



Consortium for Software Engineering Research
Consortium de recherche en génie logiciel

- Université de Montréal
- Bell Canada

SPool

**Spreading Desirable Properties
into the Design of
Object-Oriented, Large-Scale
Software Systems**



Motivation

- Acquisition of large object-oriented real-time systems by Bell
 - product selection
 - risk assessment in resp. to evolvability and maintainability
- Assumption
 - Design is key factor of influence on software quality
- Goals
 - Identify and quantify those design properties that are significant for maintainability and evolvability
 - Develop concepts for the systematic construction of high quality o-o designs



Motivation (cont.)

- SPOOL = Spreading Desirable Properties into Object-Oriented, Large-Scale Systems
- Approach:
 - empirical: study of legacy software
 - constructive: design of new methods, tools, measures
 - experimental: evaluation of new concepts



Key Questions and Subprojects

- What makes for a good o-o design?
 - Design Quality Assessment
- How do you achieve a good o-o design?
 - Design Pattern Engineering and Transformations



Design Quality Assessment

- Goal: given an o-o program, assess its quality with respect to evolution:
 - find which properties make change implementation easy/difficult
 - design properties:
 - macro-design: class relationships
 - micro-design: class-component relationships
 - quantify these properties
 - validate on programs of interest to Bell
- Participants (UdeM)
 - Ajmal Chaumon (M.Sc. student)
 - Hind Kabaili (Ph.D. student)
 - Rudolf Keller, François Lustman (investigators)



Approach

- Assumption-based approach: cohesion and coupling ARE the best indicators of design quality:
 - adapt cohesion and coupling to o-o paradigm
 - quantify
 - validate on programs of interest to Bell
- Impact-based approach: investigate which changes to a class have the largest and costliest impact:
 - classify program changes in C++
 - define impact of change
 - investigate on programs of interest to Bell



Recent Results

- List of program changes in C++
- Theoretical model of change impact
- Definition of change impact for every change
- Prototype of change impact analyzer
 - based on Gen++
 - supporting several changes of change list
- Ph.D. student training complete
 - courses, comprehensive exam
 - training in statistics



Change Impact Model and Expressions

$$I(A, c) = f(S, G, H, I, L)$$

$I(A, c)$: impact of change c made in class A

f : boolean function or set function, depending on interpretation

S : association link

G : aggregation link

H : inheritance link

I : invocation link

L : local impact

Examples of change impact expressions:

$$I(A, \text{public variable} \rightarrow \text{protected}) = SHF'$$

$$I(A, \text{delete friend}) = F(S + G + H + I)$$



Direction of Work in next 6 Months

- Cohesion/coupling
 - update of literature study
 - investigation of cohesion and coupling inside a class:
 - definition of cohesion of a class, and of the coupling between class components
 - definition or adaptation of metrics for assessing cohesion/coupling in a class
- Change impact analysis
 - run prototype analyzer at Bell
 - test analyzer on middle-sized program provided by Bell and analyze results
 - design and implement new, complete version of analyzer



Design Pattern Engineering and Transformations

- Project Goals
 - design pattern-based software composition environment (design components, interactive specification, component repository)
 - design pattern-based reverse engineering environment (pattern detection, visualization, assessment)
 - systematic transition approach from high-level to detailed design as a basis for automation, traceability, and coherence
- Participants (UdeM)
 - François Martel, Patrick Pagé, Sébastien Robitaille (term students)
 - Bouazza Bachar (M.Sc. student), Ismail Khriiss (Ph.D. student)
 - Reinhard Schauer (research associate)
 - Rudolf Keller (investigator)



Design Transformations

- Milestones achieved in last six months
 - 4-step transformation approach from collaboration to statechart diagrams
 - algorithm for integration of UML statechart diagrams: design, implementation, test on several examples, documentation
 - refinement schemas for six design patterns: detailed description, experimentation
- Direction of work in next six months
 - validation of 4-step transformation approach
 - user interface prototype generation
 - extension of refinement schema approach
- Technical reports/Papers
 - Ismail Khriiss, Mohammed Elkoutbi, and Rudolf K. Keller. Automating the synthesis of UML statecharts from multiple collaboration diagrams. In Proc. of the International Workshop on the Unified Modelling Language (UML'98), Strasbourg, France, June 1998.
 - Ismail Khriiss, Mohammed Elkoutbi, and Rudolf K. Keller. A new approach to the synthesis of behavioral specifications from scenarios. TR GELO-83, UdeM, Feb. 1998.



