型

3.5. 面向性能分析的建模指导方法

为了分析系统设计的时间性能，需要在设计模型中添加足够的信息。这些信息包括系统部署、系统行为、系统执行需要的资源、外部负载、网络结构以及系统资源几个方面。其中系统行为和系统执行需要的资源可以通过 UML 活动图、顺序图，MARTE 语言以及消息传递操作语义来描述；其它四个方面的内容则在部署图中给出。

本文从需要描述的信息、建模元素、构造型和描述方法四个方面介绍如何建模面向性能分析的设计模型。需要描述的信息指要在设计模型中描述的内容；建模元素指通过哪种 UML 建模元素描述该内容；构造型指需要为该建模元素添加的 MARTE 构造型或者消息操作构造型，以提供额外的信息；描述方法则介绍了具体的建模方法。具体内容如表 8、表 9 和表 10 所示。

表 8　部署图建模内容

描述的信息	建模元素	构造型	描述方法
消息发送负载	部署图 Artifact	消息发送	为 Artifact 建模元素添加消息发送构造型，并编辑其属性；
消息订阅负载	部署图 Artifact	消息订阅	为 Artifact 建模元素添加消息订阅构造型，并编辑其属性；
取消订阅负载	部署图 Artifact	取消订阅	为 Artifact 建模元素添加取消订阅构造型，并编辑其属性；

表 9 部署图建模内容（续）

描述的信息	建模元素	构造型	描述方法
网络结构	ConnectionPort、Node 以及 Connection	无	Node 中添加 ConnectionPort，通过 Connection 连接不同的 ConnectionPort；
系统资源	部署图 Artifact	处理器、存储设备、网络设备或者软件资源；	为 Artifact 建模元素添加相应的构造型，并编辑其属性；
系统部署	部署图 Artifact	UML process；	为 Artifact 建模元素添加相应的构造型，并编辑其属性；

通过表 8 和表 9 给出的内容，可以很容易的通过部署图描述出特定场景中消息中间件系统的外部负载、部署信息以及执行平台信息，充分描述系统的执行环境。

表 10 活动图和顺序图建模内容

描述的信息	建模元素	构造型	描述方法
系统行为	UML 顺序图和活动图中的建模元素；	无	一般的 UML 顺序图和活动图建模方法；
资源请求	活动图 Activity 或者顺序图 Message	PaStep 或者 PaBehavior	为 Activity 和 Message 建模元素应用 PaStep 构造型，并编辑属性值；
消息传递操作	活动图 Activity 或者顺序图 Message	MessageTrasport	为 Activity 和 Message 建模元素应用 MessageTrasport 构造型，并编辑属性值；

3.6. 本章小结

为了分析系统设计的性能，需要在设计模型中添加性能相关参数。本章从性能度量目标出发，通过分析影响系统性能主要因素，定义了设计模型中需要添加的性能相关参数；然后选取 MARTE 语言中的部分建模元素在设计模型中表达这些参数；最后通过扩展 MARTE，定义了与消息操作语义相关的建模元素。在下一章中，本文首先通过分析

MePAL 语言的建模能力，设计并实现了性能仿真指令，用于描述系统行为及行为的资源需求；最后根据离散仿真的基本原理，设计了基于设计模型的性能分析算法，从而解决本文第二个研究问题：自动分析设计模型，输出性能度量目标。

第四章　性能仿真指令与仿真方法的设计

为了解耦消息中间件系统设计模型的存储格式和仿真引擎的实现，简化性能分析系统的设计实现，基于消息中间件性能分析模型，本文设计并实现了性能仿真指令 PSI（Performance Simulation Instruction），PSI 从顺序图、活动图描述的系统行为以及性能相关参数的角度出发，通过分支、顺序和循环控制结构指令描述系统执行流程；通过具体的指令属性描述性能相关参数。本章首先给出了分析消息中间件性能的计算模型，说明性能分析所需要的信息；然后定义并实现了性能仿真指令，用于描述消息中间件设计模型中提取的性能仿真信息；最后介绍了基于性能仿真指令的设计模型性能分析算法。

4.1. 性能仿真指令

本文在第三章中提出的消息中间件性能分析建模语言 MePAL 基于 UML 部署图、活动图以及顺序图的部分建模元素，并在其基础上进行了扩展。由于不同 UML 建模工具模型的存储格式有较大的差异，而且对相关 UML 建模元素的支持不尽相同，为了统一基于 MePAL 语言的模型存储格式，本文设计并实现了性能仿真指令描述系统设计的动态行为及其资源需求，并定义了配置文件的格式。其中配置文件是对外部负载、系统部署以及资源特性的描述，较为简单；描述系统行为及资源需求的建模元素源自 UML 顺序图、活动图及其扩展，较为复杂。

本小节通过分析 MePAL 语言能够描述的系统行为及相关属性，设计并实现了性能仿真指令。

4.2.1 MePAL 语言定义的系统行为

MePAL 语言利用 UML 顺序图、活动图中的部分建模元素描述系统行为，利用 UML 扩展描述资源请求。下面通过分析这些建模元素的建模能力，确定性能仿真指令需要描述的内容。

MePAL 语言中顺序图相关的概念包括消息、消息类型、组合框图、交互操作类型和生命线。其中消息描述对象之间的交互，消息类型描述产生该消息的通信行为类型，组合框图用于描述包含一组消息的复合结构，交互操作类型描述了该复合结构的类型。消

息类型决定了单个系统行为的表达能力，而消息之间的关系以及交互操作类型决定了顺序图能够描述的复合结构。MePAL 语言目前定义了描述顺序、循环以及分支三种基本结构，并对异步行为提供了支持。MePAL 语言中活动图相关的的概念包括活动、控制流，以及活动的扩展概念。这些扩展描述了循环、分支结构，并对同步行为提供了支持。MePAL 语言中还定义了消息发送操作，用于定位网络开始传输时间点。此外，性能仿真指令还需要描述系统行为执行时请求的资源类型以及数量。

4.2.2 仿真指令的设计思路

通过 4.2.1 小节的分析可以看出，性能仿真指令需要能够描述顺序、循环和分支三种基本的结构，并为异步行为、同步行为和消息发送操作提供支持，而且还需要描述系统行为执行时请求的资源类型以及数量。

综上所述，本文需要设计这样的性能仿真指令，它应该满足：

1. 支持软件系统基本结构的描述，即顺序、循环以及分支；
2. 支持异步以及同步行为的描述；
3. 支持系统行为对资源的请求描述；
4. 支持消息发送操作的描述。

根据上述需求，本文决定采用 Json 格式定义性能仿真指令。Json 格式较为简单，容易与编程语言中的对象建立映射关系，便于理解和解析；而且 Json 格式天然的树形结构和属性名/值对非常适合表达复合结构和系统行为对资源的请求。综上，本文定义了以下几种性能仿真指令：

1. 原子指令：其内部不包含复合指令，具有属性描述执行该指令需要的资源类型以及资源数量；
2. 消息发送指令：特殊类型的原子指令，其语义是将消息实体通过网络发送；
3. 复合指令：其内部包含子指令，描述顺序图中的消息触发关系；
4. 分支指令：特殊类型的复合指令，只会执行其中的某一条子指令；
5. 循环指令：特殊类型的复合指令，会循环执行其中的子指令；
6. 同步指令：为原子指令，描述需要等待的进程；
7. 异步指令：特殊类型的复合指令，其子指令需要在新创建的进程中执行。

4.2.3 仿真指令的设计实现

1. 原子指令

原子指令用来描述系统行为所需要的资源类型以及资源数量，根据第三章的内容，本文定义了如下表所示的属性：

表 11 原子指令属性

指令属性	属性描述	属性取值
name	指令的名称；	字符串；
type	指令类型；	atom；
resourceType	指令执行需要的资源类型；	处理器、硬盘、网络、软件资源或者无，即不需要任何资源；
resourceSize	指令执行需要的资源数量，根据资源类型的不同，该属性的具体含义也不尽相同；	资源类型为处理器，则该属性的单位为毫秒；资源类型为硬盘、网络资源，则该属性的单位为字节；资源类型为软件，则该属性表示数量；

2. 消息发送指令

消息发送指令描述消息实体的发送行为，由于消息实体的大小、包含的属性由消息生产者，即外部环境负载定义，所以该指令只需要定义消息名称和消息类型两种属性即可。

3. 复合指令

复合指令通过它内部的子指令描述系统行为，在复合指令中描述资源需求是没有意义的，它执行所需要的资源是通过子指令定义的。本文为复合指令定义了如表 6 所示的属性：

表 12 复合指令属性

指令属性	属性描述	属性取值
name	指令名称；	字符串；
type	指令类型；	embedded；
instructions	描述该指令的子指令；	指令数组；

4. 分支指令

分支指令是一种特殊类型的复合指令，具有描述其具体行为的子指令，不同之处在于只有某一条子指令能够执行，而且每条子指令都有一条判断语句，用于说明是否执行该指令。该指令具有下表中定义的属性：

表 13 分支指令属性

指令属性	属性描述	属性取值
name	指令名称；	字符串；
type	指令类型；	branch；
instructions	子指令；	指令数组；
branch	定义在子指令中，判断是否执行该语句；	布尔表达式；

5. 循环指令

循环指令是一种特殊类型的复合指令，具有描述其具体行为的子指令，不同之处在于子指令需要多次执行，并定义了额外的 loop 属性用于计算执行次数，具有下表中定义的属性：

表 14 循环指令属性

指令属性	属性描述	属性取值
name	指令名称；	字符串；
type	指令类型；	loop；
instructions	子指令；	指令数组；
loop	计算循环次数的数学表达式；	带有概率函数的数学表达式；

6. 同步指令

同步指令是一种特殊类型的原子指令，用于描述进程间的同步，这里的进程是指一组顺序执行的指令集合。除了原子指令的属性外，该指令还具有下表中定义的属性：

表 15 同步指令属性

指令属性	属性描述	属性取值
type	指令类型；	sync；
processes	需要同步的进程名称数组；	进程名数组；

7. 异步指令

异步指令是一种特殊类型的复合指令，负责创建新进程执行内部的指令序列。该指

令的属性如下表所示：

表 16 异步指令属性

指令属性	属性描述	属性取值
name	该指令名称；	字符串；
type	指令类型；	async；
instructions	需要异步执行的指令序列；	指令数组；

4.2. 性能分析方法的设计

软件系统的运行过程即是不断的请求系统执行所需要的软硬件资源，并等待资源请求处理完成，之后继续执行直到系统退出。本小节首先介绍了离散事件仿真器的基本原理；然后定义了同本文设计的仿真算法相关的一些基本概念，以及具体的仿真算法；最后给出了性能分析算法的伪代码。

4.3.1 离散事件仿真

在系统仿真领域，离散仿真器[35]（discrete-event simulation，DES）将系统操作建模为一系列具有离散时间的事件。每一个事件都同一个系统时间相关联，并改变当前的系统状态。因而在两个连续的事件之间，系统的状态不会有任何改变，所以系统时间可以从一个事件的时间直接跳跃到另外一个事件的时间。

连续时间仿真（continuous simulation）同离散事件仿真不同，在连续时间仿真中，系统时间被分割为固定大小的时间片，系统状态随着发生在该时间片中的活动而改变，这种仿真方式也称为基于活动的仿真。由于离散事件仿真不必仿真所有的时间片，与连续时间仿真相比，具有更高的效率，而连续时间仿真更适合时间密集型系统。

另外一种常用的系统仿真方法是基于过程的仿真，在这种仿真方法中，系统中每一个活动都对应一个独立的过程，同操作系统中的一个进程相对应。过程在执行过程中会产生改变系统状态的事件。基于过程的仿真适合分析系统过程之间的交互，以及过程行为对系统状态的影响。

4.3.2 整体仿真流程

根据第三章中确定的消息处理过程，本文设计了如图 23 所示的仿真流程。外部应用通过调用消息中间件提供的标准接口产生外部负载，负载类型不同，触发的消息中间

件服务也不同。

当产生消息发布负载时，消息中间件会执行消息发布操作，并通过消息传输操作将消息发送到网络中。消息通过网络传输到达目标节点之后，会触发消息中间件的消息接收行为，最终将消息传送给订阅了该消息的外部应用。

