Multi-Paradigm Programming through Graph Rewriting
Final Report of EPSRC Grant GR/H 41300

John Glauert
jrwg@sys.uea.ac.uk
School of Information Systems
University of East Anglia
Norwich NR4 7TJ, UK

1 Introduction

The aim of the project was to explore the usefulness of graph rewriting as an implementation technique for multi-paradigm languages focussing mainly, but not exclusively, on declarative programming styles. The project built on experience using *Extended Term Graph Rewriting* [25] to implement a range of programming languages and aimed for a secure theoretical framework for combining these languages, linked to an efficient implementation model.

The stated objectives were:

- Identification of a core implementation model based on graph rewriting
- Sequential implementation of multi-paradigm languages
- Identification of primitives for concurrent computation by graph rewriting
- Parallel implementation experiments reflecting program-specified process allocation

All of these objectives were met successfully. Due to the eventual timing of the project and the strengths of the RA appointed, the balance of work was adjusted to some extent. Some unexpected contributions arose from the project, leading to further work in progress:

- Relating process calculi and graph rewriting
- Theoretical foundations of multi-paradigm programming

Below we compare progress on the project with the original plan and then discuss in more detail the contributions made. Finally we outline plans for the future of the work.

As a result of the project, 9 papers have been published in international media, 6 internal reports have been produced (some overlap), and 2 papers have been submitted or are in the late stages of preparation. The web page http://www.sys.uea.ac.uk/~jrwg/MultiPar/multipar.html will provide further details and point to many of the papers referenced below.

2 Project Progress

The project was proposed initially in October 1991 and approved after minor clarification to start in July 1992. The project started formally in November 1992.

Two unsuccessful rounds of interviews were held to employ an RA who would combine the desired practical expertise in compiling technology with skills in theoretical computer science.

Fortunately it was possible to employ Dr. Zurab Khasidashvili, a very gifted researcher into rewriting theory, from July 1993. Dr. Khasidashvili brought expertise which enabled the project to deliver new results on the theoretical foundations of multi-paradigm programming, while the focus on practical implementation was maintained by the principal investigator and his research students.

Towards the end of the project, Dr. Richard Kennaway was employed to continue some joint work started with Dr. Khasidashvili.

Core Implementation Model

A model known as *Object Graph Rewriting* (OGR) was developed as planned. Early work used Dactl [3] to simulate OGR, enabling the key design features to be identified.

Sequential Implementation of Multi-Paradigm Languages

The project focussed on the implementation of Facile [2], which combines functional and concurrent programming. The SML/NJ compiler from Bell Labs was used to implement a translator from Facile to OGR and thence to C. This system is known as OGRe for *Object Graph Rewriting Environment* and was under continuous development from early on in the project. Indeed it initially supported an earlier core model [6] which was rejected in favour of OGR.

Primitives for Concurrent Computation

Towards the end of the project a design was produced for the *Parallel OGRe Machine* (POM) [12]. The design for POM was produced quickly, but was successful because a key design aim of OGR had been to allow highly efficient sequential evaluation while not hindering parallel implementation.

Parallel Implementation Experiments

The initial vehicle for implementing POM was PVM running on a network of conventional Linux workstations. This provided a more straightforward development environment than the proposed Meiko Computing Surface. Naturally the communication costs involved in PVM are too high to provide a real test of POM so a Transputer implementation is proposed for further work.

Process Calculi and Graph Rewriting

In order to develop a semantics for OGR, work was done to simulate the model using π-calculus. The same conclusions were reached as in similar work conducted independently elsewhere that a small sublanguage, *mini asynchronous π-calculus*, is adequate for all practical purposes. With Thomsen and Leth, originally at ECRC, we are developing a new calculus, the π^{π}-calculus, based on direct communication between processes. This calculus is very closely related to OGR.

Theoretical Foundations of Multi-Paradigm Programming

A key issue in the implementation of multi-paradigm languages is the relationship between their models of computation expressed in terms of evaluation order and the forms of results expected. A very productive strand of work was pursued leading to a general theory of relative normalisation for rewriting systems.

3 Scientific and Technological Contributions of the Project

OGR and the π^π-calculus

Milner [24] showed that a "mini π-calculus" could be used to encode the lazy and strict λ-calculus. Honda and Tokoro [15] and Boudol [1] investigated an asynchronous calculus leading to the "mini asynchronous π-calculus" which has been the focus of much recent research.

In [4], Glauert developed a form of polyadic asynchronous calculus exchanging terms. The model was implemented using a graph rewriting system in which graph terms represented both processes and the names, or channels, used for communication. Hence several rewrites modelled a single communication.

This work was taken up in the project [6] with the aim of translating both the concurrent and functional aspects of the language to small processes. It was observed that all the properties of mini asynchronous π-calculus applied, and, furthermore, the resulting processes used communication channels in a very restricted way. A new model called π^π-calculus [9] was developed in conjunction with Thomsen and Leth, though work on this is still in progress. In π^π-calculus, processes are named and messages are directed to processes. There is no independent concept of channels.

