Program Comprehension in Multi-Language Systems

Bernt Kullbach, Andreas Winter,
Peter Dahm, Jürgen Ebert

4/98

**Fachberichte
INFORMATIK**

Program Comprehension in Multi-Language Systems[*]

Bernt Kullbach   Andreas Winter   Peter Dahm   Jürgen Ebert
University of Koblenz-Landau
Institute for Software Technology
Rheinau 1, D-56075 Koblenz, Germany
(kullbach|winter|dahm|ebert)@informatik.uni-koblenz.de

Abstract

This paper presents an approach to program comprehension in multi-language systems. Such systems are characterized by a high amount of source codes in various languages for programming, database definition and job control. Coping with those systems requires the references crossing the language boundaries to be analysed.

Using the EER/GRAL approach to graph-based conceptual modeling, models representing relevant aspects of single language are built and integrated into a common conceptual model. Since conceptual modeling focusses on specific problems, the integrated model presented here is especially tailored to multi-language aspects. Software systems are parsed and represented according to this conceptual model and queried by using a powerful graph query mechanism. This allows multi-language cross references to be easily retrieved.

The multi-language conceptual model and the query facilities have been developed in cooperation with the maintenance programmers at an insurance company within the GUPRO project.

1. Introduction

It is widely agreed that a software system becomes legacy as soon as it is delivered. Thus, software maintenance is becoming increasingly important. Following the worldwide benchmark project [34] only 36.9% percent of work are allocated to new development of software whereas 52.9% are spent on maintenance, modification and migration.

Program understanding plays an essential role during the phases after software development. Already in the late seventies it has been reported that about 47% of maintenance efforts are spent on program understanding whereas only about 25% are spent on modification [16]. Following other studies, the program understanding effort is said to cover 40% [19], 60% [35] or even up to 90% [37]. As such, program understanding is the key activity during software maintenance providing a large potential to improving the efficiency of software development and maintenance processes.

There are various approaches to program understanding, which follow a common structure: Source codes are translated into a general data structure which is analyzed afterwards. Different choices to representing source code information exist, such as relational databases [28], [5], [18], PROLOG databases [24], [3], object-oriented databases [29], [32], abstract syntax trees [11], [32], [40], [6], LISP images [31] or hybrid knowledge bases [23]. The repository structures are described in terms of textual languages [40], entity-relationship languages [23], [3] or formal algebraic models [32]. Coarse-grained repository definitions are given for a PASCAL-like language [28] and for C [5], [32]. A set of three data structures of different level of granularity is defined in [23].

The way chosen for representing program information implies the analysis mechanism. In the database-driven approaches convenient database query mechanisms are used. ASTLOG [6] defines a PROLOG-based query language for analyzing abstract syntax tree representations. In [23] a database-independent program query language (PQL) is introduced. An algebraic expression-based query language has been implemented within the Software Refinery Toolset [32].

In all of these these approaches the given repository structures are fixed. They only cope with one single programming language. Thus they are not much suited to support program understanding in multi-language systems which are the usual thing in today's enterprises. Most of

these real world software systems consist of several – sometimes many – languages. This especially holds, if a system is rather old, surviving through generations of programming languages and hardware architectures.

Our work is concerned with the software of the Aachener und Münchener Informatik Systeme GmbH (subsidiary of a German insurance company) which is an MVS system consisting of multiple sources in various programming languages, database definitions and job control languages. Altogether, the system consists of about 25 000 JCL-, 5 700 CSP-, 4 000 COBOL II- 3 800 Delta COBOL-, 6 000 PL/1-, 1 000 Assembler-, 100 REXX sources as well as programs written in languages like APL, SAS or Easytrieve. Furthermore 100 data models with about 3 000 entities and 60 000 attributes have to be considered.

In this paper, we present an approach to program comprehension in multi-language software systems. Section 2 describes how a multi-language model is established and how multi-language source codes can be parsed into an object-based repository. A coarse-grained conceptual model of a subset of the programming language environment as used in the insurance company is introduced. Within section 3 the analysis of this multi-language system is described. Section 4 sketches the realization of the multi-language program understanding system as part of the GUPRO-MetaCARE approach [13]. The papers ends with a conclusion and an outlook in section 5.

2. Representing multi-language systems

There exist various kinds of software maintenance tasks ranging from analyzing single program statements [40] to inspecting the overall architecture of a whole software system [41]. In building tools to support programs understanding one has to define which aspects of software are relevant for the maintenance tasks to be approached. This definition can be performed by conceptual modeling techniques which can be derived from knowledge representation (e. g. conceptual graphs [36], $\tau \acute{\epsilon} \lambda o \varsigma$ [30]) or can be based on entity-relationship modeling (e. g. [4], NIAM information structure diagrams [39], or even UML class diagrams [38]). Conceptual models are used to focus the attention on the relevant concepts of the software system and their interdependencies with respect to the given maintenance problem.

Within our work the EER/GRAL approach [15] to graph-based conceptual modeling is used. EER/GRAL is based on TGraphs [12] which are a very general class of graphs. Conceptual modeling is enabled by an extended entity-relationship dialect (EER) and the GRAL constraint language [17]. The approach is based on a formal TGraph-oriented semantics, which is defined in [8]. Together with the GraLab implementation toolkit [9], EER/GRAL pro-

vides a seamless approach object-based to modeling and implementation.

Ongoing it is addressed how a conceptual model for multi-language software systems can be obtained using EER/GRAL and how source codes can be translated into a repository suited to this conceptual model.

2.1. Building a multi-language conceptual model

An important problem with maintaining multi-language systems is to cope with references crossing the language boundaries.

In MVS systems [2] e. g. the connections between sources from different languages are defined by job control procedures. From job control procedures programs are called. Furthermore, mappings between programs and databases are defined with respect to such calls. An application in such systems consists of a number of JCL procedures [21]. For identifying all programs belonging to an application, one has to find all programs being called by the respective JCL procedures. In this context indirect calls from these programs to other programs have to be considered too. Another significant cross referencing problem arises in database migration. Changing from one database system to another requires modifications in all programs accessing the respective data. These programs have to be detected by treating both, JCL procedures and database definitions, in conjunction.

