关于自动化应用中通信协议软件独立性的研究

戴宏斌
(国电南自轨道交通工程有限公司 南京 210032)

摘 要 自动化应用中的各种设备、系统及其各层次、部分之间使用通信协议软件进行通信,从而实现数据交互和信息共享。通信协议软件的可靠性和可扩展性对自动化工程的质量和实施存在显著影响。考查了通信协议软件涉及的数据和操作,并按功能阶段进行了细致的划分。在此基础上,通过引入事务的概念,提出了一些指导性原则,这些原则有助于提高通信协议软件的独立性,使其具有良好的可靠性和可维护性,从而有益于工程质量和工程实施。

关键词 通信协议软件,独立性,事务
中图法分类号 TP3 **文献标识码** A

Research on the Independence of the Communication Protocol Software in Automation

DAI Hong-bin
(Nanjing SAC Rail-transit Engineering Co., Ltd, Nanjing 210032, China)

Abstract Communication protocol software is employed to perform communication among devices, systems and different parts of them to exchange data and share information in automation projects. Its reliability and scalability would significantly affect the quality and implementation of the projects. In this paper the data and operations in the communication protocol software are explored and then divided according to the function stage. By introducing the transaction concept, some principles for the communication protocol software are suggested, which help enhance the independence of the communication protocol software, thus improve its reliability and scalability, and as a result contribute to the quality and implementation of automation projects.

Keywords Communication protocol software, Independence, Transaction

1 引言

自动化应用中各种设备、系统,以及系统的不同层次、部分[1]之间因地域分布、逻辑架构、应用环境、设计理念等诸多差异往往采用了各不相同的数据表示,因此在通信时需要转换成某种约定的、适合通信的一致的表示形式来传输以实现数据交互和信息共享[2]。这种约定的适合通信的数据表示一般称为通信协议[3],如工业测控自动化应用中常用的Modbus协议及其扩展[4],电力监控自动化应用中常用的IEC系列协议[5]等。通信协议软件实现了某种通信协议,在通信各方之间正确、高效地转换并传输数据。

通信协议软件充当着通信各方的通信接口,实现其数据输入和输出,即使自动化系统或者设备的功能再强,可靠性再高,一旦通信协议软件发生故障,传输的数据出现错误,整个体系还是无法正常工作。较之相对稳定的系统和设备,通信协议软件由于涉及多方通信,而在不同的应用和工程中,通信各方又经常变更,因此通信协议软件也总是频繁地修改和更新。对于成熟的自动化工程单位的工程实施而言,软件部分的修改和调试中最为频繁、最为耗费时间和人力资源的部分基本就是通信协议软件了。

通信协议软件的可靠性和可维护性对自动化工程的质量和实施存在着显著的影响,合理设计和架构的通信协议软件具有优良的可靠性和可维护性,能够有效支撑整个体系持续、稳定地正常运行,并减少维护时涉及的程序规模和工作量,节省资源开销,降低维护人员门槛;反之则不仅延误工程实施,而且容易在运行中频出故障,甚或引发事故,既耗费资源,牵扯高端研发人员的精力,也影响工程的整体质量和实施进度,成为自动化工程的阿喀琉斯之踵。

通信协议软件既然作为一种软件,同样也应遵循软件的一般规律。它的设计、实现和维护也应当遵循软件的一个基本原则——独立性原则即软件的各个功能部分应当尽可能相互独立[6],这样可以有效保障通信协议软件的可靠性和可维护性。然而由于通信协议软件的开发和维护人员通常具有更强的自动化而非软件架构领域的背景,因而在通信协议软件的设计、实现中往往自然而然地更关注各种专业逻辑的划分,却习惯性地忽视通信协议软件的独立功能的解构,从而难以确保其可靠性和可维护性,乃至于严重影响工程质量和实施。

因此,本文考查了通信协议软件涉及的数据和操作,并对其进行细致的划分。在此基础上,引入类似于数据库领域的事务概念[7],给出了一些指导性原则,坚持这些原则有助于提高通信协议软件的独立性,使其具有良好的可靠性和可维护性,从而改善工程质量和工程实施。

