CN43-1258/TP
ISSN 1007-130X

计算机工程与科学
COMPUTER ENGINEERING & SCIENCE

2005 年第 27 卷第 2 期
Vol. 27, No. 2, 2005

文章编号:1007-130X(2005)02-0099-05

SOLO STUDIO:一个基于构件的应用开发环境*
SOLO STUDIO: A Component-Based Application Development Environment

蒋 韬,张 斌,赵文耘,张 志
JIANG Tao, ZHANG Bin, ZHAO Wen-yun, ZHANG Zhi
(复旦大学软件工程实验室,上海 200433)
(Software Engineering Laboratory, Fudan University, Shanghai 200433, China)

摘 要:传统的软件开发环境着眼于从头开始开发某个应用系统。而基于构件的开发方法最明显的特点是整合而不是从头开发,整合的对象是由不同的三方构件提供商开发出来的构件。本文着重于基于构件的软件开发环境的设计,并且描述了一个原型系统 SOLOSTUDIO。该系统后台有基于 WebServices 的构件库平台作为支持,SOLOSTUDIO 的主要功能包括从构件库中搜索并且下载合适的构件,整合构件进入待组装系统的构架之中,测试构架的性能并做出评估,动态演化构架使其适应不同需求的用户,生成最终的应用系统等。

Abstract: Traditional development environments emphasize a process in which an application is developed completely from scratch, yet the remarkable feature of component-based development devotes itself to integrating the components developed by third-party dealers into an application. This paper is concerned with the design of component-based development environments and we introduce a prototype system called SOLO STUDIO in order to further our study of the component-based development technology. The main functions of this environment include searching and downloading suitable components from the database of components, integrating components into the architecture of an application, evaluating the performance of the architecture, evolving the configurable architecture to fit for different users, and building the final application system.

关键词:构件的软件开发环境;构架;构架风格;构件;构架定义语言
Key words: CBDE; architecture; architecture style; component; ADL
中图分类号:TP311 **文献标识码**:A

1 引言

当前已经有许多成熟的构件标准,如 Microsoft 的 COM/DCOM/COM+技术、SUN 的 EJB 技术、OMG 的 CORBA 技术。但是,不同的构件提供商给出的标准存在极大的差别,这从根本上违背了构件技术发展的初衷——提供大粒度的软件复用并借此缩短软件系统的开发周期,提高软件系统的开发效率[1]。事实上,相当大的精力都用在了考虑如何整合这些不同的构件标准之上。SOLO 项目研究的目的在于为提出一个通用的构件标准打开一条通道。原型系统 SOLOSTUDIO 的开发最先是为了组装 COM/DCOM 构件。但是,我们逐渐发现只要能提供一种合适的构件包装方式,SOLOSTUDIO 可以用来组装当前任何三方构件。

本文提出了设计基于构件的软件开发环境(CBDE)的前提,并且描述了我们的一个原型系统 SOLOSTUDIO。SOLOSTUDIO 由 Borland Delphi 开发,可以组装代码级别的 VCL 组件,也可以组装三方构件商提供的 COM/DCOM 构件、EJB 构件。

2 SOLOSTUDIO 项目

SOLOSTUDIO 是一个基于构件的软件开发环境。为了实现该 CBDE,我们设计并且实现了一套基于构件组装的应用框架(CBCAF)。该框架融入了许多前人提出的基

* 收稿日期:2003-09-24;修订日期:2003-12-11
基金项目:上海市科委科技攻关项目(025115014);国家 863 计划资助项目(2002AA114010,2001AA113070)
作者简介:蒋韬(1978-),男,安徽合肥人,硕士生,研究方向为软件工程。
通讯地址:200021 上海市淮海中路 333 号瑞安大厦 8 楼 5892 邮箱;Tel:13816292066;E-mail:hellotao@msn.com
Address: Mail Box 5892, 8th Fl, Ruian Building, 333 Huaihai Rd Middle, Shanghai 200021, P. R. China

