Компонентно-базирани вградени системи

Васил Георгиев, ФМИ 110

v.georgiev@fmi.uni-sofia.bg
is.fmi.uni-sofia.bg

Съдържание

1. Компонентно-базирани технологии – приложимост, характеристики, подходи
2. Архитектура на компонентно-базираните системи
3. Области на приложение на компонентите
4. Компонентни ВАС
5. Проектиране и развитие на ВАС с компоненти
6. Специализирани компонентни за ВАС – Koala на Philips

Компонентно-базирани технологии

- Компонентите са по-високо ниво в йерархията на SW технологични елементи – след макросите, структурите и обектите –
  - за многократно ползване на кода
  - бързо и евтино внедряване и поддръжка
- Приложимост
  - Web-приложения
  - офис и информационни приложения, GUI, интерактивни и графично-интензивни приложения
  - телекомуникации, битова електроника, промишлена електроника, роботи

Компоненти

- Компонентите са заменими елементи на програмата, които (фиг. 4.4.1)
  - изпълняват определени пълен набор функции или логически операции
  - имат собствено затворено състояние без споделен достъп (за разлика от обектите) и дефиниран входен и изходен интерфейси
  - данни се обменят чрез съобщения – вкл. асинхронно
  - примери: бутони, логическо дърво, браузър
- Проектирането се разделя на
  - разработване и маркетинг на компонентни библиотеки (Domain Engineering; component development)
  - интегриране, адаптиране и верификация [по-сложна] на компонентни приложения (Component Based Development; system development)
- Видове (фиг. 4.4.2)
  - Commercial-of-the-Shelf (COTS) – непрозрачни – "черна кутия", обобщена функция, надеждността се подкрепя и от многократно внедряване
  - адаптивни/"бели"
  - "сиви" – компонентната библиотека осигурява език/API за настройка

Компонентна спецификация

- Компонентите се специфицират чрез техните интерфейси – IDL
- черна/сива/бяла/стъкл ена кутия (фиг. 4.5)
- Precondition – предикат върху входящите параметри
- Postcondition – предикат върху внодящите и изходящите параметри

```
interface ISpellCheck : IUnknown    {
                HRESULT check([in] BSTR *word, [out] bool *correct);
 };

interface ICustomSpellCheck : IUnknown   {
                HRESULT add([in] BSTR *word);
                HRESULT remove([in] BSTR *word);
};

library SpellCheckerLib   {
                coclass SpellChecker    {
                                [default] interface ISpellCheck;
                                interface ICustomSpellCheck;
                                                };
};
```


Компоненти – нефункционални черти

- при изпълнение (runtime)
  - производителност/време за отговор
  - надеждност
  - защита
  - безопасност за обкръжението
- при проектиране (lifecycle, минимизиране и оптимизиране на кода – продължително проектиране от порядъка на десетки човекомесеци)
  - приложимост
  - поддръжка
  - преносимост
  - тестване
  - адаптивност/настройка
  - заместимост/съвместимост
  - разширимост (на компонентните интерфейси)

Компонентна композиция

- динамично композиране – по време на изпълнение
  - изисква изпълнение върху специализирана изпълнителна платформа, която осъществява динамичното свързване на компонентите
- статично композиране – компонентно проектиране на монолитен фирмуер
  - оптимизиране на междукомпонентните референси (директни а не посредством изпълнителната среда)
  - по-практичен подход при ВАС

Компонентно проектиране

- по принцип проектирането за ВАС с компоненти с общо предназначение на дава оптимални резултати
- Елементи на компонентно-базираните технологии – фиг. 4.8
  - компонентна библиотека
  - среда за проектиране
  - изпълнителана платформа

Компонентна грануларност

- едра грануларнаст
  - некомпактни приложения, съдържащи неизползваеми области код
- фина грануларност
  - оптимизиране на изпълнимия код по ресурси
  - по-практичен подход при ВАС
- отношение към разширяването, награждането на системата и подмяна на компонентите в бъдеще

Компоненти за малки и големи ВАС

- малки ВАС – обикновено не са приложими компонентните платформи с общо предназаначение
- големи ВАС – по-важни са разходите за разработка и внедряване на продукта, разширимостта, приложението на стандартни подходи – приложимост на общи компоненти

