文章编号:1671-4598(2020)03-0119-04 DOI:10.16526/j.cnki.11—4762/tp.2020.03.025 中图分类号:V219 文献标识码:A

基于数据分发服务和 WPF 技术的试飞实时监控系统设计

山 寿,郝明哲,孙 伟
(中国飞行试验研究院 测试所,西安 710089)

摘要:针对以服务为中心的客户端服务器(C/S)架构监控模式存在着数据融合困难、传输协议单一、扩展灵活性差等不足,将数据分发服务(DDS)技术引入到试飞实时监控系统中,结合 WPF 图形引擎技术形成了一整套解决方案;在完全兼容当前架构规范的前提下,采用 DDS 技术作为数据传输中间件,有效提高了试飞监控系统的实时性,加强了多源数据融合力度,降低了试验资源的集成难度;采用 WPF 图形引擎技术构建实时监控可视化编辑平台,最终实现了空天地综合监控环境下多源数据接收、处理、分发以及可视化展示等关键技术;此方案已经在飞行试验地面实时监控中得到了成功应用。

关键词:飞行试验;实时监控;数据分发服务;数据融合;WPF

Design of Real—time Monitoring System for Flight Test Based on DDS and WPF Technology

Shan Shou, Hao Mingzhe, Sun Wei
(Chinese Flight Test Establishment, Xi'an 710089, China)

Abstract: In view of the difficulties of data fusion, single transmission protocol, and poor flexibility of extension in the service—oriented client—server (C/S) architecture monitoring mode, the data distribution service (DDS) technology is introduced into the flight test real—time monitoring system, and a complete set of solutions is formed by combining the WPF graphics engine technology. On the premise of fully compatible with the current architecture specifications, DDS technology is used as the data transmission middleware, which effectively improves the real—time performance of the flight test monitoring system, strengthens the granularity of multi—source data fusion, and reduces the integration difficulty of the test resources; WPF graphics engine technology is used to build the real—time monitoring visual editing platform, and finally realizes the multi—source data reception in the air space integrated monitoring environment , processing, distribution, visual display and other key technologies. This scheme has been successfully applied in the ground real—time monitoring of flight test.

Keywords: flight test; real—time monitoring; data distribution service; data fusion; windows presentation foundation

0 引言

随着信息网络技术的飞速发展,航空武器装备系统越来越复杂,将面临试飞风险高、试飞周期短、试飞数据量大等严峻问题。一方面未来空天飞行器试飞时需要同时对大量测试数据、高清视频数据和控制信息进行双向、持续、高速传输,来实现对其超视距的时空连续覆盖,并通过地面监控实现对故障的及时预警、处置和设备控制,以提高试飞安全性和效率;另一方面有人—无人机协同作战试飞,将会接入更多的试飞数据汇聚到飞行试验实时监控网络[1]。

从发展要求来讲,未来所建设系统需要具备三种能力,确保为三类人员提供高质量的实时数据服务:针对测试人员应该提升其业务管理能力,系统应具备集中信息管理、集中任务管理、数据质量管理、容灾与处置等能力;针对课题人员,以提升数据服务为基准,使其具备自主监控能力,系统具备简易操作逻辑、自主发布/订阅数据、自主评估/分析试验等特性;针对联合试验人员,应具备试验资源接入与分析能力,系统应具备低成本数据互联、高可靠异构网络传输、数据服务质量等特性。

然而目前飞行试验仍然采用以服务为中心的 C/S 架构监控模式,遥测数据的接收、解析处理及转发在服务端进行,客户端被动接收服务器转发的实时数据。因此现有系统不能有效解决分布式试飞资源的融合处理、数据集中治理、链路健康状态管理以及跨试验场数据分发等关键问题,本文结合目前试飞实时监控系统的特点,引入基于 DDS 的数据融合处理分发机制,探索其在未来跨试验场联合试飞、多机协同试飞等领域中的应用,实现多源数据快速接入、数据自主发布—订阅、数据服务质量,提高通信效率;采用 WPF 实现可视化控件编辑、参数—控件双向绑定,使界面开发引擎与 DDS 数据传输相结合,满足实时监控画面的高精度和流畅性,为确保试验安全、缩短试验周期、节约试验经费提供有力支撑。

