Multi-Paradigm Programming through Graph Rewriting: Case for Support

Dr. J.R.W. Glauert
School of Information Systems, UEA
Norwich NR4 7TJ

1 Purpose

The project aims to demonstrate that extended notions of graph rewriting can act as the basis for practical implementations of multi-paradigm languages, based on largely declarative principles. The languages chosen will combine process and functional programming on the one hand and functional and concurrent logic programming on the other. The objective is to support implementations on both sequential and parallel architectures.

The precursor to the main project will be research to identify a core model of graph rewriting able to support the chosen language paradigms. This model will have restricted capabilities but will be chosen to allow efficient sequential implementation.

By extending established techniques, the project will develop an implementation for sequential machines, based on the core model, and focussing primarily on integration of functional and process paradigms.

The project will finally study truly parallel implementation of multi-paradigm languages. A study will search for primitives for concurrent computation based on graph rewriting. Implementation experiments will involve direct implementation of such primitives. Initially, task granularity and work distribution will mirror the process structures of user programs.

2 Background

2.1 Previous Work

Declarative Programming

Declarative languages have attracted considerable attention over the last decade because of their concise and powerful notation, theoretical clarity, practical benefits such as automatic heap space management, and perceived potential for exploiting implicit parallelism.

The promise has yet to be realised, partly through lack of acceptable implementations and programming environments, and partly due to the difficulty of coding portions of applications which do not fit naturally into a declarative style.

It can be argued that serious sequential implementations are now available. Logic programmers have a number of usable Prolog systems to choose from and many substantial programs exist. Lisp can be used declaratively, and now functional programmers are building serious systems using the New Jersey implementation of Standard ML. Those concerned to exploit lazy evaluation can use the Hope$^+$ language developed at ICSTM during the Flagship project or the Clean language [BruEL87] from the University of Nijmegen, based directly on graph rewriting principles. These systems produce code approaching the performance of corresponding algorithms programmed in C.

Nevertheless, pure declarative systems are only gaining limited acceptance. A barrier may be that they either refuse to countenance non-declarative styles, or accommodate them in ways which are inelegant and/or violate the theoretical basis which made the languages attractive in the first place. Clear problems exist concerning input-output, other interactions associated with an operating system, and program state.

Multi-Paradigm Programming

There are proposals to integrate functional and process programming models such as early work on FP2 [HufMM85]. More recently, Facile [GiaMP89,GiaMP90] is an experimental programming language resulting from a concrete attempt to integrate the typed call-by-value λ-calculus with a process language similar to CCS. Call-by-value λ-calculus and CCS have merged symmetrically to obtain a language that supports both functional and process abstractions: functions may be defined and used to specify internal computations of concurrent processes; dynamic process creation and communication over typed

channels may occur during any expression evaluation. Functions, processes and communication channels are first class values.

Practical work on Facile at ECRC is adding the primitives of Facile to the New Jersey Standard ML compiler. Work by Milner's group at Edinburgh is proposing similar extensions to SML. Such languages provide a natural interface to non-deterministic computation, while exploiting the strengths of functional programming for deterministic computation.

A new theory of mobile processes has been developed by Milner, Parrow and Walker [MilPW89]. It models the case of dynamic reconfiguration of communications between processes. In the original CCS, communication channels were fixed statically. The π-calculus is the underlying theory of mobile processes and involves interesting naming problems, since names of channels may be passed on channels to other processes, and may conflict with local names of other channels. This work builds on previous work by Engberg and Nielsen [EngN86].

In [Mil90] Milner studied the connection between the λ-calculus (in the form of the lazy λ-calculus and the call-by-value λ-calculus) and the π-calculus. It was established that it is possible to simulate functions using processes. In [Let91] Leth established a similar connection between her process calculus (LCCS) and the above variants of the λ-calculus as well as other representations of functional languages, such as combinators and director strings.

This work enables integrated languages such as Facile to be handled in a single theoretical framework, interpreting programs as networks of processes. The principal applicant is spending some months of study leave in 91/92 at ECRC where he will be work with the Facile group on such issues. An early result of this work [GlaLT91] is a new model of functions as processes which accommodates lazy and strict evaluation strategies.

