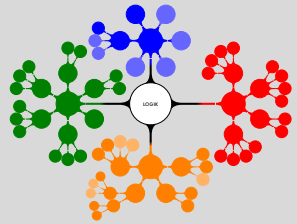
Tema 1

UniCon: Language for Universal Connector Support

ADL “UniCon”

Asignatura *Desarrollo de Software* fecha 15 marzo 2016

Manuel I. Capel
Departamento de Lenguajes y Sistemas Informaticos
Universidad de Granada



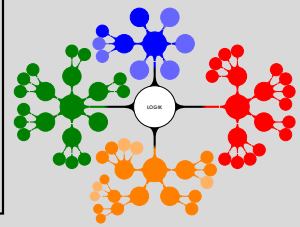
Contexto

Se pretende describir un lenguaje arquitectónico (ADL)
con configuración del sistema e “independencia entidades”.
Permite llevar a cabo el análisis de diferentes propiedades:
funcionalidad, seguridad, confiabilidad, etc.
dadas su facilidades para la abstracción de componentes.

- *Idiomas de abstracción*: ofrece tipos específicos utilizados por diseñadores.
- *Propiedades funcionales de componentes y programas*: distingue entre *filtros* y *métodos*, por ejemplo.
- *Elemento sintáctico conector*: informa de las interacciones entre componentes.
- *Función de abstracción*: correspondencia entre código y construcciones de alto nivel.
- *Abierto para incluir constructos sintácticos externos al propio lenguaje*

Artículo completo de M. Shaw

`http://research.microsoft.com/en-us/um/people/rdeline/publications/unicon-tse-95.pdf`



Estructura general del lenguaje UniCon

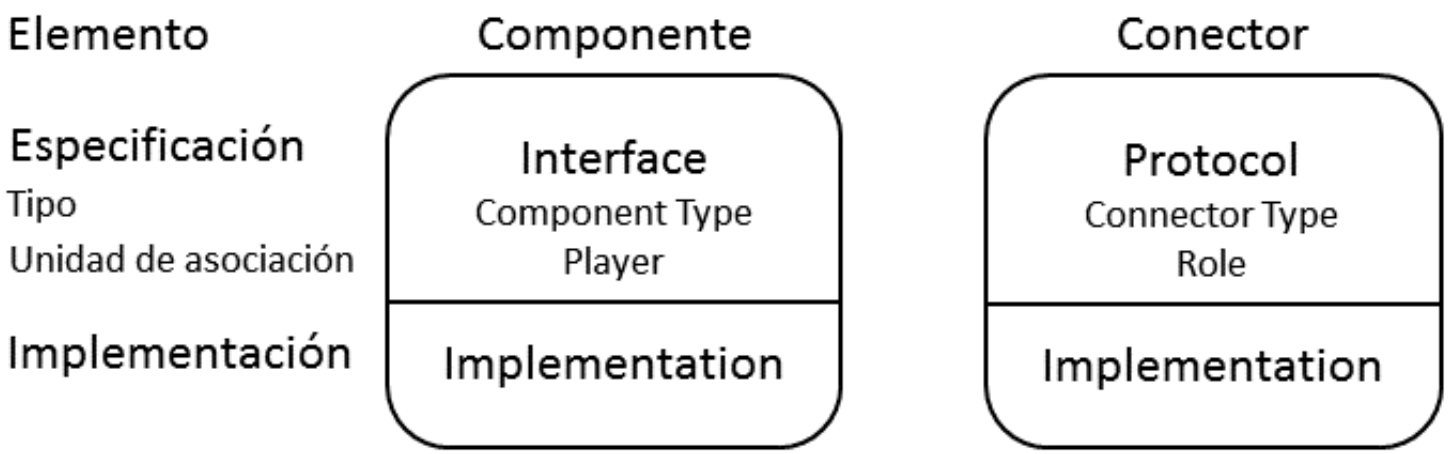
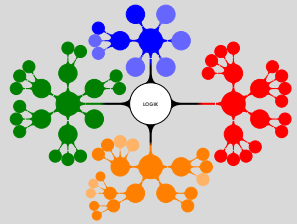
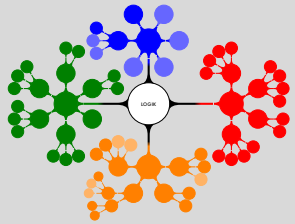


