# 病理设备中嵌入式系统异常处理①

于松涛

(徕卡显微系统(上海)有限公司 病理系统研发部, 上海 201206)

**摘 要**: 针对病理设备中嵌入式系统中的软硬件异常. 利用状态机原理提出了一种新的异常处理算法. 该算法首先根据不同类型的异常. 得到相应的异常代码; 然后根据不同的异常. 利用 XML 形式定义的异常处理表及异常处理状态机. 对该异常进行处理. 首先介绍了本算法提出的背景. 然后描述了它的具体设计及实现. 并通过一个运行示例对本算法进行了验证分析. 最后. 简要讨论了如何对本算法进行扩展.

**关键词**: 异常处理; 状态机; XML; 封装

## Exceptional Handling for Embedded System in Pathology Equipment

YU Song-Tao

(Biosystems R&D Department, Leica Microsystems LTD, Shanghai 201206, China)

**Abstract**: Based on the principle of finite state machine, the article presents one new exceptional handling algorithm aiming at software and hardware exceptional handling in embedded system running in pathology equipment. The algorithm retrieves exceptional code based on the type of exception at first, and then handles it with XML formatted exceptional handling table and the specific state machines. The article firstly introduces background on the algorithm and then describes the detailed design and implementation. After that, one example is also shown to verify the algorithm. Finally, it also discusses how to improve this algorithm briefly.

**Key words**: exceptional handling; FSM; XML; encapsulation

## 1 算法设计的背景

由于嵌入式软件系统需要与各种异构的硬件进行交互. 因此软件开发者通常需要针对不同的硬件、不同的场景以及软件具体的架构等. 花费大量的时间与精力对异常处理相关的功能进行设计与开发.

本文针对现有的嵌入式软件系统开发设计中. 有关异常处理遇到的问题. 提出了一种新的算法. 该算法将异常处理做为一个独立的模块进行集中处理. 从而降低了异常处理与正常软件流程之间的耦合性. 另外. 本算法提出了异常处理状态机的概念. 将不同的软硬件异常与导常处理进行了分类和抽象. 当异常发生时. 软件系统会首先将该信息通知给异常处理模块. 然后异常处理模块便会根据异常的类型对该异常进行处理.

本文将结合应用于病理脱水机设备中的嵌入式软件系统对本算法进行论述. 在脱水机中. 嵌入式软件系统需要与温度传感器、压力传感器、直流电路系统、交流电路系统和旋转阀机械模块进行交互. 软件与硬件之间通过控制器局域网总线(CAN. Controller Area Network)协议进行交互.

## 2 算法的设计

本部分首先给出了整个脱水机的软硬件系统架构图. 并对关键模块的功能进行了简要的描述. 然后重点介绍了如何利用 XML 与异常处理状态机等概念对本算法进行设计与实现.

### 2.1 系统架构及状态图

在病理脱水机系统中. 各种异构的硬件传感器利用 CAN 协议. 与运行于嵌入式 LINUX 中的软件进行交互. 其系统结构图如下图 1 所示.

① 收稿时间:2016-03-02;收到修改稿时间:2016-03-28 [doi:10.15888/j.cnki.csa.005413]


图 1 系统结构图

在上图中的 Slave 层中. 每一个物理的传感器在 Master 中的设备控制层中(Device Controller Layer)中. 都有一个对应的控制类的对象. 而在 Master 层中. Scheduler 类是整个 Master 层的主控类. 它继承自 QT 中的 QStateMachine 类. 从宏观上来说. Scheduler 类包括初始化、空闲、运行及异常等四个大的状态. 而每个上述每种状态又包含一到多个子状态. 其宏观状态图如图 2 所示.

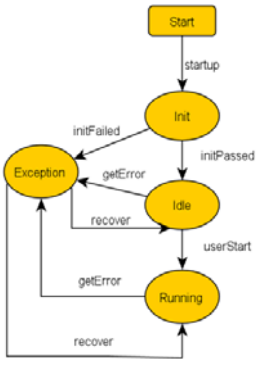


图 2 Scheduler 类的状态转换图

当下层的传感器发生异常时. Slave 会首先通过 CAN 接口将其传递到 CAN Communicator. 并最终到达设备控制层中相应的 Controller 对象. 这个时候. 该 Controller 对象便会将该异常重新解释. 得到异常代码. 并最终传递到 Scheduler 类中. 正如图 2 所示. 这个时候 Scheduler 便会立刻将自身的状态从正常状态(初始化、空闲、运行)切换到异常状态. 当该异常经过异常状态处理后. 如果可以恢复. 便会回到发生异常前的状态; 如果异常处理失败. 系统会停留在异常状态. 并启动报警器向用户报警.

