

Verificación de Sistemas Reactivos

Arquitecturas de software de sistemas reactivos

Alejandro Sánchez

Departamento de Informática
Universidad Nacional de San Luis

Maestría en Ingeniería de Software
Maestría en Calidad del Software
Especialización en Ingeniería de Software
25-26 Octubre 2013

Motivación I

Problemas de diseño arquitectónico

- Coordinación de servicios concurrentes (sistema reactivo)
- Asegurar propiedades (modales, espaciales, ...)
- Auto-adaptabilidad (run-time)
- Evolución (design-time)

Motivación I

Problemas de diseño arquitectónico

- Coordinación de servicios concurrentes (sistema reactivo)
- Asegurar propiedades (modales, espaciales, ...)
- Auto-adaptabilidad (run-time)
- Evolución (design-time)

Enfoque informal: Patrones arquitectónicos –
soluciones conocidas a problemas de diseño recurrentes

Problemas de diseño arquitectónico

- Coordinación de servicios concurrentes (sistema reactivo)
- Asegurar propiedades (modales, espaciales, ...)
- Auto-adaptabilidad (run-time)
- Evolución (design-time)

Enfoque informal: Patrones arquitectónicos –
soluciones conocidas a problemas de diseño recurrentes

Desventaja del enfoque:

Sin herramientas para
modelar, especificar, analizar y verificar
sobre las posibles soluciones

Motivación II

Utilizar framework de sistemas reactivos
(LTSs, álgebras de procesos, lógicas modales, ...)

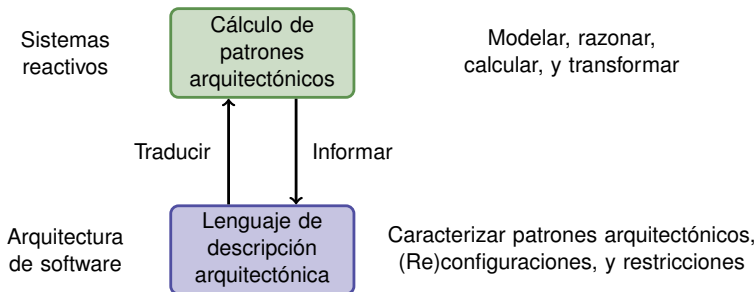
Motivación II

Utilizar framework de sistemas reactivos
(LTSs, álgebras de procesos, lógicas modales, ...)

El arquitecto de software trabaja
en un nivel de abstracción y con un lenguaje de dominio
distintos al de los sistemas reactivos

Motivación III

Modelar y razonar sobre la estructura y comportamiento de patrones arquitectónicos



Estructura de la presentación

- 1 Arquitectura de Software
 - Contexto histórico
 - Caracterización
 - Patrones arquitectónicos
 - Lenguajes de descripción arquitectónica
- 2 Archery - Un lenguaje de descripción arquitectónica
- 3 Archery - Semántica de comportamiento
 - Breve repaso de mCRL2
 - Traducción a mCRL2

Estructura de la presentación

- 1 Arquitectura de Software
 - Contexto histórico
 - Caracterización
 - Patrones arquitectónicos
 - Lenguajes de descripción arquitectónica
- 2 Archery - Un lenguaje de descripción arquitectónica
- 3 Archery - Semántica de comportamiento
 - Breve repaso de mCRL2
 - Traducción a mCRL2

Breve reseña histórica

Programación/implementación vs. diseño ('70)

Diseño vs. arquitectura ('80)

Abstracción que revoluciona la ingeniería del software
ciclo de vida del software
estudio de factibilidad - mantenimiento

¿Qué es una arquitectura de software? I

Más de 150 definiciones en SEI-CMU site (y contando)
Debido al nivel de madurez de la disciplina y su diversidad

La estructura de los componentes de un programa/sistema, sus inter-relaciones, y principios y pautas gobernando su diseño y evolución en el tiempo.

Garlan and Perry, 1995
IEEE Transactions on Software Engineering

¿Qué es una arquitectura de software? II

La organización fundamental de un sistema materializada en sus componentes, las relaciones entre estos y con su ambiente, y los principios guiando su diseño y evolución.

IEEE 1471 2000

¿Qué es una arquitectura de software? III

El conjunto de decisiones de diseño principales
que gobiernan un sistema.

Taylor, Medvidovic, and Dashofy. 2009
Software Architecture: Foundations, Theory and Practice. Addison Wesley

¿Qué decisiones son no arquitectónicas?

Para responder a esta pregunta, se retorna al
propósito primario de la arquitectura:
asegurar la satisfacción de los objetivos de negocio,
la calidad, y los requerimientos de comportamiento del sistema.