本文第2节考查了通信协议软件的软件特性,并介绍了软件独立性的相关概念;第3节将关注缺乏独立性对通信协

戴宏斌(1981—),男,硕士,工程师,主要研究方向为自动化软件研发,E-mail:xiaod00@hotmail.com。

议软件的可靠性和可扩展性的不利影响；第 4 节在对通信协议软件涉及的数据和操作进行细致分类的基础上，通过引入事务概念，提出了一些有效保障通信协议软件的独立性的指导性原则；最后进行了总结并指明一些未来的工作。

2 背景

通信协议软件实现了某种通信协议，用于在通信各方之间正确、高效地转换并传输数据。通信协议软件有时作为独立的软件系统存在，如通信管理机、前置处理机系统、协议转换器等；有时作为系统或平台的独立软件构件挂接，通过接口与之进行交互；有时则作为软件的功能子模块嵌入软件之中。但无论哪种形态，通信协议软件都用于提供协议通信的功能。

2.1 通信协议软件的软件特性

通信协议软件作为一种软件，存在以下一系列软件特性。

2.1.1 通信协议软件的可用性

通信协议软件的可用性依赖其正确性。暨通信各方的协议软件在正常的通信环境中是否正确地实现了通信协议规定的数据格式和交互流程。这一般通过点对点、端对端等通信调试流程来保证。

2.1.2 通信协议软件的性能

通信协议软件的性能受通信协议规定的数据格式、通信流程、交互方式的影响，并为通信条件和环境所制约。由于近年来总线、网络等各种通信设备和技术迅猛发展，为协议通信提供了卓越的支撑，足以保障通信过程中的数据传输速率，性能需求已经不再是通信协议软件关注的焦点。

2.1.3 通信协议软件的可靠性

通信协议软件的可靠性依赖其健壮性。与相对独立的自动化系统和设备不同，通信协议程序主要用于通信，由于现场通信环境和通信状况复杂多变，容易出现诸如通信通道失效、不稳定或者被干扰，传输数据乱序、丢失、错误等各种异常状况，同时由于通信一般涉及双方或多方，因此难免遭遇对方无应答，应答数据无效，甚至是未预期的通信数据等各种异常情况。而且各类异常情况经常混杂出现，难以判断区分。通信协议软件需要合理面对各种异常情况，发现并纠正由此导致的错误；如果无法纠正，至少需要放弃并报告错误。如果不能，那么整个体系就无法持续稳定地正确运行。因此，通信协议软件的可靠性对工程质量存在着显著影响。

2.1.4 通信协议软件的可维护性

通信协议软件的可维护性依赖其可扩展性。自动化应用中的各种系统和设备的功能相对稳定，在不同工程实施中，一般只需要依据应用需求的差异而采用不同的配置，仅在必要时做少量的更新。而且其设计、生产、实现、调试基本由同一厂商完成，周期相对较长，受每一次具体工程的影响相对较小。相比之下，由于通信一般涉及双方或多方，在不同的工程实施中，通信对方往往不断变更，而且通信环境、条件和模式也各不相同，因此协议格式、通信方式、数据流程、逻辑架构都会随之变化，通信协议软件就必然要进行各种修改、更新。而且工程实施往往还涉及多个厂商，存在各种时间、地域、环境和交流的限制，通信协议软件的更新和调试也就面临更多的困难和不确定因素。对于成熟的自动化工程单位的工程实施而言，软件部分的修改和调试中最为频繁、最为耗费时间和人力资源的部分基本就是通信协议软件了。因此，通信协议软件的可维护性对自动化工程的实施存在着显著的影响。

通信协议软件，尤其是其可靠性和可维护性，对自动化工程应用的质量和实施影响甚大，合理设计和架构的通信协议软件具有优良的可靠性和可维护性，可以有效支撑整个体系持续、稳定的正常运行，并减少维护时涉及的程序规模和工作量，节省各项资源；反之则不仅容易延误工时，阻碍工程相关部分的实施，而且容易在运行中频出故障，给运营和维护人员造成诸多不便，甚或引发事故。既耗费资源，也影响工程的整体质量和实施进度。因此，通信协议软件在设计、实现和维护时，应当遵循必要的软件工程原则，以有效保障其可靠性和可维护性。

2.2 软件模块的独立性

通信协议软件一般是由若干功能相关的模块组成的软件。作为一种软件，必然遵循软件的一般规律。软件工程的一个普遍原则是，各个模块应当尽可能相互独立，也就是软件模块应当具有较高的独立性，这样的软件具有良好的可靠性和可维护性[6]。