### 2.2 异常处理算法的设计

为了有效的描述异常处理算法. 我们首先提出了相关的术语. 在此基础上. 介绍了利用 XML 技术设计的映射表和异常处理策略表. 然后针对不同异常. 介绍了如何设计对应的异常处理类. 最后描述了异常处理算法的流程.

#### 2.2.1 相关术语

作为整个异常处理算法的基础. 我们定义了如下术语:

Event: 任何一个下位机主动抛出来的异常或者是上位机监控到的硬件的问题. 比如. 我们设计的温度传感器的最高温是 150 度. 而在某些情况下. 上位机监测到其温度已经达到 180 度. 这时候. 我们就会触发一个相应的 Event. 在算法实现中. 我们根据不同的硬件类型、传感器类型以及子系统等. 总共定义了 160 种 Event.

Scenario: 系统现在运行的状态. Scenario 的值是在我们主状态机的基础上. 加上所对应的相关试剂计算出来的. 比如. 当系统现在处于抽试剂的状态时. 如果机器正在处理的试剂类型为水. 我们定义其 Scenario 值为 212. 而当正在处理的试剂为石蜡时. 我们定义其 Scenario 为 272. 在算法实现中. 我们总共定义了 64 种 Scenario.

Error: 根据 Scenario 的不同. 同一个 Event 可以映射出一个或多个 Error. 异常处理算法根据不同的 Error. 执行不同的异常处理策略. 比如. 针对 Event 电流超限. 当系统现在处于 Scenario 004(系统空闲). 我们得到的 Error 为 511003. 在种情况下. 系统便会根据 511003 所对应的异常处理策略. 首先向用户发出一个提示报警框. 然后先关掉相关的传感器、等待 10 钞后重启. 并重新检测. 而如果系统处于 214( 浸泡试剂). 系统会首先释放压力. 并同时把相关传感器所在板上所有传感器的电流关闭. 在等 20 秒后. 会重启该板卡上所有的传感器. 在算法实现中. 我们总共定义了 121 种异常处理策略(Error).

#### 2.2.2Event-Scenario 映射表及 Error 配置表

Event-Scenario 映射表是用来得到相应 Error 的 XML 文件. 该文件在软件系统初始化时被解释. 并存放一个全局的哈希表中. 一条典型的映射记录如下图 3 所示.

```
<event eventid="500050501">
    <error errorid="513050501">
        <scenario type="single" id="200"/>
        <scenario type="range" startid="211" endid="217"/>
        <scenario type="range" startid="221" endid="227"/>
        <scenario type="range" startid="231" endid="237"/>
        <scenario type="range" startid="241" endid="247"/>
        <scenario type="range" startid="251" endid="257"/>
        <scenario type="single" id="260"/>
        <scenario type="range" startid="271" endid="277"/>
    </error>
    <error errorid="513050502">
        <scenario type="single" id="002"/>
        <scenario type="single" id="003"/>
        <scenario type="single" id="004"/>
        <scenario type="single" id="203"/>
        <scenario type="range" startid="281" endid="297"/>
    </error>
</event>
```

图 3 Event-Scenario 映射表中的记录

正如前文所述. 针对不同的 Scenario. 同一个 Event. 可以映射成一个或多个 Error. 而一个 Error 一定属于一个特定的 Event. 在映射记录的定义中. Scenario 的范围. 有两种: single 和 range. 类型 single 表示. 这条 scenario 的范围. 只有一个 scenario. 例如下面这条记录:

<scenario type="single" id="200"/>

而类型 range. 则表示在 startid 与 endid 之间(包括 startid 与 endid 本身)所有的 Scenario 都包含在该范围之间. 例如下面这条记录:

<scenario type="range" startid="211" endid="217"/>

与 Event-Scenario 映射表相似. Error 策略表也是以 XML 形式定义的. Error 策略表是用来告诉软件如何进行异常处理的. 一条典型的 Error 配置记录如下所示:

```
<Event ErrorID="513040161" ErrorType="ERROR" AlarmType="AL_REMOTE" LogLevel="High" >
    <Step ID="0" Type="ACT" Action="RS_Standby_withTissue" NextStepOnFail="5" NextStepOnSuccess="1"/>
    <Step ID="1" Type="ACT" Action="RC_ForceDraining" NextStepOnFail="5" NextStepOnSuccess="2"/>
    <Step ID="2" Type="MSG" StringID="67176212" ButtonType="RETRY" NextStepOnClickRetry="3" NextStepOnTimeOut="6" TimeOut="15m"/>
    <Step ID="3" Type="ACT" Action="RC_Filling" NextStepOnFail="5" NextStepOnSuccess="4"/>
    <Step ID="4" Type="ACT" Action="RC_Restart"/>
    <Step ID="5" Type="MSG" StringID="67176204" ButtonType="OK" NextStepOnTimeOut="6" TimeOut="15m" StatusBar="YES"/>
    <Step ID="6" Type="ACT" Action="RS_Tissue_Protect"/>
</Event>
```

图 4 Error 策略表中的记录

在 Error 记录中. 我们首先用“ErrorType”定义了 Error 的类型. 比如 ERROR, WARNING, INFO 等. 然后. 我们用“AlarmType”指定了当 Error 发生时. 调用警报器的级别. 比如 AL_DEVICE. AL_LOCAL. AL_REMOTE 等.

一条 Error 由一条或多条“step”组成. 依据“step”的顺序(从 0 开始). 软件依次对该异常进行处理. “step”共分为两类. 一类. 我们称之为 ACT. 这类对应于一个特定的异常处理类; 另一类. 我们称之为 MSG. 这类用于弹出对话框. 并接收用户的响应. 出于系统稳定性与安全性的考虑. 当用户长时间没有响应时. 我们也设计了相关的超时响应机制. 在这种情况下. 一旦超时. 系统会根据属性“NextStepOnTimeOut”自动进入到相应的下一个“step”.

2.2.3 异常处理类的设计

针对传感器类型(例如温度传感器. 压力传感器或定位传感器等)、异常类型(温度超限. 压力异常或电流异常等)以及异常发生的 Scenario 等因素. 所抽象出来的的异常处理方法. 每一个异常处理类自身也是一个独立运行的状态机. 他的生命周期对应于 Error 策略表中的 ACT 类型的“step”. 当系统运行到该“step”时. 该异常处理类的对象被创建; 而当该 “step”结束时. 该对象便被销毁. 在具体实现上. 所有的异常处理类都继承自 QT 开发平台中的 QStateMachine 类.

不同的 Error. 我们可以对应一个或多个异常处理类. 而同一个异常处理类也可以用于多个 Error. 异常处理类大体上可以分为两类. 一类. 我们称之为 Response 类型. 在 Error 策略表中以 RS_开头. 该种异常处理类常用于异常处理的第一个响应. 另一类. 我们称之为 Recovery 类型. 在 Error 策略表中以 RC_开头. 该种异常处理类一般用于异常处理结束前. 我们将系统恢复至正常状态时. 在本病理脱水机嵌入式软件系统中中. 我们总共定义了 24 种异常处理类.

根据系统设计的需要. 我们可以定义包含不同状态的异常处理类. 例如针对电流超限定义的异常处理类. 我们定义了初始化、关闭传感器、停止给容器加热、等待 30 秒、重新加热、检测电流范围等. 在上述任何步骤(状态)中. 如果本步骤执行成功. 本状态就会迁移到下一状态; 如果出错. 就会从该状态机中退出. 并中止本状态机的生命期.

另外. 我们还定义了一个特殊的异常处理类. 称之为 RC_Restart. 当异常处理算法将所有的异常都成功排除后. 系统便会调用 RC_Restart. 将系统从异常状态恢复到异常发生之前的状态. 顾名思义. RC_Restart 主要根据发生异常时的 Scenario、异常类型(Event)等因素. 做不同的逻辑处理. 当 RC_Restart 成功后. 系统便会自动切换到正常状态.

2.2.4 异常处理的流程

整个异常处理模块由两个线程构成. 分别称为 Event Handler 和 Exceptional Handler. 当整个软件初始

化时. 便会首先把上述两个线程创建. 并把 Event-Scenario 映射表和 Error 策略表中的 XML 信息解释成为两个存储在内存中的哈希表. 当异常发生后. 异常处理模块会首先将相应的 Event 和 Scenario 传给 Event Handler 线程. 然后 Event Handler 线程便会查找到相应的 Error. 并最终找到相应的 Error 策略记录. 此时. Event Handler 会将该 Error 配置的第一个“step”(step 0)的信息发送到 Exception Handler 线程.