于构件组装的思想，如基于事件和消息的构件通讯机制[2]（Event-based Corresponding Mechanism）、构件端口（Port）、连接器（Connector）、构件配置管理。框架的实现列表如图1所示。

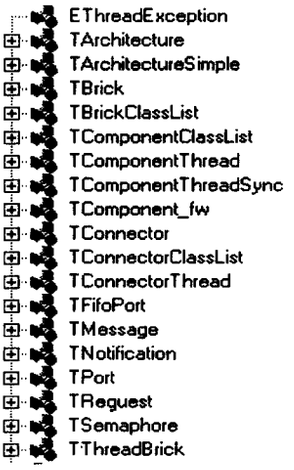


图1 基本应用框架

SOLOSTUDIO和那些有待被组装的应用系统一样，本身也使用并且符合这套框架的规范，我们称之为组装系统的自演化（Self Evolution），即该组装环境本身也由构件组装完成。图2显示了SOLOSTUDIO的主体构架，其中的构件和连接器都继承框架中的基类。SysADT构件用于维护构架描述语言（ADL），SoloComponent构件用于显示组装系统的构架图形并且允许用户生成、删除和动态演化系统构架，AemComponent构件用于维护被组装系统的构架实例以及其动态演化的性能。

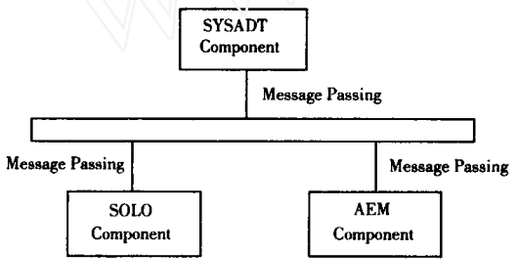


图2 SOLOSTUDIO的构架图

2.1 从SYSADT构件看构件的包装

SYSADT构件包装了一个COM构件MSXML3.0，该COM构件是XML文档的DOM接口。由于COM/DCOM构件只能按照它们自己的方式进行交互，即通过COM Event（事件产生器）和COM EventSink（事件接受器）来完成构件之间的交互，构件之间的行为不能由组装用户决定。这显然不满足构架组装的要求，通过对于每个构件的包装，每个COM构件作为组装构件的内部对象从而和外界的系统构架没有直接的关系，COM构件之间的事件被包装转化为请求（Request）和通知（Notificaiton）消息[3]。构件包装的内部构架如图3所示。

构件的包装信息主要是构件的描述信息，需要描述的信息包括构件常规信息（记录构件的基本信息）、构件刻面信息（供构件库检索构件使用的信息）、接口信息（包括接口ID、接口名称、接口方法和事件。这些信息能在可视化的构

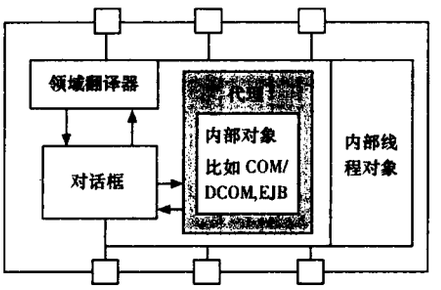


图3 构件的包装

件组装平台中提供给用户察看，用户可以根据系统需要选择使用的方法和事件）。生成的包装类应该包括以下内容：

（1）接口变量。一个COM/DCOM构件中可能包括多个接口，每个接口都用一个独特的变量来表示，调用构件的方法时，就调用该方法相应的接口变量的方法。

（2）构件方法。对于构件的每个方法，都生成一个相应的包装类的方法，该方法的名称和参数都和构件的方法相同，该方法的实现就是调用该方法所属接口变量的相应方法。

（3）端口方法。为构件的每个端口都生成一个方法。该方法有一个参数，就是该端口的接收消息。方法的实现就是将端口中的一系列操作转化为可以执行的代码。初始化方法主要完成构件的实例化和事件连接处理。消息处理方法则负责在该构件接收到消息时，根据消息的内容调用相应的端口方法。对象包装器应该提供下列服务：当对象接口的访问路线被调用时，包装器把调用和其返回值转化成一个通知，接着把这个通知传送到下层的连接器中。因此，内部对象的接口决定构件发出的通知的类型。