收稿日期:2019-12-06; **修回日期**:2020-01-13。
基金项目:国防基础科研项目(JCKY2016205B006)。
作者简介:山 寿(1987-),男,陕西西安人,工程师,主要从事飞行试验数据处理与软件技术方向的研究。

1 关于 DDS

1.1 DDS 简介

DDS（data distribution service for real－time systems）作为分布式实时数据分发的主流技术，已经在证券交易、国外靶场资源整合方案中得到了充分验证，DDS 标准为 OMG 组织发布，该规范标准化了分布式实时系统中数据发布、传递和接收的接口和行为，定义了以数据为中心的发布－订阅机制，提供了一个与平台无关的数据模型[2]；而且 DDS 还定义了大量的 QoS 策略，协调可预测性与执行效率之间的平衡，很好地配置和利用系统资源，支持复杂网络环境下的数据传输需求等[3]。

DDS 规范的核心就是标准化了以数据为中心的分布式应用“发布/订阅”通信模型－DCPS 模型[4]，该模型隔离了网络传输层与上层应用层的耦合性，将数据的发送抽象为了数据发布者（Publisher），将数据的接收抽象为了数据订阅者（Subscriber），将传输的数据信息抽象成了主题（topic），将逻辑上隔离的虚拟网络抽象成了域空间（Domain），如图 1 所示。

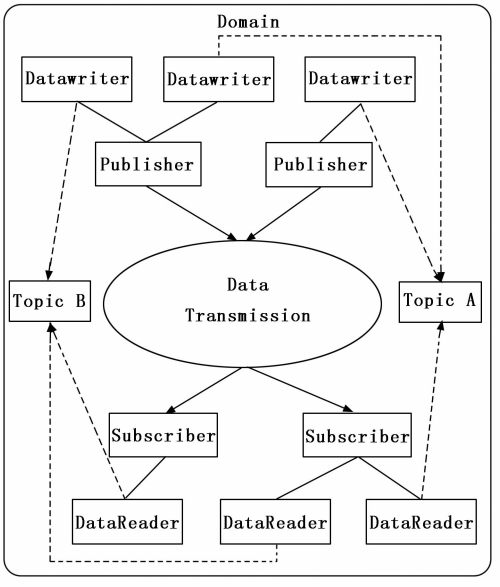

图 1 DDS 发布订阅模型

域空间（Domain）是 DDS 服务依据域 id 建立的逻辑上独立的通信网络，具有相同主题的订阅者和发布者才能建立发布/订阅关系，其中 QoS 服务质量贯穿整个 DDS 通信过程[5]。

1.2 WPF

WPF 是 Windows Presentation Foundation 的缩写，是 MicroSoft 公司新一代图形化显示系统，具有多种优越特性：基于 DirectX，具有更高的画面精度；显示设备的无关性，具有可移植性[6]。同时引入了可扩展应用程序标记语言（eXtensible Application Markup Language，XAML），这是一种全新的标记语言，它是专门用于应用程序 UI 的定制而开发的，可以在声明性 XAML 标记中创建可见的 UI 元素，使 UI 设计与运行时业务逻辑分离[7]。XAML 代码可以使用图形设计工具 Microsoft Expression Blend 创建，方便开发人员直接修改界面，提高了效率。

1.3 服务质量策略（QoS）

服务质量策略 QoS（quality of service policies）是一种网络传输策略，DDS 规范定义多个 Qos 策略，尽可能地满足客户对通信质量的需求[8]。DDS 的数据传输机制采用的传输协议，包含有 TCP、UDP、RTPS _ UDP 等，这些协议的使用采用配置文件的方式灵活配置，结合 OMG 组织定义的 22 种 QOS 策略可以满足系统的各种传输要求，其工作流程如图 2 所示。