默认情况下. Exception Handler 线程首先会进入等待状态. 并一直监控是否有 Error 到达. 一旦收到 Event Handler 的 Error 中的一条“step”. 便会根据该“step”创建相应的异常处理类对象. 并启动相应的状态机. 当处理完成(无论是成功还是失败). Exception Handler 会将该“step”处理的状态发送回 Event Handler. 而 Event Handler 会根据收到的返回值. 再次调用下一个“step”. 并发送给 Exception Handler 线程. 而在同时 Event Handler 又会进入等待状态.

当所有的“step”都处理完成. 系统便会根据处理结果决定现在系统的状态: 如果处理成功. 整个软硬件系统便会进入正常状态. 反之. 便会仍处于异常状态. 并弹出对话框通知用户做进一步的处理. 整个异常处理的流程如下图 5 所示.

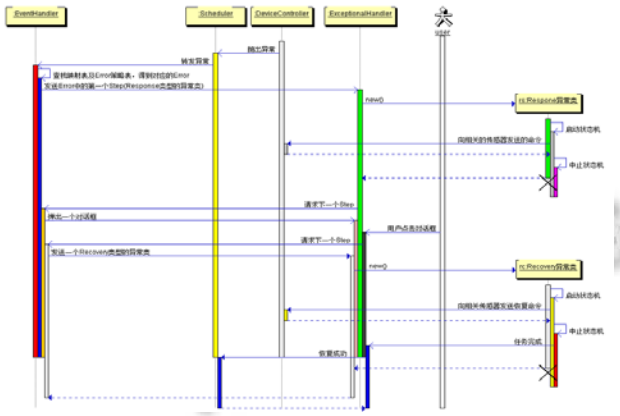

图 5 异常处理时序图

在具体实现中. 我们利用了 QT 平台中的 QThread 类进行线程相关的创建. 同步及销毁等.

## 3 运行实例及分析

### 3.1 运行实例简要描述

在本实验中. 我们首先制造一个液位传感器(Level Sensor. 详见图 1)温度超限. 从而验证异常处理算法是否按我们所设计的流程对系统进行异常处理. 在本实例中. 我们触发的 Event 为 500010321(液位传感器温度超限),Scenario 为 200(脱水程序中的自检). 在 Event-Scenario 映射表中. 所对应的 Error 如下图 6 所示.

```
<event errorid="500010321">
    <error errorid="513010320">
        <scenario type="single" id="200"/>
        <scenario type="range" startid="211" endid="217"/>
        <scenario type="range" startid="221" endid="227"/>
    </error>
    <error errorid="513010332">
        <scenario type="range" startid="281" endid="287"/>
        <scenario type="range" startid="291" endid="297"/>
    </error>
</event>
```

图 6 Event-Scenario 表中的记录

在 Error 策略表中. 本 Error(513010320)的描述如下图所示.

```
<Event Code="1" ErrorID="513010320" ErrorType="ERROR" AlarmType="AL_REMOTE">
  <Step ID="0" Type="ACT" Action="RS_Standby_withTissue" NextStepOnFail="1" NextStepOnSuccess="2"/>
  <Step ID="1" Type="MSG" StringID="67176004" ButtonType="OK" StatusBar="TBS"/>
  <Step ID="2" Type="MSG" StringID="67176020" ButtonType="RETRY" NextStepOnClickRetry="3"/>
  <Step ID="3" Type="ACT" Action="RC_Levelsensor_heating_Overtime" NextStepOnFail="5" NextStepOnSuccess="4"/>
  <Step ID="4" Type="ACT" Action="RC_Restart"/>
  <Step ID="5" Type="MSG" StringID="67176004" ButtonType="OK"StatusBar="TBS"/>
</Event>
```

图 7 Error 策略表中的记录

### 3.2 运行实例验证

如下图 8 所示. 首先我将液位传感器外接于变阻器中. 并将变阻器的电阻调大. 利用阻抗的感理. 令该传感器认为温度已经达到了超限值(摄氏 180 度).