2.2 编程语言

我们选择Delphi最初的考虑在于Delphi对于COM/DCOM构件的良好支持。虽然在JAVA下也可以通过编写桥代码的方式应用COM/DCOM构件，但需要大量额外的工作。此外，Delphi的Object Pascal的语言特征为开发基于构件的开发环境提供了很大的方便：

（1）接口作为明确的实体而存在，可以统一对待接口和类，接口封装构件内部实现；

（2）每个接口都具有全球范围内独一无二的IID；

（3）虽然需要编写代码来注册所有的构件，但我们可以和像JAVA的Reflection技术一样通过构件的名字来获取构件的实例；

（4）Delphi支持动态连接库，具有DLL和DPL两种形式，我们采用了DPL技术，保证所有的构件可以动态加入到开发环境中而不需要重新启动整个系统。

2.3 应用组装过程

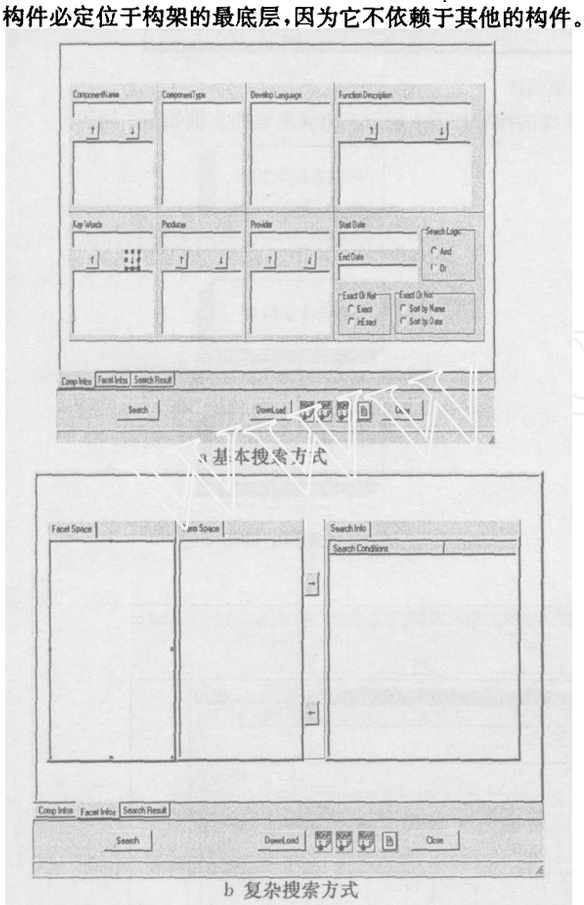
在SOLOSTUDIO环境中的整个应用组装过程包括：首先在定义了构件的需求之后，可以到构件库中查找相关的构件并且下载到本地。当然很有可能搜索到的不止一个构件，这时候需要由组装用户找到最合适的构件。每个构件必须经过包装和连接配置才能够被整合到构架之中，因为不太可能存在一个三方构件完全满足用户的需求，构件的接口所能提供的服务总是或多或少。总的来说，应用组装过程是一个涉及到搜索构件、选择构件和配置构件的迭