模块独立性是指每个模块只完成软件要求的独立的子功能，并且与其他模块的联系最少且接口简单。软件之所以强调模块的独立性，一方面模块独立的软件固然比较容易开发，这是由于能够分割功能而且接口可以简化。更重要的是独立的模块比较容易纠错和维护，具有良好的健壮性和可扩展性。这是因为在模块独立的软件中，错误传播范围小，修改设计和实现需要的工作量比较小，需要扩充功能时能够方便地“插入”模块。总之，独立性是优秀设计的关键，而设计又是决定软件质量的关键环节。通信协议软件作为一种软件，也同样遵循这一规律。

模块的独立程度可以由两个定性标准度量，这两个标准分别称为内聚和耦合。

内聚标志了一个模块内各个元素彼此结合的紧密程度，它是信息隐蔽和局部化概念的自然扩展。简单地说，理想的内聚的模块只做一件事情。

耦合是对软件结构内各个模块之间互连程度的度量。耦合是影响软件复杂程度的一个重要因素。应该采取的原则是：尽量使用数据耦合，少用控制耦合，限制公共环境耦合的范围，完全不用内容耦合。

坚持遵循独立性原则有助于提高通信协议软件的可靠性和可维护性，从而有力保障工程质量和工程实施。然而遗憾的是，由于通信协议软件的开发和维护人员通常具有更强的自动化而非软件架构领域的背景，因而在通信协议软件的设计、实现中往往自然而然地更关注各种专业逻辑的划分，常常习惯性地忽视通信协议软件的独立功能的结构。

下面，以一个最简单的例子来说明缺乏独立性对通信协议软件的可靠性和可维护性的不利影响。

3 通信协议软件缺乏独立性的问题

通信协议软件在设计和实现时可以从不同的视角来看待和划分。例如，以监控自动化应用中的一个最简单的实现遥信、遥测、遥控数据通信的通信协议软件进行简要示意。它可以划分为遥信数据的传输和处理，遥测数据的传输和处理，遥控数据的传输和处理等功能模块，如图 1 所示；也可以划分为

数据处理、数据通信功能模块，进而将数据通信功能模块分解为监测方向数据通信、控制方向数据通信等子功能模块；将数据处理功能模块分解为遥信数据处理、遥测数据处理、遥控数据处理等子功能模块，如图 2 所示。依据这两种方式实现的通信协议软件经过调试后也许都是可用的，但是其可靠性和可维护性则差异很大。

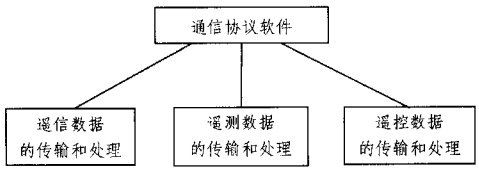

图 1　基于自动化视角的通信协议软件划分示意图

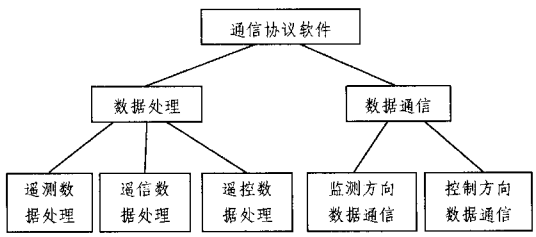

图 2　基于软件视角的通信协议软件划分示意图

遗憾的是，通信协议软件的开发人员面对数据的时候，往往并非首先考虑要分几个阶段来依次完成对数据的各项处理；而是因为专业背景的缘故条件反射地分析数据的专业类型。于是，通信协议软件常在没有经过相对周详的分析和设计的情况下，凭借专业的习惯直觉性地采用了前一种设计和实现方式。

这是一种典型的低独立性的方式，存在严重的控制耦合乃至内容耦合。虽然就自动化专业的视角来看，焦点更加集中在某种专业数据的整个操作流程，对专业人员更为自然；但是，从软件设计的理念出发，这样把数据的传输、封装、转换、处理等很多互不相关的功能阶段拼凑在同一个模块之中，使之相互纠缠，互相影响，表现出软件工程中非常忌讳的低内聚、高耦合特性，这种低独立性会在这样一些方面严重影响通信协议软件的可靠性和可维护性。

低独立性会妨害错误原因和位置的判断和追踪。例如，发现遥信数据错误，但是由于整个遥信数据处理流程都交织在一起，难以判断到底是传输过程、处理过程还是其它操作导致了遥信数据错误。