当产生消息订阅负载时，消息中间件会执行消息订阅操作，并通过消息订阅操作以及设计人员指定的路由策略，在某些节点的消息订阅表中添加消息订阅信息。

当产生取消订阅负载时，消息中间件会执行取消订阅操作，并通过取消订阅操作以及设计人员指定的路由策略，在某些节点的消息订阅表中删除消息订阅信息。

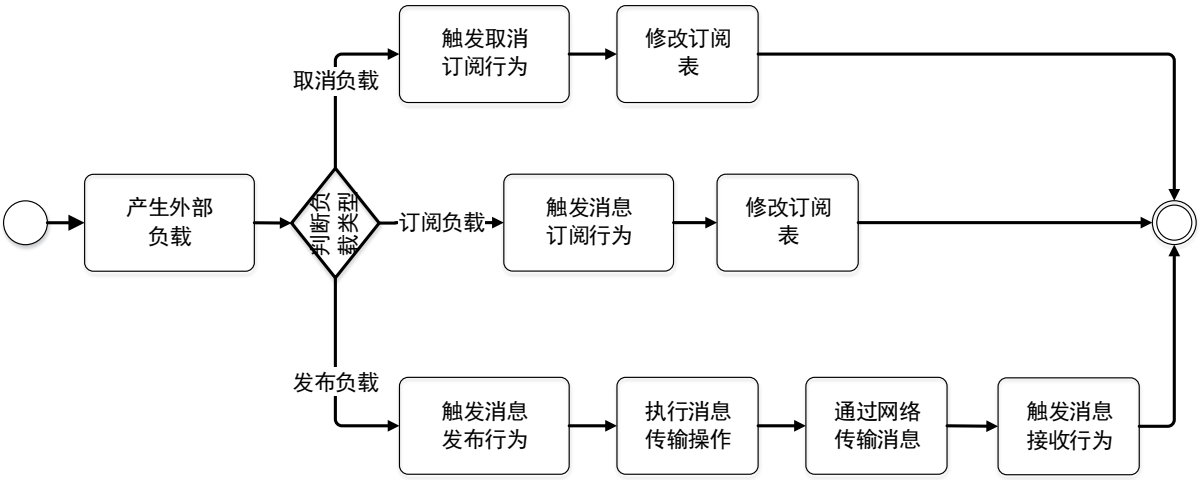

图 23　面向性能分析的整体仿真流程图

通过该仿真流程可以看出，消息中间件的系统行为由消息触发，而消息中间件的系统行为驱动了消息在系统中的传输。因而本文需要同时仿真消息流和系统行为，以及它们之间的交互关系。

4.3.3 消息流仿真

为了仿真消息在系统中的传递，本文设计并实现了如图 24 所示的消息处理流程。消息由外部应用产生之后，会根据消息的类型进行不同的处理。

当产生的消息为订阅消息时，只需要根据消息路由策略修改某些节点的订阅表，在其中添加或者删除订阅消息。

当产生的消息为发布消息时，则首先将其加入到消息发送队列中；在消息传输操作执行时，会根据路由策略、订阅表以及消息属性确定消息发送的目标节点；接下来将消息从消息队列中弹出；最后经过网络将消息传输到目标节点。

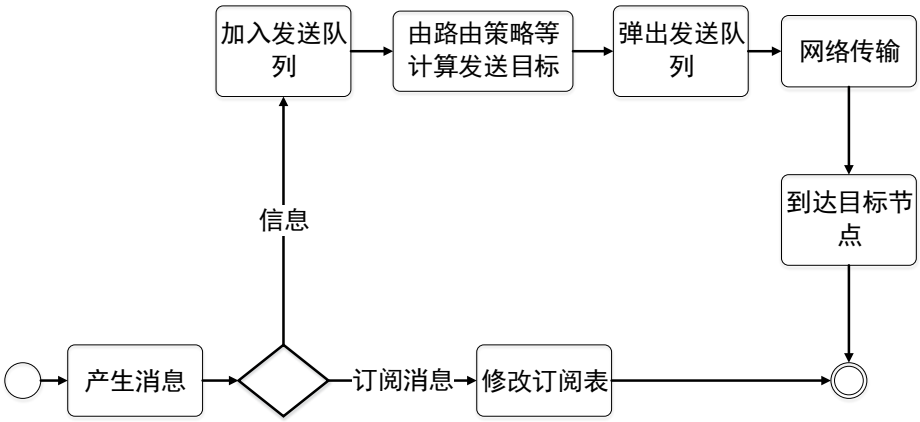

图 24 消息仿真流程图

4.3.4 系统资源特性

软件系统在运行过程中，通过进程不断请求软件资源或者硬件服务来完成自身的任务。这里的进程指顺序执行的性能仿真指令集合，指令中包含了系统执行过程中对各类资源的请求。不同类型的系统资源具有不同的特性，在系统执行过程中其行为也不尽相同，因而在这里介绍了各类系统资源的相关特性。

计算节点中的系统资源一般包括处理器、硬盘、网络以及共享资源四类，其中共享资源即为竞争资源，用于控制多个进程之间的并发访问。一般情况下，如果存在多个进程请求同一资源，会将没有获得资源的进程按照一定的策略放入优先权队列中；如果该资源可用，便选取优先权队列中的第一个进程满足其资源请求。

本文将软硬件资源抽象为优先权队列，接受进程的资源请求，并产生相应的事件驱动仿真的进行。下面对各类资源的特性，以及该资源能够产生的事件进行详细介绍。

1. 处理器资源

处理器资源用于满足进程对计算资源的请求，对应仿真指令中的资源请求类型为处理器。由于判断某条语句的计算量较为困难，因而本文允许设计人员直接给出需要的计算时间，单位为毫秒。目前较为常用的调度策略为时间片轮转，通过赋予进程给定长度的时间片，将该处理器的计算资源分配给计算资源请求进程，直到时间片耗尽，便选取下一个请求进程继续处理。

处理器资源在赋予请求进程固定长度的时间片之后，其状态不会改变，直到时间片耗尽、进程请求其它资源或者进程结束。在将处理器资源分配给某一进程，并将其自身设置为不可用之后，会产生一个处理器资源请求满足事件，该事件发生时间的计算公式

为：

$$\text{OccureTime} = \text{switch}\ ?\ \text{switchTime}\ :\ 0 + \min(\text{remaineTime}, \text{requiredTime})$$

在公式中，OccureTime表示该事件的发生时间，switch 用于判断是否出现进程切换，switchTime描述进程切换时间，并同剩余时间（remaineTime）和请求时间（requiredTime）中较小的值相加，剩余时间表示进程中剩余的可分配时间，初始值大小为时间片大小。在分配处理器资源后，还需要重新计算进程资源的时间片大小，计算公式为：

$$\text{remaineTime} = \text{remaineTime} > \textit{requiredTime}\ ?\ \text{remaineTime} - \textit{requiredTime} : 0$$

2. 硬盘资源

硬盘资源用于满足软件系统对文件的读写请求，对应性能仿真指令中的资源请求类型为硬盘。本文允许设计人员在性能仿真指令中给出需要读写的数据量的大小，并采用先入先出的策略由优先权队列中选取下一个处理进程，处理结束后，便选取下一个请求进程继续处理。

硬盘资源在处理某个进程的资源请求时，其状态不会改变，直到处理完当前进程的资源请求。在将硬盘资源本身分配给某一进程，并将其自身设置为不可用之后，会产生一个硬盘资源满足事件，该事件发生时间的计算公式为：

$$\text{OccureTime} = \text{requiredSize} / \text{speed} + \text{headTime}$$

在该公式中，requiredSize指读写请求的数据量大小，单位为字节；speed指该硬盘的读写速度；headTime指在硬盘资源实际读写之前需要的时间，如磁盘寻道时间，由用户在部署图中给出。

3. 网络资源

网络资源用来满足软件系统对网络数据传输的需求，对应性能仿真指令中的资源请求类型为网络资源。本文允许设计人员在性能仿真指令中给出需要读写的数据量的大小，并采用先入先出的策略由优先权队列中选取下一个处理进程，处理结束后，便选取下一个请求进程继续处理，直到请求队列为空。

网络资源在处理某个进程的资源请求时，其状态不会改变，直到处理完当前进程的资源请求。在将网络资源本身分配给某一进程，并将其自身状态设置为不可用之后，会产生一个网络资源满足事件，该事件发生时间的计算公式为：

$$\text{OccureTime} = \text{requiredSize} / \text{speed} + \text{headTime}$$

该公式同硬盘资源满足事件的时间计算公式相同，不再进行说明。需要说明的是，出于效率考虑，船舶控制系统中的消息中间件一般采用 UDP 协议发送消息，不需要确认消息的接收。

4. 共享资源

共享资源用来同步进程对共享软件资源的竞争行为，对应性能仿真指令中的资源请求类型为软件资源。本文允许设计人员在性能仿真指令中给出需要的共享资源数量，并采用先入先出的策略从优先权队列中选取下一个请求进程，满足其资源请求。处理结束后，查看现有资源数量是否能够满足下一个请求进程，如果可以，则继续处理。

不同于前三种资源，共享资源在满足当前资源请求后，不需要等待便可以继续处理下一个资源请求。因而共享资源也需要设计人员在顺序图中通过消息进行释放，释放共享资源对应的资源请求类型为 Release。

4.3.5 系统行为仿真

离散仿真引擎控制器负责事件的处理，它循环处理处理系统中产生的事件，并根据待处理进程请求资源的类型将其交由不同的资源处理。图 25 给出了仿真分析算法流程图。

在初始化阶段，仿真系统为部署图中的每个进程分别创建一个进程加载事件（ProcessLoadEvent），且事件的发生事件为系统时间 0；之后通过该事件获取将要执行的进程，以及执行下一条进程指令所需要的系统资源；然后根据需要的资源类型，将执行进程放入对应资源的进程队列，等待具体系统资源的处理；系统资源按照一定的策略从进程队列中选取将要处理的进程，并按照 4.3.4 中的公式创建资源处理事件放入事件队列中，等待下一次处理。

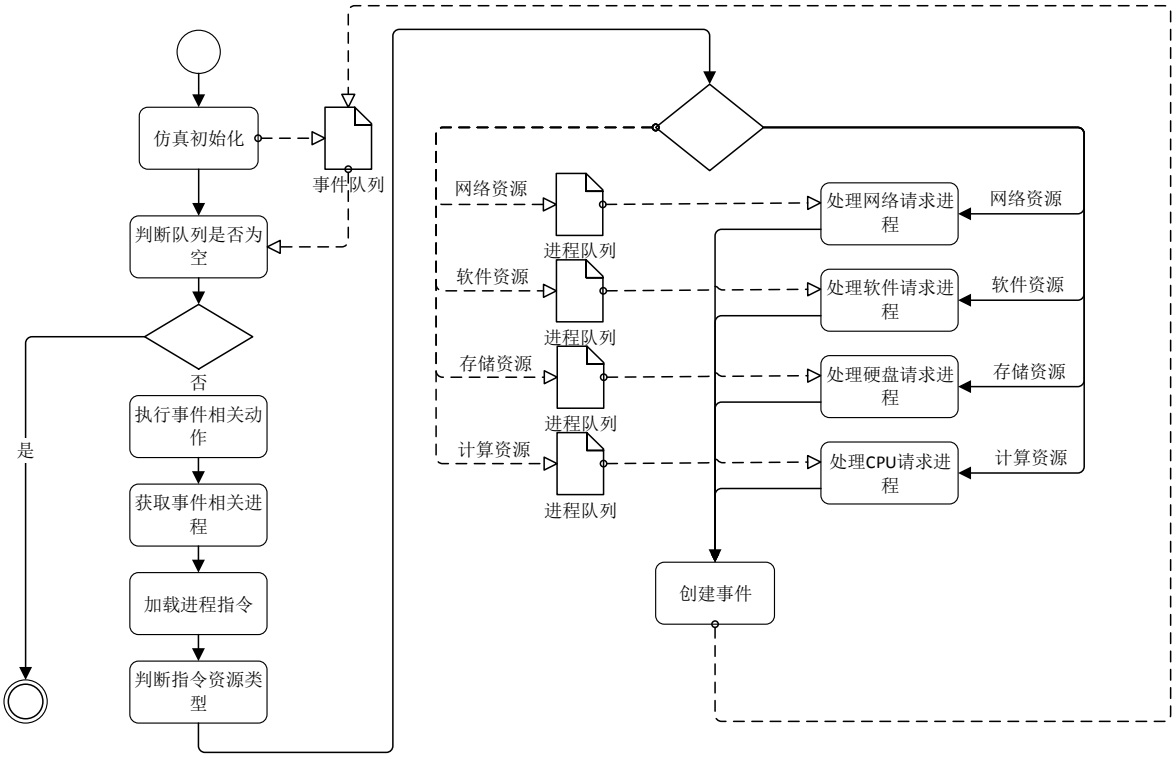

图 25　系统行为仿真流程图

表 17 给出了仿真分析算法的伪代码，其基本流程同图 25 基本相同，这里不再详细介绍。

表 17　仿真分析算法伪代码

算法：基于离散事件仿真器的软件系统性能仿真

输入：软件系统包括的进程，软件系统部署的执行平台

```
01 //initialize this system with process load event.
02 for each process in system do
03     new ProcessLoadEvent() -> event
04     insert event into event list
05 end
06
07 for each event in event list do
08     execute event routine
09     get associated process from event -> process
10     switch(resource type of process) do
11         case processor do
12             dispatch process to processor
13         end
14
```

```
15          case storage do
16              dispatch process to storage
17          end
18
19          case network do
20              dispatch process to network
21          end
22
23          case passive do
24              dispatch process to passive
25          end
26
27          case release do
28              release passive resource
29          end
30      end
31      // notify all the idle resource.
32      for each resource in system do
33          if resource is idle do
34              notify resource
35          end
36      end
37  end
```