Implementation through Graph Rewriting

Graph rewriting has been used as a technique for implementing functional languages. The origins of the graph rewriting approach are in the thesis by Wadsworth [Wad 71] where he introduces the technique as an optimisation of term rewriting; identical subterms are shared using pointers. It is now widely accepted as the standard way of implementing functional programming languages. Peyton-Jones [Pey87] provides a very comprehensive survey of the state-of-the-art. He highlights work on the G-Machine and the lazy language LML by the Programming Methodology Group at Chalmers University [Aug84, Joh84] which has produced a very efficient sequential implementation. A number of variants of the G-Machine have been proposed including [Pey91].

Probably the fastest lazy functional language implementation is the Concurrent Clean System [PlaEN91]. This language is based on the theory of Term Graph Rewriting (TGR) [BarEG87] which has been the focus of the Semagraph ESPRIT II BRA in which the applicant is involved.

Extended forms of TGR have been investigated using the Dactl notation [GlaKS90]. Kennaway and Glauert [Ken88, Ken90, Gla90] show that this notation can be used to implement functional languages, but more significantly, Dactl has been used to translate concurrent logic languages such as GHC, chosen as the basis for the kernel language of the Japanese Fifth Generation Computing Systems project [GlaP88] and to implement Standard ML [GlaHKP88], including the imperative features. Recent work addresses languages integrating functional and logic ideas [GlaP91].

Very recently, work by the Facile project at ECRC [GlaLT91] has shown that a useful subset of a process notation similar to the π-calculus can be mapped directly to a graph rewriting system using Dactl.

Parallel Implementation of Functional Languages

Despite the promise of functional languages for the exploitation of implicit parallelism, most schemes for parallel implementation of functional languages rely on annotations supplied by the programmer or provide a limited number of parallelism templates.

Early experiments with the ZAPP model focussed on the divide-and-conquer template [McBS87]. Recent, more general work on skeletons is reported in [DarFM91].

The Concurrent Clean system [PlaEN91] uses annotations, as does, in effect, early work on GRIP [HamP90]. A parallel implementation on Transputers has been developed on top of the ZAPP model [McBS91] at UEA. Impressive speedups can be achieved by these methods, although trial and error are still needed to select a good pattern of annotation.

Languages integrating functional and concurrent programming allow explicit expression of parallelism without requiring a separate layer of notation. One approach to implementation of such languages is to exploit concurrency if and only if the program generates processes. A more sophisticated approach would use congruences, or transformations between equivalent process representations with the same observable behaviour but different degrees of concurrency.

2.2 Research Experience

Dr. John Glauert joined the School of Information Systems at the University of East Anglia as Information Technology Lecturer in 1984 and has been Senior Lecturer in the department since January 1991.

He is member of the Declarative Systems Project is one of a number of research groups active at the University. Other leading members of the research group are Prof. Ronan Sleep, Dr. Richard Kennaway (SERC Advanced Fellow), and Dr. Simon Brock.

The current focus of the group is to provide contributions to knowledge about new generation languages and architectures through both the theory and practice of graph rewriting. Dr. Glauert was chief designer of the Dactl language and is involved in the Semagraph ESPRIT II Basic Research Action.

Recent research grants held are:

SemaGraph: ESPRIT II Basic Research Action. (Jul 89–Jun 92). Lead by UEA. In collaboration with CWI Amsterdam, Nijmegen University, ICSTM and ENS Paris. UEA portion 240kECU.

European Declarative System, ESPRIT II Technology Integration Project. (Jan 89–Dec 91). In collaboration with ICL, Bull, Siemens, ECRC. UEA portion 500kECU.

The Dactl Interface for New Generation Computing. SERC/Alvey. (May 86–Apr 89). In collaboration with ICL, ICSTM and Manchester University. £260k.

3 Programme

The objective of the proposed project is to explore the thesis that Extended Term Graph Rewriting techniques are a suitable basis for implementing Multi-Paradigm languages with a declarative flavour, yielding the expected benefit of providing an existence proof in terms of a usable implementation. It is to be expected that results on sequential implementation will be forthcoming and the applicant is confident that progress on parallel implementation techniques will be made.

The proposed programme of work breaks naturally into a number of phases which will run consecutively, though with some overlap.

3.1 Identifying a Core Graph Rewriting Model

An essential first step is to identify a simple but powerful model of graph rewriting on which implementation work may be based. With the applicant's background, it is to be predicted that this model will build on the Dactl notation. Unlike Dactl, the model would place considerable syntactic constraints on the forms of rewriting rules allowed and the graph structures manipulated would not be as free – it is intended to develop a proper notion of type. If possible, a fixed reduction strategy would be associated with the model, rather than the fine-grain control markings of Dactl.

