Fuxi:一种敏捷的嵌入式应用开发环境

王忠斌

深圳市海创达资讯技术有限公司

计算新技术实验室

(e-mail: victor@fuxi.org 邮码:518048)

黄继东

深圳市卓讯达科技发展有限公司研发部

(e-mail: jidong@iwt.com.cn 邮码:518040)

摘要：本文提出了一种基于软件架构的、面向方面的敏捷软件的方法学，将表达力和效率作为嵌入式应用的两个最重要的关注加于考虑：表达力带来敏捷性，效率带来可用性。依据关注分离的原则，将系统分解为功能方面和若干个为功能方面提供支持的技术方面，我们主张不同的关注采用不同的编程手段；在功能方面采用高表达力的说明性编程，而技术方面采用高效率的过程性编程；最后通过抽象机将功能方面和技术方面动态地编织起来，完成系统的全部功能。Fuxi 语言与 Fuxi 环境是上述观点的载体，其本身也是采用敏捷软件的方法学进行开发的。

关键词：嵌入式系统；面向方面；程序设计语言；说明型语言；Fuxi 语言；中间件；Fuxi 平台

Fuxi: An Agile Embedded Application Development Environment

Victor Z. Wang

Laboratory of Novel Computing

Hitrend Information Technologies, Inc.

(e-mail: victor@fuxi.org ZIP:518048)

Derek J. Huang

R&D Department

Shenzhen Intelliworks Technologies Co., Ltd.

(e-mail: jidong@iwt.com.cn ZIP:518040)

Abstract: In this paper, we proposed an architecture-based, aspect-oriented methodology of agile software development, which takes the expressivity and the efficiency as two major concerns of embedded applications; expressivity results in agility, and efficiency yields usability. Under the separation of concerns, we differentiate the system into functional aspect and several technical aspects which give supports to the functional one. Different concern, different method. At the functional aspect, we takes declarative programming, and at the technical aspects procedural programming. An abstract machine, as a pivotal of the architecture, weaves all these aspects dynamically at runtime, to achieve all the functionalities of the system.

Fuxi programming language and its environment are the embodiment of the above methodology; and Fuxi environment itself was developed through such an agile software process.

Key words: embedded system, aspect-oriented, programming language, declarative language, Fuxi language, middleware, Fuxi platform

1. 前言

受市场竞争和技术进步的影响，嵌入式产品的需求变化往往贯穿于产品开发的始末。往嵌入式应用开发过程中注入敏捷性[2,4]，适应需求的不断变化就显得十分重要。由于嵌入式开发受环境和资源的制约，开发人员之间的交互，特别是同客户之间的沟通就要比普通的台式环境困难得多了。因此，K. Beck、A. Cockburn 等人的敏捷软件过程在嵌入式应用中实施就存在了一定的困难。

本文提出一种基于软件架构的方法，结合面向方面[3，5]的软件技术，通过模块化嵌入式应用的各个技术方面，达到控制系统的复杂度，提高软件开发过程对需求变化的适应性，从而达到向嵌入式应用开发过程中注入敏捷性的目的。

我们的基本思维是，将系统的设计空间划分为功能性(Functionality)和技术性

(Technology)两个正交的维度(或方面)。在系统的功能性维度里，主要涉及系统的功能、行为、人机界面等方面内容；而技术性维度主要是那些为功能性内容提供支持的技术，如并发性、分布性、移动性支持等。在功能性维度，我们采用一种平台无关的、面向对象的说明型语言——Fuxi 语言编程，可以快速地获得可执行的软件原型(runnable prototype)，甚至是可工作的软件。而在技术性维度，则采用 C/C++在 Fuxi 元对象模型框架下编程，为系统提供高效率的技术实现。在运行时，由 Fuxi 抽象机将功能性和技术性两方面内容动态地编织(weave)在一起，完成系统的全部功能。

我们所主张的软件过程的敏捷性与 K. Beck 和 A. Cockburn 等人所提出的敏捷性在方法学上存在着不同。A. Cockburn 等人漠视了软件架构的价值，主张不同的软件采用不同的方法，这样就很难获得软件的可重用性[8]。而 Fuxi 技术则一开始就是在软件架构的框架下，采用关注分离(Separation of Concerns)的思想[6,7]，主张在统一的架构下不同的关注采用不同的方法。我们认为，在软件的开发过程中，容易变化的部分主要集中在产品的功能性方面，敏捷性应主要针对产品的功能性而言，而技术性部分则应强调高效率和可重用性。