4.3. 本章小结

基于 UML 图描述的设计模型以及第三章中定义的性能相关参数，本章设计并实现了性能仿真分析算法。为了解耦模型存储格式与仿真算法，并更好的集成顺序图和活动图描述的系统行为，本章设计并实现了性能仿真指令 PSI（Performance Simulation Instruction），PSI 从顺序图、活动图描述的系统行为以及性能相关参数的角度出发，通过分支、顺序和循环等控制结构指令描述系统执行流程；接下来，通过介绍离散事件仿真的基本原理，以及不同系统资源的特性，本文设计并实现了性能仿真算法，并给出了该算法的流程图以及伪代码实现。下一章将给出消息中间件性能分析系统的设计和实现，在该系统的基础上，本文对实际的设计模型进行了性能分析，以验证方法的有效性。

第五章　消息中间件性能分析系统及实验验证

基于前文选取的消息中间件系统建模语言、定义的性能仿真指令和设计的基于离散事件的性能分析方法，本章设计并实现了消息中间件系统设计模型性能分析系统，并基于该系统对某船舶控制系统的消息中间件进行了性能分析实验，以验证本文提出方法的有效性。本章首先给出了消息中间件系统设计模型性能分析系统的需求、架构与详细设计，并通过实际使用场景展示了系统的功能；接下来，为了验证基于设计模型进行性能分析的有效性，基于某船舶控制系统消息中间件系统的设计模型，进行了性能分析实验，然后对实验结果和实际执行结果进行了分析和比较，实验结果证明本文的方法可以有效的分析消息中间件系统设计的性能。

5.1. 消息中间件性能分析系统

本文在前面章节中重点阐述了设计模型的建模内容、性能仿真指令的设计思路以及核心概念，然后设计了基于性能仿真指令的性能分析方法，并给出了性能仿真过程的部分伪代码实现。基于本文提出的性能相关参数和性能分析方法，本文设计并实现了消息中间件性能分析系统。本节首先介绍了系统需求，接下来给出系统的架构与实现，最后通过实际使用场景展示了系统的功能。

5.1.1. 系统需求

根据第三章中对性能相关参数的需求分析，以及第四章中设计的性能分析算法，本文的消息中间件性能分析系统主要由可视化建模工具、性能仿真指令生成模块和系统设计性能仿真引擎这三个功能模块组成。如图 26 所示：

图 26　系统功能模块

按照功能模块的划分，消息中间件性能分析系统的关键功能需求主要包括以下几个方面：

可视化建模工具：

1. 支持建模 UML 顺序图、活动图和部署图；
2. 提供模型解析接口，或者能够生成 XML 格式的模型文件；
3. 支持建模元素扩展，能够通过 UML 的 Profile 机制方便的添加建模元素。

性能仿真指令生成模块：

1. 能够解析建模模块生成的模型文件；
2. 能够生成第四章中描述的 Json 格式的性能仿真指令；
3. 易于扩展，支持更多类型的性能仿真指令。

系统设计性能仿真引擎：

1. 能够自动解析 Json 格式的性能仿真指令；
2. 能够仿真系统运行，并收集系统运行时数据；
3. 能够输出本文第三章讨论的性能度量目标。

5.1.2. 系统架构

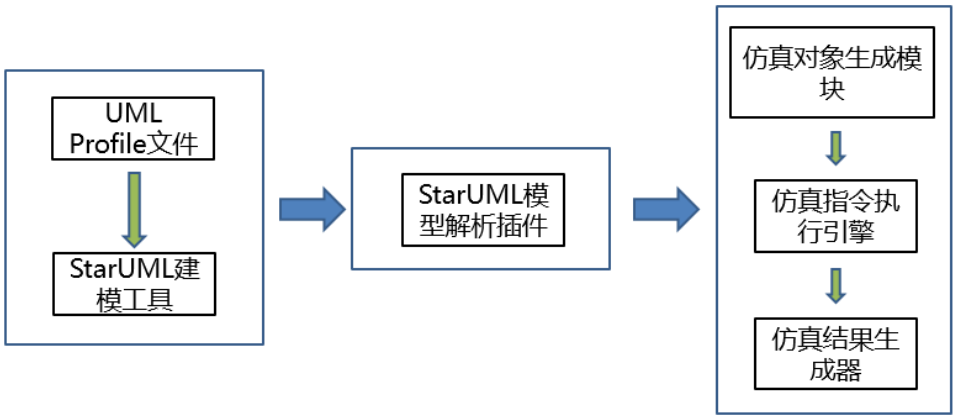

图 27 性能分析系统架构

根据上一节中对系统的分析，本文把系统划分为可视化建模工具、性能仿真指令生成模块和系统设计性能仿真引擎三部分。其中可视化建模工具负责向用户提供 UML 顺序图、活动图和部署图编辑功能，并且允许添加本文扩展的 UML 建模元素；性能仿真指令生成模块负责解析用户绘制的设计模型，提取顺序图、活动图中的性能相关信息并将其转化为性能仿真指令，提取部署图中的执行平台信息并将其转化为配置文件；系统

设计性能仿真引擎读取性能仿真指令以及配置文件，按照第四章中提出的方法仿真系统运行，并输出第三章中给出的性能度量目标。图 27 给出了消息中间件性能分析系统架构，接下来对各个子模块进行详细的介绍。

StarUML 建模工具：

由于本文基于 UML 顺序图、活动图和部署图，有多种可视化建模工具可以使用。StarUML[36]是一个开源的 UML 建模工具，较为稳定；对 UML 的 Profile 扩展机制提供了较好的支持，能够方便扩展 UML 建模元素；同时 StarUML 为开发人员提供了模型访问编程接口，便于模型的解析。

UML Profile 文件：

StarUML 本身并不支持 MARTE 建模语言，因而需要对其进行扩展，添加建模性能相关参数的能力。利用 StarUML 提供的 UML 扩展机制，开发人员只需要按照规定的格式编写 Profile 扩展文件，并将其放至给定的目录，就可以在 StarUML 中设计模型时使用。

StarUML 模型解析插件：

StarUML 还为模型解析提供了编程接口，允许开发人员通过给定的接口，访问和编辑设计模型。编程接口通过 ActiveXObject 技术对外提供，支持多种类型的编程语言。本文需要的功能较为简单，实现设计模型到性能仿真指令的生成，以及配置文件的生成即可，因而选用脚本语言 Javascript 编写 StarUML 的插件。

仿真对象生成模块：

性能仿真指令并不能直接执行，需要生成相应的类对象。每个性能仿真指令对应某个指令类的对象；部署图描述的执行平台信息，如处理器、硬盘等，也需要通过配置文件生成相应的对象。而且本文以多次仿真，计算平均值的方式获取性能度量目标，每次执行都解析文本格式的性能仿真指令显然较慢。利用该模块读取性能仿真指令以及系统配置信息文件，生成对应的对象，建立各个对象之间的关联，最后交给仿真指令执行器能够有效的提高仿真效率。

仿真指令执行器：

仿真指令执行模块以仿真对象生成模块产生的性能仿真对象，以及资源对象作为输入，以仿真运行时产生的事件作为输出。该模块定义了基本的指令类、事件类、资源类

以及控制仿真系统运行的控制器。代码生成器模块便是利用本模块提供的类生成的对象。

仿真结果生成器：

仿真结果生成器负责记录仿真系统运行过程中产生的同性能度量目标相关的数据，然后对这些数据进行统计，并以 HTML 的格式保存。

5.1.3. 系统的详细设计

UML Profile 文件

StarUML 支持标准的 UML 的 Profile 机制，但是对 Profile 文件的格式进行了自定义。Profile 文件格式包括三个部分：构造型（stereotype）定义，标签（tag）定义，类型（data type）定义。其中构造型用来标示扩展，构造型中的标签定义了该构造型扩展的属性集合，而数据类型定义了该扩展中需要的新的数据类型的名称。

StarUML 模型解析插件：

模型解析插件利用 StarUML 提供的编程接口解析顺序图、部署图，生成性能仿真指令和配置信息，以文件的形式保存。配置信息的生成较为简单，表 18 给了出根据顺序图生成指令的伪代码。

算法由两个函数组成，generate_instruction 函数负责解析顺序图，输入顺序图，并生产性能仿真指令；get_children 函数负责解析消息数组。算法执行时，首先获取顺序图中的所有消息（02 行代码），然后交由 get_children 函数处理；get_children 函数循环处理传入的消息数组（第 9 行至第 35 行）：如果下一条消息是当前消息的返回，则当前消息没有子消息，可直接生成原子性能仿真指令（第 10 行至第 17 行）；如果下一条消息不是当前消息的返回，则循环读入消息直到找到当前消息的返回，将其子消息交由 get_children 函数处理，然后根据具体的消息类型创建复合指令（第 19 至第 32 行）；处理完成后，get_children 返回生成的性能仿真指令。

表 18　顺序图生成性能仿真指令算法

算法：顺序图生成性能仿真指令

输入：带有性能标注的 UML 顺序图

输出：性能仿真指令

```
01 function generate_instruction(diagram)
02 messages of sequence diagram -> messages
03 get_children(messages) -> result
04    return result
05 done
06
07 function get_children(messages)
08    array of instructions -> children
09    for 1, messages.length -> i do
10       if messages[i+1] is return of messages[i] do
11          switch (messages[i].type) do
12             case normal: new normal instruction -> atomic
13             case message transport: new message instruction -> atomic
14             case sync: new sync instruction -> atomic
15          done
16          push atomic into children
17          i + 2 -> i
18       else
19          array- > instructions
20          while (messages[i+1] is not return of messages[i]) do
21             push message[i+1] into instruction
22             i + 1 -> i
23          done
24
25          switch (messages[i].type) do
26             case loop:   new Loop Instruction -> embedded
27             case branch: new Branch Instruction -> embedded
28             case async:  new async instruction -> embedded
29             default : new Embedded Instruction -> embedded
30          done
31
32          get_children(instructions) -> embedded.instructions
33       done
34    done
35    return children
36 done
```

仿真指令执行模块：

该模块定义了系统性能仿真的核心类，包括指令、事件、资源三部分，以及用于控制仿真运行的控制器类。

1. 性能仿真指令

根据第四章中定义的性能仿真指令，本文设计了如图 28 所示的指令类。Instruction 为指令接口，定义了指令对象提供的对外接口；NormalInstruction 表示原子指令，定义了描述资源需求的属性；EmbddedInstruction 类表示复合指令，包括 Instruction 类组成的子指令。每种指令的具体含义以及包括的属性可以参考第四章中的相关内容。

