Graph Based Programming

Enhancing Prototype Based Programming
Outline

• Motivation

• Three pillars of graph based software development

• Past work

• Graph based programming

• Future work
Motivation

• Hypergraphs (Topic Maps, RDF, ..) from area of knowledge representation

• Interest begun in SEP and Diplomarbeit

• Regarding hypergraphs as general and strong data model

• Idea: Translate principles from other data models into graphs and add specific semantic there
The three pillars

• **Applications** use graph based data models (learn from relational databases, xml, ..)

• **Software engineering** process uses graph based artefact management (learn from models, processes, UML, IDEs, ..)

• Programmers use graph based **programming languages** (learn from object oriented programming, AOP, functional p., rule-based p., ..)
Related work

• Chair:
  • Sysiphus, REQuest, RAT (artefact management)

• Other:
  • Areas: knowledge representation, programming languages, software engineering, ...
Past work with hypergraphs

- Associative glossary (SEP 2001)
- Non typed visual database (DA 2002)
- Secure knowledge exchange (Medusa 2003)
- Configurable data models (SE Prakt 2003)

⇒ Graph based applications
**Associative Glossary**

**Hagel**


**Niederschlag**

_Abhaengigkeiten_

_Generalisierungen_

_Wolken_  
_Gewitter_  

_Niederschlag_

_Niederschlag_ ist moegliche Konsequenz von _Wolken_  
_Niederschlag_ ist Konsequenz von _Gewitter_
Visual Database
(DA 2002)
Secure, Flexible Knowledge Exchange (Medusa)
Configurable Data Exchange
Graph based programming

• Main idea: Program execution is message passing through a graph along edges

• Inspired by Smalltalk, Self (prototype-based, type-less languages)

• Objects, classes and methods are all handled equally as nodes in the graph
Classes and Instances

- **trait**: Class
- **C**: foo()
- **P**: "Method"
- **D**: shared
- **D**: delegate
- **C**: child
- **P**: parent
- **obj**: "Instance"
- **meth**: "Method"

Dispatch path:
1. foo()
2. ?
3. shared
4. "Method"

Members and Associates

"Class"

trait

D: shared

"Instance"

obj

A: y

"Associate"

ass

C: x

P:

"Member"

data

D: delegate
C: child
P: parent
A: associate
Inheritance and Overriding
Method evaluation

- Trait
  - "Class"
  - D: shared
  - C: foo()
  - P: meth

- Method
  - "Method"
  - P: copy

- Object
  - "Instance"
  - P:
  - C: x
  - P:

- Data
  - "Member"

D: delegate
C: child
P: parent
Instantiation
Nesting and Delegation

Diagram:

- Parent2
- Parent1
- Child
- Delegate2.1
- Delegate2.2
- Delegate1.1
- Delegate1.2
- Delegate.1
- Delegate.2

Connections:
1. From child to foo
2. From delegate.1 to delegate.2
3. From parent1 to delegate1.1
4. From delegate1.1 to delegate1.2
5. From parent2 to delegate2.1
6. From delegate2.1 to delegate2.2
Dynamic Delegation
Expected direct benefits

• Slim set of construction elements: Nodes, edges, delegation, nesting and sharing
• Combining features of prototypes and classes
• Dynamic nested overrriding
Implementation

- Python prototype for message dispatching and programming language
- GUI Prototype as IDE for graph creation and visualization
Next steps

- Finishing prototype, playing around with new language
- Paper about graph based programming with Axel Rauschmayer (LMU) until July
- DFG Antrag for a “The three pillars” project with LMU until end of August
Future ideas

- **Components:**
  - Prototypes, connectors and ports
  - Interactive layout of architecture

- **Meta programming:**
  - expose meta protocol
  - provide reflection (no meta constructs needed!)

- **Advanced Typing:**
  - Stateful multi-methods
  - Views, perspectives

- **Distributed applications**
Thanks for patience.