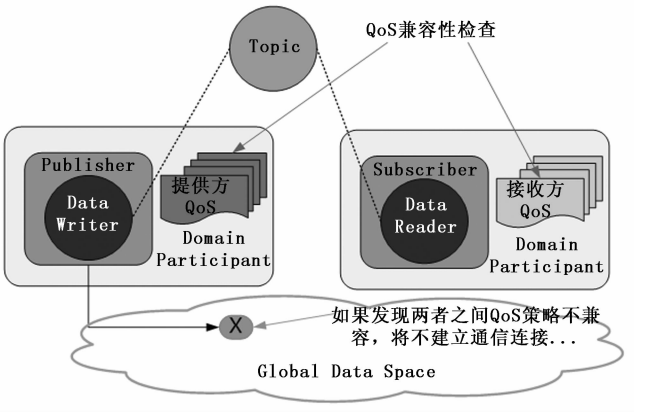

图 2 DDS 发布订阅模型

DDS 支持最大带宽传输模式和可靠传输两种模式。其中，最大带宽传输模式以尽可能的最大传输速率进行数据传输，如果传输的协议采用非可靠的传输协议（如：UDP 协议）时，若发生数据丢包，将不会产生数据重传，该传输模式适用于仅对于最新数据感兴趣的持续输出系统；可靠传输模式是在 DDS 传输层上提供的一种可靠数据传输服务，依赖底层传输协议而提供的一种可靠传输服务。针对复杂网络环境下的试飞任务，可根据当前网络状态和试飞科目需求，通过 QoS 策略可以实现整个网络的控制，增强其对复杂网络环境下的适应性和鲁棒性。

2 基于 DDS 的试飞实时监控平台

基于 DDS 的试飞实时监控平台采用一种可以自发现，分布式的软件构架，多个节点之间通过计算机网络进行消息传递实现复杂网路环境下综合监控多源数据流的接收、解析、处理和分发任务，主要包括以下两部分：

1）DDS 服务中间件，将 DDS 服务中间件部署于各个监控终端，实现整个平台的基础通信环境。

2）数据融合处理分发，实现基础配置、多源数据处理、集中数据处理、数据分发等功能。

基础通信环境可支持 UDP、TCP 等通讯协议，能够结合现有遥测实时监控的具体场景以及相关数据模型，建立试飞数据的主题化配置与管理；多源数据处理具备空天地多源数据的接收、解析、处理以及融合处理能力；集中数据处理能够结合现有数据封装协议与飞行试验数据配置信

息实现数据解析，能够基于 DDS 实现计算业务的封装与分发，支持多个计算节点的业务派发能力，能够实时检测各节点工作状态，可根据各计算节点的计算能力动态调整各节点计算业务，具备数据存储功能，并具有管理和发现能力，支持数据回放，同时实现 22 种 QoS 服务策略；数据分发支持订阅发布模式，并能够兼容现有 C/S 架构通信模式，将数据分发到各个监控软件。

2.1　总体架构设计

平台整体架构采用 OpenDDS 作为数据通信中间件，OpenDDS 是对象管理组织 OMG 的实时数据分发系统的 C++开源实现[9]，系统进行分布式处理，每个分布节点之间使用 DDS 进行数据的传输，采用 ORM 映射技术处理数据库和实体之间的关系，来提供数据服务业务，利用 WPF 技术中的 XAML 引擎来搭建实时监控界面的可视化，使用任务容器引擎，支持多业务的数据解析和处理，整体架构如图 3 所示。

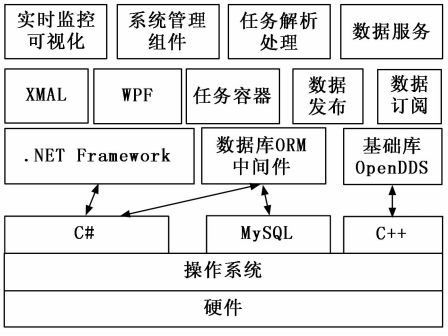

图 3　系统整体架构图

使用 DDS 中间件替代了原有的 Socket 原生编程通信底层，利用 DDS 的协议特点，创建一个更适应于试飞环境下的数据传输机制，系统的分布式结构由若干个可以独立执行的程序模块组成，它们分布于一个分布式处理系统的多台计算机上被同时执行。软件采用 Windows 的 . Net 框架进行开发，使用 XAML 引擎来进行实时编程界面的开发，如控件、数据双向绑定、二维和三维图形、事件路由等，可使开发人员和设计人员用来构建和重用 UI 的工具更加丰富。