即使判断出错误的原因，也由于各种操作交相掺杂，原先的环境可能已经变化而难以处理错误。例如判断出是由于通信故障导致的数据传输错误，重新传输数据可以改正错误。但是由于数据处理和数据传输操作交互进行，致使数据状态已经变化而无法重新传输原来的数据来改正错误。

即使错误可以处理，但是由于各种操作互相影响，单个错误处理的影响可能渗透到其它各个部分。例如错误是由于遥信数据处理异常造成的，需要在遥信数据的处理过程中进行相应的修正，但是由于遥信数据的处理和传输操作交杂在一起，相互影响，以至于不得不对传输操作也进行调整，而遥测数据的传输和处理模块以及遥控数据的传输和处理模块也涉及传输操作，于是又不得不进行相应的调整。结果仅仅是一个错误的处理可能就涉及整个软件的所有部分。

当自动化工程变更、应用变化时，通信协议软件大约是修改和更新最为频繁的软件。这种缺乏独立性的设计和实现，往往会使得维护人员不得不总是面对牵一发而动全身的窘境。例如，通信架构从现场总线变更为网络，由于通信协议软件的每一个模块都涉及数据的传输，因此，所有的模块都需要更新。明明只是数据传输方式发生了变更，却不得不为此梳理并修改整个通信协议软件的逻辑流程。

为了保障通信协议软件的可靠性和可维护性，其设计和实现应当具有尽可能高的独立性。为此，需要尽可能实现高内聚、低耦合，也就是将相关的操作尽可能集中一处，而独立的操作尽可能分开。需要特别注意的是，这里所谓相关和独立的着眼点是软件功能而非专业逻辑。也就是说应当使功能一致的操作紧密相连，而非同一种专业数据的整个流程尽在一处。为此，接下来对通信协议软件涉及的数据和操作进行考查和分类。并在此基础上提出一些有利于提高通信协议软件独立性的原则。

4　保障通信协议软件的独立性

自动化应用中的通信协议软件的目标是实现数据通信，一般会涉及以下种类的数据。

本地数据 D_{Loc} (Locale Data)：用于本地逻辑的数据，例如用于本地系统的访问和更新，用于通过 HMI 向用户展示并进行交互等等。这些数据与具体的自动化应用相关，为本地的逻辑服务，一般采用方便本地逻辑的数据表示方式。

可传输数据 D_{Tran} (Transmittable Data)：本地数据为了便于本地逻辑使用，一般包含的信息容量有限，而且参与通信的各方由于应用逻辑差异，各自本地数据的表示可能相差甚远。因此，出于交互和性能的需要，一般先要将本地数据转换为便于传输的数据，这些数据往往以另一种表示方式表征本地数据的信息内容。

通信数据 D_{Com} (Communication Data)：本地数据转换为可传输数据之后一般不会直接传输。这是因为除了可传输数据，为了能够对通信流程进行控制，并保障传输数据的正确，通信各方还需要额外传输如地址、校验码等用于辅助通信的数据，因此需要把可传输数据和这些数据按照协议格式封装成通信报文进行传输，这才是真正传输的通信数据。

通信状态数据 DCS (Communication Status Data)，在通信的各方，一般会存在一些用于表示当前的通信状态和特征的数据，这些数据与本地逻辑无关，也不会被传输，主要用于表征和记录当前的通信状态，以控制通信流程，使之满足应用的需求。

自动化应用中的通信协议软件的目标是实现数据通信。因此主要需要实现这样一些功能：本地数据的逻辑处理，本地数据和可传输数据的转换，可传输数据封装为通信数据以及相应的解封装，通信数据传输等等。为了实现这些功能，往往会涉及以下类型的操作。

本地逻辑操作 Op_{Log} (Logical Operation)，主要是对本地数据进行操作，实现各种本地的逻辑功能。例如访问本地系统，或者通过 HMI 展现数据状态，或者通过交互接受控制指令等等。这与具体的自动化应用需求相关，不同的通信协议软件的本地逻辑操作千差万别。

数据转换操作 Op_{Tran}（Transformation Operation），数据的本地逻辑表示与用于通信传输的表示一般是不同的，需要进行转换。数据转换操作用于将本地数据转换为包含相同信息的可传输数据以用于传输，或者将通信得到的可传输数据转换为包含相同信息的本地数据用于本地逻辑操作。