A conceptual model for a multi-language software system capturing such kind of analysis has to include concepts representing the relevant source code aspects (e. g. JCL procedures or programs) as well as the conceptual relationships between them (e. g. calles or isCalledBy). Multi-language models have to be composed from single language models by identifying their interconnections.

Ongoing, the integration of MVS-JCL [2] and COBOL [22] shall be sketched as an example.

Example. Figure 1 shows a coarse-grained conceptual model for MVS-JCL. According to this, a *JclProcedure* consists of *Steps*. JCL *Steps* are to call *Programs*, provide bindings between physical data *Files* and logical file names (*DdStmt*) and specify accesses to hierarchical databases via *Psb* definitions.

In Figure 2 a conceptual model for COBOL is introduced: *CobolPrograms* contain *CobolProcedures* which can call other *CobolProcedures* or other (not necessarily COBOL) programs (modeled as *callsProc* and *callsProgram* relationship, respectively). COBOL programs may also have access to data *files* (*accessesFile*). In the COBOL code this access is specified indirectly by a logical file reference (*DdFileRef*). As we will see the connection to a physical file can only be established in conjunction with a JCL pro-

cedure. For this reason, the *accessesFile* relationship in figure 2 is marked dashed. □
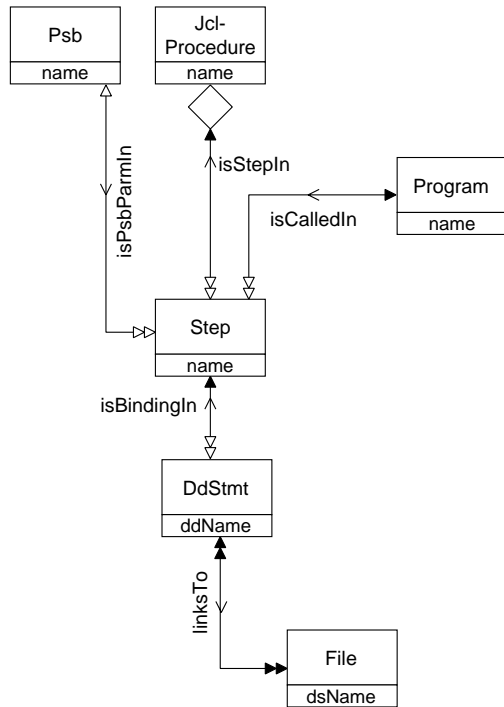

Figure 1. JCL single-language model

Figure 2. COBOL single-language model

Within this multi-language model we are primarily interested in concepts relevant for inter-language cross-references. As such, concepts like internal definitions e. g. the COBOL data division are not represented in the model.

Building a common conceptual model requires the corresponding single-language models to be integrated. In this respect a very general integration strategy is adapted from the EER/GRAL approach to conceptual modeling [13]:

(A) Concepts which are similar are generalized into a common super concept. If they denote the same information they are unified. and

(B) Concepts from different models, which have interconnections are connected by further conceptual relations.

When integrating the conceptual models for JCL and COBOL presented in figures 1 and 2, two common concepts can be identified: Firstly, a *Program* is called within a JCL *Step* as well as by a *CobolProgram*. Secondly, both models include bindings resp. accesses to *Files*. According to strategy (A), the concepts *Program* and *File* have to be unified from both models.
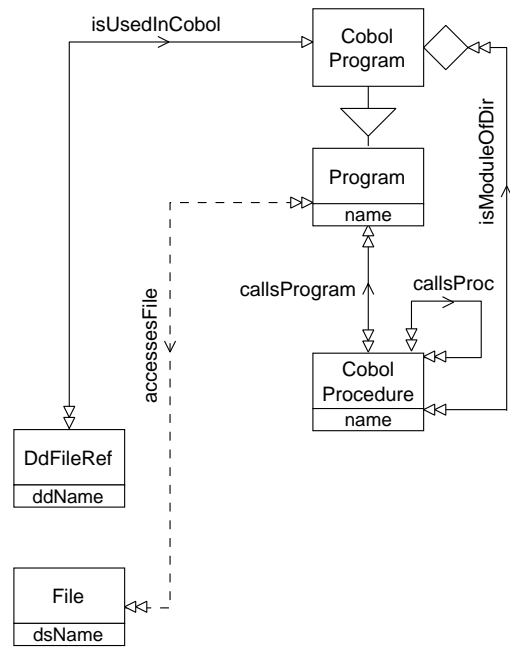
As an integration of type (B), a conceptual relation *accessesFile* between *Program* and *File* can be established, indicating the access of a *Program* to a *File*. The logical file reference (*DdFileRef*), used in COBOL programs is mapped to a physical file in a JCL step. So, an *accessesFile* conceptual relation between a *CobolProgram* and a *File* can only be established, iff the *File* is bound by some *ddStmt* in a call to a COBOL program within a JCL step. Furthermore, the *ddStmt* has to match the name of the logical file reference *DdFileRef* specified in the COBOL program called. This condition is specified by the GRAL constraint which completes the integrated conceptual model shown in figure 3.

Just as in the case of COBOL, conceptual models of other programming languages can be added to this model, e. g. using an integration of type (A) *Pl1Programs* or *CspApplications* etc. can be integrated as additional subconcepts of *Program*.