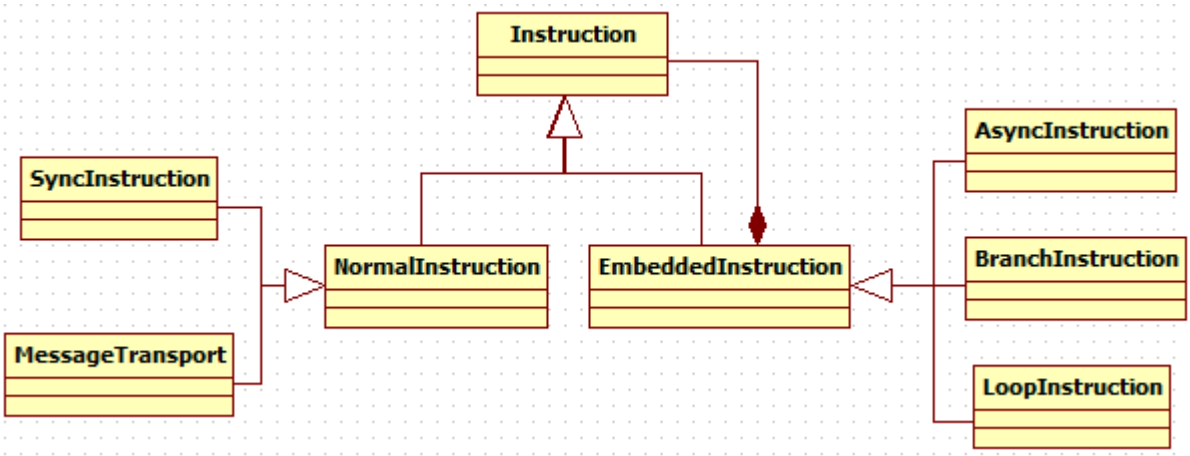

图 28 指令类及其关系

2. 离散事件

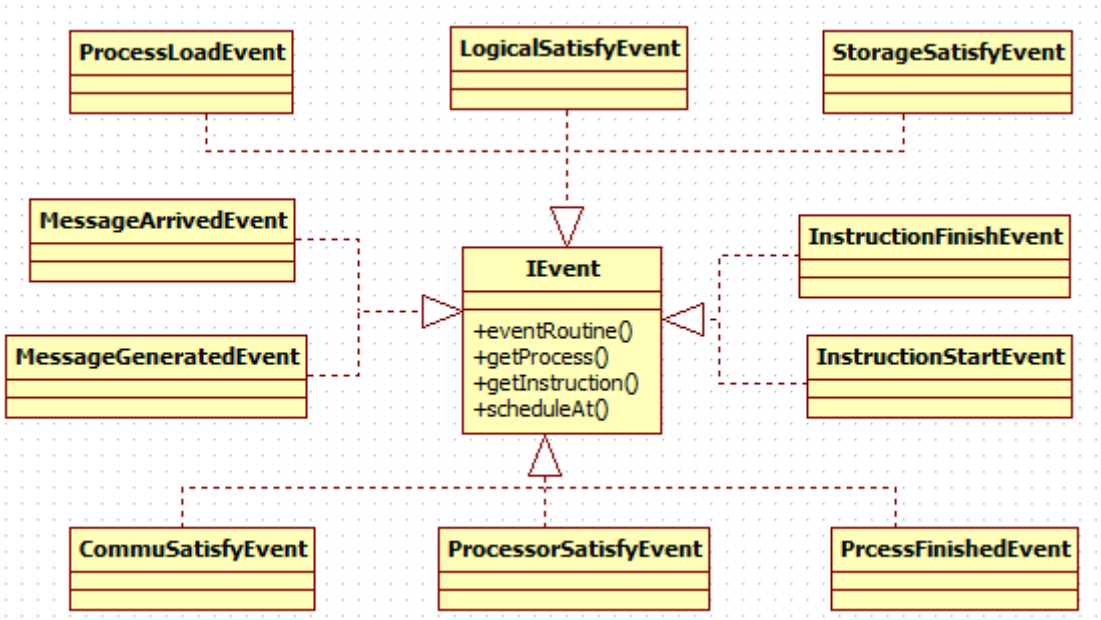

图 29 事件类

在离散仿真系统中，事件用于驱动仿真的运行。本文通过仿真系统运行过程中的资源请求以及消息流分析系统设计的性能，为此定义了如图 29 所示的 10 种事件。IEvent 定义了事件的抽象接口，eventRoutine 为该事件发生时执行的动作， scheduleAt 返回事件的发生时间，getProcess 和 getInstruction 方法分别返回同该事件相关的进程对象和性能仿真指令对象，可以为空。表 19 给出了图 29 中各种类型事件的描述。

表 19 事件描述

事件名称	描述
ProcessLoadEvent	表示进程的加载；
ProcessFinishedEvent	表示进程执行结束；
InstuctionStartedEvent	表示方法开始执行；
InstructionFinishedEvent	表示方法正结束；
MessageGeneratedEvent	表示有一个新消息产生；
MessageArrivedEvent	表示有一个消息到达某计算节点；
CommuSatisfyEvent	表示网络资源满足了某个进程的网络资源请求，在事件发生时处理结束；
StorageSatisfyEvent	表示存储资源满足了某个进程的存储资源请求，在事件发生时处理结束；
ProcessorSatisfyEvent	表示计算资源满足了某个进程的计算资源请求，在事件发生时处理结束；
LogicalSatisfyEvent	表示软件资源满足了某个进程的软件资源请求，在事件发生时处理结束；

3. 系统资源

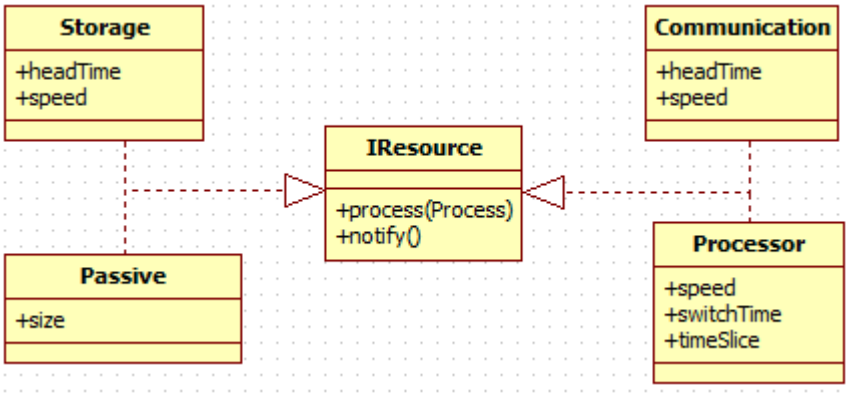

图 30 资源类

通过 4.2 节的分析，本文设计了如图 30 所示的资源类，表示不同类型的系统资源。IResource 定义了资源类对外提供的接口；Storage 表示存储资源；Communication 表示网络资源；Processor 表示计算资源；Passive 表示软件资源。每类系统资源的特性可以参考 4.2 节中的相关内容，这里不再赘述。

仿真结果生成模块：

仿真结果生成模块负责记录系统运行时产生的数据，这些数据包括发生的不同类型事件，以及各类资源进程队列随系统运行的变化。因而该模块需要监听仿真系统中发生的事件，以及资源进程队列的变化情况。IObserver 接口提供了两个方法，分别用于记录事件的到达以及队列长度的变化；ReportGenerator 则根据 IObserver 收集的数据，生成性能度量目标，作为设计人员进行性能分析的量化依据。

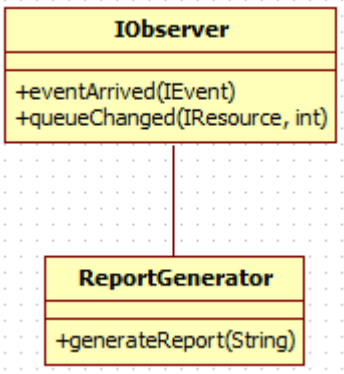

图 31 仿真结果生成模块组成结构

5.1.4. 系统展示

本系统的核心功能是通过绘制消息中间件系统的设计模型，分析顺序图和部署图，生成性能仿真指令和配置文件，最终仿真分析系统设计的性能。其中模型绘制、性能仿

真和性能度量目标输出是本文的核心场景，本小节从这几个使用场景出发，通过系统截图和对应的说明文字，对系统的功能进行详细演示。

使用场景 1 绘制部署图及性能相关参数

部署图用来描述软件系统的执行平台，以及软件系统的部署情况。启动 StarUML 后，打开 Model Explorer 面板，右键<<deploymodel>> Deploymodel，添加部署图。为了在部署图中添加性能相关参数，右键 Artifact 建模元素，选择 Profile for MARTE 标签页。如图 32 所示。

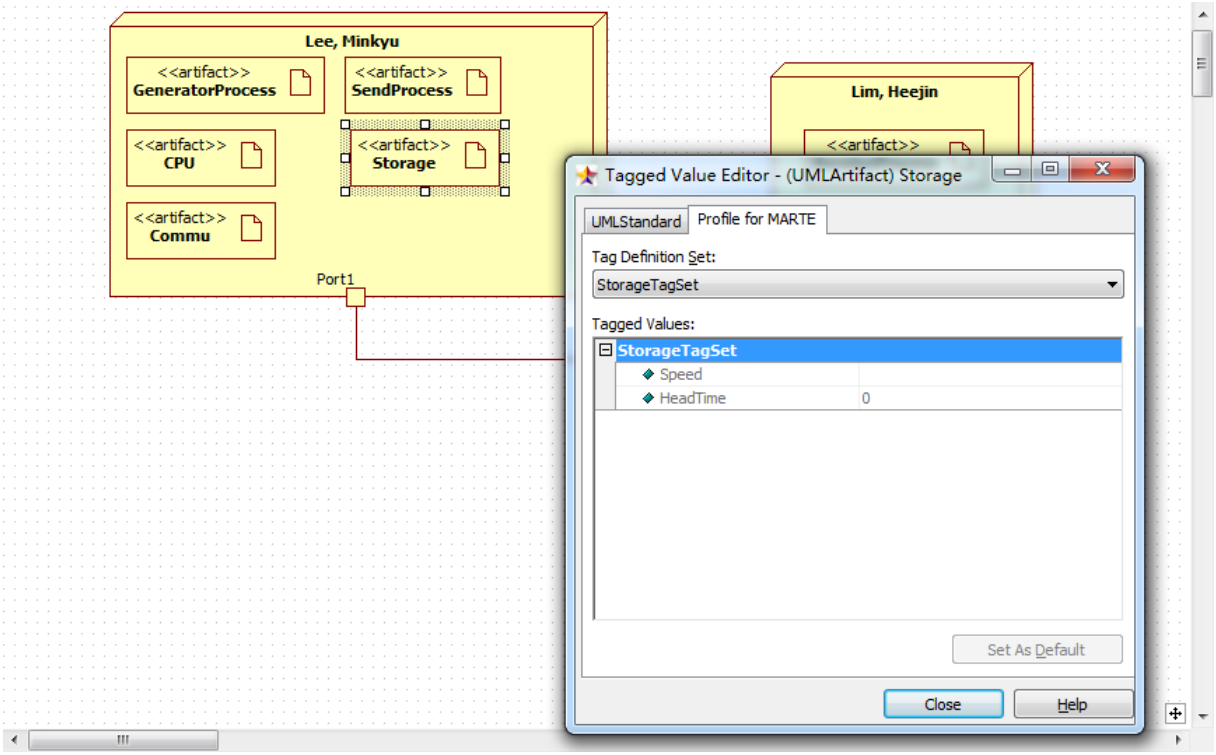

图 32 绘制部署图

使用场景 2 绘制顺序图及性能相关参数

启动 StarUML 工具，即可使用 StarUML 提供的建模元素，以及本文提供的 MARTE 扩展建模系统动态行为，并添加相关的性能相关参数。如图 33 所示。