Pointers to the properties required in this model come from earlier implementation studies on functional [Ken88, Ken90], logic [GlaP88, GlaP91], and process [GlaLT91] languages. The MONSTR model used by the FLAGSHIP project [Ban88, Ban89] can be seen as an early attempt to identify such a model. There are also many features in common with Paragon [BolHK90, BolHK91] which was developed to address some perceived shortcomings in existing rewrite notations which make it difficult to represent the message passing and concurrency.

There is an emerging common core of features suggested by these studies so it is believed to be a reasonably short term task to develop the model.

3.2 Sequential Implementation Techniques for Multi-Paradigm Languages

A study of sequential implementation techniques will be made and an implementation produced of a simple multi-paradigm language, focussing initially on functional and process paradigms.

For pure functional languages, sequential techniques based on graph rewriting are well advanced and will be exploited wherever possible. Studies of the abstract machines on which implementations are based do

not always reveal the expected results [Kin90], however, since the true success of some implementations depends on many subtle optimisations which are not always reported in the literature.

While direct evaluation of a graph rewriting model may involve large numbers of very small tasks, each performing one rewrite, successful implementations elide long sequences of rewrites and minimise the number of accesses to the graph heap. For functional implementations, suspension of tasks can be avoided. This will not be possible entirely when considering non-deterministic languages which will require at least pseudo-parallelism for implementation.

The existing implementations of the full Dactl language include a reference interpreter, an occam-based compiler running on multiple transputers, and a sequential compiler generating C code [KinG91]. It is intended to build on the latter implementation during this phase.

One target machine would be single-transputer systems as a step towards parallel experiments in later phases.

3.3 Implementation Primitives for Concurrent Computation

The project will aim to exploit concurrency using processes with an intermediate granularity. These must be large enough to amortise costs of process creation and communication, but not so large that non-determinism is excluded and work distribution problems arise.

It will still be necessary to have very efficient process creation, communication, and access to remote memory. This phase of the project will examine the core model, aiming to identify appropriate implementation primitives for parallel execution, making minor adjustments to the model if required.

Dally and Wills [Dal89] propose some concurrency primitives and recent work on message passing architectures will be studied.

3.4 Experimental Parallel Implementation of Multi-Paradigm Languages

Using available parallel architecture, principally the Meiko Computing Surface at UEA, experiments will be performed to implement the concurrency primitives. A parallel implementation will be constructed to test the suitability of the approach. The project is not about new hardware design and will use current commercial parallel machines as its target.

In the first instance, task granularity and work distribution will mirror the process structures of user programs. This will have been reflected in the process networks used in the sequential implementation. It is expected that any approach which would attempt to exploit more implicit parallelism – or discard spurious explicit parallelism – would begin by transforming source programs.

3.5 Research Techniques

The project will be based in the School of Information Systems at the University of East Anglia. It will involve a research assistant working under the direction of the applicant, Dr. John Glauert.

The project will form part of the work of the Declarative Systems Project and as such will benefit from interaction with the work of Prof. Ronan Sleep and Dr. Richard Kennaway and Dr. Simon Brock on other ESPRIT and SERC projects. This will be both in the area of theoretical research into graph rewriting and practical expertise in implementation of novel architectures on Transputers.

The project will exploit the best compiler technology for sequential implementation of declarative languages.

The project will adopt the same approach to computing infrastructure as the rest of DSP, using networked Macintosh II workstations and word processing facilities connected to central Unix servers. Access to the UEA Meiko Computing Surface will enable serious experimentation with multi-processor implementations.

4 Resources.

The proposal is for a programme of work lasting 36 months.

4.1 Staff.

Support is requested for a postgraduate research worker at level RA 1A. The work involves a good deal of theoretical and language design work requiring an experienced researcher able to take initiative.

4.2 Travel.

Travel funds are requested to enable research visits to other groups engaged in related work and for attendance at specialist workshops. Current examples are the Dagstühl Seminar on Compiler Technology and Parallelism for Functional Languages (See [HanW91]) and International Workshops on the Parallel Implementation of Functional Languages (See [Pla90, GlaH91]). It will also be very desirable to visit ECRC which has an interest in the results of this work.

International conferences of significance for this work are: FPCA, the Lisp & Functional Programming conference, and PARLE.

