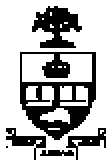
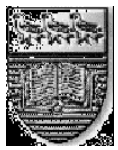
Software Evolution Through Program Transformations: An Experience Report

Kostas Kontogiannis,
Johannes Martin
Kenny Wong
Richard Gregory
Hausi Muller
John Mylopoulos



Introduction

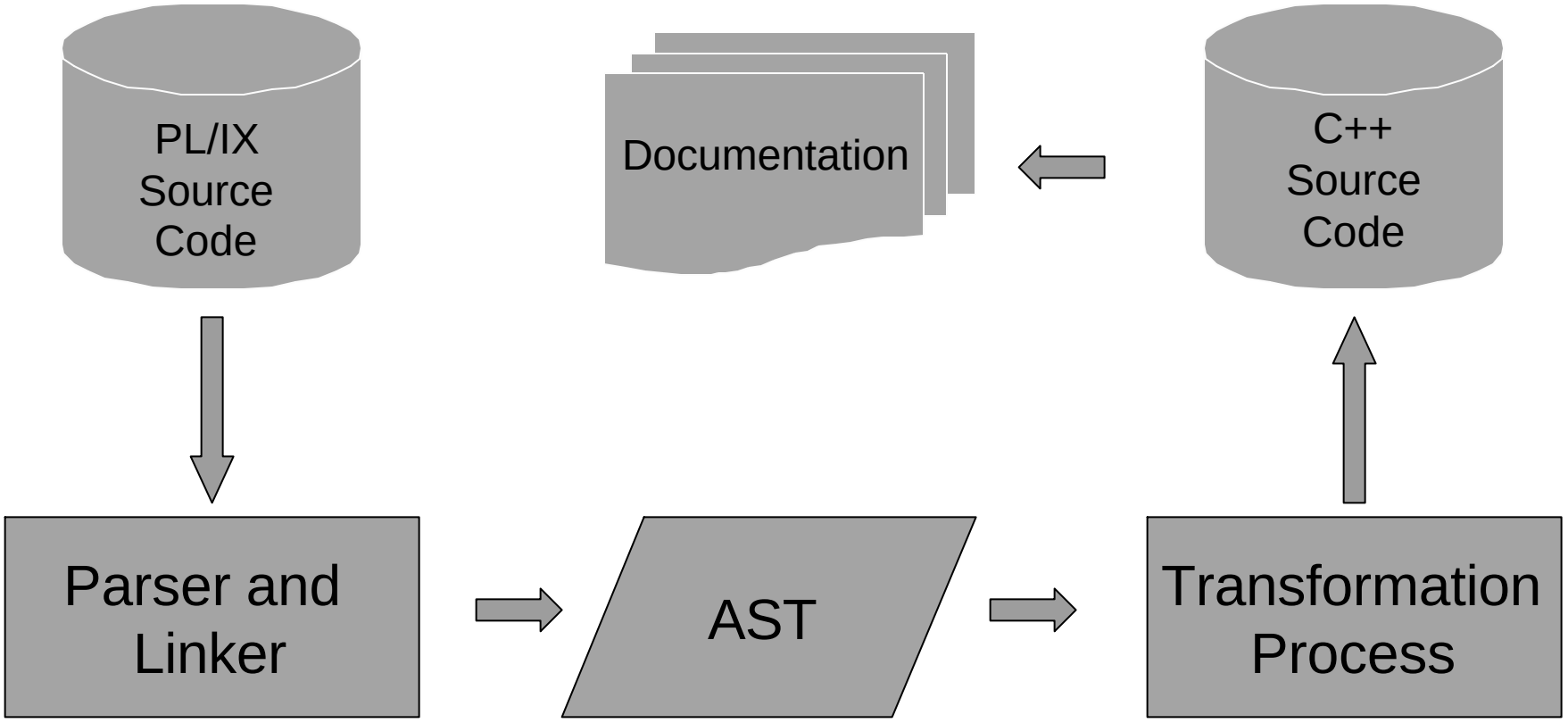
- Over 800 billion lines of code exist worldwide written in more that 500 languages
- Operating platforms and supporting utilities evolve constantly
- Research task:
  - provide support for massive source code changes
  - allow for legacy code to be kept up-to-date
  - support custom designed performance enhancements