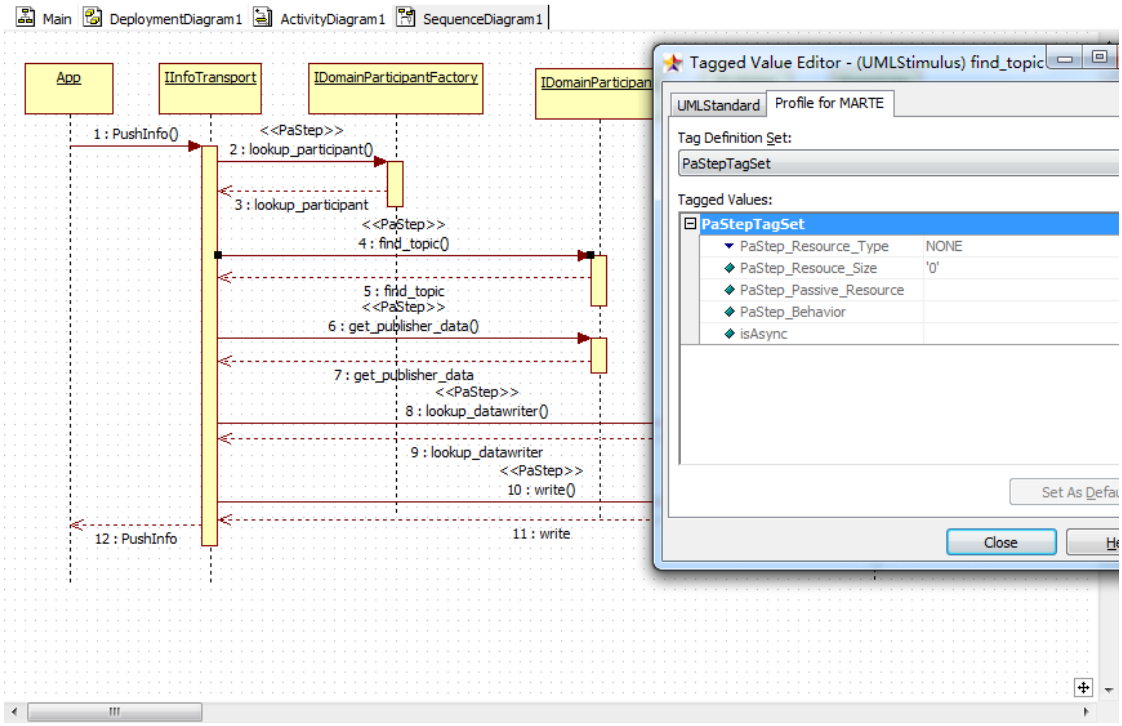

图 33 顺序图建模界面

使用场景 3 启动性能仿真

选择要仿真的部署图，打开 StarUML 的 Tools 菜单，选择 Performance Analysis，系统就会自动解析当前工程中包含的模型，生成性能仿真指令和配置文件，并仿真系统设计。如图 34 所示。

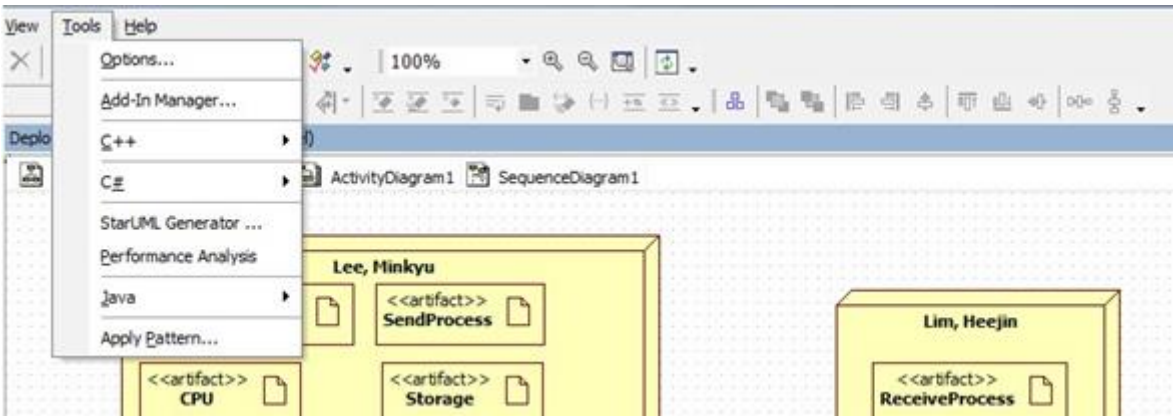

图 34 启动性能仿真

使用场景 3 生成性能度量目标

生成性能仿真指令和配置文件之后，打开 StarUML 的 Tools 菜单，选择 Run Simulation，系统会自动生成性能仿真代码，并编译执行，将性能度量目标以文本格式保存在当前项目目录下，生成的性能度量目标文件如图 35 所示。

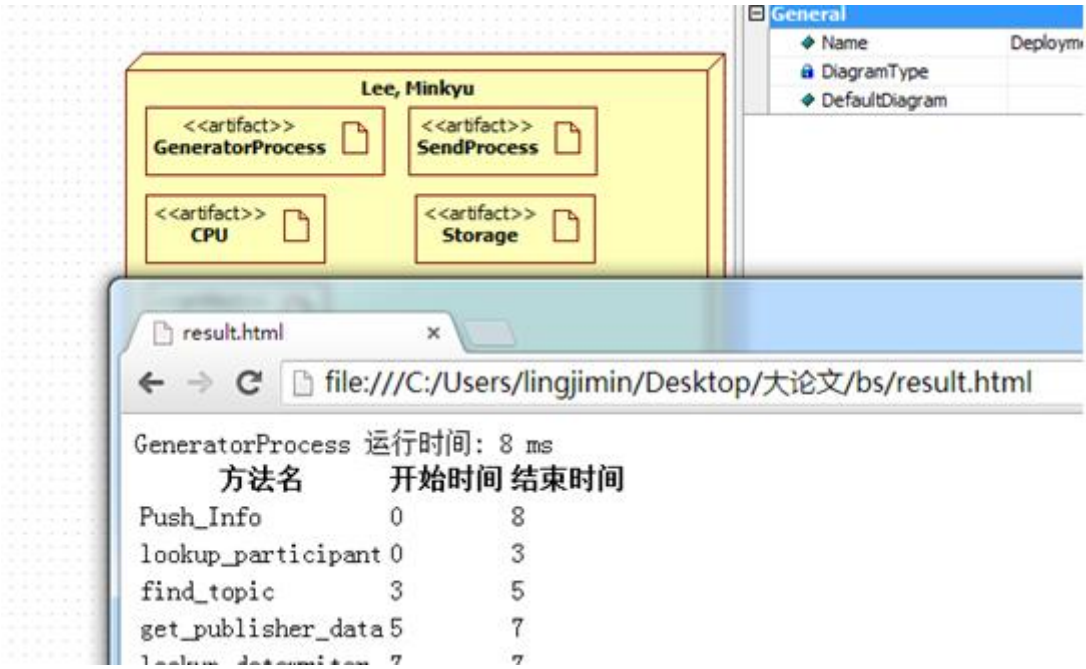

图 35 输出性能度量目标

5.2. 基于船舶消息中间件系统的实验验证

为了验证本文提出的基于设计模型分析系统设计性能方法的有效性，本文对某船舶控制系统的消息中间件系统进行了基于设计模型的性能分析。船舶领域对软件系统具有较为严格的性能需求，如果不能满足，可能会引起严重的后果。传统的方法是在系统实现后，通过性能测试发现问题，再尝试解决遇到的问题。这种方法可能会浪费大量的人力和时间，因为很多性能问题是设计缺陷导致的，不是代码优化能够解决的，只能重构已实现的系统。因此，在设计阶段就对系统设计进行分析，及早发现设计中的不足，能够避免不必要的人力和时间浪费。

本节以某船舶控制系统中的消息中间件为实验背景，首先介绍了实验的内容以及过程，并给出了用于性能分析的设计模型；然后对实验结果进行了分析和总结。

5.2.1. 实验过程及结果

该实验包括两个阶段。在第一阶段，设计人员为了实现同一节点上多应用共享消息中间件，采用了共享内存的方式同外部应用进行交互，并利用 TCP 协议传输消息，其系统架构如图 36 所示。其中 InfoInteratel_dmd 向外部应用构件提供消息操作接口，将待发送的消息写入共享内存；节点代理不断循环读取共享内存中的消息，将其发送到网络

中。

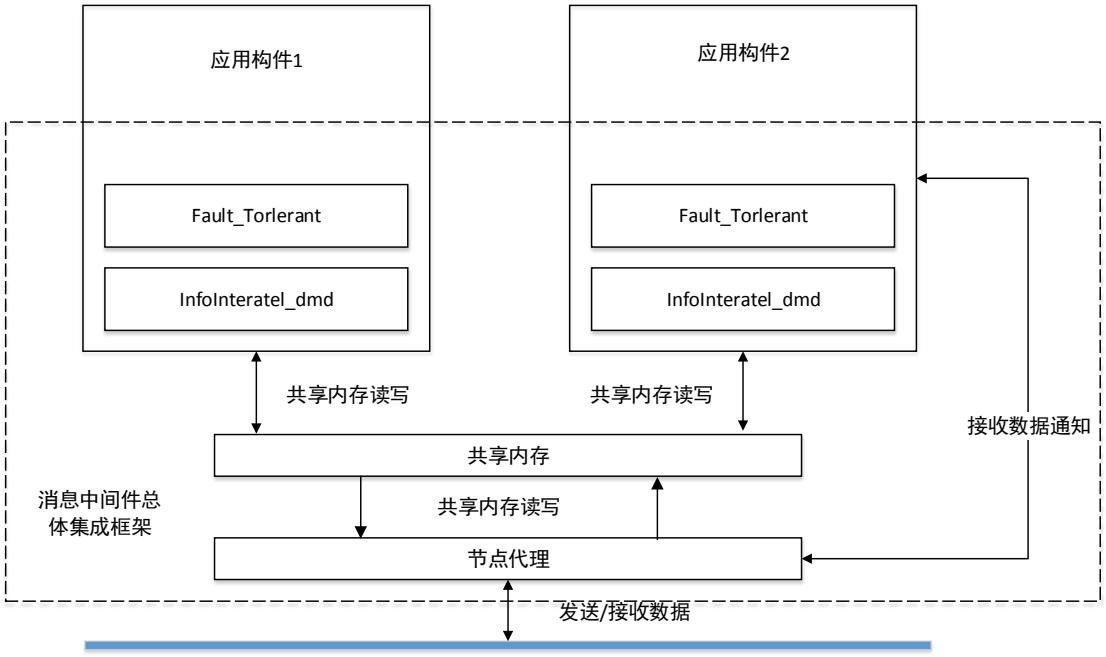

图 36 初始设计总体集成框架执行流程示意图

通过对该系统进行性能测试，得到了如图 37 所示的性能测试结果，发现其性能无法满足系统需求。

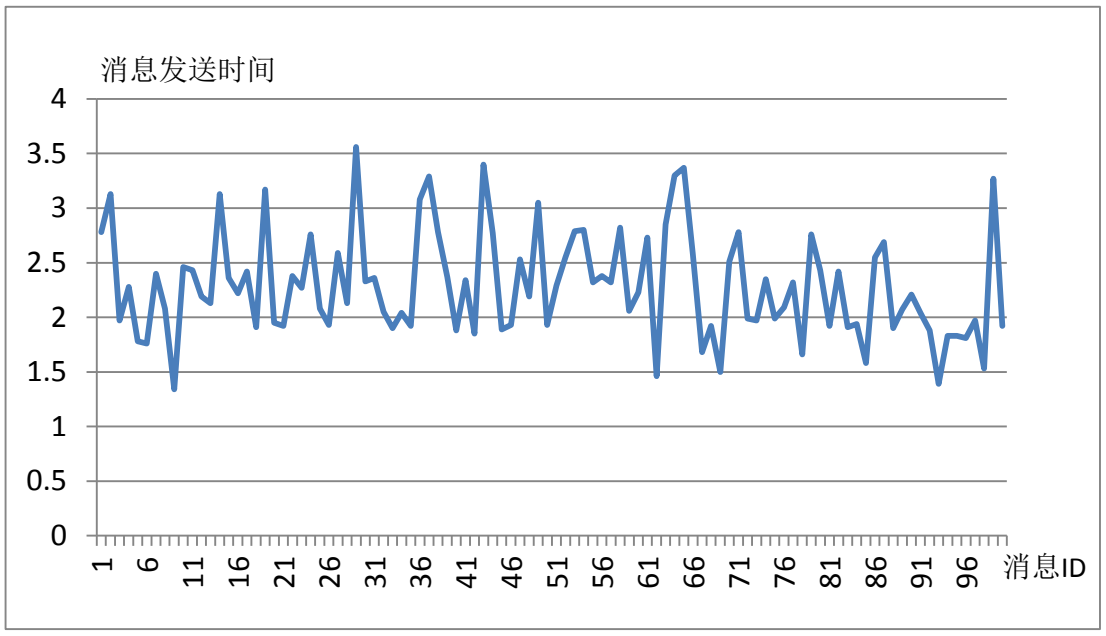

图 37 初始设计性能测试结果

设计人员随后对初始系统设计的设计模型进行了性能分析。图 38 给出了描述消息发送接口系统设计的顺序图，其中 create_datawriter 操作需要获取读写共享内存的锁 Mutex，而 add_info 操作会释放消息信号量 Semaphore。

