迪爱斯三台合一产品平台刨析

1. 平台架构

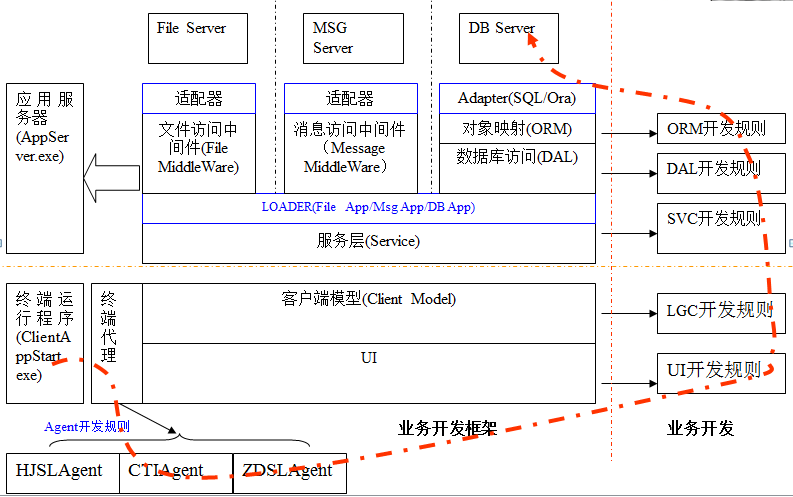


图 1-1

1. 从图1-1 可以看出公司的平台框架是一个重客户端业务系统，从架构上来分析目前这种划分是一个比较普遍的划分方式，对于C/S系统客户端由宿主程序和代理组件构成，这种设计是比较好的一种方式。

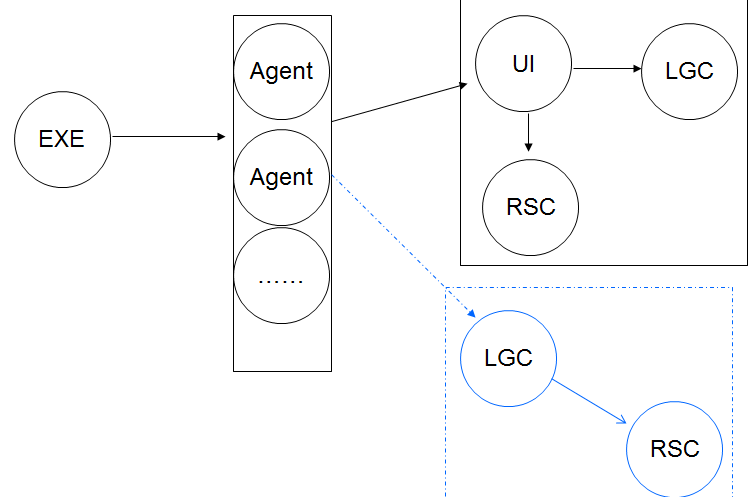
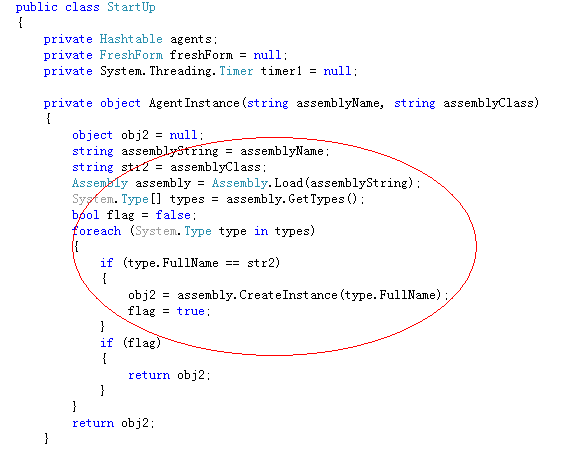


图 1-2

1. 这个图体现了UI框架的思想，这个思想可以很好的隔离和构建模块化组件。
2. 但在UI框架里，界面控制和数据是UI框架的骨架和血液，在数据的处理方面还是比较简单的。并且和UI控件的绑定也是非常淳朴的。没有体现出WPF中DependencyProperty 优势；在控制方面也缺少一些必要支持；
3. 在UI界面框架中，对开发人员提供了事件通知机制，但该事件机制类似控件的冒泡事件机制，这种机制有他的开发便捷性，但对于多界面事件通知时往往会大大加大开发和维护难度。Reported会使代码理解比较困难，不便于维护。问题更出在，我们的组件不论是接口还是函数都喜欢使用隐晦的参数。导致跳过了编译器的校验，只有在执行时才能发现错误，而有些可能在某种条件下才会出现错误。给系统稳定性带来巨大隐形炸弹。
4. 代码中有很多关键点采用的直接反射，这种做法可能会带来内存泄漏问题，一般采用AppDomain来加载。但更好的办法是使用IOC或MEF框架。由IOC对象容器来帮你处理对象生命周期问题。