代过程。搜索可以自动化，但选择和配置构件过程则需要设计经验和人的创造性活动。

2.4 搜索和选择构件

一般来说，应用程序组装用户会根据自然语言的关键字来搜索构件，如图 4a 所示。这些关键字包括一些构件的基本信息，如构件名称、构件类型、版本、厂商、开发语言等。同时，SOLOSTUDIO 提供更加复杂的搜索方式，如通过刻面或者术语来搜索构件，如图 4b 所示。由于提供的搜索条件不可能足够精确，搜索的结果往往是构件的一个集合，这时候就需要设计人员根据构件的自我描述信息选取合适的构件。

SOLOSTUDIO 提供了一个窗口用于显示搜索出来的构件的自我描述以及端口信息，这方便了设计人员对构件的复杂性作出判断。例如，一个下部端口没有提供服务的构件必定位于构架的最底层，因为它不依赖于其他的构件。



a 基本搜索方式

b 复杂搜索方式

图 4 构件的搜索

2.5 面向类型的构件配置

SOLOSTUDIO 拥有一个图形编辑器，如图 5 所示，可以显示和允许用户编辑系统构架图形。我们知道，每个构架都需要具有一个构架风格，构架风格是构架的规范。SOLOSTUDIO 的构架风格包括：每个构件都拥有一个上端口和一个下端口，上端口用于发出请求和接受通知，下端口用去接受请求和发出通知，构件和构件之间的依赖关系体现在层次上。例如，如果 A 构件总是需要得到 B 构件的服务，那么 A 构件应该位于 B 构件的下层。当前的端口数目并不允许改变，也就是说，一个端口既可以接受请求，也可以发出通知，这样可能导致端口功能不明确。可以提供比较的是，ICS OF UCI 开发的基于构件开发的 WREN[4] 环境则提供了请求端口和服务端口的区别，并且不限制端口的数目。这样，语义上的分离对于构架的测试和动态演化都是非常有利的。

2.6 构架的动态演化

首先，明确的架构模型需要有精确的架构模型。既然模型的动态变化必须要被实例化到架构的实现中去，必须要提供一个从模型到实现的映射。其次，能够描述动态架构的变化。这里的变化指架构模型的变化，包括这些操作，例如添加、删除和替换某个构件或者连接器，改变整个架构的拓扑结构。还有就是对于模型的查询也应该是修改的一部分，因为对于架构的修改经常是依赖于系统的配置。再次，能够管理动态架构的变化，虽然架构风格的规则和连接器可以用来防止动态架构带来的对系统一致性的危害，一种用来管理应用程序变化的一套机制还是必要的。约束在这里起到了很大的作用，而且还应该提供一种交互式的修改机制。最后是具有重用的动态架构的基础工具，这个工具的作用有：(1)用来维持模型和构架实现的一致性；(2)使模型中的变化映射到实现当中；(3)保证架构的约束不被违反。为了实现构架的动态演化，SOLOSTUDIO 提供运行如下对组装用户可见的编程接口：

```
Component Interface
    Start()
    Finish()
    Handle(request)
    Handle(Notification)
Connector Interface
    Start()
    Finish()
    Handle(request)
    Handle(Notification)
    addTopPort()
    removeTopPort()
    addButtonPort()
    removeButtonPort()
Architecture Interface
    Start()
    Finish()
    Handle(request)
    Handle(Notification)
    addComponent(Component)
    removeComponent(Component)
    addConnector(Connector)
    removeConnector(Connector)
    Weld(Connector, Component)
    Weld(Component, Connector)
    Weld(Connector, Connector)
    UnWeld(Connector, Component)
    UnWeld(Component, Connector)
    UnWeld(Connector, Connector)
```

其中，Start()方法和 Finish()方法用于初始化和终止构件、连接器或架构的运行。如果一个构件正在处理消息，那么 Finish()方法并不立刻结束整个组件的运行，而是等到消息处理完毕以后才结束构件的运行。面向应用程序的构件可以 Override 方法改变这些构件的行为，比如，大多数新加入架构的构件用 Start()方法来保持和整个系统的同步，而连接器提供两个额外的方法：Weld()和 UnWeld()方法，在系统运行的时候添加或者删除一个构件。