2. Fuxi 语言

Fuxi 语言是一种面向对象的说明型程序设计语言。作为一种面向对象的程序设计语言，Fuxi 继承了 C++/JAVA 的风格，是一种类型安全的对象系统；作为一种说明型语言[9, 10]，Fuxi 结合了函数型、逻辑型、约束型语言的特点，广泛地从函数型语言(如 SML、Erlang、Miranda、Haskell)、逻辑型语言(如 Prolog、Gödel 等)、函数逻辑型语言(如 PLG、BABEL、CURRY 等)中汲取养分。可以说，Fuxi 是一种说明型的 Java。

Fuxi 语言的设计目标就是，将那些与实现计算的相关技术从语言的构成中剔除掉，留下那些可以为各领域专业人员所共同接受的概念(函数、规则、类等)，让程序成为一种纯洁的对计算本身的描述，从而方便其他领域的专业人士(如产品策划、美工、甚至客户)使用。

2.1 Fuxi 的文法

Fuxi 文法可以概括为以下的伪公式：

Fuxi 文法 = 类似 Java 的类型框架 + 模式匹配的方法构成

以下是一个计算 Fibonacci 数的 Fuxi 程序：

```
import fuxi.*
public active class FibonacciApp: Applet {
   Fib(0) = 1
   Fib(1) = 1
   Fib(int n) = Fib(n - 1) + Fib(n - 2)
   public OnError(ERROR_MEMORY_ALLOC) -> Console.Println( “内存不足！” )
   public Activate() = let{ int n = Console.Readln().ToInteger() } in {
      Console.Println( “请输入一个整数:” )
      Console.Println( “Fib(“ + n + “)=” + Fib(n) )
   }
}
```

图 1．一个简单的 Fuxi 程序

图 1 是一个典型的 Fuxi 控制台应用程序，它计算 Fibonacci 数。从这个例子中我们可以看出：

1. Fuxi 的文法和 C++、JAVA 或 C# 极其相近；
2. Fuxi 采用模式匹配的方式定义，如 Fib 函数，包含三个模式 Fib(0)、Fib(1)和 Fib(n)；
3. Fuxi 的方法区分函数、子句和触发器，函数的模式和定义体之间采用‘=’连接，

而子句采用‘<-’连接，触发器采用‘->’连接；

4. Fuxi 支持惰性计算，即表达式只有在用到时才计算值。局部变量 n 在建立时并没有被计算，而只是在被引用时才进行计算。

2.2 对象的正交风格化

正交的对象风格化(Orthogonal Stylization of Objects)是对象的正交持续性(orthogonal persistence)的扩展，指的是对象风格和对象类型正交。对象风格指对象在执行过程中表现的姿态，例如，是否拥有自己独立的线程、是否具有持续性、是否可以在网上移动等；而对象类型指对象的结构和行为，对象类型通常是类。我们可以把对象的风格想象成产品的颜色，

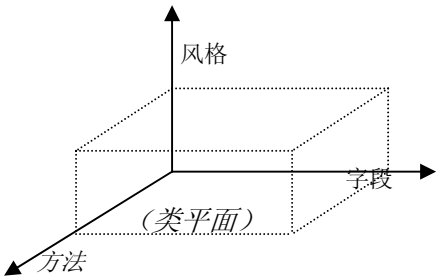

图 2. 实例的三维空间

```
class A { … }
active A a_act // 主动式对象 a_act
mobile A a_mob // 可移动对象 a_mob
persistent A a_per // 持续性对象 a_per
remote A a_rem // 远程对象 a_rem
@WebSecurity A a_sec // 自定义风格对象
```

图 3. 对象风格的使用

产品的设计师通常关注产品结构本身的设计，而忽略对颜色的考虑。然而，当产品成型后，工人可以为产品着上各种不同的颜色。

Fuxi 在语言级对风格提供支持，可以让程序员只关注类结构本身的设计，关注问题的本身，而忽略对同问题方案正交的那些技术性方面的考虑。

Fuxi 语言目前支持的系统风格(built-in styles)有 active, mobile, persistent, remote，同时 Fuxi 也提供对用户自定义风格的支持。用户自定义风格使得 Fuxi 具备了面向方面的基本特征。所谓风格，其实就是和风格相关的方面同某个类进行动态编织的标注(annotation)。用户自定义风格包括两个部分：风格定义(style specification)和风格实现(style implementation)。风格定义是 Style 类的派生类，其定义体中包含一组触发器，当程序执行到了某个特定的位置，其上下文(context)同某个触发器的参数匹配，从而激活风格实现中的某个方法。风格实现，为 StyleImpl 的派生类，通常采用本地实现。风格定义中的触发器的名称是预定义的，它们包括：

```
public class WebSecurity : Style {
    static WebSecurityImpl    security()
    before(GET_ATTR, _, _ ) -> security.Check()
    before(INVOKE, _) -> security.Check()
    …
}
```