A sligthly more complicated situation arises, when databases are added to the given integration. IMS (hierarchical) databases as an example consist of a tree-like structure of *Segments*. A subtree of this structure is mirrored in PSB specifications which are used by *Steps*. The JCL step is responsible for connecting the physical segments to programs. Again we have an integration step of type (B) where the access from programs to segments is defined by an additononal conceptual relationship (c. f. *accessesSegment* in figure 4). A respective constraint has to be defined with re-
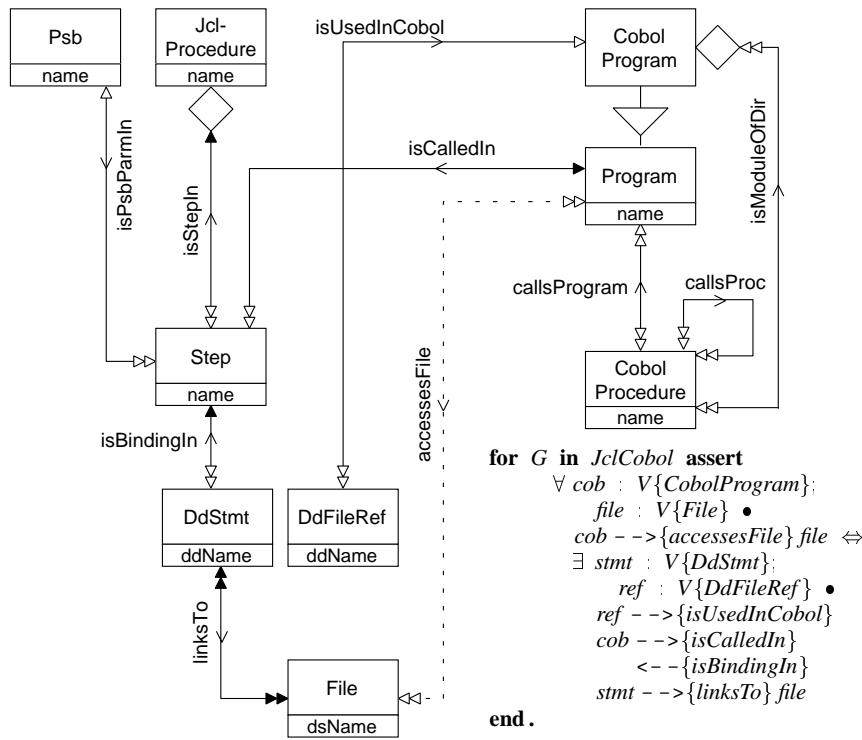
Figure 3. Integrated conceptual models for JCL and COBOL (*JclCobol*)

spect to IMS-DBD, PSB, JCL, and the respective programming language:

```
for  G  in  Macro  assert

  ∀ prog  :  V{Program};   seg  :  V{Segment}  •
      prog -->{accessesSegment} seg  ⇔
      prog <--{isCalledIn}<--{isPsbParmIn}
            -->{usesSegRel}
            (<--{isRoot}  |  <--{isChild}  |
                <--  {isParent}) seg

end .
```

The integration as presented so far is part of the multi-language model representing the programming environment at Aachener und Münchener as depicted in figure 4[1]. The model has been developed together with maintenance engineers at the insurance company [10]. It is the result of an integration of MVS/JCL, COBOL, PL/I, CSP/ESF, IMS-DBD, PSB, as well as SQL DML and DDL.

---

[1] The EER dialect depicts generalization by the usual triangle notation but also allows an alternative notation by nesting object types which is very useful for clustering EER diagrams.

2.2. Filling a multi-language repository

The multi-language model defines the repository structure of a program comprehension tool. Following the EER/GRAL-approach the repository is held in an object-based graph representation.

In order to fill this repository, source codes have to be translated into the underlying graph representation. Because the software system consists of an enormous amount of documents, it is not feasible to fill the repository at once. As a consequence, the translation process has to allow an arbitrary parsing order of sources from various languages. Furthermore, the parsing facilities should enable updating (or even removing) repository entries, caused by changes in software components.

These requirements are met, using a four step parsing approach [27]. The first step checks if the document is already represented in the repository in a former version. If so, it is removed. The document itself is parsed in a second step. In the third step, this representation is integrated into the repository and a fourth step is to ensure the integrity constraints. These parsing steps are controlled by a set of rules, which are derived from the conceptual model.

The demand of parsing source codes from different languages in an arbitrary order also affects the definition of the conceptual model. The model has to assure that incom-
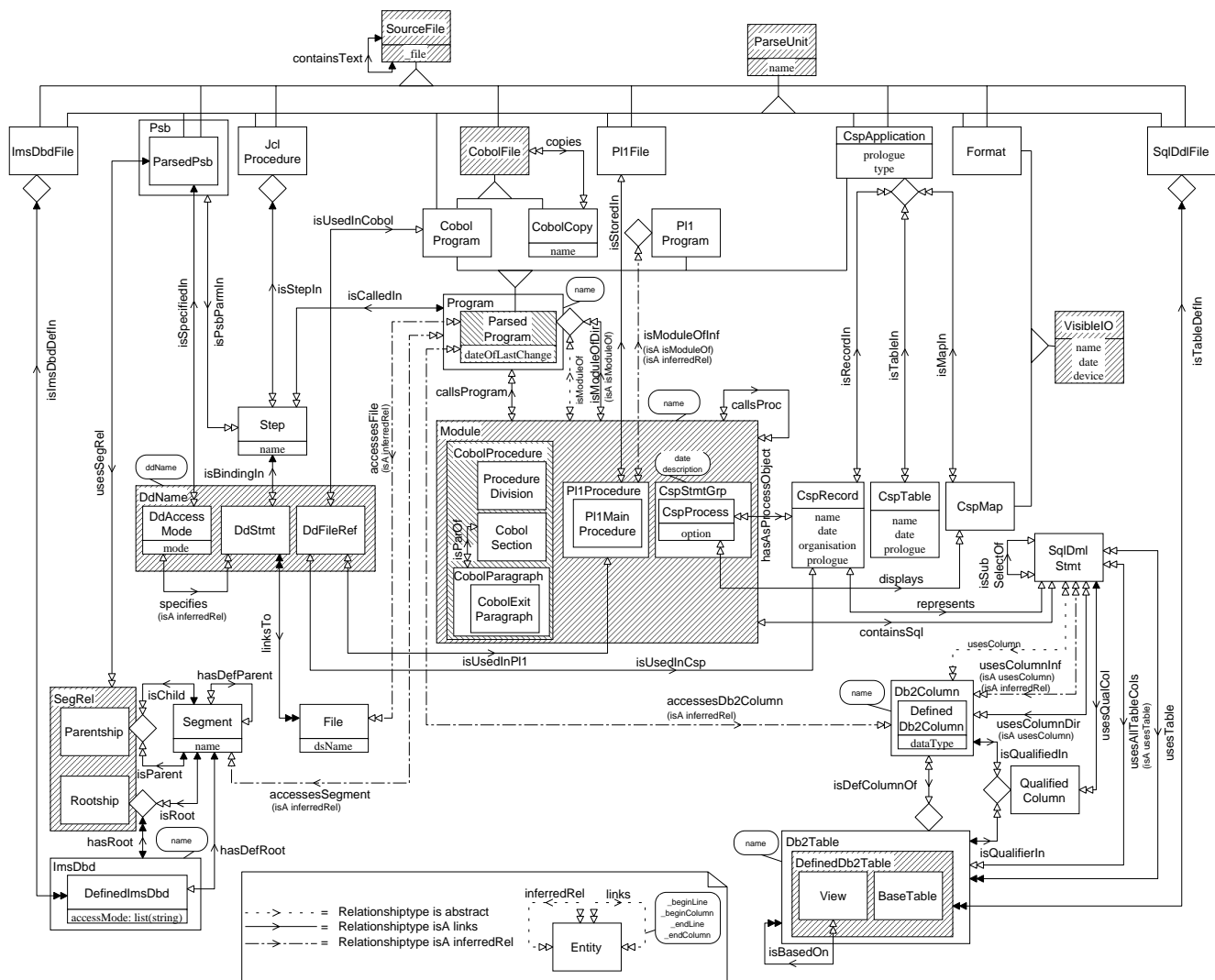
Figure 4. EER-part of the multi-language conceptual model (*Macro*)