Handle(request)和 Handle(Notification)方法作为处理消息的对话框，保存这个构件在接受消息和发送消息时所需的动作和行为。当然，这个对话框不是黑盒的，允许用户使用自己定义的行为序列以及需要处理的消息。

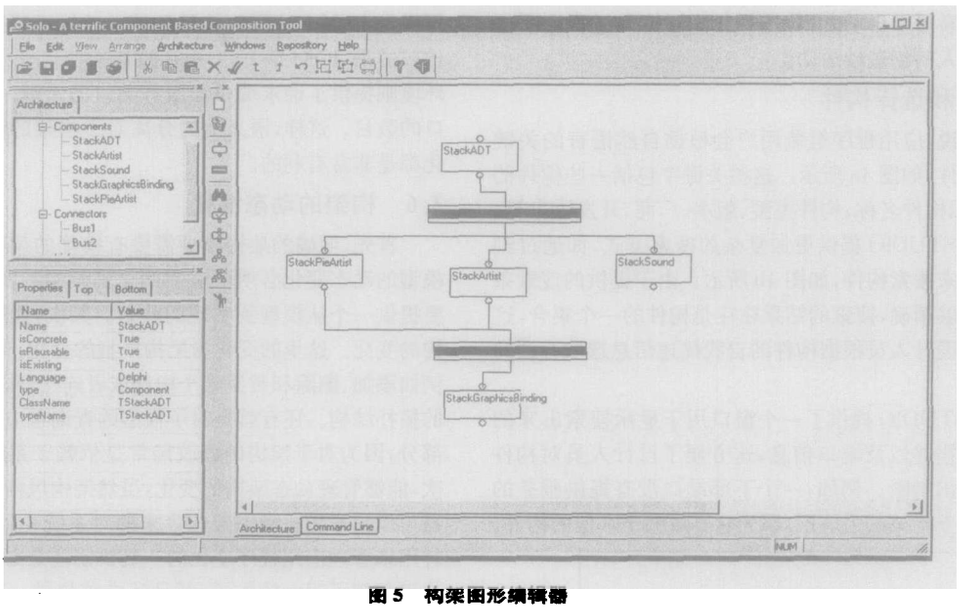


图 5　构架图形编辑器

3　实例研究

下面详细说明基于构件的开发工具 SOLOSTUDIO 的应用实例。

3.1　一个简单的计算器

该计算器包括三个构件：TCalculatorADT、TCalculatorArtist 和 TGrahicsBinding。其中，所有计算器中需要保留的数据结构和计算逻辑都保存在 TCalculatorADT 构件之中，但它并不负责用户界面的显示和用户事件的响应。这些功能全部放在 TGraphicsBinding 构件中，TCalculatorArtist 构件仅仅是保存用户界面布局信息的构件。整个系统的构架如图 6 所示。组装出来的实例如图 7 所示。