Design Pattern Engineering

- Milestones achieved in last six months
 - version 1.0 of prototype environment
 - initial phase of case study Bell system 3
- Direction of work in next six months
 - evolution of prototype environment
 - several case studies with Bell systems
 - 3 student summer projects
- Technical reports/Papers
 - Reinhard Schauer and Rudolf K. Keller. Pattern visualization for software comprehension. In Proc. of the Sixth Intl. Ws. on Program Comprehension (IWPC'98), Ischia, Italy, June 1998
 - Rudolf K. Keller and Reinhard Schauer. Design components: Towards software composition at the design level. In Proc. of ICSE'98, Kyoto, Japan, April 1998. IEEE.
 - Rudolf K. Keller and Reinhard Schauer. The Complet model: Component-based software development from process to code. Technical Report GELO-86, Universite de Montreal, Montreal, Quebec, Canada, March 1998.
 - Bouazza Bachar. Towards automatic detection of design patterns. Master's thesis, Universite de Montreal, Montreal, Quebec, Canada, May 1998. In French.



Implementation and Experimentation

- (1) Introduction
- (2) Reverse Engineering
- (3) Source Code & Design Repository
- (4) Visualization
- (5) First Pattern Detection Results
- (6) Student Summer Projects

SPOOL

**Spreading Desirable Properties
into the Design of
Object-Oriented, Large-Scale
Software Systems**



Reverse Engineering - Background

- Analysis of *Bell System 3*
 - ca. 300.000 lines of code, 1221 header files
- Use of Gen++ to run pattern queries
 - Gen++ is bound to the AT&T Cfront Compiler.
 - Gen++ reads the abstract semantic graph (ASG) produced by the AT&T Cfront Compiler.
 - For pattern detection, the whole system needs to be treated as one compilation unit.
 - However, AT&T Cfront Compiler cannot handle 300.000 lines of code in one compilation run.
- Some other problems encountered
 - files missing, unreasonable include nesting, many unresolved references, right setting of macros,...



Reverse Engineering - Conclusion

- Conclusion for Gen++ prototyping
 - 6 person days to modify System 3 so that it could be compiled with the AT&T Cfront compiler
 - 20-30 min per analysis run
 - no interactivity, no visualization
- Our solution
 - Use Gen++ (Datrix in future) as the source code parser
 - Spool the C++ source code into our UML repository
 - Run OQL pattern detection queries on the UML repository
 - interactive, direct visualization, multi-phase detection