plete conceptual relationships can be derived when all necessary informations are captured by once parsing the defining sources. In order to delete former code versions, the remove step has to keep the information that has been caused by other source codes parsed in an earlier step in the repository.

For a detailed description of this parsing processes see [27].

3. Querying multi-language systems

In order to retrieve information about a multi-language system, the repository is analyzed by a general, model-independent query mechanism. The query facility suited to the EER/GRAL approach is provided by the graph query language GReQL (Graph REpository Query Language) [26]. GReQL is an expression language offering restricted first-order logic being especially suited to querying TGraph structures. As a powerful means, GReQL queries may include paths expressions as defined by GRAL [17]. This allows complex relationships between the concepts represented in the repository to be included. GReQL comes along with an extensible library of built-in functions and predicates. These include TGraph-specific functions and relations as well as aggregate functions like counting, average computing, and such.

In the following, the GReQL language shall be explained in some detail along with the MVS application example:

Example. To get the general idea of all software modules making up an application, the maintenance programmer has

to find out which programs are called by the JCL procedures defining this application. As said before, indirect calls are of interest too. A GReQL query addressing this task looks as follows:

```
FROM    jcl : V{JclProcedure},
        called : V{Program}
WITH    jcl
          <--{IsStepIn}
          <--{IsCalledIn}
          (<--{IsModuleOf}
           -->{CallsProgram})*
        called
REPORT  jcl.name, called.name
END
```

This query embodies the basic GReQL construct which is the FROM-WITH-REPORT expression. The FROM clause introduces variables for the concepts and conceptual relationships to be used. Within the example query these are of type *JclProcedure* and *Program*, respectively. The WITH clause imposes a restriction on the possible values of the variables defined. According to the conceptual model in figure 3 (which is a subpart of the model in figure 4) the call of a program from a JCL procedure is represented by a path connecting an object of type *JclProcedure* (jcl) by an object of type *Program* (called). In the case of direct calls this path has to follow an *isStepIn* and an *isCalledIn* relation each in opposite direction. This is denoted by the path-expression jcl <--{IsStepIn} <--{IsCalledIn} called. For including the indirect calls the path description <--{IsModuleOf} -->{CallsProgam} has to be inserted. Each *Program* object may contain some modules, reached by an *IsModuleOf* relationship in reverse order, which themselves call other *Program* objects. To retrieve all indirectly called programs the reflexive, transitive closure (denoted by *) over this subpath has to be calculated. The REPORT clause defines how the output of a query is composed. Within our example only the *JclProcedure* and *Program* identifiers are reported. These are specified in the conceptual model as name attributes. The REPORT clause again may contain further FROM-WITH-REPORT expressions resulting in nested queries.

Figure 5 shows a query screenshot presenting the same question as a nested GReQL query together with the query result in a tabular form. The output of a FROM-WITH-REPORT expression is always a bag. Within the implementation this bag is interpreted and visualized as a table.   □

In order to give a deeper insight into GReQL and its use in querying multi-language systems the IMS-DBD maintenance problem sketched in section 2.1 should be approached.

Example. If one has to replace an IMS database by a relational one, all programs using this database are affected. Unfortunately, programs are connected to IMS databases
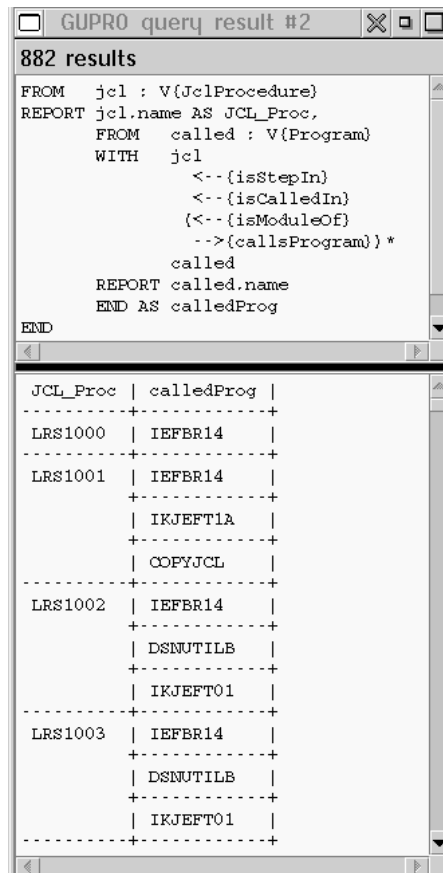

Figure 5. GUPRO Query

only indirectly via JCL and PSB definitions. Starting from a *ImsDbd* object (ims) those *Programs* (prog) are of interest, which can be reached by a path following a *hasDefRoot* relationship to a *Segment* object which itself is accessed by the *Program* through *Psb* and *Step*-objects. The complete path according to the conceptual model in figure 4, is given as follows:

```
ims -->{hasDefRoot}
    (-->{isChild} | -->{isParent} |
     -->{isRoot})
    <--{usesSegRel} -->{isPsbParamIn}
    <--{isCalledIn}
    (<--{isModuleOf} -->{callProgram})*
prog
```