图 4. 一个风格说明的例子

before(GET_ATTR,<attr-name>) -> {…}
after(GET_ATTR, <attr_name>,
    <attr_value>) -> { … }
before(SET_ATTR,<attr_name>,<value>)
 -> { … }
after(SET_ATTR, <attr_name>, <value>)
-> { … }
before(INVOKE, <method_name>, <param-list>) -> { … }
after(INVOKE, <method_name>, <param-list>) -> { … }
after(BACKTRACK, <method_name>, <param_list>) ->{ … }

3. Fuxi 的软件架构

Fuxi 的软件架构是以元对象[1]GOBJ 结构(Generic Object)为基础，以抽象机为枢纽的软件架构；抽象机向这些对象提供归约计算和反向推理的支持，同时抽象机也是不同关注(concerns)实现的对象按其风格标注进行动态编织的编织器(Weaver)。

3.1 Fuxi 的元对象结构

每个 Fuxi 对象都是从一个 GOBJ (Generic Object)元对象派生出来的。Fuxi 的基本库为我们定义了这个元对象。除了 GOBJ，基本库还定义几个 GOBJ 的派生类，其结构如图 5 所示。Fuxi 对象通常需要两个虚拟方法表(VTable)，分别表示功能性和技术性两个维度。其中技术性 VTable 是每个对象必须具有的，GOBJ 中的唯一字段就是指向这个 VTable 的指针。

在技术性维度中，有一段为所有对象所共有的部分，称为公共接口。Fuxi 平台规定了功能接口的原型和语义。公共接口主要包括：Fuxi 对象的识别函数、反射函数、数据操作函数、调试支持函数等。其中反射函数是获取对象功能性信息的工具，是技术性和功能性进行编织的手段。

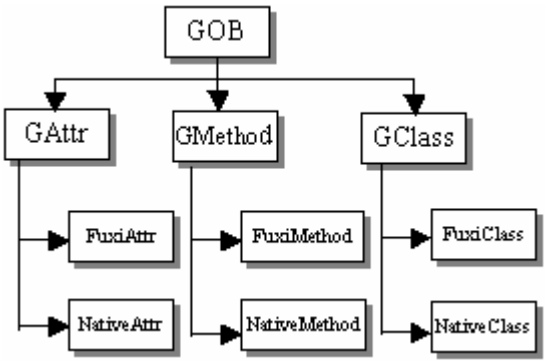

图 5. Fuxi 元对象层次

技术性 VTable 和功能性 VTable 所指的内容存在着很大的不同，前者是由本地机器指令组成的可执行的代码段(过程)，而后者则是由 GMethod(Generic Method)接口派生出来的对象。根据实现机制的不同，这些 GMethod 对象可以是：FuxiMethod、NativeMethod 等，分别表示由 Fuxi 对象图(FOG)实现的 Fuxi 方法、本地指令实现的本地方法。通过 GOBJ 中的反射函数 GetMethod，可以获取功能性 VTable 中的这些 GMethod 对象。

3.2 Fuxi 抽象机

在 Fuxi 架构中，Fuxi 抽象机起着联系各方关注的枢纽作用。Fuxi 抽象机采用图计算模型[10]，对一种称为 Fuxi 对象图(FOG)的计算图进行变换求解。Fuxi 抽象机模型是由 Frame 和 Trail 栈组成的双堆栈结构，既可以计算又可以推理。原来用于实现核心语言(KL)，后经过面向对象和面向方面的扩展。

3.3 Fuxi 平台

Fuxi 平台是一个 Fuxi 程序能够执行的最小环境，它包括 Fuxi 抽象机(FAM)和 Fuxi 的基本包体系。用户可以根据实际需要对 Fuxi 平台进行扩展，向平台增加新的包。

在 Fuxi 的包体系中，包分为两种类型：语言包和方面包(Aspectual Package)。其中，语言包是供 Fuxi 程序调用的的包，如 SHOM (表单对象模型，一种说明型的 GUI)；而方面包则是 Fuxi 平台所提供的风格实现，供其它 Fuxi 本地类或抽象机调用；而 Fuxi 应用程序不能直接调用方面包。