Figure: Estructura general de un lenguaje de descripción arquitectónico (ADL)



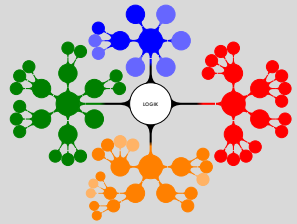
Función de los componentes en UniCon

- Correspondencia con *unidades de compilación* de un lenguaje imperativo.
- La **Interfaz** define los *compromisos computacionales* que puede realizar y las *restricciones*, que indican sobre su utilización.
- Especificación de **Interfaz de Componente**:
  - Tipo o subtipo del componente
  - Funcionalidad
  - Garantías del cumplimiento de invariantes globales
- Información acerca del rendimiento.



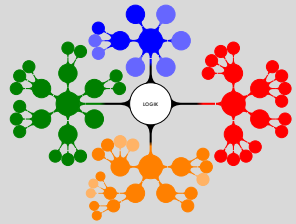
Sintaxis de los componentes en UniCon: Players

- Entidades con nombre que aparecen en las interfaces de los componentes
- Player es una entidad con tipo:
  - Tipo o subtipo del componente
  - Atributos opcionales: *signatura*, *especificaciones funcionales*, *restricciones de uso*, *información específica* para el componente
  - Garantías del cumplimiento de invariantes globales
- Son unidades semánticas a través de las cuales el componente puede interaccionar
- "Property lists" :$\{atributos, valores - -asociados\}$
- Los players de una interfaz están condicionados por su entorno



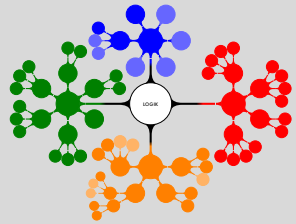
Ejemplo de especificación de componente

```
COMPONENT KKKK
  INTERFACE IS
    TYPE Filter
    PLAYER input IS StreamIn
    SIGNATURE("line")
    PORTBINDING(stdin)
    END input
    PLAYER output IS StreamOut
    SIGNATURE("line")
    PORTBINDING(stdout)
    END input
    PLAYER error IS StreamOut
    SIGNATURE("line")
    PORTBINDING(stderr)
    END input
```

Ejemplo de especificación de componente-II

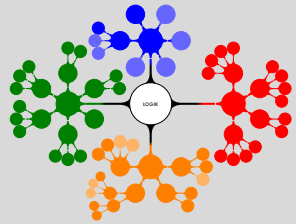
```
COMPONENT Stack
  INTERFACE IS
    TYPE Computation
    PLAYER apilamiento IS PLBundle
      MEMBER(init_stack; RoutineDef; SIGNATURE(;"void"))
      MEMBER(stack_is_empty; RoutineDef; SIGNATURE(;"int")
          )
      MEMBER(push; RoutineDef; SIGNATURE("char␣*"; void))
      MEMBER(pop; RoutineDef; SIGNATURE("char␣**"; void))
    END apilamiento
    ...
  END INTERFACE
```

Implementación de componentes

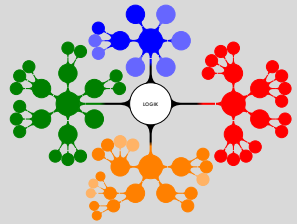
- Implementaciones *primitivas*: formas de localizar una definición en un lenguaje de programación
- Implementaciones *compuestas*: instancian un conjunto de componentes y los configuran con conectores
- Concepto de *variante* en la implementación de componente
- Implementación mediante “wrappers"

```
IMPLEMENTATION IS
  VARIANT stack IN "stack.c"
  IMPLTYPE(source)
  END stack
END IMPLEMENTATION
```

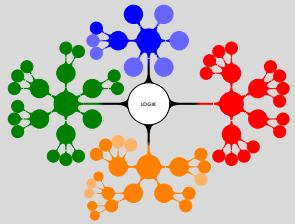
Ejemplo de implementación compuesta

```
IMPLEMENTATION IS
  USES rev INTERFACE reverse
  USES stk INTERFACE stack
  USES lib INTERFACE libc
  BIND input TO ABSTRACTION
    MAPSTO (rev.libc.fgets)
  END input
  BIND output TO ABSTRACTION
    MAPSTO (rev.libc.fprintf)
  END output
 ...
 ESTABLISH C-PLBundler WITH
   rev.stackness AS participant
   stk.stackness AS participant
   MATCH ((rev.stackness.new,
    stk.stackness.init_stack),(rev.stackness.no_more, stk.
        stackness.stack_is_empty), (rev.stackness.stash,
stk.stackness.push), (rev.stackness.deliver,
stk.stackness.pop))
 END C-PLBundler
 ...
END IMPLEMENTATION
```