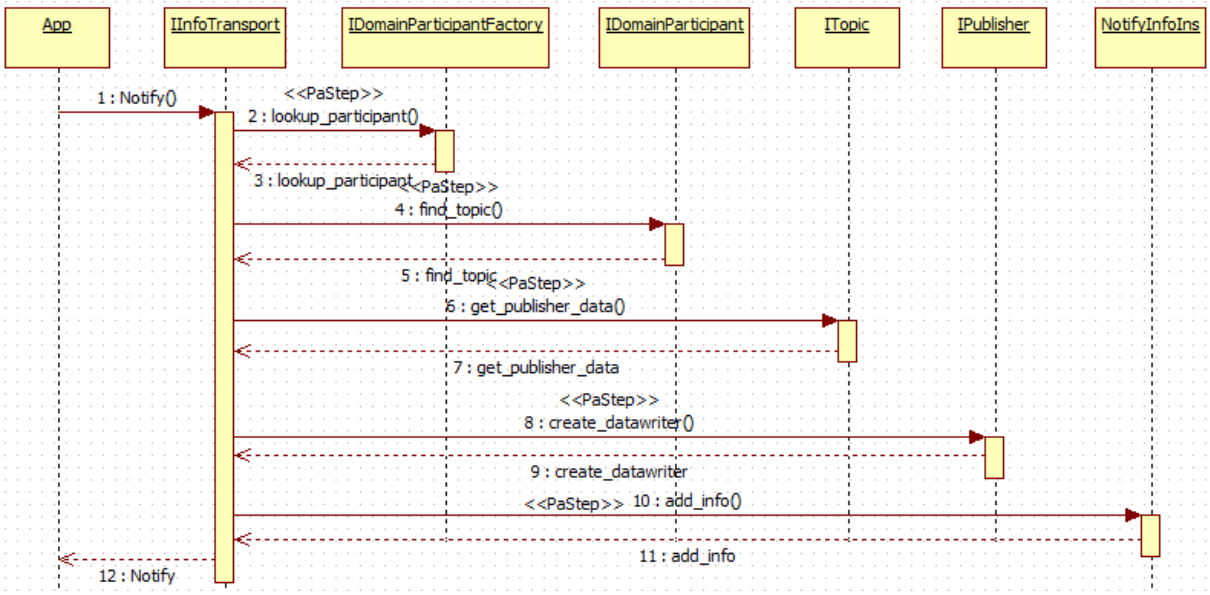

图 38　第一阶段实验消息中间件发送接口设计

图 39 给出了描述节点代理发送消息系统行为的顺序图，Notify 方法为循环结构，循环次数设置为-1，表示无限循环；lookup_participant 系统方法需要获取消息信号量 Semaphore；get_publisher_data 方法需要获取共享内存锁 Mutex；write 系统方法具有 MessageTransport 消息操作语义构造型，表明该操作将消息发送到网络中。

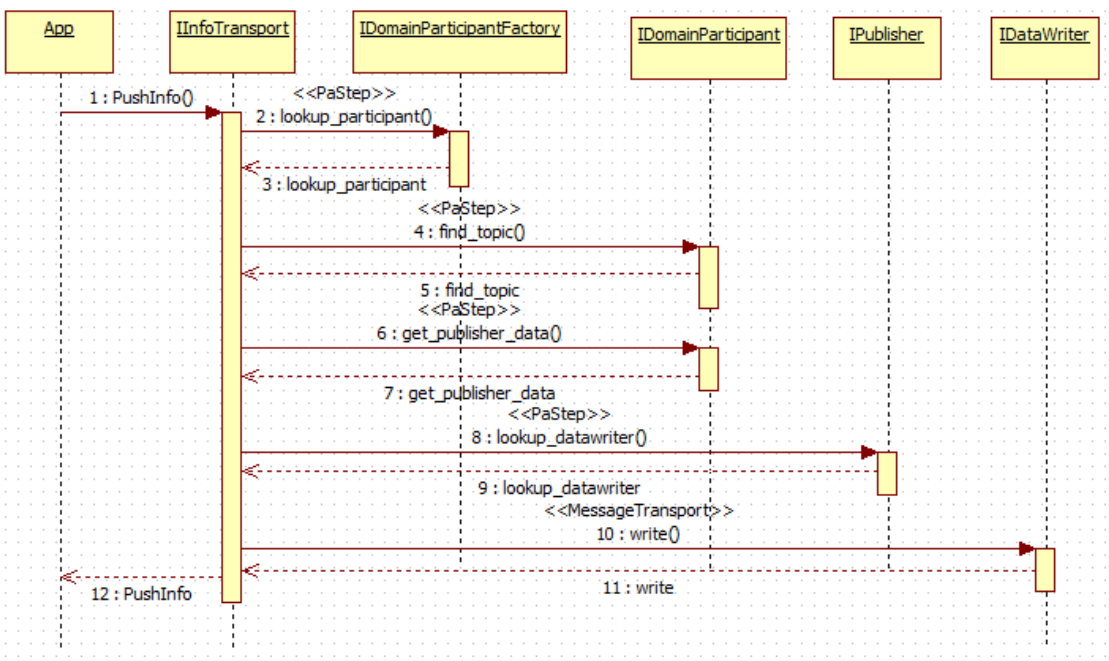

图 39　第一阶段实验节点代理设计

图 40 给出了描述系统部署的部署图，包括两个计算节点和一个路由设备。人工制品（artifact）GenreatorProcess 以及 SubscriberProcess 描述该次实验的外部负载。

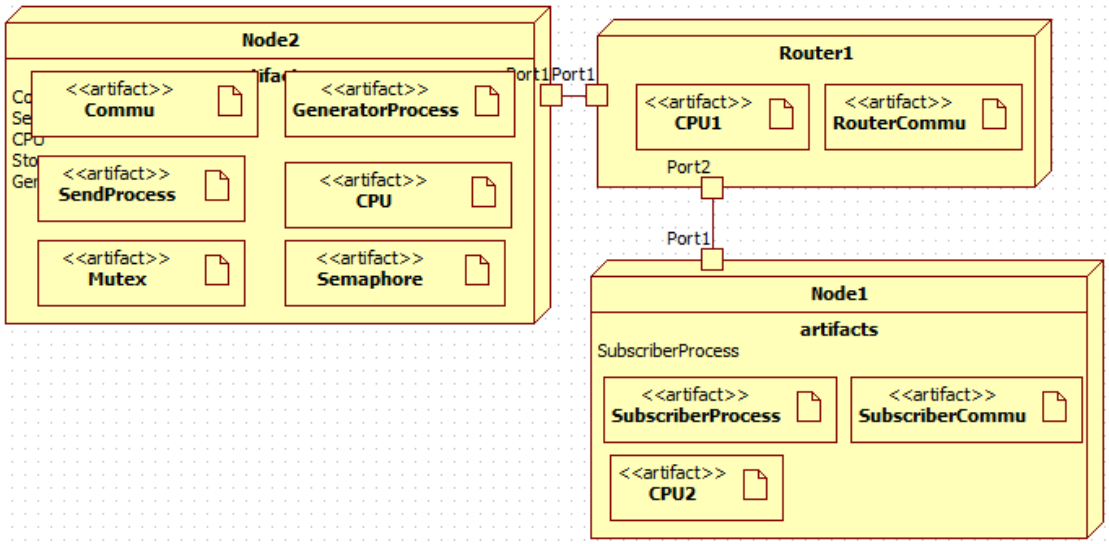

图 40 第一阶段实验部署图

在 Node2 计算节点中，人工制品 GenerateProcess 具有 SendMessage 构造型，具有如图 41 所示的属性值，描述了消息产生外部负载，其各个属性的含义请参考第三章中的内容；人工制品 SendProcess 表示节点代理进程，具有 UML 的 Process 构造型；其它的人工制品描述该计算节点上具有的系统资源，一个处理器，一个存储设备，一个网络设备以及两个用于控制共享内存访问的软件资源。Router1 描述路由设备，负责连接不同的计算节点，具有一个处理器和一个网络设备。

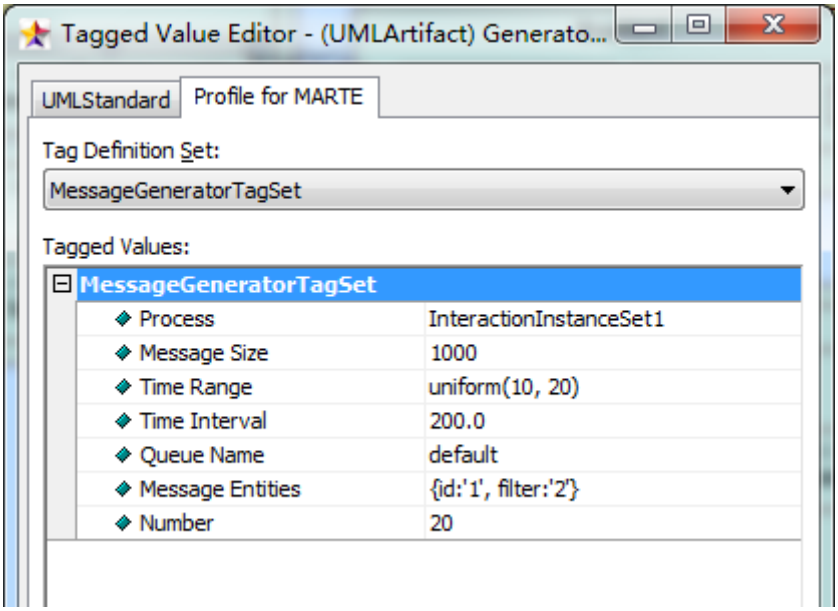

图 41 消息发送外部负载配置

Node1 节点中，人工制品 SubscribeProcess 具有 SubscribeMessage 构造型，用于描述消息订阅外部负载，其各属性值如图 42 所示；其它两个人工制品表示该计算节点执