2.2　平台运行环境

基于 DDS 和 WPF 技术的试飞实时监控系统运行环境如图 4 所示，试飞过程中指挥员、各专业课题组人员，通过监控终端和监控大屏进行实时安全监控和任务监控。后台各个节点通过 DDS 服务中间件来进行通信。

后台业务处理是系统设计的关键，目的是完成飞行数据的解析，能够解析实时数据和离线数据。主要包括计算服务，实现多源数据流的接收、处理和分发；监控服务，实现解析后数据的可视化展示；管理服务，实现配置信息管理和监控整个网络节点的信息管理；数据库服务，实现实时数据的快速存取和业务信息的数据交互。

2.3　并行计算与多任务管理

在真实飞行试验环境下，通常多架机同时进行数据遥

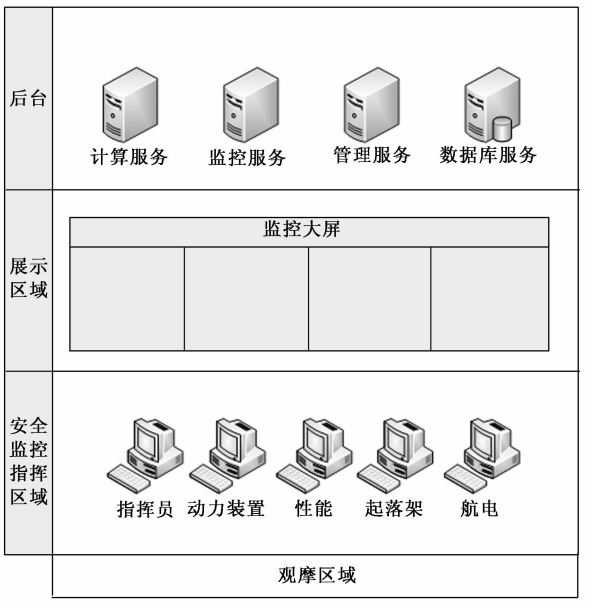

图 4　系统运行环境

测传输与地面安全监控，因此需要采用分布式的多节点同时计算，来进行多任务的并行计算，同时提供管理平台来对多任务实施全流程管控。并行计算与多任务管理功能由分布式处理平台提供，本分布式处理平台由管理端和计算节点端组成，如图 5 所示。

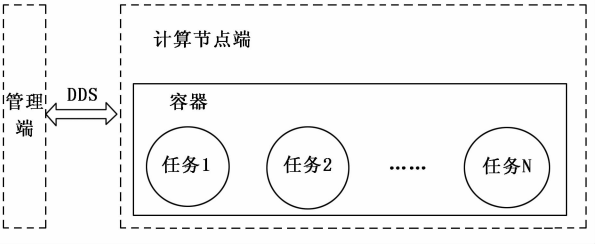

图 5　多任务管理结构图

1）管理端负责收集汇总各计算节点上报的心跳、任务信息、节点机信息等内容，管理端实时显示各节点的任务状态和节点机状态，以便用户动态调整各节点任务状况。管理端和计算端的通信通过固定的 DDS 话题提供整个通信指令和上报链路；

2）容器是一个任务集合容器，负责接收由管理端下达的控制指令和上报计算端信息，用于对任务进行添加删除启动停止等内部管理工作；

3）任务负责执行具体业务逻辑，例如数据的接收、数据的解析和数据的处理等。任务可动态分配到单个或多个 CPU 核心当中以调整不同的系统性能和并发性。

2.4　实时业务处理分发

实时业务处理分发是系统设计的关键，目的是完成飞行数据的解析，能够解析实时数据和离线数据。并且能够通过 DDS 中间件将解析后的数据进行分发，计算服务必须要能够高效的对数据进行解算，并且能够快速的将数据进

行转发。

由试验配置信息综合管理模块提供试验信息解析，根据解析后的参数进行接收、解析、处理、以及融合处理，具体步骤如下。

1）数据接收，完成对多源数据提供参数接收任务，如 PCM 格式数据、IENA 格式数据和 iNET 格式数据，该任务的主要工作是可以使用不同的接收机制，从不同的信号源中接收原始数据；