In the conceptual model of the multi-language system, the access of *Programs* to *segments* is also represented by the *accessesSegment* relationship defined in the GRAL constraint on page 4. This, of course, can be used in the GReQL query:

```
FROM      ims : V{ImsDbd};
          prog : V{Program}
WITH      ims
             -->{hasDefRoot}
             <--{accessesSegment}
             (<--{IsModuleOf}
             -->{CallsProgram})*
          prog
REPORT    ims.name, prog.name
END
```

This query yields pairs of IMS databases and those programs which might be affected by modifying the database specification. □

GReQL queries are evaluated by an eval/apply evaluator using an automaton-driven strategy for calculating path expressions efficiently (with respect to the repository content) [1] and including a query-optimizer [33]. To provide non GReQL wizards with querying facility a form-based user interface (with restricted power) has been implemented.

4. The current implementation

The program understanding tool for multi language-systems presented before has been developed as one instance of the GUPRO MetaCARE toolset [13].

This toolset provides a generic environment which is parameterized by a specification of the actual maintenance problem in order to derive concrete program understanding tool instances. This problem specification is defined by an EER/GRAL conceptual model as the one defined in figure 4. Software is parsed by parsers suiting to this model and represented in an object repository. These parsers are to a large extent generated by the PDL (Parser Description Language) parser generator [7]. PDL extends the Yacc parser generator [25] by EBNF syntax and by notational support for compiling textual languages into TGraphs. A source code independent GReQL query facility, also working on the underlying TGraph model, is used for static source code analysis. The GUPRO MetaCARE system has been coded in C++ and runs under Sun Solaris and IBM AIX as well as under IBM OS/2.

The toolset for multi-language program understanding has been built by parameterizing the GUPRO system with the conceptual model defined in figure 4. Additionally, parsers for each language included in this model have been built as sketched in section 2. Analyzing the multi-language system as described in section 3 is performed by the general GReQL retrieval component. The multi-language GUPRO instance is currently being integrated into the software production and maintenance process at Aachener und Münchener.

Further GUPRO instance have been developed e. g. for supporting the understanding of ANSI85-COBOL [20] and C [14], both on a very fine-grained level of granularity.

This GUPRO C instance is used within the Reverse Engineering Demonstration Project [42]. An other GUPRO instance exists for analyzing the conceptual models used in the EER/GRAL-approach. Here GUPRO is applied to understanding the models used in GUPRO themselves.

5. Conclusions and future work

We presented an approach to program understanding in multi-language systems which has been used to represent the programming environment of the Aachener und Münchener insurance company. The work is formally based on the EER/GRAL modeling approach. In accordance with this, the single programming languages of a multi-language environment are modeled using an EER dialect together with the GRAL constraint language. Based upon common concepts the resulting models are combined into a single model using a general integration strategy. Stepwise parsing technology is used to fill a repository defined by the conceptual model with the varying source code information. Program comprehension is enabled on the repository using the GReQL query language. The GReQL queries presented make use of paths expressions that exactly represent the cross references between the concepts of different languages in the model. The queries are used at the Aachener und Münchener to easily retrieve multi-language cross references. As such, the tool will provide the maintenance engineers with increased productivity.

In addressing the special needs of the Aachener und Münchener, the GReQL querying facility has been extended with a form-based query interface. Ongoing, the query facility will be integrated with a browsing component to let the user combine query and navigation as desired. This component is also used for presenting query results in terms of source code instead of the tabular representation depicted in figure 5.

The multi-language program comprehension tool is just one instance of the GUPRO MetaCARE environment. In future, additional GUPRO instances will be developed. A fine-grained multi-language tool will be a desired goal. In this context new problems like different kinds of parameter exchange formats have to be encountered. But GUPRO and EER/GRAL seem to offer adequate support to this and upcoming problems.

References

[1] M. Behling. Ein Interpreter regulärer Pfadausdrücke. Studienarbeit S-525, Universität Koblenz-Landau, Fachbereich Informatik, Koblenz, Dezember 1997.

[2] G. D. Brown. *System 370/390 JCL.* Wiley, New York, 3 edition, 1991.

[3] G. Canfora, L. Mancini, and M. Tortorella. A Workbench for Program Comprehension during Software Maintenance. In A. Cimitile and H. A. Müller, editors, *Proceedings Fourth Workshop on Program Comprehension, March 29-31 1996, Berlin, Germany*, pages 30–39. IEEE Computer Society Press, Los Alamitos, 1996.

[4] P. P. Chen. The Entity–Relationship Model — Toward a Unified View of Data. *ACM Transactions on Database Systems*, 1(1):9–36, March 1976.

[5] Y.-F. Chen, M. Y. Nishimoto, and C. V. Ramamoorthy. The C Information Abstraction System. *IEEE Transactions on Software Engineering*, 16(3):325–334, March 1990.

[6] R. F. Crew. ASTLOG: A language for examining abstract syntax trees. In *Proceedings of the Conference on Domain-Specific Languages (DSL-97)*, pages 229–242, Berkeley, Oct.15–17 1997. USENIX Association.

[7] P. Dahm. PDL Reference. Technical Report (to appear), Universität Koblenz-Landau, Institut für Softwaretechnik, Koblenz, 1998.

[8] P. Dahm, J. Ebert, A. Franzke, M. Kamp, and A. Winter. TGraphen und EER-Schemata, formale Grundlagen. Projektbericht in Vorbereitung, Universität Koblenz-Landau, Institut für Softwaretechnik, Koblenz, 1998.

[9] P. Dahm, J. Ebert, and C. Litauer. Das EMS-Graphenlabor 3.0. Projektbericht, Universität Koblenz-Landau, Institut für Softwaretechnik, Koblenz, 1994.

[10] P. Dahm, J. Fricke, R. Gimnich, M. Kamp, H. Stasch, E. Tewes, and A. Winter. Anwendungslandschaft der Volksfürsorge. Projektbericht 5/95, Universität Koblenz-Landau, Institut für Softwaretechnik, Koblenz, 1995.