Clements et al. 2011
Documenting Software Architectures. Addison Wesley

Caracterización de los patrones arquitectónicos

Un patrón arquitectónico es una descripción de tipos de elementos y configuraciones, y un conjunto de restricciones sobre como usarlos.

Bass, Clements, Kazman. 2003
Software Architecture in Practice. Addison Wesley

Luego de modificar la configuración

- ¿Cómo se relacionan las configuraciones?
¿son equivalentes? ¿son refinamientos?
- ¿Se verifican todavía las restricciones del patrón arquitectónico?
- ¿Qué otras restricciones son verificadas?
- ¿Podemos simular y prevenir modificaciones infractoras?

Caracterización I

Lenguaje de descripción arquitectónica (ADL)

Principales abstracciones

- **Componente.**
 - Encapsula funcionalidad y/o datos
 - Restringe acceso a través de interfaz
 - Tiene dependencias explícitas de su contexto requerido
- **Conector.** Elemento arquitectónico a cargo de la tarea de efectuar y regular las comunicaciones entre componentes.
- **Configuración.** Conjunto de asociaciones específicas entre los componentes y conectores de la arquitectura de un sistema.

Caracterización II

Lenguaje de descripción arquitectónica (ADL)

Enfoques para soportar patrones arquitectónicos

- Tipos de elementos
- Restricciones
 - Satisfacción por construcción.
Operaciones de reconfiguración específicas del patrón y restricciones implícitas
 - Satisfacción por verificación.
Operaciones genéricas de reconfiguración y restricciones explícitas a verificar

Estructura de la presentación

- 1 Arquitectura de Software
 - Contexto histórico
 - Caracterización
 - Patrones arquitectónicos
 - Lenguajes de descripción arquitectónica
- 2 Archery - Un lenguaje de descripción arquitectónica
- 3 Archery - Semántica de comportamiento
 - Breve repaso de mCRL2
 - Traducción a mCRL2

Principales características

Archery:

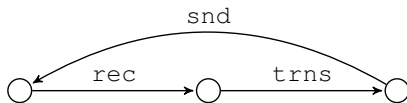
- Lenguaje de descripción arquitectónica
- La semántica de comportamiento esta dada por una traducción a mCRL2
- Soporta patrones arquitectónicos con un enfoque de satisfacción de restricciones por verificación

Patrones

- Un **patrón** comprende un conjunto de elementos
- Actúan como **tipos**

```

pattern PipeFilter()
element Pipe()
  act acc, fwd;
  proc Pipe() = acc.fwd.Pipe();
  interface in acc; out fwd;
element Filter()
  act rec, trns, snd;
  proc Filter() =
    rec.trns.snd.Filter();
  interface in rec; out snd;
end
  
```



Elementos

- Modelan componentes o conectores
- Actúan como **tipos**

```

pattern PipeFilter()
element Pipe()
    act acc, fwd;
    proc Pipe() = acc.fwd.Pipe();
    interface in acc; out fwd;
element Filter()
    act rec, trns, snd;
    proc Filter() =
        rec.trns.snd.Filter();
    interface in rec; out snd;
end
  
```

- Cada **elemento** tiene un **comportamiento** asociado
 - mCRL2 Act y Proc
 - Inicia en primer proceso
 - Puertos in/out:
 - flujo de agua
 - $p.q$, $p + q$, $c \rightarrow p \diamond q$
 - sin sumatoria

Instancias

Una **arquitectura** es una **instancia** de un patrón

- describe una **configuración** de instancias de elementos
- puede ser **jerárquicamente compuesta**
- tiene un comportamiento **emergente**

```
pf : PipeFilter =  
  architecture PipeFilter()  
  instances  
    f1 : Filter = Filter();  
    f2 : Filter = Filter();  
    p1 : Pipe = Pipe();  
  attachments  
    from f1.snd to p1.acc;  
    from p1.fwd to f2.rec;  
  interface  
    f1.rec as rreq;  
    f2.snd as sres;  
end
```

Estructura de la presentación

- 1 Arquitectura de Software
 - Contexto histórico
 - Caracterización
 - Patrones arquitectónicos
 - Lenguajes de descripción arquitectónica
- 2 Archery - Un lenguaje de descripción arquitectónica
- 3 Archery - Semántica de comportamiento
 - Breve repaso de mCRL2
 - Traducción a mCRL2

Secciones de especificación

- Behaviour
 - Acciones `act`
 - Procesos `proc`
 - Inicialización `init`
- Data
 -

Procesos secuenciales

Dados procesos p y q , expresiones booleanas c , y expresiones de datos d de tipo D

- **Acciones:** $a, b(d)$
- **Composición secuencial:** $p.q$
- **Composición alternativa:** $p + q$
- **Recursión:** $p \triangleq q.p, \text{proc } p = q.p;$
- **Condicional:** $c \rightarrow p \diamond q, (c) \rightarrow p <> q$
- **Sumatoria:** $\sum_{d:D} p(d), \text{sum } d:D. p(d)$