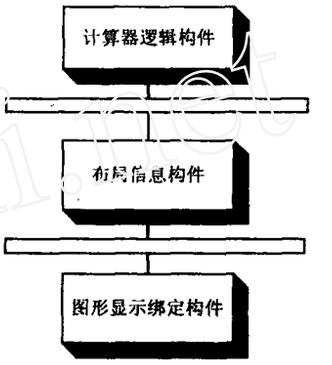


图 6　计算器的构架描述

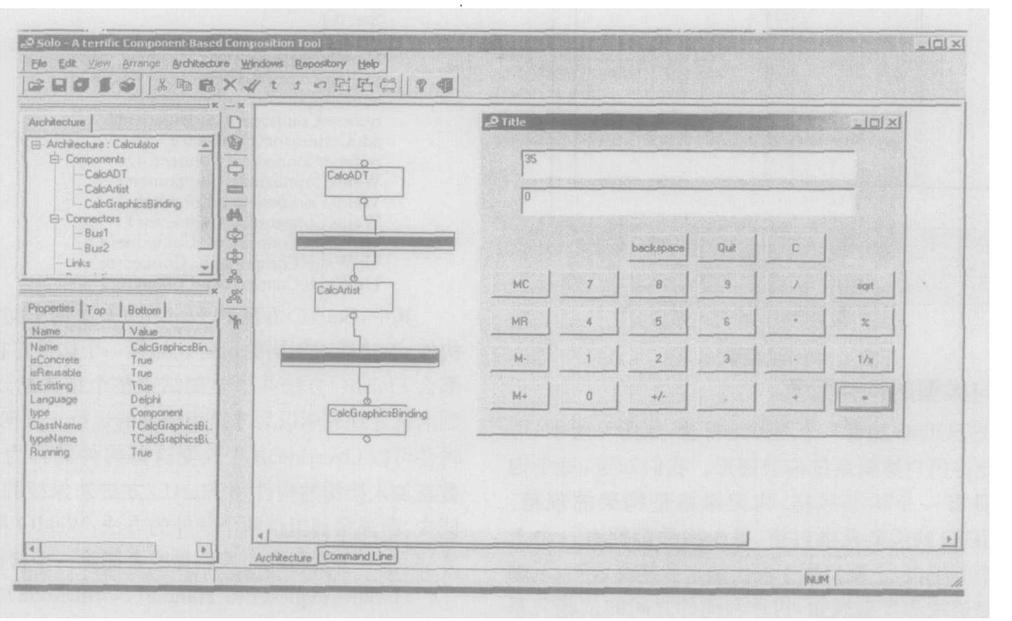


图 7　计算器的实例

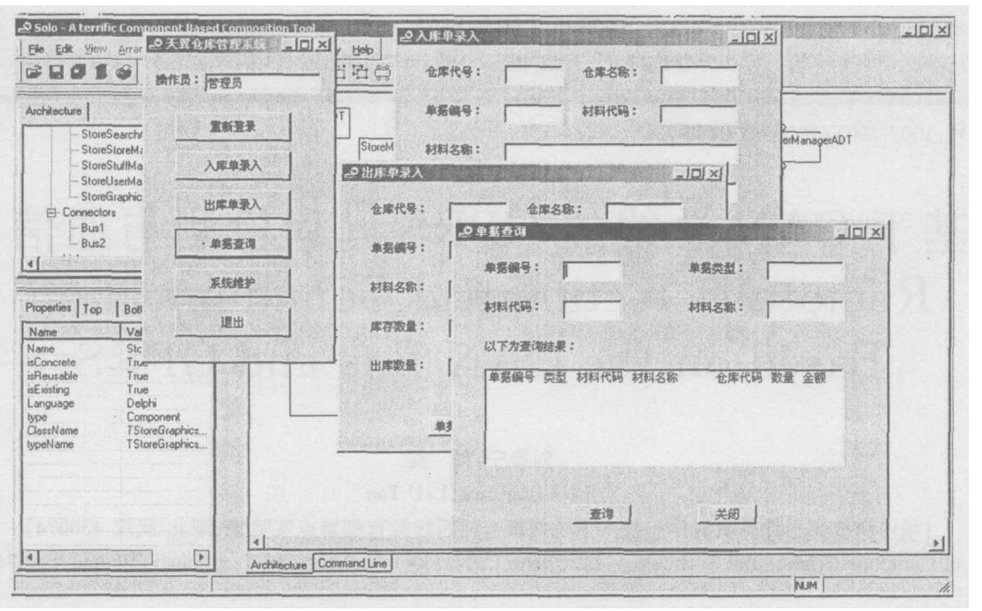


图 9　仓库管理系统的实例

3.2　仓库管理系统的实例

一个仓库管理系统是管理通用仓库的软件产品，其基本的功能包括用户登陆、产品入库、产品出库、库产品查询等。我们用构件 DataModule 来记录整个系统构架的内部数据结构，构件 Login 表示用户登陆，构件 InSheet、OutSheet、SqlGen、QueryResult 分别表示入库、出库和查询功能。这些构件都是 COM 构件，对外提供可以调用的接口，通过图 3 的包装器转化为面向事件的构件，这些构件通过连接器发送消息 Request、Notification 来相互作用。UI 是图形用户接口构件，它负责向计算机终端输出图形界面，完成和用户的交互。图 8 显示了该仓库管理系统的构架全貌。