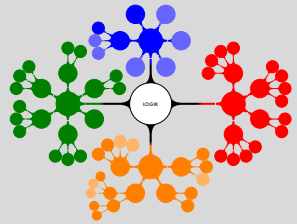
Función de los conectores en UniCon

- Definición de las relaciones de conexión entre los componentes
- No son *cosas para conectar* los componentes entre sí
- Un conector consiste en un *protocolo* y una *implementación*
- Proporcionan las reglas de las conexiones
- Para cada conector se especifica un *protocolo*:
  - Tipo o subtipo (p.e.: RPC, broadcast, encauzamiento, etc.)
  - Reglas que cumplen los tipos
  - Asertos que restringen al conector completo (temporalización y ordenamiento)
  - Seguridad y compromisos que han de respetarse en la interacción



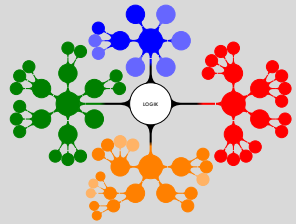
Sintaxis de los conectores en UniCon: Roles

- Entidades con nombre que aparecen en los *protocolos* de los conectores
- Un *Rol* es una entidad con tipo:
  - Tipo o subtipo de conector
  - Atributos opcionales: *signatura*, *especificación funcional*, *restricciones* de uso
  - Garantías del cumplimiento de invariantes globales
- Son unidades semánticas a través de las cuales el conector actúa como intermediario
- Los tipos suelen ser primitivos y sirven para identificar a los players que actúan
- *Property lists*: especificaciones detalladas de los roles



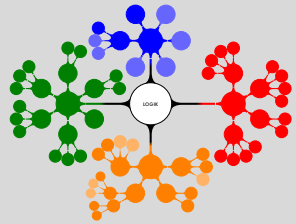
Ejemplo de especificación de conector

```
CONNECTOR KKKK
  PROTOCOL IS
    TYPE Pipe
    ROLE source IS source
    MAXCONNS(1)
    END source
    ROLE sink IS sink
    PORTBINDING(1)
    END sink
  END PROTOCOL
```

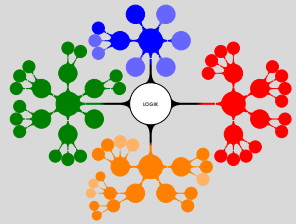
Ejemplo de especificación de conector–II

```
CONNECTOR C-PLBundler
  PROTOCOL IS
  TYPE PLBUndler
  ROLE participant IS participant
  END PROTOCOL
/*******/
  IMPLEMENTATION IS
    BUILTIN
  END IMPLEMENTATION
  END C-PLBundler

/*******/
  IMPLEMENTATION IS
    BUILTIN
  END IMPLEMENTATION
  END C-Proc-call
```

Ejemplo de especificación de conector–II

```
CONNECTOR RTM-realtime-sched
PROTOCOL IS
TYPE RTScheduler
ROLE load IS load
END PROTOCOL
IMPLEMENTATION IS
BUILTIN
END IMPLEMENTATION
END RTM-realtime-sched

/*************/
CONNECTOR RTM-remote-proc-call
PROTOCOL IS
TYPE RemoteProcCall
ROLE definer IS definer
ROLE caller IS caller
END PROTOCOL
IMPLEMENTATION IS
BUILTIN
END IMPLEMENTATION
END RTM-remote-proc-call
```