2）数据的解析，是根据试验配置文件提供的解析信息，对数据进行初次解析，将所有数据参数从原始数据包中提取出来，等待后续任务进行处理；

3）数据的处理，是对由解析任务所提取的参数，根据试验配置文件中提供的校准信息进行相应处理，并将处理结果暂时保存至内存等待后续任务进行使用；

4）数据的融合处理，是由用户提供类似试验配置信息格式或以接口形式提供动态链接库进行融合处理；

5）对于处理后的数据（一次处理或融合处理），由 DDS 进行传输，根据不同的话题定义，发布至不同的话题以供链路传输。

2.5 基于 XAML 的编辑引擎设计

界面编辑引擎旨在为数据融合编程可视化显示提供一个开发、发布平台，编辑引擎基于 .NET WPF 开发，提供界面可视化编辑引擎，通过基于 XAML 的编辑器引擎，用户通过浏览基本控件提供布局控件（Border、Grid、Rectangle、TabControl 等）、功能控件（Button、CheckBox、ComboBox、Image、ListBox、RadioButton、TextBox 等）、以及图表等控件，自定义控件库中的显示控件，以拖拽的方式设计不同试飞科目监控画面，如飞控、动力装置、航电等，同时将可视化控件和参数进行绑定，能够与 DDS 数据传输相结合，来满足数据的接收显示，最后将设计好的 XAML 文件序列化且录入关系数据库。平台支持多源数据融合，数据源端可将接收的 PCM 格式数据、IENA 格式数据、INET 格式数据进行融合显示，用户也可以同时将多架飞机的数据进行融合的综合显示。

运行阶段，首先配置需要实时监控的试飞任务参数，然后生成话题在任务中进行注册，当话题注册完毕后，执行任务时，显控软件会接收到配置的参数。经数据接收、解析、处理后，都由 DDS 进行传输，按照多参数定制话题，由计算服务传递至监控服务。监控服务由两种方式接收数据：DDS 接收模块接收经由 DDS 链路传递的数据至编辑引擎当中，或使用兼容模式下的 UDP 与已有系统进行连接。

1）基于 Canvas 的布局

系统中的控件库分为两大类：布局控件（Layout Control）和可视化子控件（Sub Control）。WPF 的布局控件都在 System. Windows. Controls. Panel 这个基类下面，一般使用 WPF 提供的各种控件在界面进行布局，同时对各种子控件（如按钮、文本框，下拉框等）进行排列组合。但是对于复杂的布局情况，每当布局控件内的子控件改变其位置时，布局系统就可能触发一个新的处理过程，子控件数量会严重影响系统整体性能。本系统引入 Canvas 布局面板，同时为各个子控件添加鼠标事件，实现控件的旋转、平移等动态修改。由于 Canvas 画布只是一个存储控件的容器，不会自动调整内部元素的排列及大小，因此具有良好的性能优势。

定义拖拽方向枚举：

```
public enum DragDirection
{
TopLeft = 1,
TopCenter = 2,
TopRight = 4,
MiddleLeft = 16,
MiddleCenter = 32,
MiddleRight = 64,
BottomLeft = 256,
BottomCenter = 512,
BottomRight = 1024,
}
```

XAML 布局代码：

```
<Canvas Margin="0,0,0,0" Background="White">
<Rectangle Fill="Red"
Stroke="Azure"
Width="64"
Height="64"
Canvas. Left="210" Canvas. Top="20"/>
<Canvas>
```

2）试飞参数双向绑定（Binding）

试飞安全监控要求传感器数据或总线数据变化时，监控画面能够实时更新，传统的定时器操作会丢失瞬态数据，而事件机制又会增加代码的复杂度，本系统采用 Binding 技术实现试飞数据和可视化控件的双向绑定，一但监控的试飞数据发生变化，可视化控件的值会自动更新，实现代码如下所示：

```
Public class Parameters: INotifyPropertyChanged
{
public event PropertyChangedEventHandler ParamPropertyChanged;
private string x;
public string X;
{
get {return x}
set { x = value;
if(this. ParamPropertyChanged ! =null)
{
this. ParamPropertyChanged. Invoke(this,new
PropertyChangedEventArgs("X"));
}}
}
this. graph. SetBinding(Graph. DataSourceProperty,new Binding() {
```

（下转第 143 页）