Related Work

- Three basic approaches to the problem:
  - Formal methods & language semantics
  - Grammar-based parse tree transformations
  - Repository-based transformation rules

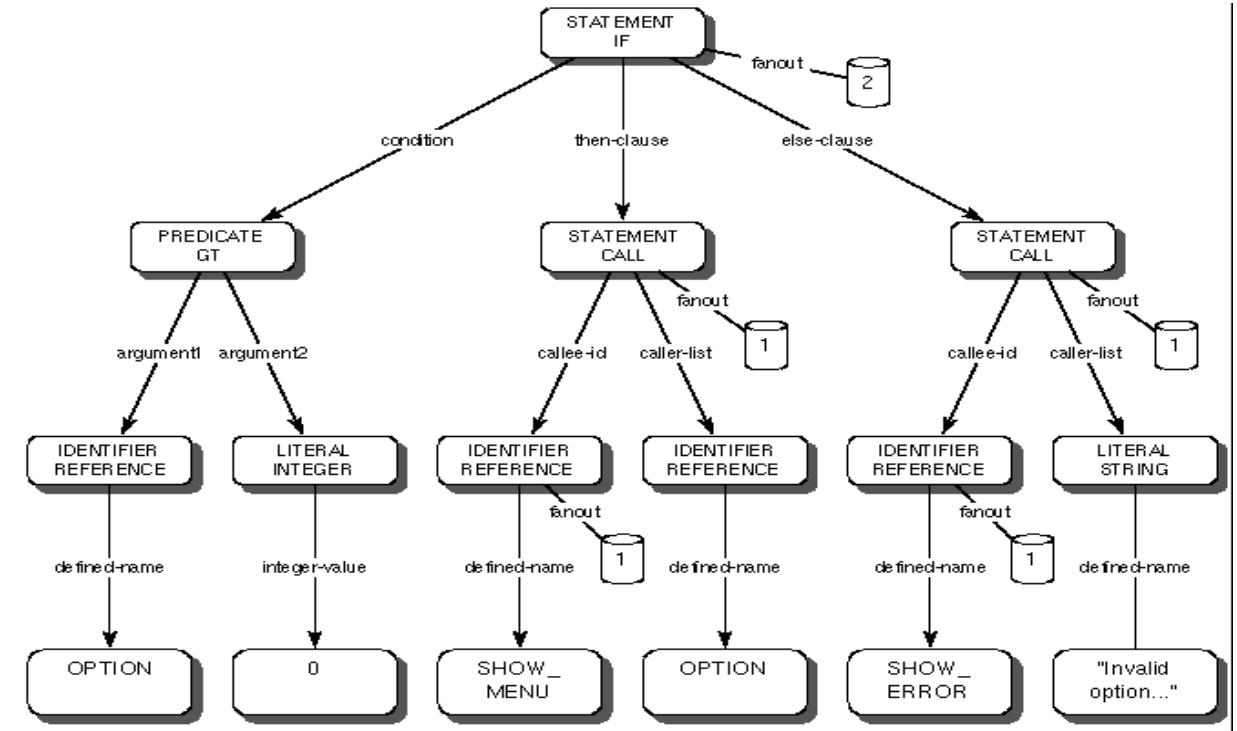
- Pattern matching is a key ingredient of all approaches:
  - Mapping patterns from one language to another
  - Syntax-directed editing

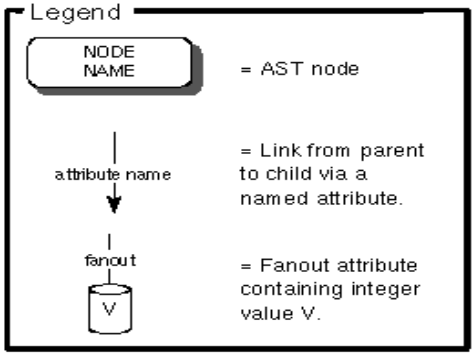
System Architecture



Source Code Representation

```
If(OPTION > 0) THEN
    SHOW_MENU(OPTION)
ELSE
    SHOW_ERROR("Invalid
                option")
```