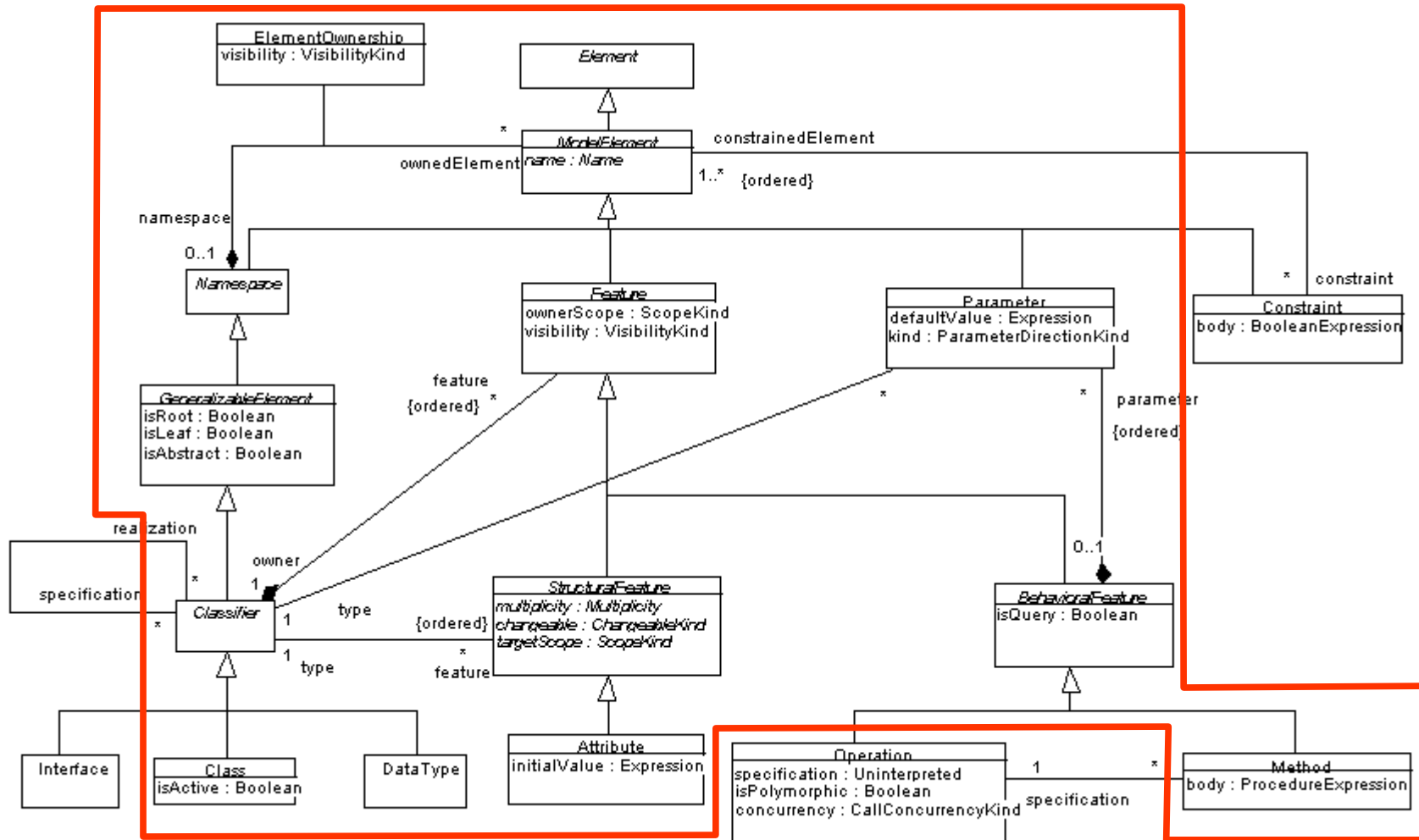


Source Code & Design Repository

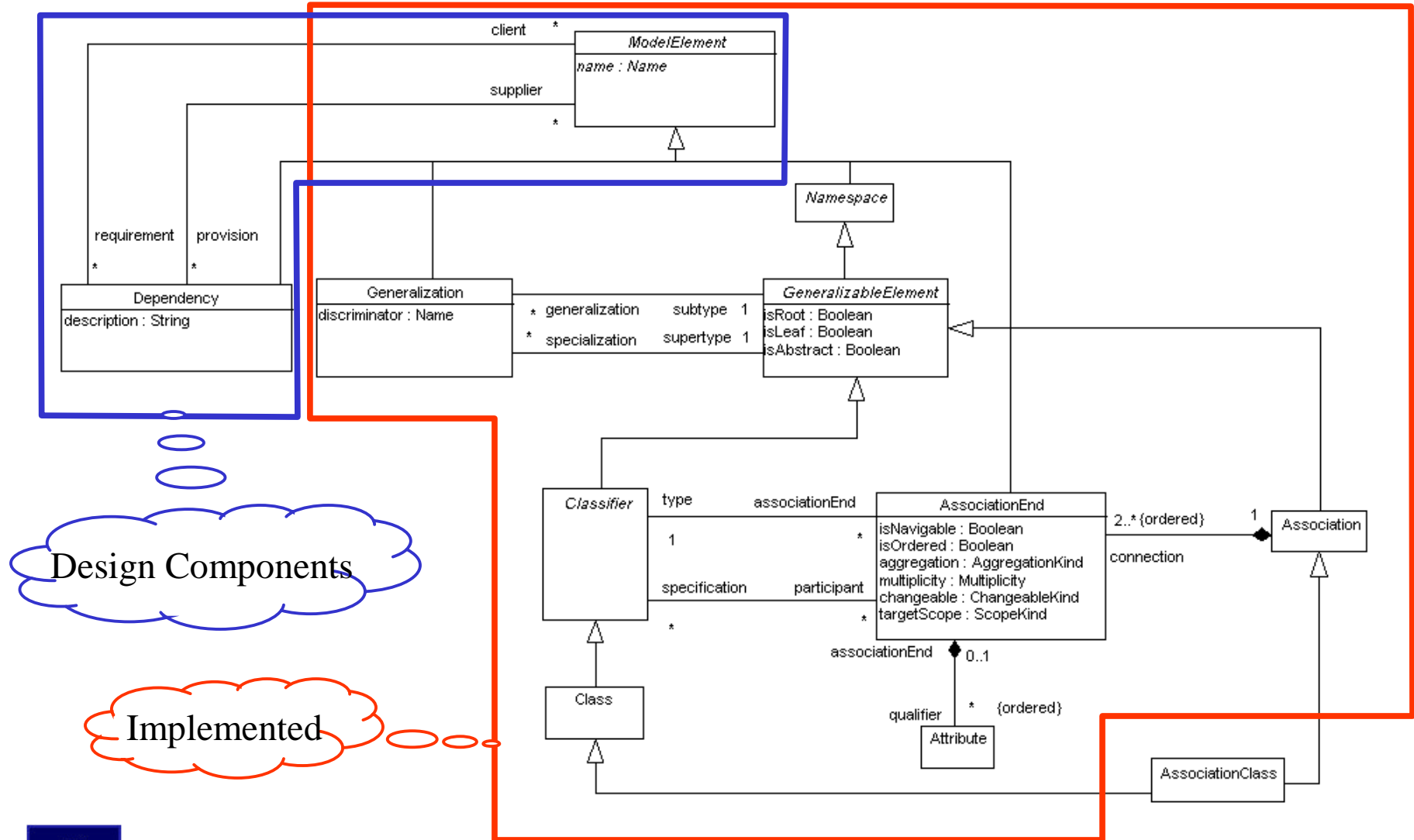
- UML 1.1 Metamodel
 - Implemented Metamodel Packages
 - Core Backbone
 - Core Relationships
 - Common Behavior Actions
 - Model Management
- CDIF Intermediate Format
 - JavaCC program for parsing UML/CDIF