[11] P. T. Devanbu. GENOA – A Customizable, Language and Front-End independent Code Analyzer. *Proc. 14th International Conference on Software Engineering, Melbourne*, pages 307–317, 1992.

[12] J. Ebert and A. Franzke. A Declarative Approach to Graph Based Modeling. In E. Mayr, G. Schmidt, and G. Tinhofer, editors, *Graphtheoretic Concepts in Computer Science*, number 903 in LNCS, pages 38–50, Berlin, 1995. Springer.

[13] J. Ebert, R. Gimnich, and A. Winter. Wartungsunterstützung in heterogenen Sprachumgebungen, Ein Überblick zum Projekt GUPRO. In F. Lehner, editor, *Softwarewartung und Reengineering - Erfahrungen und Entwicklungen*, pages 263–275. Gabler, Wiesbaden, 1996.

[14] J. Ebert, B. Kullbach, and A. Panse. The Extract-Transform-Rewrite Cycle - A Step towards MetaCARE. In P. Nesi and F. Lehner, editors, *Proceedings of the 2nd Euromicro Conference on Software Maintenance & Reengineering*, pages 165–170, Los Alamitos, 1998. IEEE Computer Society.

[15] J. Ebert, A. Winter, P. Dahm, A. Franzke, and R. Süttenbach. Graph Based Modeling and Implementation with EER/GRAL. In B. Thalheim, editor, *15th International Conference on Conceptual Modeling (ER'96), Proceedings*, number 1157 in LNCS, pages 163–178. Springer, Berlin, 1996.

[16] Fjeldstad and Hamlen. Application Program Maintenance Study — Report to Our Respondents. IBM Coropration, DP Marketing Group, 1979.

[17] A. Franzke. GRAL: A Reference Manual. Fachbericht Informatik 3/97, Universität Koblenz-Landau, Fachbereich Informatik, Koblenz, 1997.

[18] J. E. Grass. Object-oriented design archaeology with CIA++. *Computing Systems*, 5(1):5–67, Winter 1992.

[19] S. Henninger. Case-Based Knowledge Management Tools for Software Development. *Journal of Automated Software Engineering*, 4(3), July 1997.

[20] M. Hümmerich. Entwicklung und prototypische Implementation eines konzeptionellen Modelles zum Reverse-Engineering von ANSI85-COBOL-Programmen. Studienarbeit S 380, Universität Koblenz-Landau, Fachbereich Informatik, Koblenz, Juni 1995.

[21] IBM, editor. *MVS/Extended Architecture JCL Reference*. International Business Machines Corporation, North York, Ontario, 5 edition, September 1989.

[22] IBM, editor. *VS-COBOL II, Application Programming Language Reference*. International Business Machines Corporation, San Jose, 8 edition, March 1993.

[23] S. Jarzabek. PQL: a language for specifying abstract program views. In W. Schafer and P. Botella, editors, *Proc. 5th European Software Engineering Conf. (ESEC 95)*, volume 989 of *Lecture Notes in Computer Science*, pages 324–341, Sitges, Spain, Sept. 1995. Springer-Verlag, Berlin.

[24] S. Jarzabek and T. P. Keam. Design of a Generic Reverse Engineering Assistant Tool. In L. Wills, P. Newcomb, and E. Chikofsky, editors, *Proceedings Second Working Conference on Reverse Engineering, July 14-16 1995, Toronto, Ontario, Canada*, pages 61–70. IEEE Computer Society Press, Los Alamitos, Cal., 1995.

[25] S. C. Johnson. YACC — Yet another compiler - compiler. Computing Science Technical Report No. 32, Bell Laboratories, Murray Hill, N.J., 1975.

[26] M. Kamp. GReQL - Eine Anfragesprache für das GUPRO–Repository 1.1. Projektbericht 8/96, Universität Koblenz-Landau, Institut für Softwaretechnik, Koblenz, Januar 1996.

[27] M. Kamp. Managing a Multi-File, Multi-Language Software Repository for Program Comprehension Tools – A Generic Approach. In *Proceedings Sixth International Workshop on Program Comprehension (IWPC'98), June 24-26, 1998, Ischia, Italy (to appear)*. IEEE Computer Society Press, 1998.

[28] M. A. Linton. Implementing Relational Views of Programs. *Proc. ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Developement Environments*, pages 132–140, May 1984.

[29] H. A. Müller, M. A. Orgun, S. R. Tilley, and J. S. Uhl. A Reverse Engineering Approach To Subsystem Structure Identification. *Journal of Software Maintenance: Research and Practice*, 5(4):181–204, Dec. 1993.

[30] J. Mylopoulos, A. Borgida, M. Jarke, and M. Koubarkis. Telos: Representing Knowledge About Information Systems. *ACM Transactions on Information Systems*, 8(4):325–262, October 1990.

[31] P. Newcomb. Legacy System Cataloging Facility. In L. Wills, P. Newcomb, and E. Chikofsky, editors, *Proceedings Second Working Conference on Reverse Engineering, July 14-16 1995, Toronto, Ontario, Canada*, pages 52–60. IEEE Computer Society Press, Los Alamitos, Cal., 1995.

[32] S. Paul and A. Prakash. Querying source code using an algebraic query language. In *Proceedings of the International Conference on Software Maintenance 1994*, pages 127–136. IEEE Computer Society Press, Sept. 1994.

[33] D. Polock. Ein statischer Optimierer für GRAL- und GReQL-Ausdrücke. Diplomarbeit D-414, Universität Koblenz-Landau, Fachbereich Informatik, Koblenz, September 1997.

[34] Rubin Systems Inc. The Worldwide Benchmark Project. Final Report 1997, 1998.

[35] P. Selfridge. Integrating Code Knowledge with a Software Information System. In L. Hoebel, editor, *Proceedings of the 1990 Knowledge-Based Software Engineering Conference*, pages 183–195, 1990.

[36] J. Sowa. *Conceptual Structures. Information, Processing in Mind and Machine*. The Systems Programming Series. Addison-Wesley, Reading, 1984.

[37] T. A. Standish. An essay on software reuse. *IEEE Transactions on Software Engineering*, SE-10(5):494–497, Sept. 1984.