行需要的系统资源，包括一个处理器以及一个网络设备。

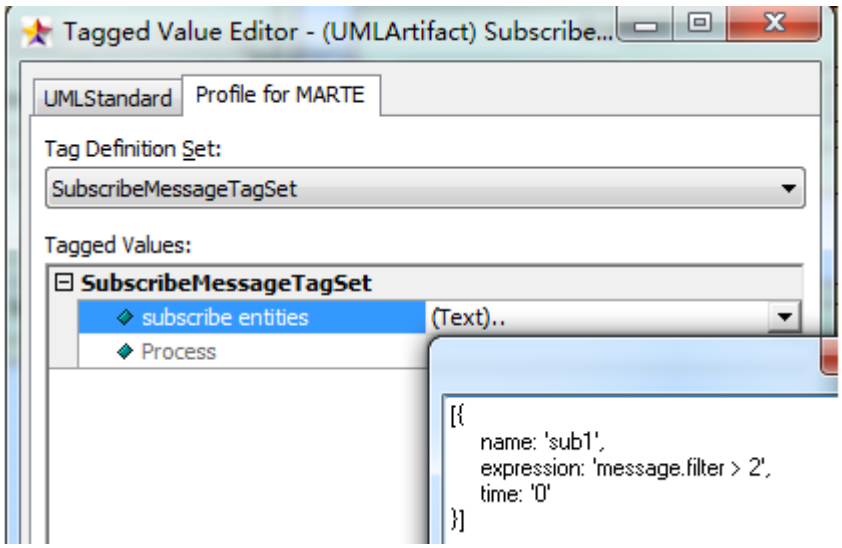

图 42　消息订阅外部负载配置

初始设计的性能分析结果如图 43 所示，可以发现分析结果同性能测试结果较为接近，推测可能是由设计问题导致的性能问题。

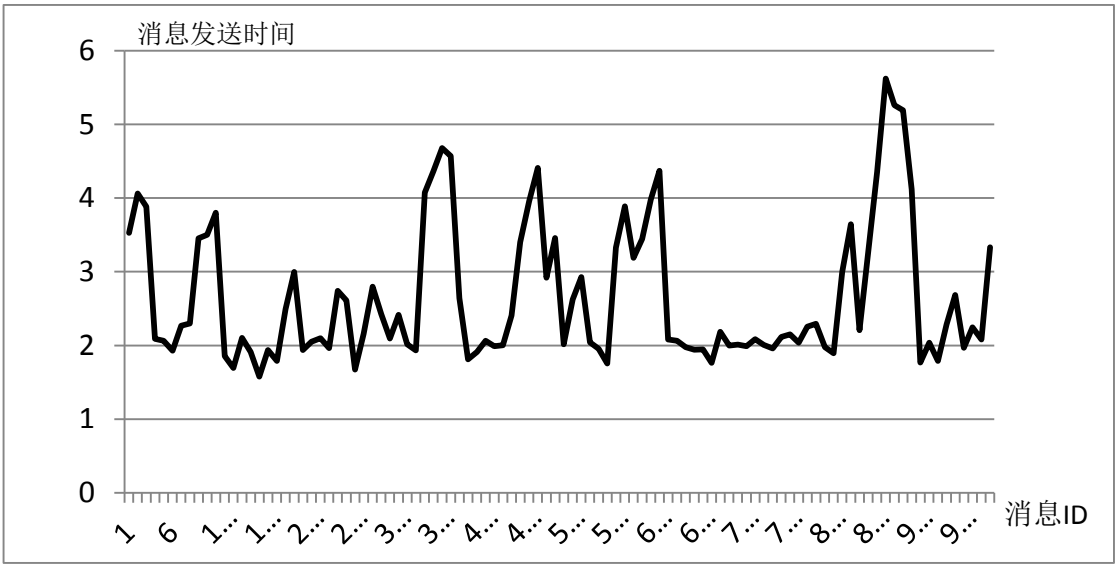

图 43　初始设计性能分析结果

在第二阶段，由于初始设计可能存在性能缺陷，设计人员对初始设计进行了改进，其系统架构如图 44 所示。

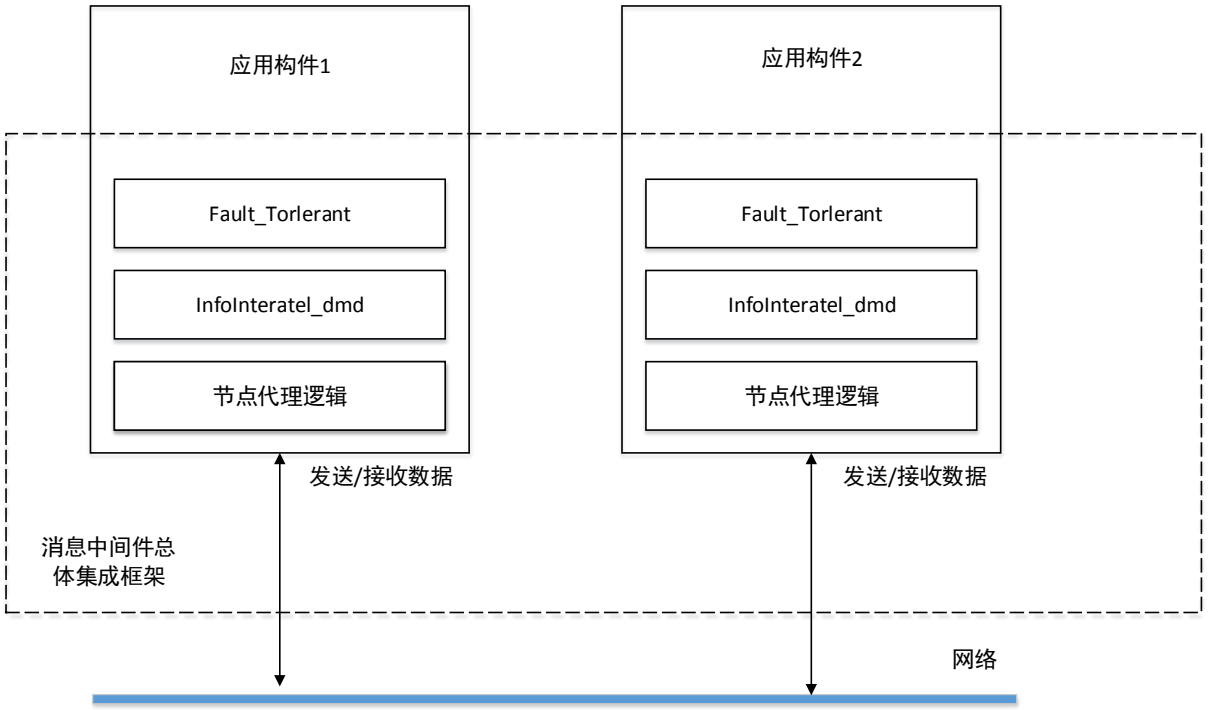

图 44 修改后总体集成框架执行流程示意图

该系统架构将消息中间件接口和网络操作实现集成在 infoInteral_dmd 模块中，并取消了共享内存设计，实现各个应用构件在发送/接收数据时不从节点代理进行中转，减少了处理中间环节；采用 UDP 组播技术管理消息的订阅，减少了数据传输量以及传输延迟。该系统设计的变更主要体现在网络技术，以及系统执行过程中对资源的请求，消息接收/发送流程的变化并不大，所以不再给出描述消息发送行为的顺序图。

改进后系统设计的性能分析结果如图 45 所示。该性能分析结果同初始设计相比，性能有 30%的提升。

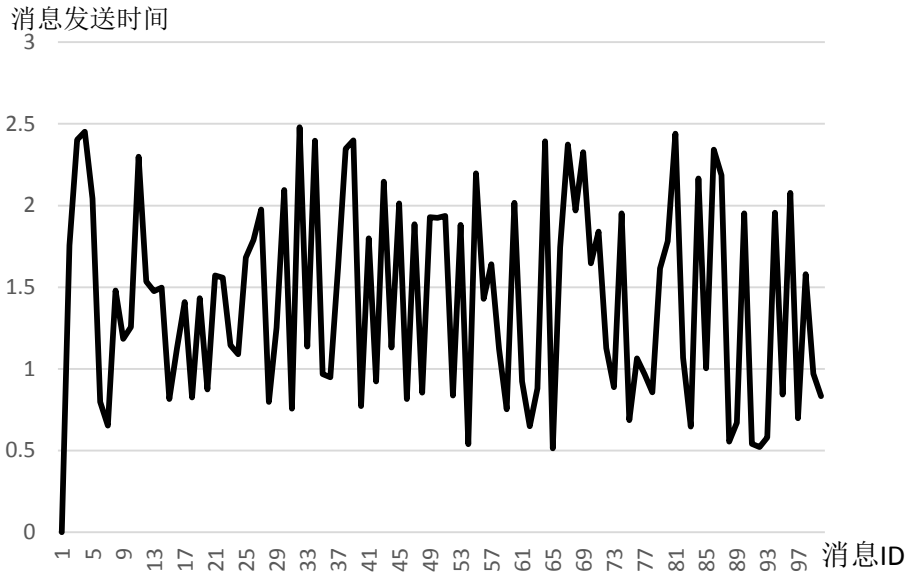

图 45　改进设计后设计性能分析结果

设计人员按照改进后的系统设计实现消息中间件系统，其性能测试结果如图 46 所示，该结果显示性能提升了 35%。

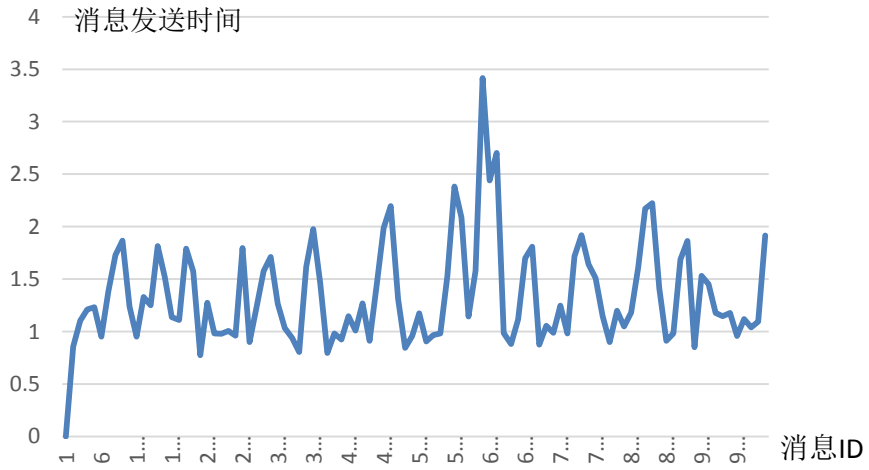

图 46　改进设计后性能测试结果

5.2.2. 实验结果分析

在第一阶段，设计人员利用本文提出的方法对系统设计进行性能分析，发现性能分析结果和性能测试结果较为接近，从而猜测是由设计缺陷造成的性能问题。在第二阶段，设计人员改进了初始系统设计，并对其进行性能分析，发现性能有较大的提升。然后实现了改进后的系统设计，并对其进行性能测试，测试结果表明系统性能确实有了较大的提升，从而说明本文提出的方法对于设计人员评估系统性能，改进系统设计有一定的实

际意义。

5.3. 本章小结

基于面向性能分析的设计模型和基于性能仿真指令的仿真分析算法，本章实现了消息中间件性能分析系统，并基于该系统进行了设计模型性能分析实验，以验证本文方法的有效性。本章首先给出了消息中间件性能分析系统的需求、架构和详细设计，然后介绍了可视化建模工具、性能仿真指令生成模块和系统设计性能仿真引擎这三个核心模块，并通过实际使用场景展示了系统功能；接下来，为了验证本文的方法，以船舶控制系统中的消息中间件为背景，基于其设计模型进行了性能分析实验，然后同实际执行结果进行了比较，实验结果证明本文的性能分析方法可以较好的分析消息中间件设计模型性能，为设计人员提供量化评估依据。

结论和展望

论文工作总结

根据目前国内外对基于模型的软件系统性能分析的研究现状和发展趋势，以及某船舶控制系统中消息中间件的实际项目需求，本文提出并设计了一种基于 UML 设计模型的性能分析方法，该方法支持在设计模型中建模性能相关参数，并自动分析系统设计的性能。具体而言，本文的主要工作内容由以下几个方面组成：

1. 根据消息中间件的特点，确定需要添加的性能相关信息，以及这些信息的建模方法。

目前在设计阶段分析系统设计性能的方法，都是针对一般性的系统，没有考虑消息中间件这类系统的特点。本文在相关研究的基础上，并根据消息中间件的系统特性，选取、定义了用于分析消息中间件系统设计性能的性能度量目标。由此出发，确定需要在设计模型中添加的性能相关信息，从而为后面的分析提供足够的数据。设计模型一般使用 UML 描述，它本身并不具备建模这些性能相关信息的能力，本文通过借鉴 MARTE 语言的相关概念，对 UML 进行了扩展，定义了消息中间件性能分析语言 MePAL。

2. 提出一种自动化的设计模型性能分析方法，该方法应该能够自动分析设计模型，输出需要的量化结果，同时具备良好的效率。

为了减轻开发人员的负担，方便相关人员在设计阶段评估系统设计的性能，本文提出了一种自动化的设计模型性能分析方法，该方法解析带有性能相关信息的设计模型，将其转换为其它易于处理的格式，并通过多次运行获得性能度量目标的平均结果。

3. 设计并实验了消息中间件性能分析系统，并通过实验验证了本文提出方法的有效性。

为了验证本文提出方法的可行性，本文设计并实现了消息中间件设计模型性能分析系统，然后基于该系统对某船舶控制系统的消息中间件进行了性能分析实验，最后对分析结果进行总结和分析。实验结果证明本文的方法可以能够较好的分析船舶控制系统领域内消息中间件系统设计的性能。

进一步工作展望

本文虽然已经取得了一定的成果，但仍然存在有待进一步研究的问题，具体包括以下几个方面：

1. 尽管实验结果显示本文的方法可以有效的分析消息中间件的系统性能，但同实际运行结果还有一定的差距。差距存在的原因主要有两方面：一是由于资源行为的仿真粒度过粗，无法精确地反应真实的资源行为；二是消息语义的建模能力有限，缺乏某些行为的建模能力。针对这两个问题，在接下来的研究中，一方面通过分析真实资源的行为，对资源行为的模拟进一步细化；另一方面通过分析现有建模元素无法描述的消息中间件语义，对其进行扩充。

2. 对 MePAL 提供更为完善的工具支持。目前实现的消息中间件性能分析系统仅支持 MePAL 语言中建模系统行为的部分建模元素，针对这个问题，在接下来的工作中，本文会进一步完善系统的实现。

参考文献

[1] Oracle Corporation, http://docs.oracle.com/cd/E19148-01/820-0533/aeraq/index.html 2010.