UML 1.1 - Core Backbone

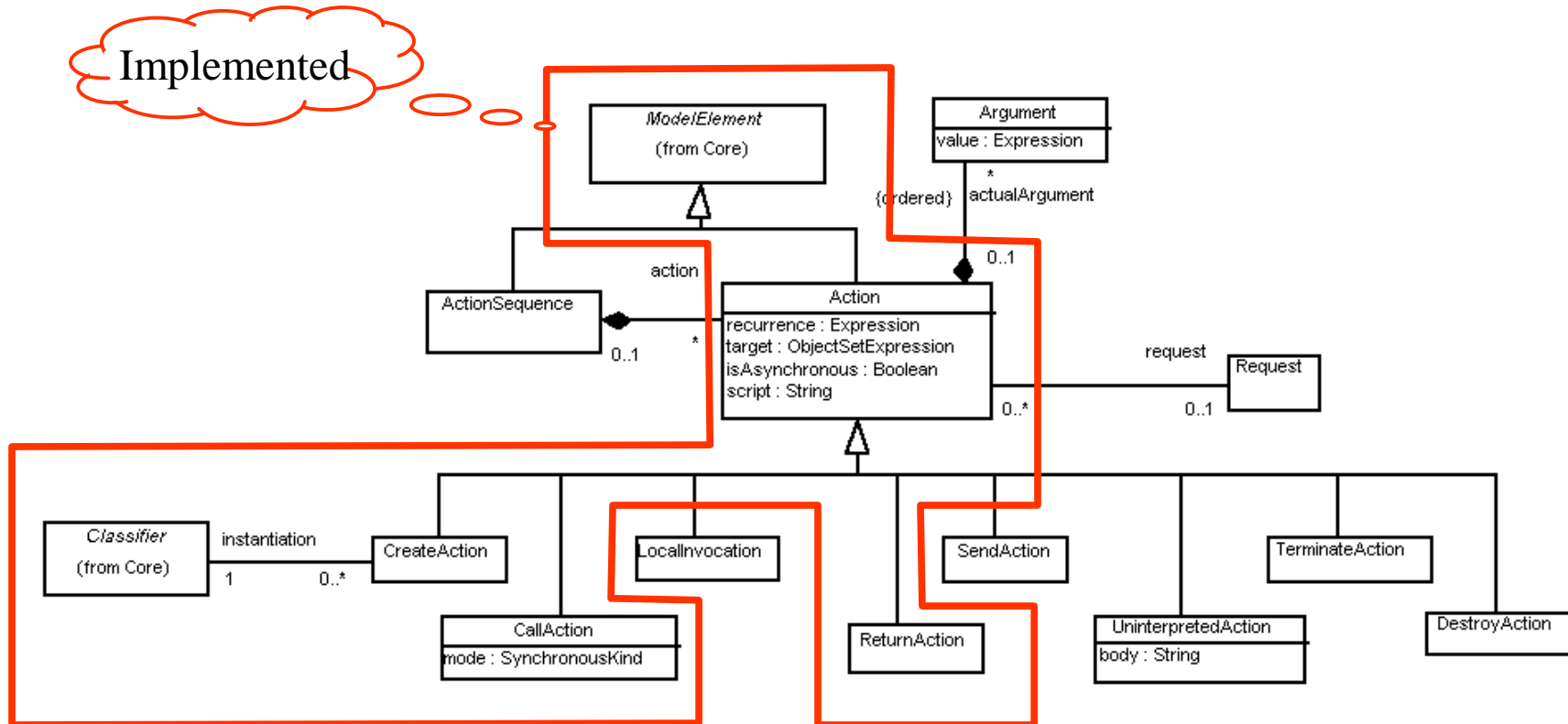


UML 1.1- Core Relationships

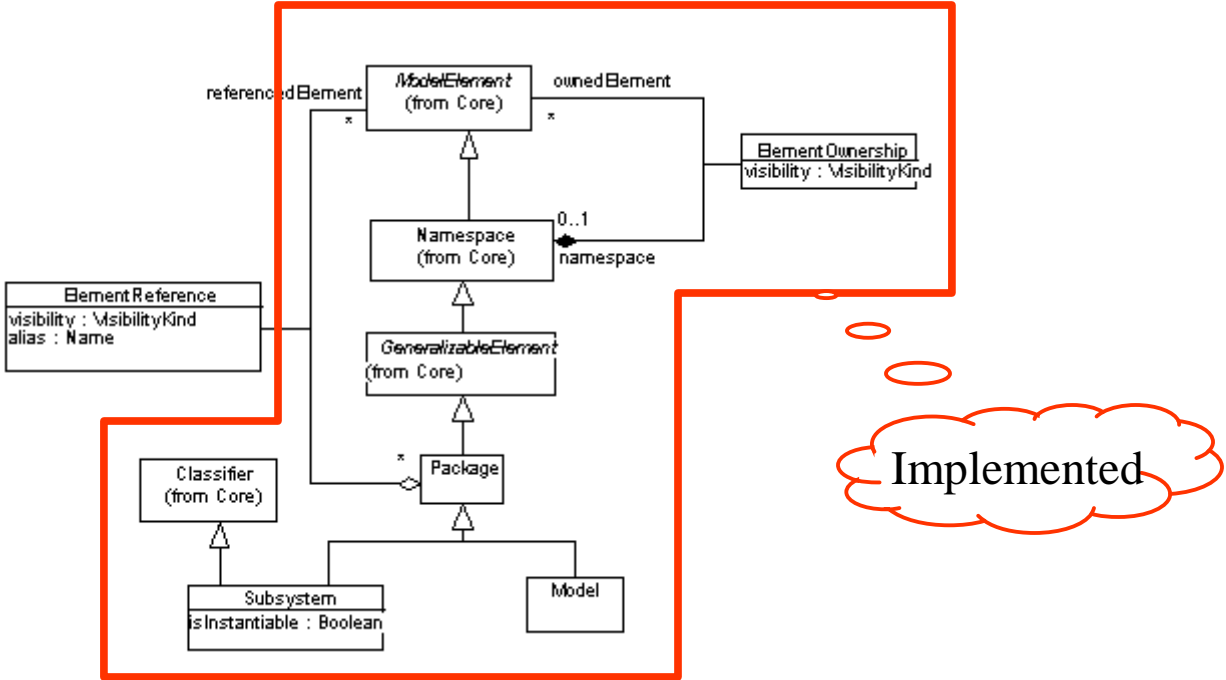


UML 1.1 - Common Behavior Actions

Implemented



UML 1.1 - Model Management



UML / CDIF Intermediate Language

- Grammar for UML 1.1 / CDIF
- JavaCC (Compiler-Compiler) - parser for UML/CDIF
- Interface for semantic routines using the *Visitor* pattern; Interface implementations for
 - reformatting the input stream and writing it into another output file.
 - writing the input stream into our database (not done in the course of this student project).



UML / CDIF - Simple Example

```
CDIF,SYNTAX "SYNTAX.1" "02.00.00",
      ENCODING "ENCODING.1" "02.00.00"
(:HEADER
  (:SUMMARY
  )
)

(:META-MODEL
)

(:MODEL
  (Package 01)
  (Model 02
    (name M)
    (isRoot -True-)
  )
)

(Class 03
  (name A)
  (isAbstract -False-)
  (isLeaf -False-)
)
(Class 04