[38] Unified Modeling Language. Notation Guide. Technical Report Version 1.1 alpha 6 (1.1 c), Rational Software Corporation, Santa Clara, 21 July 1997.

[39] G. M. A. Verheijen and J. van Bekkum. NIAM: An Information Analysis Method. In T. W. Olle, H. G. Sol, and A. A. Verrijn-Stuart, editors, *Information Systems Methodologies*, pages 537–589. North Holland, Amsterdam, 1982.

[40] C. H. Wells, R. Brand, and L. Markosian. Customized Tools for Software Quality Assurance and Reengineering. In L. Wills, P. Newcomb, and E. Chikofsky, editors, *Proceedings Second Working Conference on Reverse Engineering, July 14-16 1995, Toronto, Ontario, Canada*, pages 71–77. IEEE Computer Society Press, Los Alamitos, Cal., 1995.

[41] K. Wong, S. R. Tilley, H. A. Müller, and M.-A. D. Storey. Structural Redocumentation: A Case Study. *IEEE Software*, pages 46–54, January 1995.

[42] WorldPath Information Services. Reverse Engineering Demonstration Project. http://www.worldpath.com/reproject/, 1998.

Available Research Reports (since 1995):

1998

4/98 *Bernt Kullbach, Andreas Winter, Peter Dahm, Jürgen Ebert.* Program Comprehension in Multi-Language Systems.

3/98 *Jügen Dix, Jorge Lobo.* Logic Programming and Nonmonotonic Reasoning.

2/98 *Hans-Michael Hanisch, Kurt Lautenbach, Carlo Simon, Jan Thieme.* Zeitstempelnetze in technischen Anwendungen.

1/98 *Manfred Kamp.* Managing a Multi-File, Multi-Language Software Repository for Program Comprehension Tools — A Generic Approach.

1997

32/97 *Peter Baumgartner.* Hyper Tableaux — The Next Generation.

31/97 *Jens Woch.* A component-based and abstractivistic Agent Architecture for the modelling of MAS in the Social Sciences.

30/97 *Marcel Bresink.* A Software Test-Bed for Global Illumination Research.

29/97 *Marcel Bresink.* Deutschsprachige Terminologie des Radiosity-Verfahrens.

28/97 *Jürgen Ebert, Bernt Kullbach, Andreas Panse.* The Extract-Transform-Rewrite Cycle - A Step towards MetaCARE.

27/97 *Jose Arrazola, Jürgen Dix, Mauricio Osorio.* Confluent Rewriting Systems for Logic Programming Semantics.

26/97 *Lutz Priese.* A Note on Nondeterministic Reversible Computations.

25/97 *Stephan Philippi.* System modelling using Object-Oriented Pr/T-Nets.

24/97 *Lutz Priese, Yurii Rogojine, Maurice Margenstern.* Finite H-Systems with 3 Test Tubes are not Predictable.

23/97 *Peter Baumgartner (Hrsg.).* Jahrestreffen der GI-Fachgruppe 1.2.1 'Deduktionssysteme' — Kurzfassungen der Vorträge.

22/97 *Jens M. Felderhoff, Thomas Marx.* Erkennung semantischer Integritätsbedingungen in Datenbankanwendungen.

21/97 *Angelika Franzke.* Specifying Object Oriented Systems using GDMO, ZEST and SDL'92.

20/97 *Angelika Franzke.* Recommendations for an Improvement of GDMO.

19/97 *Jürgen Dix, Luís Moniz Pereira, Teodor Przymusinski.* Logic Programming and Knowledge Representation (LPKR '97) (Proceedings of the ILPS '97 Postconference Workshop).

18/97 *Lutz Priese, Harro Wimmel.* A Uniform Approach to True-Concurrency and Interleaving Semantics for Petri Nets.

17/97 *Ulrich Furbach (Ed.).* IJCAI-97 Workshop on Model Based Automated Reasoning.

16/97 *Jürgen Dix, Frieder Stolzenburg.* A Framework to Incorporate Non-Monotonic Reasoning into Constraint Logic Programming.

15/97 *Carlo Simon, Hanno Ridder, Thomas Marx.* The Petri Net Tools Neptun and Poseidon.

14/97 *Juha-Pekka Tolvanen, Andreas Winter (Eds.).* CAiSE'97 — 4th Doctoral Consortium on Advanced Information Systems Engineering, Barcelona, June 16-17, 1997, Proceedings.

13/97 *Jürgen Ebert, Roger Süttenbach.* An OMT Metamodel.

12/97 *Stefan Brass, Jürgen Dix, Teodor Przymusinski.* Super Logic Programs.

11/97 *Jürgen Dix, Mauricio Osorio.* Towards Well-Behaved Semantics Suitable for Aggregation.

10/97 *Chandrabose Aravindan, Peter Baumgartner.* A Rational and Efficient Algorithm for View Deletion in Databases.

9/97 *Wolfgang Albrecht, Dieter Zöbel.* Integrating Fixed Priority and Static Scheduling to Maintain External Consistency.

8/97 *Jürgen Ebert, Alexander Fronk.* Operational Semantics of Visual Notations.

7/97 *Thomas Marx.* APRIL - Visualisierung der Anforderungen.

6/97 *Jürgen Ebert, Manfred Kamp, Andreas Winter.* A Generic System to Support Multi-Level Understanding of Heterogeneous Software.

5/97 *Roger Süttenbach, Jürgen Ebert.* A Booch Metamodel.

4/97 *Jürgen Dix, Luis Pereira, Teodor Przymusinski.* Prolegomena to Logic Programming for Non-Monotonic Reasoning.

3/97 *Angelika Franzke.* GRAL 2.0: A Reference Manual.

2/97 *Ulrich Furbach.* A View to Automated Reasoning in Artificial Intelligence.

1/97 *Chandrabose Aravindan, Jürgen Dix, Ilkka Niemelä .* DisLoP: A Research Project on Disjunctive Logic Programming.

1996

28/96 *Wolfgang Albrecht.* Echtzeitplanung für Alters- oder Reaktionszeitanforderungen.

27/96 *Kurt Lautenbach.* Action Logical Correctness Proving.