数据封装/解封装操作 $Op_{Enc/Dec}$（Encapsulation/Decapsulation Operation），为了能够对通信流程进行控制，并且保障传输数据的正确，不仅需要传送可传输数据，还需要同时传送一些状态和控制数据。数据封装操作把可传输数据和这些数据按照协议格式封装成通信报文以便传输；而数据解封装操作则将获得的通信数据依据协议格式进行解析，从中提取出通信状态和控制数据以及可传输数据。

通信操作 Op_{Com}（Communication Operation），通信协议的最主要目标就是实现数据通信，通信操作依照一定的通信逻辑完成各个通信动作，以实现通信需求。在通信过程中，通信操作除了需要发送和接收通信数据，还需要适时对各种本地的通信状态数据进行相应的更新并据此实现正确的通信流程。

通信协议软件涉及的各类数据，及其在对应的各种操作下的转换关系如图 3 所示。

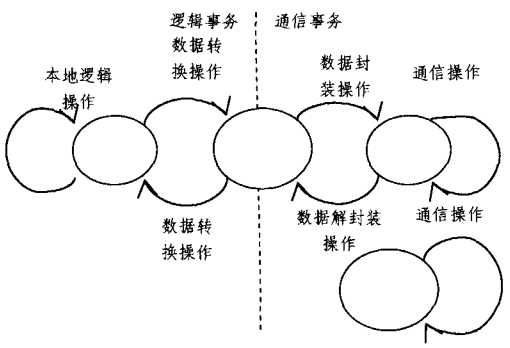

图 3　数据及相关操作示意图

前文已经对通信协议软件经常涉及的数据和操作进行了分类，如果能够依据适合的原则，把相关性较小的数据类型以及操作种类加以区分并隔离，将有助于提高通信协议软件的独立性。为了便于规范化的描述和表达，这里引入数据库领域的一个相关概念——事务。事务（Transaction）是定义的一个操作序列，这些操作要么全做，要么全不做。在数据库领域，事务须具备 ACID 特性。这里并不深究，只是借助事务作为独立的操作序列这样一个概念以助于有效地描述和理解。

在高独立性的软件中，相关的操作尽可能集中，而独立的操作尽可能分离，以实现高内聚、低耦合。需要注意的是，所谓相关和独立是指软件功能而非专业逻辑。通信协议软件的功能需求主要集中于数据通信和与自动化应用相关的本地逻辑处理这两个方面。因此，定义两种事务：通信事务和逻辑事务。为了将相关性较低的数据种类以及操作类型分离，规定：

1）通信事务 $Tran_{Com}$（Communication Transaction）是仅由通信操作和数据封装/解封装操作组成的操作序列。

$$tran_{Com}=\{op_1,op_2,\cdots,op_n\} \tag{1}$$

其中，$op_i\in(Op_{Enc/Dec}\cup Op_{Com})$，$i=1,2,\cdots,n$。

例如一个典型的通信事务可能形如：

$$tran_{Com1}=\begin{cases}op_{Enc/Dec1}：封装通讯请求\\op_{Com1}：发送通讯请求\\op_{Com2}：获得通讯应答\\op_{Com3}：更新通讯状态数据\\op_{Enc/Dec2}：解封装通讯应答\end{cases}$$

2）逻辑事务 $Tran_{Log}$（Logical Transaction）是仅由本地逻辑操作和数据转换操作组成的序列。

$$tran_{Log}=\{op_1,op_2,\cdots,op_m\} \tag{2}$$

其中，$op_j\in(Op_{Tran}\cup Op_{Log})$，$j=1,2,\cdots,m$。

例如一个典型的逻辑事务可能形如：

$$Tran_{Log1}=\begin{cases}op_{Tran1}：解析通讯获得可传输数据\\op_{Log1}：将数据展现给用户\\op_{Log2}：接受用户的控制请求\\Op_{Tran2}：将控制请求转换为可传输数据\end{cases}$$