Ejemplo: Implementación heterogénea de un encauzamiento-I

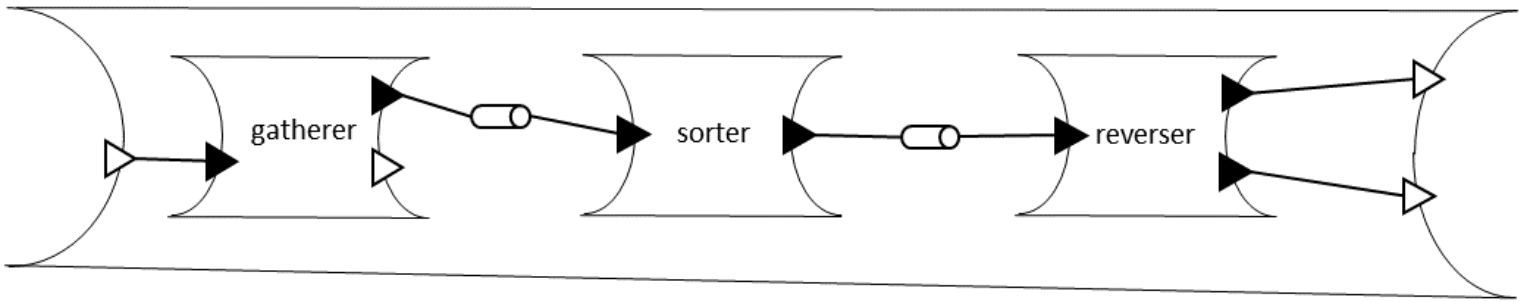
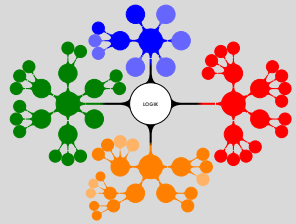


Figure: Vista de alto nivel de la arquitectura “Encauzamiento”



Ejemplo: Implementación heterogénea de un encauzamiento-II

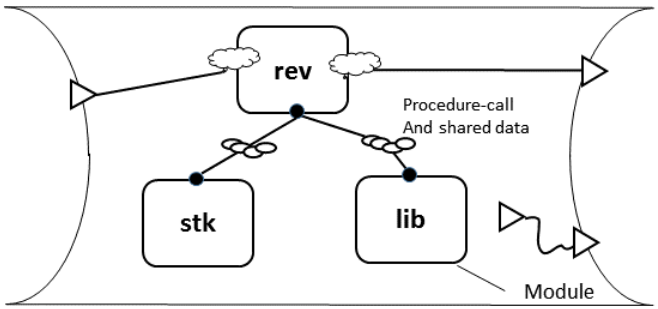
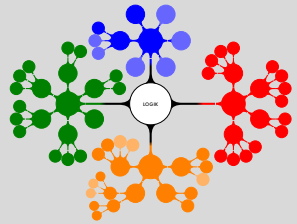


Figure: Implementación del componente “reverser”)



Vista de alto nivel de un “encauzamiento” (KWIC)

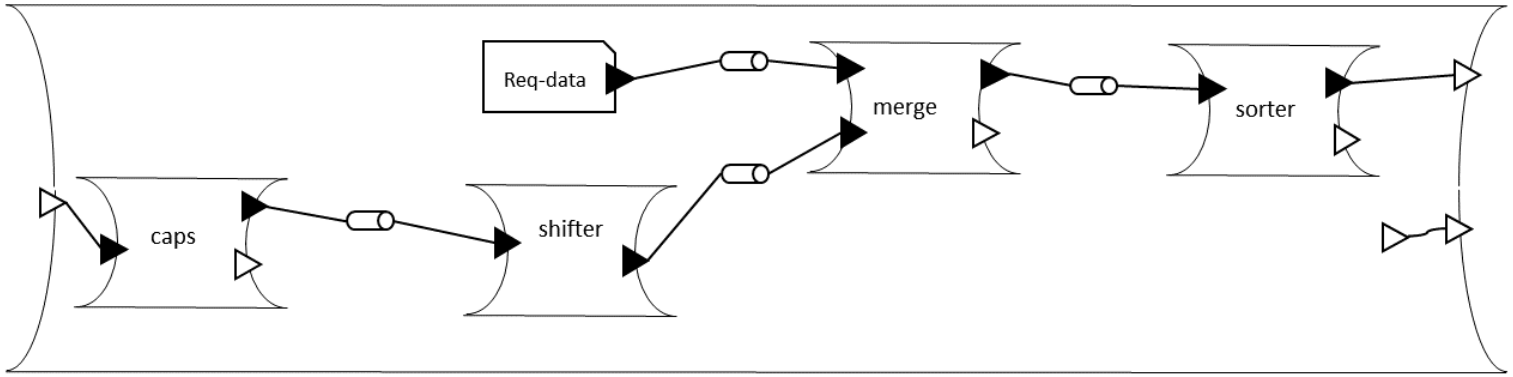


Figure: Arquitectura de la herramienta KWIC