26/96 *Frieder Stolzenburg, Stephan Höhne, Ulrich Koch, Martin Volk.* Constraint Logic Programming for Computational Linguistics.

25/96 *Kurt Lautenbach, Hanno Ridder.* Die Lineare Algebra der Verklemmungsvermeidung — Ein Petri-Netz-Ansatz.

24/96 *Peter Baumgartner, Ulrich Furbach.* Refinements for Restart Model Elimination.

23/96 *Peter Baumgartner, Peter Fröhlich, Ulrich Furbach, Wolfgang Nejdl.* Tableaux for Diagnosis Applications.

22/96 *Jürgen Ebert, Roger Süttenbach, Ingar Uhe.* Meta-CASE in Practice: a Case for KOGGE.

21/96 *Harro Wimmel, Lutz Priese.* Algebraic Characterization of Petri Net Pomset Semantics.

20/96 *Wenjin Lu.* Minimal Model Generation Based on E-Hyper Tableaux.

19/96 *Frieder Stolzenburg.* A Flexible System for Constraint Disjunctive Logic Programming.

18/96 *Ilkka Niemelä (Ed.).* Proceedings of the ECAI'96 Workshop on Integrating Nonmonotonicity into Automated Reasoning Systems.

17/96 *Jürgen Dix, Luis Moniz Pereira, Teodor Przymusinski.* Non-monotonic Extensions of Logic Programming: Theory, Implementation and Applications (Proceedings of the JICSLP '96 Postconference Workshop W1).

16/96 *Chandrabose Aravindan.* DisLoP: A Disjunctive Logic Programming System Based on PROTEIN Theorem Prover.

15/96 *Jürgen Dix, Gerhard Brewka.* Knowledge Representation with Logic Programs.

14/96 *Harro Wimmel, Lutz Priese.* An Application of Compositional Petri Net Semantics.

13/96 *Peter Baumgartner, Ulrich Furbach.* Calculi for Disjunctive Logic Programming.

12/96 *Klaus Zitzmann.* Physically Based Volume Rendering of Gaseous Objects.

11/96 *J. Ebert, A. Winter, P. Dahm, A. Franzke, R. Süttenbach.* Graph Based Modeling and Implementation with EER/GRAL.

10/96 *Angelika Franzke.* Querying Graph Structures with G^2QL.

9/96 *Chandrabose Aravindan.* An abductive framework for negation in disjunctive logic programming.

8/96 *Peter Baumgartner, Ulrich Furbach, Ilkka Niemelä .* Hyper Tableaux.

7/96 *Ilkka Niemelä, Patrik Simons.* Efficient Implementation of the Well-founded and Stable Model Semantics.

6/96 *Ilkka Niemelä .* Implementing Circumscription Using a Tableau Method.

5/96 *Ilkka Niemelä .* A Tableau Calculus for Minimal Model Reasoning.

4/96 *Stefan Brass, Jürgen Dix, Teodor. C. Przymusinski.* Characterizations and Implementation of Static Semantics of Disjunctive Programs.

3/96 *Jürgen Ebert, Manfred Kamp, Andreas Winter.* Generic Support for Understanding Heterogeneous Software.

2/96 *Stefan Brass, Jürgen Dix, Ilkka Niemelä, Teodor. C. Przymusinski.* A Comparison of STATIC Semantics with D-WFS.

1/96 *J. Ebert (Hrsg.).* Alternative Konzepte für Sprachen und Rechner, Bad Honnef 1995.

1995

21/95 *J. Dix and U. Furbach.* Logisches Programmieren mit Negation und Disjunktion.

20/95 *L. Priese, H. Wimmel.* On Some Compositional Petri Net Semantics.

19/95 *J. Ebert, G. Engels.* Specification of Object Life Cycle Definitions.

18/95 *J. Dix, D. Gottlob, V. Marek.* Reducing Disjunctive to Non-Disjunctive Semantics by Shift-Operations.

17/95 *P. Baumgartner, J. Dix, U. Furbach, D. Schäfer, F. Stolzenburg.* Deduktion und Logisches Programmieren.

16/95 *Doris Nolte, Lutz Priese.* Abstract Fairness and Semantics.

15/95 *Volker Rehrmann (Hrsg.).* 1. Workshop Farbbildverarbeitung.

14/95 *Frieder Stolzenburg, Bernd Thomas.* Analysing Rule Sets for the Calculation of Banking Fees by a Theorem Prover with Constraints.

13/95 *Frieder Stolzenburg.* Membership-Constraints and Complexity in Logic Programming with Sets.

12/95 *Stefan Brass, Jürgen Dix.* D-WFS: A Confluent Calculus and an Equivalent Characterization..

11/95 *Thomas Marx.* NetCASE — A Petri Net based Method for Database Application Design and Generation.

10/95 *Kurt Lautenbach, Hanno Ridder.* A Completion of the S-invariance Technique by means of Fixed Point Algorithms.

9/95 *Christian Fahrner, Thomas Marx, Stephan Philippi.* Integration of Integrity Constraints into Object-Oriented Database Schema according to ODMG-93.

8/95 *Christoph Steigner, Andreas Weihrauch.* Modelling Timeouts in Protocol Design..

7/95 *Jürgen Ebert, Gottfried Vossen.* I-Serializability: Generalized Correctness for Transaction-Based Environments.

6/95 *P. Baumgartner, S. Brüning.* A Disjunctive Positive Refinement of Model Elimination and its Application to Subsumption Deletion.

5/95 *P. Baumgartner, J. Schumann.* Implementing Restart Model Elimination and Theory Model Elimination on top of SETHEO.

4/95 *Lutz Priese, Jens Klieber, Raimund Lakmann, Volker Rehrmann, Rainer Schian.* Echtzeit-Verkehrszeichenerkennung mit dem Color Structure Code — Ein Projektbericht.

3/95 *Lutz Priese.* A Class of Fully Abstract Semantics for Petri-Nets.

2/95 *P. Baumgartner, R. Hähnle, J. Posegga (Hrsg.).* 4th Workshop on Theorem Proving with Analytic Tableaux and Related Methods — Poster Session and Short Papers.

1/95 *P. Baumgartner, U. Furbach, F. Stolzenburg.* Model Elimination, Logic Programming and Computing Answers.