以 $Op_{ComTran}$ 表示所有通信事务中的操作的集合，以 $Op_{LogTran}$ 表示所有逻辑事务中的操作的集合。那么：

$$Op_{ComTran}\subseteq(Op_{Enc/Dec}\cup Op_{Com}) \tag{3}$$

$$Op_{LogTran}\subseteq(Op_{Tran}\cup Op_{Log}) \tag{4}$$

很显然：

$$Op_{ComTran}\cap Op_{LogTran}=\varnothing \tag{5}$$

通过定义通信事务和逻辑事务这两种不同类型的事务，并规定各自允许包含的操作类型，可以有效地将不同类型的操作隔离，使得本地逻辑操作、数据转换操作与数据封装\解封装操作、通信操作各自集中于相应的功能部分，相互独立，不会互相直接干扰。

分析各种事务包含的操作类型可知，通信事务和逻辑事务涉及的数据类型如图 4 所示。

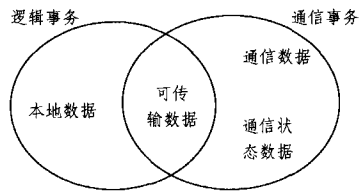

图 4　不同事务涉及的数据类型示意图

可以看出，通信事务仅涉及可传输数据、通信数据和通信状态数据，不可以操作本地数据。相对地，逻辑事务仅涉及可传输数据和本地数据，不可以操作通信数据以及通信状态数据。这样就将本地数据同通信数据及通信状态数据进行了隔离。当然，两种事务都不免涉及可传输数据，但是如果更仔细一点地考查两种类型的事务中与可传输数据相关的操作，就会发现，可传输数据作为通信事务中数据封装操作的输入和数据解封装操作的输出，同时也作为逻辑事务中数据转换操作的输入和输出，实际上是成为了通信事务和逻辑事务的数据接口，这一点在图 3 中表现得尤为明显。由此可知，通信事务和逻辑事务之间的耦合方式是以可传输数据作为接口的数据耦合，是典型的低耦合方式，具有高独立性。

为了进一步降低两类事务在可传输数据上的耦合程度，这里规定：数据封装和解封装操作不允许修改可传输数据的内容。这就将可传输数据的传送同其产生和使用隔离，其传送由通信事务实现，而产生以及使用则交由逻辑事务，不相混淆，从而进一步降低通信事务和逻辑事务的相关性，强化各自

的独立性。

在通信协议软件的设计和实现中坚持遵循上述的这些规定和原则，例如在功能模块划分时，低层模块尽可能只包含一个事务，其上的模块尽可能包含同类型的事务，可以有效地帮助通信协议软件实现高内聚、低耦合，从而具有较高的独立性，有助于保障通信协议软件的可靠性和可维护性。这是因为，划分通信事务和逻辑事务可以将各自处理的数据以及涉及的操作尽可能地隔离开来，从而也将相应的错误尽可能地隔离在各自的功能区域中，这有利于错误的发现、追踪、处理和解决。

例如如果错误发生在逻辑事务中，在涉及的可传输数据正确的情况下，就不会是由于通信操作导致，而应当对数据转换和本地逻辑进行梳理。反之，如果错误发生在通信事务中，那么在可传输数据正确的情况下，就与本地逻辑处理无关，而需要考虑由于现场通信环境故障导致通信过程异常的可能性。

由于不相关的数据以及操作被尽可能隔离开来，相互影响就被尽可能降低。因此，当错误被定位后，不会因为不相关操作的相互影响而难以处理。例如，逻辑事务并不涉及通信，所以如果逻辑事务中发生了错误，就不会由于正在或者已经进行了通信而无法处理错误。反之，在一个通信事务中，本地数据不发生变化，因此，当发生了通信错误时，也不会由于本地数据已经发生了变化致使无法通过重新传输原来的数据来修正错误。

由于不同事务中的错误被隔离在各自的功能区域中，因此，错误的处理也将同样处于相同的功能区域中。例如，本地数据逻辑错误的处理将处于逻辑事务中，不会直接对通信过程造成不利影响；而通信流程错误的处理也将限制于通信事务中，不至于使正确的本地数据处理逻辑失效。

当工程变更、应用变化时，通信协议软件往往也因为需求变更而必须频繁维护。划分通信事务和逻辑事务可以将各自处理的数据以及涉及的操作尽可能隔离开来，从而将变更也限制在尽可能小的范围内。例如，如果通信环境由现场总线变更为网络，那么主要是通信事务中的通信操作需要采用新的实现；而如果是通信协议版本更新，那么数据封装\解封装操作才是软件更新需要关注的焦点。在这两种情况下，只要作为数据耦合接口的可传输数据不发生变化，逻辑事务就不受影响。反之，如果发生了类似自动化系统升级这样的本地逻辑需求的变化，此时逻辑事务就需要随之修改以满足需求，同样地，只要作为数据耦合接口的可传输数据不发生变化，通信事务就无需更新。变更被有效限制在相关的范围内，通信协议软件的维护中不至于出现任何变动都有可能导致修改整个软件这种牵一发而动全身的恐怖局面。