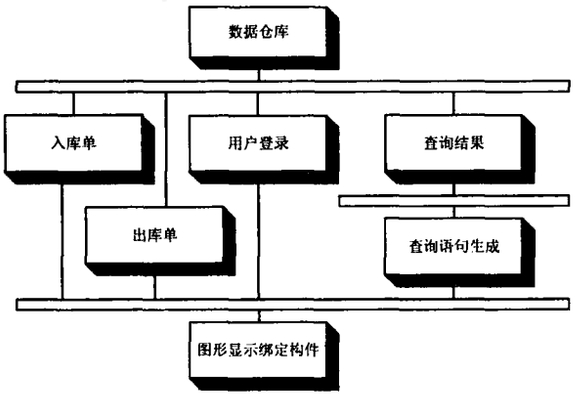


图 8　仓库管理系统的构架描述

实现了各个 COM 构件并且加入到构件库中，通过提供的刻面信息在构件库检索到所需要的构件，在 SOLOSTUDIO 构架组装平台上按照图 8 的方式组装完毕，自动生成系统。组装的系统如图 9 所示。

用户完成组装以后，点击登录按钮，构架内部会发送一个请求 Request 消息到 Login 组件，Login 组件接受到用户输入的登录信息以后，发送一个请求 Reqeust 消息到 DataModule 组件，由它负责修改整个构架的状态，并且保存和验证用户信息，接着发送一个通知 Notify 消息到下层。下层各个构件在接受到通知 Notify 消息之后得知登陆成功并且可以访问数据库，接着用户可以提出入库、出库、查询等请求。所有这些请求和通知都以消息的形式在 SOLOSTUDIO 的框架中传递。

4　结束语

本文讨论如何设计一个基于构件的软件开发环境，并且描述了一套原型系统 SOLOSTUDIO。该系统主要着眼于探索基于构件的开发方法以及 COM/DCOM、EJB 这些商用的组件对象模型标准与构架领域的整合问题，对于构件的搜索、下载、包装和组装过程进行有益的探索。这里提出今后的工作的方向，

(1) 继续研究商用对象组件模型，如 CORBA、COM/DCOM、EJB 等和领域构架的整合问题，目标在于实现这两个领域的无缝连接；

(2) 由于 Web Services 技术的出现，WSDL 代表构件接口的标准化，用 WSDL 代替我们自己的构件接口描述文件，实现对 Web Services 构件的包装，从根本上解决组装不同开发语言、异构环境下的构件的问题；

(3) 如何让 SOLOSTUDIO 和商业的开发环境如 Eclipse 等紧密地整合。

参考文献：

[1] 杨芙清，梅宏，李克勤，等. 支持构件复用的青鸟Ⅲ型系统概述[J]. 计算机科学，1999，26(5)：50-55.

[2] Richard N Taylor, Nenad Medvidovic, Kenneth M Anderson, et al. A Component- and Message-Based Architectural Style for GUI Software[J]. IEEE Trans on Software Engineering, 1996, 22(6): 390-406.

[3] D C Luckham, J Vera. An Event-Based Architecture Definition Language[J]. IEEE Trans on Software Engineering, 1995, 21(9): 717-734.

[4] Chris Luer, David S Rosenblum. An Environment for Component-Based Development [R]. Technical Report # 00-28, WREN, 1999.