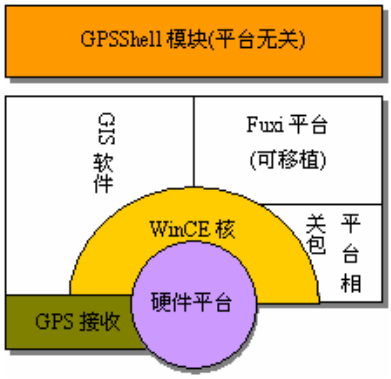

图 6. Fuxi 平台的应用案例

4. 一个应用案例

某公司经过市场调查，发现车载 GPS 具有巨大的市场潜力，决定开发 GPS 产品。此时市场上已经有几款 GPS 产品在销售，因此开发时间就被压缩得很短，要求尽可能快地出产品。

根据甲方快出产品的要求，我们选择了性能较好的 Intel 270 芯片和环境友好的 Win CE。当我们介入时发现，甲方对 GPS 的需求极其模糊，根本就提不出具体的产品需求报告。鉴于形势，我们提出了兼顾各方关注的 Fuxi 平台方案(见图 6)。这一方案可以概括为以下公式：

嵌入式 OS + Fuxi 平台 + Fuxi 的用户平台相关的扩展 + Fuxi 应用模块

Fuxi 应用模块是采用 Fuxi 语言编写的功能方面的程序，与平台无关的。在我们的案例中，

这部分内容主要由 FAE 和用户共同开发的(借助于 Fuxi 的 IDE)，并且可在 PC 上执行。在项目开发的过程中，这部分内容经过了多次的修改，最终形成了几个不同的个性化版本。

而 Fuxi 平台的类库和用户平台扩展部分则由系统工程师来根据需要采用 C/C++编程，并且把可移植、可重用作为软件设计的要求。由于功能性与技术性关注的正交分离，系统程序员就不会因应对需求的变化而分散注意力，而把精力集中在高品质的程序开发上。

在试产之后，甲方又提出降低单台成本的要求。最终选择了三星 2440 芯片和 Linux。由于 Fuxi 功能模块的平台无关性，不需要作修改；而 Fuxi 平台的类库等在一开始就把可移植性作为开发的标准，因此，很快就完成了这部分的移植工作。

5. 结论

就嵌入式应用开发的方法学而言，表达力与效率是两个必须给予关注的方面：表达力带来敏捷性，而效率带来可用性。Fuxi 技术在尊重敏捷软件倡导者们所提倡的价值理念的同时，反对完全自由的敏捷性，而主张敏捷性应当源自对软件架构和对系统不同方面关注的理解。因而提出了一种以抽象机为核心的、面向方面的软件架构，将表达力与效率作为两个最重要的关注而加于考虑，并主张不同的关注采用不同的编程手段，即功能性方面采用说明性编程，而技术性方面采用过程性编程。

Fuxi 语言和 Fuxi 平台是上述观点的载体。通过一个具体的开发案例，证明了我们的观点在嵌入式领域中的适用性。同时，Fuxi 环境本身也是采用了敏捷软件方法学进行开发的，在开发尚未完成时就已经获得了应用。

6.参考文献

[1] Baker, J. and Hsieh, W., “Runtime Aspect Weaving through Metaprogramming”. *Proc. of AOSD 2002*, ACM Press. 20002, pp.86-95

[2] Cockburn. A., *Agile Software Development*. Boston: Addison-Wesley, 2002.

[3] Elrad, T., Filman, R.E. and Bader, A. “Aspect-oriented Programming”. *Communications of the ACM*, *44* (10), October 2001, pp.29-32.

[4] Highsmith, J., Cockburn, A., “Agile Software Development: Business of Innovation”, *IEEE Computer*, *34*(9) Sep. 2001, pp.120-122

[5] Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J. and Griswold, W.G. “Getting Started with AspectJ”. *Communications of the ACM*, *44* (10), October 2001, pp.59-65.

[6] Kiczales, G., Mezini, M., “Separation of Concerns with Procedures, Annotations, Advice and Pointcuts”, ECOOP 2005, LNCS 3586, pp.195–213

[7] Ossher, H. and Tarr, P. “Using Multidimensional Separation of Concerns to (Re)Shape Evolving Software”. *Communications of the ACM*, *44* (10) 2001, pp.43-50.

[8] Turk, D., France, R. and Rumpe, B. “Limitations of Agile Software Processes,” *Proc. of Third International Conference on extreme Programming and Agile Processes in Software Engineering*, Italy, May 2002.

[9] 王鼎兴等, *逻辑程序设计语言及其实现技术*, 清华大学出版社 1996

[10] 郑纬民等, *函数型程序设计语言*, 清华大学出版社 1997

王忠斌：男，1967 年出生，总工程师兼实验室主任；主要研究方向：程序设计语言，软件架构学，新型计算技术（普适计算、全域计算等），集成化开发环境，嵌入式系统。