图 8 传感器外接电阻

这个时候,在日志文件中. 会输出如下信息:

2016-03-14 15:40:18.204;Instrument has switched to exceptional state;

2016-03-14 15:40:18.212;Scheduler enters the exceptional handling state.

2016-03-14 15:40:45.967;Do response action RS_Standby_withTissue.

2016-03-14 15:40:46.296;Debug message: Get action: RS_Standby_withTissue

2016-03-14 15:40:46.820;Debug message: RS_Standby_WithTissue, in state SHUTDOWN_FAILD_HEATER
2016-03-14 15:40:47.333;Debug message: RS_Standby_WithTissue, in state RTBOTTOM_STOP_TEMPCTRL
2016-03-14 15:40:48.333;Debug message: RS_Standby_WithTissue, in state CHECK_TEMPMODULE_CURRENT
2016-03-14 15:40:51.820;The instrument encountered the error 513010320 during scenario RS_Standby_withTissue; and tried to do the recovery action Resolved.
2016-03-14 15:40:51.821;513010320;Error;Please clean the level sensor.

这表明. 软件系统已经进入到了对 Error 513010320 的异常处理过程中. 软件系统首先进入异常处理状态. 它会首先进入“Step 0”Response 类型的异常处理类对象 RS_Standby_WithTissue. 并进入它的状态机内进行异常处理. 一旦完成. 它便会进入第二个“Step 1“. 弹出一个对话框. 让用户清洁相关的液位器. 如下图 9 所示:


图 9 用户提示对话框

此时. 我们将液位传感器恢复到内接状态. 从而温度也恢复正常. 然后. 点击上图对话框的按钮. 这时. 系统便会执行“Step 2”RC_Levelsensor_heating_Overtime. “Step 3”RC_Restart 等使机器最终恢复正常状态. 并继续正在运行的程序.

## 4 对算法扩展的思考

当前版本的异常处理算法. 一次只处理一个系统异常. 也就是说. 当在异常处理过程中. 下位机发生的异常. 该软件系统是忽略的; 并且. 该算法同时也只处理一个异常. 关于第一点. 我们可以考虑在系统中增加一个硬件监控模块 HWMonitor 模块. 无论系统是在正常状态(Idle 和 Running). 还是在异常状态(Exception)状态. 软件都会利用一个 500ms 的定时器来检测是否有来自于下位机的异常. 而当系统将当前异常处理完成. 会首先查看是否有另外的异常存在. 如果有. 会继续处理. 直到所有的异常都处理完成. 关于第二点. 可以考虑利用多线程技术. 同时对多个异常进行处理但是. 要考虑到这几个异常之间的相关性. 以及时序等因素. 我们需要设计一个通用的线程间同步的机制对多个异常进行同时处理.

另外. 如何实现运行时的动态配置 Event-Scenario 表和 Error 策略表等. 也是本算法的扩展方向. 关于这一部分. 可以结合嵌入式系统自身的特点. 利用 Sqllite 等轻量级文件数据库对本算法进行扩展与优化.

## 5 结语

本文介绍了一个针对病理系统中的脱水机上的嵌入式系统异常处理的算法. 本算法利用 XML 以及异常处理类等方法. 使算法有着很好的实用性. 加之 QT 开发平台的支持. 又使本算法也比较易于实现. 另外. 在算法具体实现上. 也充分借鉴了面向对象程序设计中的 OCP. SRP 等设计原因. 另外. 本文也利用实验认证了算法的实用性. 同时也简要讨论了算法的扩展性.

通过以上的论述. 不仅仅是针对病理系统中脱水机、包埋机等. 我们希望也可以为相关的嵌入式软件系统的异常处理提供一个有益的参考.

## 参考文献

1 Sutter H. Exceptional C++(英文版).北京:机械出版社,2006.
2 Stevens R, Rago S. Advanced Programming in the UNIX Environment. Second Edition. New York: Addison Wesley Professional. 2005.
3 Larman C.李洋,郑 等译.UML 和模式应用(原书第 3 版). 北京:机械工来出版社,2006.
4 Summerfield M.白建平等译.QT 高级编程.北京:电子工业出版社,2011.
5 Apache Software Foundation. Apache Xerces Project. http://xerces.apache.org/.
6 QT Development Platform. Qt Company. http://www.qt.io/developers/.
7 yEd Graph Editor. yWorks. http://www.yworks.com.