Procesos concurrentes

- **Multi-acción:** $a|b$ or $a(d)|b(d)$
- **Composición paralela:** $p \parallel q$
- **Comunicación:** $\Gamma_C(p)$, $\text{comm}(C, p)$
con C un conjunto de $a_1|..|a_n \rightarrow b$
- **Permitir:** $\nabla_V(p)$, $\text{allow}(V, p)$
con V un conjunto de multi-acciones
- **Renombrar:** ρ_R , $\text{rename}(R)$
con R un conjunto de renombrado $a \rightarrow b$
- **Ocultar:** τ_I , $\text{hide}(I)$
con I un conjunto de acciones

Ejemplo pipes and filters

```

pattern PipeFilter()
element Pipe()
  act acc,fwd;
  proc Pipe()=
    acc.fwd.Pipe();
  interface
    in acc; out fwd;
element Filter()
  act rec,trns,snd;
  proc Filter()=
    rec.trns.snd.Filter();
  interface
    in rec; out snd;
end

```

```

pf : PipeFilter =
  architecture PipeFilter()
instances
  f1 : Filter = Filter();
  f2 : Filter = Filter();
  f3 : Filter = Filter();
  p1 : Pipe = Pipe();
attachments
  from f1.snd to p1.acc;
  from p1.fwd to f2.rec;
  from p1.fwd to f3.rec;
interface
  f1.rec as rreq;
  f2.snd as sres;
end

```

Instancias de elementos

- Los elementos se usan como plantillas
- Las acciones se hacen únicas
- Los puertos dependen de las conexiones

```
proc
  Filter_f1 = rec_f1.trns_f1.snd_f1_acc_p1.Filter_f1;
  Filter_f1_init = Filter_f1;
  Filter_f2 = rec_f2_fwd_p1.trns_f2.snd_f2.Filter_f2;
  Filter_f2_init = Filter_f2;
  Filter_f3 = rec_f3_fwd_p1.trns_f3.snd_f3.Filter_f3;
  Filter_f3_init = Filter_f3;
  Pipe_p1 = acc_p1_snd_f1.(
    fwd_p1_rec_f2+fwd_p1_rec_f2|fwd_p1_rec_f3+fwd_p1_rec_f3
  ).Pipe_p1;
  Pipe_p1_init = Pipe_p1;
```

Procesos concurrentes I

$$\tau_H(\rho_R(\nabla_V(\Gamma_C(\prod_{i \in \text{instances}} p_i))))$$

- C : reglas de comunicación (attach.)
- V : multi-acciones permitidas
 - acciones de sincronización de C
 - puertos en la interfaz (a ser renombrados)
 - V_{arch} del conjunto de instancias anidadas
 - V_{einst} acciones no puerto
- R : renombre de cada acción generada en un puerto
- H : acciones no puerto

Procesos concurrentes II

```
init
rename ({rec_f1 -> rreq_pf, snd_f2 -> sres_pf},
  allow ({rec_f1, snd_f2, snd_f3, synch_fwd_p1_rec_f2,
    synch_fwd_p1_rec_f3, synch_snd_f1_acc_p1,
    trns_f1, trns_f2, trns_f3},
    comm ({
      fwd_p1_rec_f2|rec_f2_fwd_p1 -> synch_fwd_p1_rec_f2,
      snd_f1_acc_p1|acc_p1_snd_f1 -> synch_snd_f1_acc_p1,
      fwd_p1_rec_f3|rec_f3_fwd_p1 -> synch_fwd_p1_rec_f3},
      Filter_f2_init || Filter_f3_init ||
      Filter_f1_init || Pipe_p1_init
    )
  )
);
```

Animación de la arquitectura pipes and filters

LpsXSim

Transitions

| Action | State Change |
|---------------------|------------------------------------|
| synch_fwd_p1_rec_f2 | s3_Filter_f2 := 2, s4_Pipe_p2 :... |
| synch_fwd_p1_rec_f3 | s1_Filter_f1 := 2, s4_Pipe_p2 :... |
| rreq_pf | s2_Filter_f2 := 2 |

Trace

| # | Action | State Change |
|---|---------------------|--------------------------|
| 0 | | s3_Filter_f2 := 1, s1... |
| 1 | rreq_pf | s2_Filter_f2 := 2 |
| 2 | trns_f1 | s2_Filter_f2 := 3 |
| 3 | synch_snd_f1_acc_p1 | s2_Filter_f2 := 1, s4... |

Current State

| Parameter | Value |
|--------------|-------|
| s3_Filter_f2 | 1 |
| s1_Filter_f1 | 1 |
| s2_Filter_f2 | 1 |
| s4_Pipe_p2 | 2 |