The OGR model used as the core language by the project is strongly related to the π^π-calculus. The theoretical basis of the language can be described as multiset rewriting of graph terms but an alternative view is to see certain terms as process descriptors, while other terms represent messages sent to those processes. A communication becomes a single rewrite. OGR appears in an early form in [5] and more recently in [12] which discusses implementation details.

Relative Normalisation

Investigations into evaluation strategies for multi-paradigm languages underlined that evaluation to a number of forms was of interest: normal forms; head-normal forms; and weak head-normal forms for example. The literature revealed a range of specific treatments of normalisation by neededness for different rewriting systems and to different forms.

Based on Expression Reduction Systems (ERS) [17], higher-order rewriting systems which encompass both the λ-calculus and term rewriting, a general theory of relative neededness was developed [7, 13] (also in [8]). Further work on Conditional ERS was undertaken in collaboration with van Oostrom and is reported in [18] (and [19]).

The theory was further extended to a more abstract framework which applies also to graph rewriting and other rewriting notions [10, 20]. Novel work was done on the semantics of such systems based on event structures and on abstract models of sharing [22], capturing such concepts as redex families in the λ-calculus.

Our framework for abstract reduction has proven particularly fruitful, and with Kennaway we are working to develop a sound notion of transfinite abstract reduction [16].

OGRe and POM Implementation

Facile was used as the multi-paradigm language to prove the effectiveness of OGR. The target language chosen was C in order to achieve independence from any particular hardware platform.

As the functional part of Facile assumes the same model of computation as Standard ML, previous implementations have been based on modified versions of the SML/NJ compiler. OGRe, the implementation testbed for OGR, was also written in SML and links to the SML/NJ Lambda level by enhancing the compiler. This enables parsing, type-checking, and many optimizations, to be performed using the existing compiler code.

Input to OGRe can be a Facile program (using SML syntax), or a representation of a λ-expression, or a set of OGR rules. Output is generally as (optimised) OGR, or C. It is also possible to generate Dactl, or LaTeX. In addition, OGRe provides a simulation of OGR programs for debugging purposes.

Translation of Facile programs can lead to inefficient OGR rules. A range of optimizations are employed during compilation, leading to larger threads of computation than might be produced by the initial OGR code.

Even on a sequential machine it is necessary for the OGRe runtime system to simulate concurrency. However, this is done by maintaining a very simple task queue which is only employed when multiple threads are spawned or a sequential thread dies. Generated C code makes extensive use of macro definitions. A portable garbage collector was written as a student project [14].

OGR was designed to be amenable to parallel implementation without sacrificing efficiency on sequential machines. Scoping rules allow common subexpressions in data terms to be shared or copied as desired, without observable effect. Since OGR processes cannot examine the contents of other processes, it is possible to make the POM housekeeping operations (which coordinate a distributed implementation) appear like standard OGR processes. Messages sent to these special processes take the normal form, but are handled by built-in system "rules" which will send messages to remote processors as desired [12].

POM is integrated seamlessly with the sequential implementation so that negligible costs are paid if a parallel program is executed on a single machine. This is because marking a task with the potential to be exported to a remote processor causes almost no overhead if the task is in fact executed locally.

4 Conclusions and Further Work

The project has achieved a great deal but many new opportunities for research have become apparent.

The feasibility of OGR as an implementation technique for multi-paradigm languages has been established. However, due to the unpredicted development of the project in the area of theoretical work, there is still scope for further optimisation and thorough comparative testing of the sequential implementation of OGR with implementations using other techniques.

Although nothing more than experiments were planned for POM, it is very attractive to pursue the parallel implementation, especially by using a more closely-coupled system - either a shared-memory multiprocessor or a distributed memory processor with high bandwidth message passing communication.

The strand of theory which has been developed has led to some work mentioned before which has been submitted or is under development [22, 16, 9]. Further funding is being sought from EPSRC to develop this work further.

OGR has a concept of objects, but cannot be said to be truly object-oriented. A research student, Lee Jeong-Ho, is developing such an object-oriented language for programming in the OGR style.

The flexible and portable memory management scheme used to implement OGR makes it possible to consider developing the model as a communication and process management interface for applications written in conventional languages. An application has been made to the EC INCO Keep-In-Touch programme and is under consideration at present. This would use results from the present project to develop concurrent multimedia systems.

It is interesting to note that VRML2 provides a graphical representation of scenes and interactions which could benefit from some graph rewriting techniques developed by this project.

Follow-on funding in these areas may be sought from EPSRC.

The papers below marked "• " were produced in connection with the project.

References

[1] G. Boudol: *Asynchrony and the π-calculus*, INRIA Report 1702, INRIA Sophia-Antipolis, 1992.

