Data Models for Conceptual Structures

Roger T. Hartley
Heather D. Pfeiffer
Basic CS Graph
Definitional CP Graph

AGT -> CATCH

PERSON -> AGT
PERSON -> POSS

PTNT -> BALL

BALL -> POSS
CP Overlay Graph
Complete CP Procedural Graph
Small Example

- CS Graph example
- Simple CP overlay examples
- CP overlay across Definition Graphs
- CP Model
PersonBirth is Graph

- Person
- BirthDate
- CHRC
- Birth
- PTIM
- Date
Person is DefGraph

- Person
  - CHRC
    - Age
  - CHRC
  - Birth
    - PTIM
      - Date
BirthDate is OvGraph
Now is DefGraph
ComputeNow is OvGraph

Now

Date

ComputeNow
Age is OvGraph
CurrentAge is PartModel
CGIF for Conceptual Structures

- **CG**
  
  \[ CG ::= (Concept | Relation | Actor | SpecialContext | Comment)* \]

- **Concept**
  
  \[ Concept ::= "[" Type(1)? \{CorefLinks?, Referent?\} Comment? "]" \]

- **Relation**
  
  \[ Relation ::= "(" Type(N) Arc* Comment? ")" \]

- **Actor**
  
  \[ Actor ::= "<" Type(N) Arc* "|" Arc* Comment? ">" \]

- **SpecialContext**
  
  \[ SpecialContext ::= Negation | "[" SpecialConLabel ":"CG "]" \]

- **Comment**
  
  \[ Comment ::= DelimitedStr(";") \]
Data Model

ADTs

- Definition of types and structures
- Operations on those types
DTD Structure

<!ELEMENT cg (concept | relation | actor | specialcontext | cgcomment)*)>
<!ELEMENT concept (contypelabel?, (coreflinks | referent | ((coreflinks, referent) | (referent, coreflinks)))?, concomment?)>
<!ELEMENT relation (reltypelabel, arc*, relationcomment?)>
<!ELEMENT actor (reltypelabel, arc*, actorcomment?)>
<!ELEMENT specialcontext (negation | (specialconlabel, cg))>
<!ELEMENT cgcomment (#PCDATA)>
Creation of Data Models

• Haskell Language
• By Hand – using XmlSpy
Haskell Data Model
(Basic CS Constructs)

type CG = ([CNode], [RNode])
type Label = String
data CNode = Concept Label Referent
data RNode = Relation Label InArcs OutArc
type InArcs = [CNode]
type OutArc = CNode
data Referent = Nil | Literal Literal | Graph CG
data Literal = LitString String | Name String | Marker String
Haskell Simple Example

let sit = Concept “Sit” Nil in
        ([], [Relation “AGT” [sit] Concept “Cat” Literal Name “Fred”,
              Relation “LOC” [sit] Concept “Mat” Nil])
Haskell Data Model
(Add Co-references)

type CG = ([CNode], [RNode])
type Label = String
type CoRef = String
data CNode = Concept Label Referent |
          DefConcept Label CoRef Referent |
          BoundConcept CoRef
data RNode = Relation Label InArcs OutArc
type InArcs = [CNode]
type OutArc = CNode
data Referent = Nil | Literal Literal | Graph CG
data Literal = LitString String | Name String | Marker String
Haskell Example
(With Co-references)

([], [Relation “AGT”
    [DefConcept “Sit” “x” Nil]
    Concept “Cat” Literal Name “Fred”,
    Relation “LOC”
    [BoundConcept “x”]
    Concept “Mat” Nil])
Haskell Grammar
(Part 1)

CG : Node
    | Node CG

Node : Relation
    | Concept
    | Actor
    | Negation
Haskell Grammar
(Part 2)

Relation: '(' TypeExp Arcs ')

Actor   : '<' id Arcs '|' Arcs '>

Negation: '~' '[' CG ']

Concept : '[' TypeExp ':' Referent ']
          | '[' TypeExp '*' id ':' Referent ']
          | '[' TypeExp ']' 
          | '[' TypeExp '*' id ']'
          | '[' ']' 
          | '[' ':' Referent ']'
Graph Data Model Types

• Pointer Type
• Adjacency List Type
• Adjacency Matrix Type
Haskell XML Schema
Haskell Concept XML Schema
Haskell Concept Attribute

- Name: CoRef
- Type: xs:string
- Use: optional
Haskell
Relation
XML Schema
Haskell Actor XML Schema
Graph Tuple
XML Schema
Graph
Tuple
Concept
XML
Schema
Graph Tuple Concept Attribute

- Name: uniquecon
- Type: xs:ID
- Use: required
Graph RCC Tuple
XML Schema

RCC-tupleType

relation

typelabel

Relation Node - gives a conceptual relationship between concept types

0..∞

c-posin-out

concept

concept list in pairs with concepts in and out, does not have to have a concept in, only out

1..∞

concept

Concept Node - gives the conceptual type information

Generated with XMLSpy Schema Editor www.xmlspy.com