Repository

- Rigi example:

Type	parse.c	File
FileGlobals	parse.c	memory_table
FileGlobals	parse.c	assignment_parse
FileLocals	parse.c	simple_pattern_parse
Includes	parse.c	memory.h

- Telos domain model example:

```
(File Sclass  (RigiClass RefineClass)
  (RigiContainer RigiProgrammingObject
                 RefineProgrammingObject)
  ((Name String)
   (fileGlobals seq(Identifier))
   (fileLocals seq(Identifier))
   (Includes set(File))))
```

A Case Study

- Work with a 300KLOC legacy software system of highly optimized code written in PL/IX
- Components of this system need to be translated to C++
- Develop tools which semi-automate the translation process to C++
- Make sure that translated code performs as well as the original code

Methodology

- First migration effort was completed by hand; an expert
- programmer required ~10weeks to migrate 7.8KLOC
- Prototype migration tools were based on the heuristics used by the expert
- Migrated code was 50% slower than original; expert identified bottlenecks and hand-transformed migrated code so that it performs much faster.
- Expert heuristics were, again incorporated into the tool

PL/IX to C++

- The transformation process consists of three main steps:
  - Transform PLI/IX declaration items and data types to their corresponding C++ data types
  - Generate support C++ libraries (macros for reference components, class definitions for major data structures)
  - Generate C++ source code that is structurally and behaviorally similar to the legacy source code

Type Transformations

PL/IX to C++

```
dcl
  1 l_bag    based   REFLECT_ATTR,

    2 bag              bit(32),
   .2 *,
     3 oper_count    bit(16),
       max_opnds     lit('2**11-1'),
     3 opcd           bit(16),

   .2 *,
     3 ind            bit(8),
     3 x              bit(24),
```

```
define C_MAX_OPNDS 2**11-1
struct any_L_BAG {
        union BAG {
                int             bag;
                struct any_L_BAG_2 {
                        short int   oper_count;

                        short int   opcd;
                };
                struct any_L_BAG_6 {
                        unsigned char   ind;
                        int             x:24;
                };
   }; };
```

PL/IX to C++

```
dsinit: proc(pn);
        var_containing: proc;
                        …….
        end procedure var_containing;

        overlap: proc(m1,m2)
                        ……..
        end procedure  overlap;
                        ……..
end procedure dsinit;
```

```
class Dsinit {
  public:
     static void   dsinit(int pn);
  private:
     static void   var_containing();
     static void   overlap(int m1, int m2);
};
```

PL/IX to C++

```
dsmrgs: proc(bb,always) returns(integer) exposed;
dcl  bb              integer value,
     always      bit value,
     (i,l)          integer;

 l = make_empty_list;
   if ns(bb) = 0
    then l = merge(1,l) ;

    else
      do I = fs(bb) to fs(bb)+ns(bb)-1;

          l = merge(succ(i),l);
      end;
   return(l);
end procedure dsmrgs;
```

```
int Dsmrgs::dsmrgs(int bb, boolean always)
{
 int            bb;
 boolean        always;
 int            i, l;

 l = make_empty_list();
 if ((C_NS(bb) == 0)) {
      l = Dsmrgs::merge(1, l) ;
   }
 else {
     for (i = C_FS(bb) ; i <= ((C_FS(bb) +
     C_NS(bb)) - 1); i++)
          l = Dsmrgs::merge(C_SUCC(i), l);
   }
     return l;
}
```

Overall System Performance

Subsystem Performance

Translator Performance

Human Effort

Conclusions

- Semi-automatic transformation of large volume of code is feasible
- Migrated code suffers no deterioration in performance
- Incremental migration process feasible
- Technique readily applicable to PL/xx family of languages
- Technique reduces migration effort by a factor of 10
- New prototype for C to RPG transformations targeting AS/400 users