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

[3] J.R.W. Glauert, J.R. Kennaway, and M.R. Sleep: *Dactl: An Experimental Graph Rewriting Language*, Proceedings, 4th International Workshop on Graph Grammars, Bremen, 1990. Springer LNCS, Vol. 532, Springer-Verlag, 1991.

[4] • J.R.W. Glauert: *Asynchronous Mobile Processes and Graph Rewriting*, Proceedings PARLE'92, Champs Sur Marne, Paris, June 1992, Springer LNCS, Vol. 605, 1992.

[5] • J.R.W. Glauert: *Parallel Implementation of Functional Languages Using Small Processes*, Proceedings, International Workshop on Parallel Implementation of Functional Languages, RTWH Aachen, September 1992.

[6] • J.R.W. Glauert, L. Leth, and B. Thomsen: *A New Process Model for Functions*, in [25], Chapter 18, Wiley, 1993.

[7] • J.R.W. Glauert and Z. Khasidashvili: *Relative normalization in orthogonal expression reduction systems.* International workshop on conditional (and typed) term rewriting systems, CTRS'94, Springer LNCS, vol. 968, N. Dershowitz, N. Lindenstrauss eds. Jerusalem, p144, 1994.

[8] • J.R.W. Glauert and Z. Khasidashvili: *Minimal and Optimal relative normalization in orthogonal expression reduction systems.* Report SYS-C94-06. UEA Norwich, 1994.

[9] • J.R.W. Glauert, Z. Khasidashvili, L. Leth, and B. Thomsen: *Syntax and Semantics of the π^π-Calculus*, Report SYS-C95-10. UEA Norwich, 1995.

[10] • J.R.W. Glauert and Z. Khasidashvili: *Relative normalization in Deterministic Residual Structures.* In: Proc. of the 19th International Colloquium on Trees in Algebra and Programming, CAAP'96, Springer LNCS, Vol. 1059, p180, 1996.

[11] • J.R.W. Glauert and Z. Khasidashvili: *Relative normalization in stable deterministic residual structures.* Report SYS-C96-01, UEA Norwich, 1996.

[12] • J.R.W. Glauert: *Parallel Implementation through Object Graph Rewriting*, Proceedings, International Workshop on Parallel Implementation of Functional Languages, Bonn, September 1996.

[13] • J.R.W. Glauert and Z. Khasidashvili: *Minimal relative normalization in orthogonal expression reduction systems.* In Proc. of the 16th International Conference on Foundations of Software Technology and Theoretical Computer Science, FST&TCS'96, Springer LNCS, to appear.

[14] • D. Green: *Parallel Implementation of an Object Graph Rewriting Model*, CMP-3P4Y Project, UEA Norwich, 1996.

[15] K. Honda and M. Tokoro: *An object calculus for asynchronous communication*, Proc. ECOOP 91, Springer LNCS, Vol 512, p133, 1991.

[16] • J.R. Kennaway, Z. Khasidashvili, J.R.W. Glauert, and M.R. Sleep: *Transfinite abstract reduction*, In Preparation, 1996.

[17] Z. Khasidashvili: *On higher order recursive program schemes*, Proc. CAAP'94, Springer LNCS, Vol. 787, 1994.

[18] • Z. Khasidashvili and V. van Oostrom: *Context-sensitive conditional expression reduction systems.* In proc. of the International Workshop on Graph Rewriting and Computation, SEGRAGRA'95. In Electronic Notes in Computer Science, Elsevier Science B.V., p141, 1995.

[19] • Z. Khasidashvili and V. van Oostrom: *Context-sensitive Conditional Rewrite Systems.* Report SYS-C95-06. UEA Norwich, 1995.

[20] • Z. Khasidashvili and J. R. W. Glauert: *Discrete Normalization and Standardization in Deterministic Residual Structures.* In proc. of the 5^{th} International Conference on Algebraic and Logic Programming, ALP'96, Springer LNCS, vol. 1139, p135, 1996.

[21] • Z. Khasidashvili and J.R.W. Glauert: *Discrete normalization and standardization in stable deterministic residual structures.* Report SYS-C96-02, UEA, Norwich, 1996.

[22] • Z. Khasidashvili and J. R. W. Glauert: *Zig-zag and extraction families in stable non-duplicating Deterministic Residual Structures.* Submitted.

[23] R. Milner: *The Polyadic π-Calculus: A Tutorial*, Technical Report ECS-LFCS-91-180, Edinburgh University. 1991.

[24] R. Milner: *Functions as processes*, Mathematical Structures in Comp. Science, Vol 2, p119, 1992.

[25] M.R. Sleep, M.J. Plasmeijer, and M.C.J.D. van Eekelen (eds): *Term Graph Rewriting: Theory and Practice*, Wiley, 1993.