  (name B)
  (isAbstract -False-)
  (isLeaf -True-)
)
(Dependency monID
  (description "Allo \" Ca c'est \\ un string.")
)
(Generalization sept
  (discriminator unBeauNom)
)
(Package.generalization.Generalization 05 01 02)
(Class.generalization.Generalization 06 03 04 )
)
```



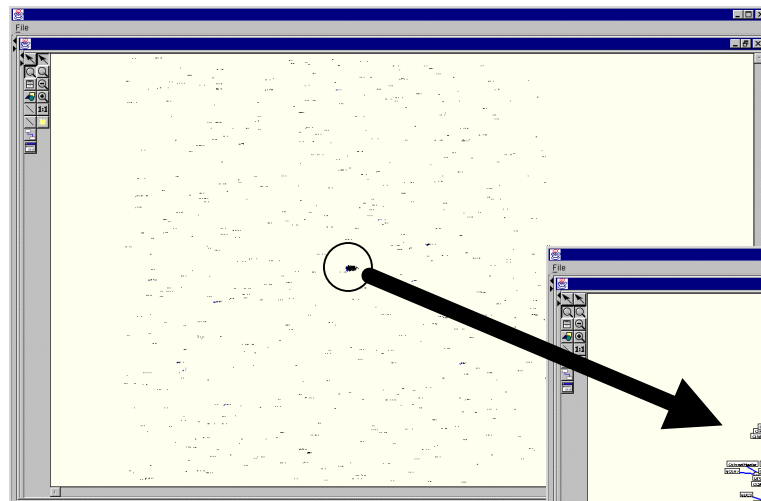
Visualization

- Architectural Review of the Design Visualization System
 - diagram manager (View Handler pattern)
 - internal frames for UML packages
 - synchronization of diagrams
 - extensibility for additional services, such as undo, tracing, logging, etc. (Command Processor pattern)
 - configurable visualization strategies (Strategy pattern)
- Integration of “Dot” and “Neato”
 - DOT - for directed graphs
 - hierarchical arrangement of graphic objects (variation of the *dag* algorithm).
 - NEATO - for undirected graphs
 - symmetric arrangement of associations.