[2] Schuldt H. Message-oriented Middleware[J].

[3] Terpstra W W, Behnel S, Fiege L, et al. A peer-to-peer approach to content-based publish/subscribe[C]//Proceedings of the 2nd international workshop on Distributed event-based systems. ACM, 2003: 1-8.

[4] Eugster P T, Felber P A, Guerraoui R, et al. The many faces of publish/subscribe[J]. ACM Computing Surveys (CSUR), 2003, 35(2): 114-131.

[5] XU J, XU W. Summarization of Message-oriented Middleware [J]. Computer Engineering, 2005, 16: 029.

[6] Mühl G, Fiege L, Buchmann A. Filter similarities in content-based publish/subscribe systems[M]//Trends in Network and Pervasive Computing—ARCS 2002. Springer Berlin Heidelberg, 2002: 224-238.

[7] Baldoni R, Virgillito A. Distributed event routing in publish/subscribe communication systems: a survey[J]. DIS, Universita di Roma La Sapienza, Tech. Rep, 2005: 5.

[8] Banavar G, Chandra T, Strom R, et al. A case for message oriented middleware[M]//Distributed Computing. Springer Berlin Heidelberg, 1999: 1-17.

[9] 樊志强, 张莉, 李琳. 一种面向构件的 C4ISR 技术体系结构建模方法[J]. 系统仿真学报, 2011, 23(7): 1297-1304.

[10] Smith C U, Williams L G, 唐毅鸿, 等. 软件性能工程[J]. 2003.

[11] Curry E. Message-oriented middleware[J]. Middleware for communications, 2004: 1-28.

[12] Object Management Group, http://www.uml.org/ 2013.

[13] Marzolla M. Simulation-based performance modeling of UML software architectures[C]//Dipartimento di Informatica, Università Ca'Foscari di Venezia. 2004.

[14] Object Management Group, "UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE) RFP", document.

[15] Smith C U. Performance Engineering ofSoftware Systems[J]. Addison-Wesley, 1990, 1: 990.

[16] Woodside M, Franks G, Petriu D C. The future of software performance engineering[C]//Future of Software Engineering, 2007. FOSE'07. IEEE, 2007: 171-187.

[17] Bittner K. Use case modeling[M]. Addison-Wesley Longman Publishing Co., Inc., 2002.

[18] Baldoni R, Contenti M, Piergiovanni S T, et al. Modeling publish/subscribe communication systems: towards a formal approach[C]//Object-Oriented Real-Time Dependable Systems, 2003.(WORDS 2003). Proceedings of the Eighth International Workshop on. IEEE, 2003: 304-311.

[19] Rathfelder C, Klatt B, Sachs K, et al. Modeling event-based communication in component-based software architectures for performance predictions[J]. Software & Systems Modeling, 2013: 1-27.

[20] Mühl G, Schröter A, Parzyjegla H, et al. Stochastic analysis of hierarchical publish/subscribe systems[M]//Euro-Par 2009 Parallel Processing. Springer Berlin Heidelberg, 2009: 97-109.

[21] Poernomo I. The meta-object facility typed[C]//Proceedings of the 2006 ACM symposium on Applied computing. ACM, 2006: 1845-1849.

[22] Fuentes-Fernández L, Vallecillo-Moreno A. An introduction to UML profiles[J]. UML and Model Engineering, 2004, 2.

[23] S. Balsamo and M. Marzolla. A Simulation-Based Approach to Software Performance Modeling. In Proceedings of the 9th European software engineering conference held jointly with 10th ACM SIGSOFT international symposium on Foundations of software engineering, pages 363–366. ACM Press, 2003.

[24] Becker S, Koziolek H, Reussner R. The Palladio component model for model-driven performance prediction[J]. Journal of Systems and Software, 2009, 82(1): 3-22.

[25] OMG: UML Profile for Schedulibility, Performance and Time Specification. Adopted Specification (2005), http://www.omg.org/docs/formal/05-01-02.pdf.

[26] Object Management Group, http://www.omgmarte.org/node/2 2010.

[27] Rabiner L, Juang B. An introduction to hidden Markov models[J]. ASSP Magazine, IEEE, 1986, 3(1): 4-16.

[28] Khan R H, Heegaard P E. Translation from UML to Markov model: A performance modeling framework for managing behavior of multiple collaborative sessions and instances[C]//Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on. IEEE, 2010, 1: 677-686.

[29] Cortellessa V, Di Marco A, Inverardi P. Model-based software performance analysis[M]. Springer, 2011.

[30] Gelenbe E, Pujolle G. Introduction to queueing networks[M]. Chichester: Wiley, 1987.

[31] Plateau B, Atif K. Stochastic automata network of modeling parallel systems[J]. Software Engineering, IEEE Transactions on, 1991, 17(10): 1093-1108.

[32] Bause F. Queueing Petri Nets-A formalism for the combined qualitative and quantitative analysis of systems[C]//Petri Nets and Performance Models, 1993. Proceedings., 5th International Workshop on. IEEE, 1993: 14-23.

[33] Tai S, Totok A, Mikalsen T, et al. Message queuing patterns for middleware-mediated transactions[M]//Software Engineering and Middleware. Springer Berlin Heidelberg, 2003: 174-186.

[34] Hapner M, Burridge R, Sharma R, et al. Java message service[J]. Sun Microsystems Inc., Santa Clara, CA, 2002.

[35] Hoare R R, Ahn J, Graves J. Discrete event simulator: U.S. Patent Application 10/043,847[P]. 2002-1-11.

[36] Wong S. StarUML Tutorial[J]. 2007.

攻读硕士学位期间取得的学术成果

[1] 徐龙, 路红, 张莉. 基于 MARTE 的模型性能仿真分析[A]. 中国计算机学会 2013 年全国软件与应用学术会议(NASAC2013) [C].

附录 1 本文支持的数学表达式

1. 数学操作符

本文支持常见的数学操作符，各操作符及其优先级如下表所示，其中优先级由上至下递减：

表 20 本文支持的数学表达式

操作符
变量赋值(var = value)
取非(!)
乘(*)、除(/)、取模(%)
加(+)、减(-)
大于等于(>=)、小于等于(<=)、小于(<)、大于(>)

2. 概率表达式

表 21 本文支持的概率表达式

概率表达式	输入参数	描述
CONST	常量；	常量表达式，每次都返回输入参数；
UNIFORM	起始值，结束值；	均匀分布，参数为起始值与结束值；
POISSON	泊松分布均值；	泊松分布，输入参数为该分布均值；
NORMAL	均值与标准差；	正态分布，输入参数为均值和标准差；
LOGDIST	对数分布的 scale 与 shape；	对数分布，输入参数为 scale 与 shape；
ENUMDIST	离散值与该离散值的取值概率列表；	离散分布，输入参数为离散值与该离散值的取值概率列表；

附录 2 MePAL 语言元模型

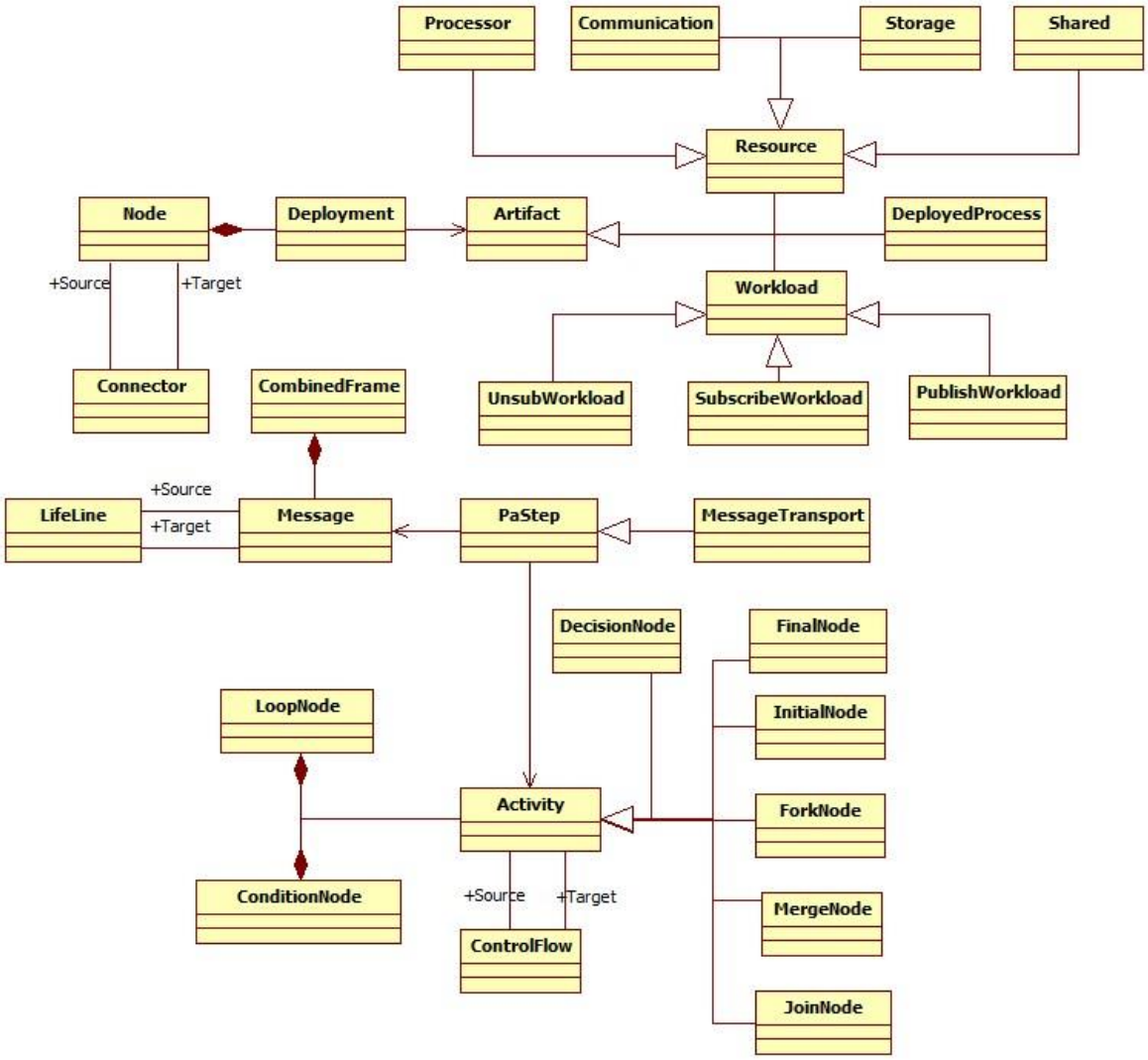

图 47　MePAL 语言元模型

致谢

本文工作是在导师张莉教授的悉心指导和帮助下完成的，她给了我许多有益的帮助，使得本文得以顺利完成。导师严谨的学风、求实的精神和正直的人品不仅使我受益匪浅，而且将永远激励我在专业研究中探索前进。

感谢我们实验室的全体成员，他们给我提供了良好的学习、研究环境。特别感谢路红和连小利两位同学，无论在学习和生活中，他们都给了我很多无私的帮助。感谢樊志强同学和凌济民同学，他们在项目开发和讨论中带给我很多新的想法，非常高兴可以与他们一起共度研究生两年半的珍贵时光。感谢实验室的其他师兄师姐，特别是路红师姐，在我工作陷入僵局时帮助我找到新的思路。

在攻读学位期间，计算机系的诸位老师都给了我很多的帮助，在此一并致谢。同时，感谢所有曾在学习和生活中帮助过我的同学和朋友，你们给我的关心和帮助是我最珍贵的回忆和财富。

最后，谨以此文献给我的父母以及其它关心和爱护我的人，他们无私的爱是我奋斗的动力。