Funds are requested for one journey a year to Europe, and for a visit to the US during the lifetime of the project for both the RA and principal applicant,.

4.3 Equipment.

Equipment requirements are dictated by the nature of the project, and by local infrastructure policy.

A networked Macintosh II workstation, is requested for the use of the RA. It will require 8 to 16 MBytes of main memory to support a declarative development environment such as New Jersey ML. The most cost effective model cannot be determined at this stage as new products are on the point of announcement. Indications suggest that a suitable system will cost around £6500.

Access will be possible via Ethernet to the UEA Meiko system and to standard network services such as Email and News.

4.4 Other

Maintenance is requested for the equipment supporting this project calculated at 8% of purchase cost per annum. Consumables and other charges, including laserwriter cartridges, magnetic media , and dial-up, etc. have been costed at £340 per year per person.

5 Applications, Collaboration, and Exploitation

A multi-paradigm approach based on linking declarative and process languages makes it possible for components of a complete software system to exploit the most appropriate paradigm for the purpose. Use of declarative languages will be encouraged by the provision of an environment in which their limitations could be overcome.

Since the model concerned would support concepts of process creation and communication, there is the promise of effective distributed and parallel implementations. The algorithms expressed in the language would allow dynamic process configurations to be created, extending the range of applications which could benefit from parallel execution.

The applicant is engaged in collaboration with ECRC on the Facile project which is interested in some of the issues addressed in this proposal.

References

[Aug84] L. Augustsson, A compiler for lazy ML, in: Proc. ACM Symposium on Lisp and Functional Programming, pp 218-227, 1984.

[Ban88] R. Banach, & P. Watson, Dealing with State on Flagship: the MONSTR Computational Model, Proceedings CONPAR 88, September 1988.

[Ban89] R. Banach, Dataflow Analysis of Term Graph Rewriting Systems, Proceedings PARLE'89, Volume II, LNCS 366, pp 55-72, 1989.

[BarEG87] H.P. Barendregt, M.C.J.D. van Eekelen, J.R.W. Glauert, J.R. Kennaway, M.J. Plasmeijer, & M.R. Sleep, Term graph rewriting, Proc. PARLE conference, LNCS 259, pp 141–158, 1987.

[BolHK90] D. Bolton, C. L. Hankin and P. H. J. Kelly, Parallel object-oriented descriptions of graph reduction machines, Future Generation Computer Systems 6, pp 225-239, 1990.

[BolHK91] D. Bolton, C. L. Hankin and P. H. J. Kelly, An Operational Semantics for Paragon: A Design Notation for Parallel Architectures, New Generation Computing, 9 pp 171-197, 1991

[BruEL87] T. Brus, M.C.J.D. van Eekelen, M.O. van Leer, & M.J. Plasmeijer, Clean: a language for functional graph rewriting, Proc. Third Int. Conf. on Functional Programming Languages and Computer Architectures, Oregon, USA, LNCS 274, pp 364-384, 1987.

[Dal89] W.J. Dally and D.S. Wills, Universal Mechanisms for Concurrency. Proc. PARLE'89. LNCS 365, pp 19-33, 1989.

[DarFM91] J. Darlington, A.J. Field, P.G. Harrison, D. Harper, G.K. Jouret, P.J. Kelly, K.M. Sephton, & D.W. Sharp, Structured Parallel Functional Programming, in [GlaH91], 1991.

[EngN86] U. Engberg, & M. Nielsen, A calculus of communicating systems with label passing, Report DAIMI PB--208, Computer Science Department, University of Åarhus, 1986.

[GiaPM89] A. Giacalone, P. Mishra, & S. Prasad, Facile: A Symmetric Integration of Concurrent and Functional Programming, IJPP, Vol 18, No 2, pp 121-160, 1989.

[GiaPM90] A. Giacalone, P. Mishra, & S. Prasad, Operational and Algebraic Semantics for Facile: A Symmetric Integration of Concurrent and Functional Programming, in Proceedings of ICALP 90, LNCS 443, pp. 765-780, 1990.

[Gla90] J.R.W. Glauert , Compiling Functional Languages Based On Graph Rewriting, in [Pla90], 1990.

[GlaH91] H. Glaser, & P. Hartel, (editors), Proc. 3rd International Workshop on the Parallel Implementation of Functional Languages, Technical Report Series, CSTR 91 91-07, Department of Electronics and Computer Science, University of Southampton, June 1991.