Компонентни платформи

- MS DCOM/.NET, Sun EJB, OMG CCB и др.
  - поддържани черти: функционалност, гъвкавост, динамична адаптивност, упростено проектиране и поддръжка
  - неподдържани черти: производителност, ресурси, надеждност
- директно приложение на компонентни платформи
  - CORBA – телекомуникации)
  - COM/DCOM, .NET – промишлени роботи и контролери
- модифицирани компонентни модели – разширена и фокусирана функционалност
  - OPC (OLE process control Foundation)
- ограничен компонентен модел - напр. приложение само за спецификация на интерфейсите (IDL)

Компоненти и РВ-задания

- Пряк подход – 1:1
  - лесен анализ и проектиране
  - по-ниска ресурсна ефективност
- Етапи:
  - анализ и проектиране
    - свързване на задания с компоненти
    - алокация на атрибутите
    - спецификация на интерфейсите
  - компилация и свързване
    - генерация на свързващ код (напр. от IDL)
    - зареждане на компонентите
    - компилация за РВОС

Компонентен модел Koala

- платформа на Philips за ВАС в битова електроника
  - ADL (Architecture Definition Language) – с явен графичен синтаксис – фиг. 4.13
  - статично свързване при компилация – без динамична интерпретация – за бързодействащи и компактни приложения
  - предназначен за генерации от сходни продукти (диверсификация, осъвършенстване, наследственост, ресурсни ограничения, автономност)

Koala – черти

- логическите връзки между компонентите са определени статично (свързване при проектирането и се осъществяват чрез компилация)
- възможност за надстройка на компонентите – при по-високи ресурсни възможности на продукта– фиг. 4.14
  - за тази цел от компонентите се отделят provide и require интерфейсни примитиви, които имплементират "универсални" интерфейси (спецификации на IDL)

Koala – компонентен модел

- достъпа на компонентите да външни функции е чрез requires интерфейси
- описанието на компонентите включва описание на функциите и на интерфейсите им с текстов CDL (Component Definition Language, C-ситаксис)

```
component CTunerDriver       {
provides  ITuner ptun;
          IInit pini;
requires  II2c ri2c;         }
```

- всеки интерфейси се описва с две имена – [глобален] тип (напр. `ITuner`) и името на конкретна инстанция от библиотеката интерфейси (напр. `ptun`)
  - това позволява повече от една инстанция за даден тип интерфейс на компонента – напр. различни интерфейси-контроли на силата на звука за тонколоните и за слушалките на дадено мултимедийно устройство

Koala – интерфейсен модел и конфигурация

- всеки requires интерфес се свързва към точно един provides интерфейс; provides интерфейсите са свързани към 0 или повече requires интерфейси
- компонентите се характеризират с име и тип
- интерфейсните връзки са от съвпадащи типове
- всеки Koala-продукт е съставен компонент – конфигурция от множество компоненти (тип. десетки) и свързването на интерфейсите им

```
component CTvPlatform                          {
          provides IProgram pprg;
          requires II2c slow, fast;
          contains
                   component CFrontEnd cfre;
                   component CTunerDriver ctun;
          connects
                   pprg = cfre.pprg;
                   cfre.rtun = ctun.ptun;
                   ctun.ri2c = fast;           }
```

- вложените компоненти се характеризират с име-инстанция (локален обхват) и тип (глобален обхват) – заменимост и модулност

Koala – функционално свързване

- при свързване на функции (компоненти) с интерфейси от различен тип – вместо да се кодират свързващи модули на C, които забавят проектирането на системата и изпълнението на кода – се използват модули за функционално свързване – фиг. 4.17.1

```
within m {
cfre.rtun.SetFrequency(x) = ctun.ptun.SetFrequency(x);
cfre.rtun.GetFrequency() = ctun.ptun.GetFrequency();
cfre.rtun.EnableOutput(x) = chip.pout.EnableOutput(x);
}
```

- изборно изпълнение на функции, ключове (4.17.2):

```
1?f(x):g(x);   /*  -> f(x)
```