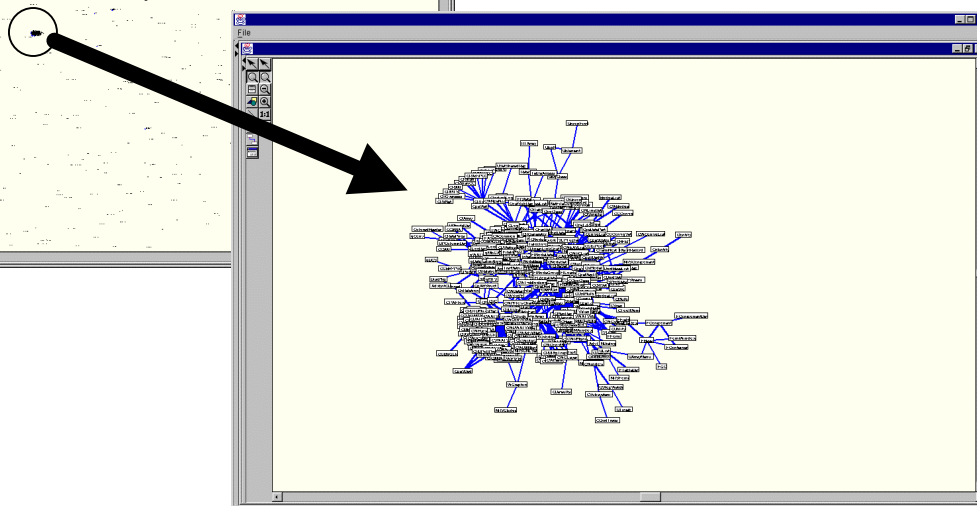


Bell System 3 - Aggregation Relationships

- Aggregation relationships (layout with “neato”)

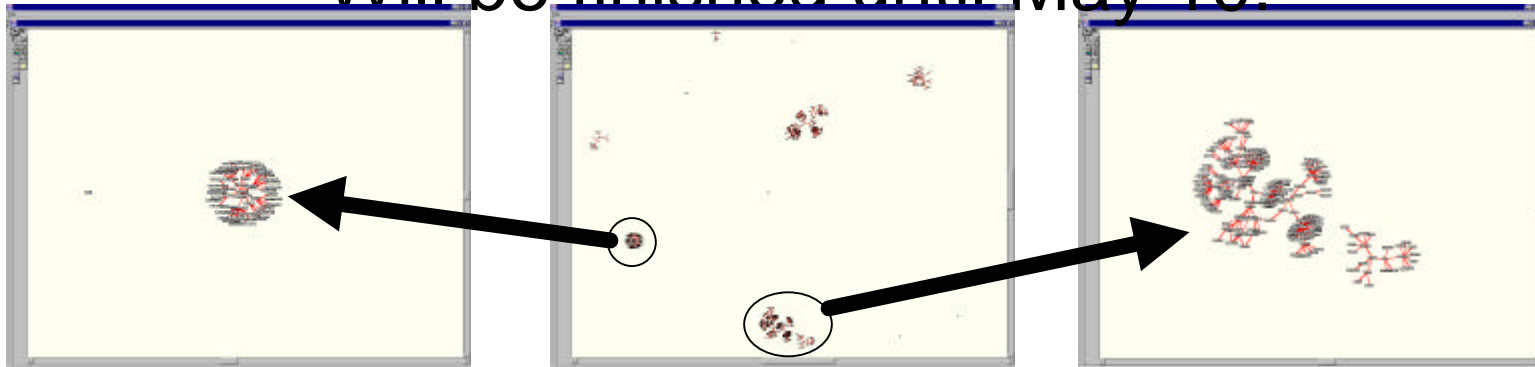


- Ca. 300.000 lines of code (C++)
- 1373 classes
- 2408 aggregation relationships



Bell System 3 - Class Hierarchy

- Layout generated with Neato
 - Will be finished until May 10!



- Tree Layout generated with Dot will be available until the meeting.



Student Summer Projects

- Graphic Editor Evolution (Design Components)
 - Extension of UML Graphic Editor
 - Implementation of Design Assembly
- C++/UML Software Repository
 - Fully-fledged C++ Source Code Repository
 - OQL-Based Pattern Detection
- Software Migration
 - Use of Design Patterns for Software Migration