可见，在通信协议软件的设计和实现中坚持遵循这些原则可以帮助有效隔离不相关的数据以及操作，从而将错误处理以及功能更新限制在尽可能小的功能范围中，不会因为不相关数据或操作的相互影响而扩散。这样做的核心目标是尽可能提高通信协议软件的独立性，以保障通信协议软件的可靠性和可维护性。因此，在应用允许的情况下，可以加强限制或者增加规则以进一步提高通信协议软件的独立性。

例如如果通信协议允许，可以规定，可传输数据以一段数据流的形式表示，而数据封装\解封装操作除了复制可传输数据以外，不可以对其进行其它任何方式的访问。从而使得可传输数据在通信事务中成为彻头彻尾的黑盒，进一步实现数据以及操作的隔离。

同样，如果应用允许，可以规定逻辑事务中只允许在开始和结束阶段存在数据转换操作；逻辑事务中的本地逻辑操作和数据转换操作不允许交织进行。这一方面将数据的转换和其处理过程隔离在逻辑事务的不同阶段，进一步隔离可传输数据与本地数据以及各自相应的处理。另一方面逻辑事务被强制为输入→处理→输出，这种典型的具有高内聚特性的单入单出模式，有益于进一步提高通信协议软件的独立性。

结束语 通信协议软件的可靠性和可扩展性对自动化工程的质量和实施存在显著影响。本文考查了通信协议软件涉及的数据和操作，并进行了细致的划分。在此基础上，通过引入类似于数据库领域的事务概念，提出了一些指导性原则，坚持这些原则有助于提高通信协议软件的独立性，使其具有良好的可靠性和可维护性，从而有益于工程质量和工程实施。

不过，由于通信协议软件的开发和维护人员大多具有更强的自动化而非软件架构的专业背景，容易习惯性的从专业逻辑的视角来看待通信协议软件，从而在通信协议软件的设计、实现和维护中会自然而然地忽视软件设计的原理和规则，这就需要转变观念和视角，从不习惯慢慢过渡到自然而然地遵循这些原则，这是需要时间和代价的。但是这样做有助于保障通信协议软件的可靠性和可维护性，有益于工程质量和工程实施，因此是需要的，也是值得的。

文中提出的是一些一般性的原则，对于各个自动化领域的各类型通信协议软件，都具有参考和指导意义，有助于改善通信协议软件的独立性，使其具有良好的可靠性和可维护性。未来的工作包括针对具体的自动化应用如监控应用，和特定的协议类型，如 IEC 系列协议、DNP 协议、Modbus 协议及其扩展[4,5]等等，如何基于协议本身的特点和应用数据的特征，依据文中提出的原则，定制适合应用的通信协议软件框架，合理设计各类数据表示，清晰区分各种操作以及功能阶段，从而将这些独立性原则具体贯彻到通信协议软件的设计、实现和维护中。

参 考 文 献

[1] Gupta R A, Chow M Y. Networked control system: Overview and research trends[J]. IEEE Transactions on Industrial Electronics, 2010, 57: 2527-2535

[2] Sauter T, Treytl A. Communication systems as an integral part of distributed automation systems[M]. Distributed Manufacturing, London: Springer, 2010

[3] Clarke G, Reynders D, Wright E. Practical modern SCADA protocols[M]. Europe: Elsevier, 2004

[4] 卢文俊，冷杉，杨建军，等. 基于 Modbus 协议的控制器远程监控系统[J]. 电力自动化设备，2003，23(6)：54-56

[5] 蔡运清. IEC 870-5 系列及 DNP 3.0 协议简介[J]. 电力系统自动化，1998，22(1)：49-51

[6] Kemerer C F. Software complexity and software maintenance: A survey of empirical research[J]. Annals of Software Engineering, 1995, 1(1): 1-22

[7] Gray J. The transaction concept: Virtues and limitations: Proceedings of the Very Large Database Conference[C]// Cannes, September 1981