[GlaHKP88] J.R.W. Glauert, K. Hammond, J.R. Kennaway, and G.A. Papadopoulos. Using Dactl to Implement Declarative Languages. Proceedings CONPAR 88, September 1988.

[GlaKS90] J.R.W. Glauert, J.R. Kennaway, & M.R. Sleep, Dactl: an experimental Graph Rewriting language, in Proc. 4th International Workshop on Graph Grammars and Their Application to Computer Science, Bremen, March 1990. LNCS, to appear, 1991.

[GlaLT91] J.R.W. Glauert, L. Leth, & B. Thomsen, A New Translation of Functions as Processes, University of East Anglia, 1991.

[GlaP88] J.R.W. Glauert & G.A. Papadopoulos, A Parallel Implementation of GHC, Proceedings, International Conference on Fifth Generation Computer Systems 1988. ICOT, Tokyo, December 1988.

[GlaP91] J.R.W. Glauert & G.A. Papadopoulos, Unifying Concurrent Logic and Functional Languages in a Graph Rewriting Framework, Proceedings, 3rd Panhellenic Computer Science Conference, Athens, May 1991.

[HufMM85] J.-M. Hufflen, A. Marty, J.-Ch. Marty, & Ph. Schnoebelen, FP2: the language and its formal definition, Report RR LIFIA 26, Laboratoire d'Informatique Fondamentale et d'Intelligence Artificielle, 1985.

[HamP90] K. Hammond, & S.L. Peyton Jones, Parallel Haskell: the GRIP implementation, in [Pla90], 1990.

[HanW91] C. Hankin, & R. Wilhelm, (editors), Functional Languages: Optimization for Parallelism, Dagstuhl-Seminar-Report 3, Universität des Saarlandes, Saarbrücken, Germany, 1991.

[Joh84] T. Johnsson, Efficient compilation of lazy evaluation, in: Proc. ACM Conference on Compiler Construction, pp 58–69, 1984.

[Ken90] J.R. Kennaway, Implementing Term Rewrite Languages in Dactl, Theo. Comp. Sci., v.72, pp 225-250, 1990

[Ken88] J.R. Kennaway, The correctness of an implementation of functional Dactl by parallel rewriting, UK IT 88 Conference Publication, IEE, 1988.

[Kin90] I. King, The Efficiency and Generalisation of Various Abstract Machines, in [Pla90], 1990.

[KinG91] I. King I. and J.R.W. Glauert, Generating Native Code for a Generalised Graph Rewriting Language, University of East Anglia, 1991.

[Let91] L. Leth, Functional Programs as Reconfigurable Networks of Communicating Processes, Ph.D Thesis, Imperial College, London University, 1991.

[McBS87] D.L. McBurney & M.R. Sleep. Transputer-based experiments with the ZAPP architecture. Proc. PARLE conference, LNCS 259, Springer Verlag 1987.

[McBS91] D.L. McBurney & M.R. Sleep, Graph Rewriting as a Computational Model, in: Concurrency: Theory, Language and Architecture, ed Yonezawa A. and Ito T., LNCS 491, 1991.

[Mil90] R. Milner, Functions as Processes, Technical Report 1154, INRIA Sophia Antipolis, February 1990.

[MilPW89] R. Milner, J. Parrow, & D. Walker, A Calculus of Mobile Processes, Parts I and II, TR ECS-LFCS-89-85, Edinburgh University, June 1989

[Pey87] S.L. Peyton Jones, The Implementation of Functional Languages, Prentice-Hall, London, 1987.

[Pey91] S.L. Peyton Jones, The spineless tagless G-machine: a second attempt, in [GlaH91], 1991.

[Pla90] M.J. Plasmeijer, (editor), Proc. 2nd International Workshop on Parallel Implementation of Functional Languages, TR90-16, Univ. Nijmegen, Oct 1990.

[PlaEN91] M.J. Plasmeijer, M.C.J.D. van Eekelen, E.G.J.M.H. Nöcker, & J.E.W. Smetsers, The Concurrent Clean System — Functional Programming on the Macintosh, Proceedings of the 7th Int. Conf. of the Apple European University Consortium, Paris 1991., Summarized in: Wheels for the Mind, an Apple University Publication, Volume 5, Number 3, Fennite Publications Ltd, London, UK, 1991.

[Wad71] C.P. Wadsworth Semantics and pragmatics of the lambda-calculus, Ph.D. thesis, University of Oxford, 1971.