1. 代码中有大量命令接口，最好使用面向对象的方式，而不是直接字符串来传递参数；这是OO的思想，需要融入在整个平台的血液中。如图

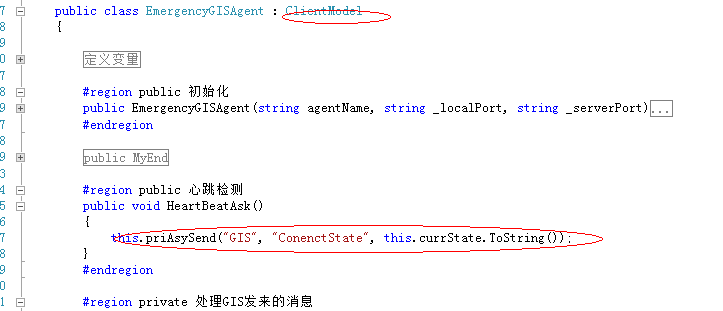


图 1-1

可以看到函数基本都是用字符串作为命令来执行，而且基类里有消息发送功能，代码里又实用Socket来发送消息。导致代码功能重复。而且通信组件也没有一点封装。



1. 配置文件比较零碎，不论对开发人员还是实施人员，配置位置的多点，文件的多个，都导致配置是个灾难。并且配置还是采用XMLDOC的方式，后期需要调整非常困难。最好使用对象化方式，既便于理解也便于维护。如图



这种滞后的设计，会导致很多问题不能在前期暴漏出来。是把风险留在用户来检验的模式。我认为这种方式是不可取的。

1. 代码组件的调用不区分平台的核心组件、平台组件和业务组件，导致很多功能让开发人员混淆。重复代码组织混乱。
2. 平台组件



1. 平台项目76个，看起来非常庞大，划分下来的组件实际上很少。由于层次过细的划分和没有模块化的编排导致一个很简单的组件由大量项目组成。导致后期维护和重构都加大难度。虽然平台也分框架组件和产品组件，但每个组件设计还有欠缺有些职责不清如登录组件其实并不是平台架构核心组件这样导致组件物理存放和组件所属很难确定。目前以三台合一项目在生产上其实只用到如下组件：**服务器宿主组件**、**客户端宿主组件**、**消息组件、ORM(单表维护和脚本查询、数据库适配)、登录验证组件、远程调用组件**，作为平台基础类库能为开发提供的实用类库太少。
2. 数据访问组件，即ORM。目前该功能比较单一，还欠缺很多基本功能。导致没有起到ORM减少基本维护工作，反而增加了代码开发量。尤其是把sql放在代码之外，这种方式本来是让项目修改比较方便，对小项目应该是一个不错的方案。但如果系统复杂那可就是灾难了。
3. 通信组件，Remoting和21通信平台。对于21通信平台，目前还是比较成熟的通讯组件，并且和我们公司的基础组件如CTI已经做了大量历史性工作。是可以优化他并延续发展的组件之一。对于Remoting的发展早已经被WCF架构取代，微软也明确不再支持。所以作为主流的WCF是平台通信组件发展的必要组件。
4. 登录组件，功能比较简单。但该组件应该成为一个独立模块。尤其验证方式采用WCF来降低耦合统一公司的SSO。
5. UI框架组件，在UI框架中为我们提供了界面的加载、事件通知机制。这两个功能都是UI框架必须的功能，对于界面加载Agent的方式本意是需要框架和注入模块逻辑解耦。这种方式是个很不错的思想。是可以延续的，但还需要强化和明确Module的职责，此模块是针对界面框架域的。而不是平台的组件。在平台产品中发现大量把逻辑写在父窗体上，靠冒泡事件来触发。首先，这种把逻辑写在界面上的基于脚本模式本来就不是一个好的办法。更不用说还是写在父窗体上了。这样加大了代码跟踪难度和维护成本。在UI框架中还欠缺大量必须功能组件，如页面导航组件、界面展现组件、异常错误页面、环境变量组件、通信组件、数据缓存组件、并行线程组件等等。只有这些组件才能让开发人员开发更高效。
6. UI控件，在这里几乎是个空白，界面开发几乎没有封装，由其是Grid控件使用起来非常麻烦。失去了配置列头的意义。
7. 业务组件封装比较面向过程，使代码组织和理解比较困难。基本需要跟踪理解。
8. 平台组织
9. 代码组织从层的角度来看，对目前业务需求是足够细了。但对于从业务模块化组件化来说没有很好的剥离，才导致平台和生产项目中竟有170多个项目存在。对于维护是难以想象的。
10. Dll的生成以及依赖关系复杂，本来由于项目较多导致有大量dll产生，项目之间不是直接引用，而且生成目录位置还有一定讲究，除此之外还要设定私有目录来加载dll。真是一个浩大工程。
11. 配置文件和dll一样，是相互关联的。由于配置文件被部署到多个目录并且，配置文件的格式写法都各异。很容易让开发和实施人员遗漏。
12. 代码命名很容易让人误解。命名和主流不太一致。并且有很多命名是非常不好的。如变量全大写拼音等。
13. 业务组件功能和平台系统功能混杂。很难发现平台功能组件功能需要到哪里找。
14. 开发模式
15. 平台的架构和开发模式是统一的。如果开发的专业化分工，那就需要平台对不同专业进行划分。人不是全才，要做个简单的系统可能还能将就，但随着系统的不断复杂。体系化不断完善。对开发人员的要求也越来越高。只有技能的细分才能既降低开发人员水平。有能适应用户对系统的高要求。
16. 开发可以划分成：界面交互、业务编排这两部分，这里的界面交互主要是侧重用户交互体验和根据业务来设计对应交互体验的UI控件和特效。而业务编排这部分则侧重业务的组织和再利用。

业务编排图



界面开发模型图

1. 业务需求

略