Mobile
Resource
Guarantees

IST-2001-33149 (MRG)

Annual report for year 1

14th January 2003

Funded by the European Community's "Information Society Technologies" Programme (1998–2002) under the FET proactive initiative on Global Computing.

Contents

Part I

Executive Summary

I.1 Management Information

I.1.a Manpower resources

The following personnel have been employed at the two sites during 2002:

Edinburgh
Lennart Beringer: 1 Jan – 30 Jun and 1 Oct – 31 Dec 2002 (9 months)
Michal Konečný: 1 November – 31 December 2002 (2 months)
Kenneth MacKenzie: 1 Jan – 30 Jun and 1 Oct – 31 Dec 2002 (9 months)

Munich
Roberto Amadio: 1 Aug – 31 Dec 2002 (5 months)
Hans-Wolfgang Loidl: 1 Aug – 31 Dec 2002 (5 months)
Olha Shkaravska: 18 Jan – 31 Dec 2002 (11.5 months)

This gives a total of 20 man-months of work at Edinburgh and 21.5 man-months in Munich; the plan was for 30 man-months at Edinburgh and 24 man-months at Munich. The manpower available in 2002 was therefore 77% of what was planned. The shortfall is mainly due to delays in appointing staff (all except Beringer and MacKenzie). Another factor was assignment of Beringer and MacKenzie for three months each to a closely related EPSRC project, the one that sparked the original MRG proposal, which finished in October. Although they were not available to MRG during that period, they gained experience that will prove valuable in their work for MRG.

If existing research staff serve to the end of their contracts, the situation in 2003 and 2004 will be as follows:

2003: Edinburgh 36 man-months, Munich 25 man-months
2004: Edinburgh 36 man-months, Munich 24 man-months

This gives a total for 2002–2004 of 92 man-months for Edinburgh, against a planned total of 90 man-months, and 70.5 man-months for Munich, against a planned total of 72 man-months; overall, the total is 162.5 man-months, almost exactly the same as the planned total of 162 man-months.

The allocation of manpower of staff employer on MRG to workpackages during 2002 has been as follows:

WP1: Edinburgh 12 mm, Munich 0 mm
WP2: Edinburgh 2 mm, Munich 9.5 mm
WP3: Edinburgh 3.5 mm, Munich 5.5 mm
WP4: Edinburgh 2 mm, Munich 4 mm
WP5: Edinburgh 0 mm, Munich 1 mm
WP6: Edinburgh 0 mm, Munich 0 mm
WP7: Edinburgh 0 mm, Munich 0 mm
WP8: Edinburgh 0 mm, Munich 1.5 mm
WP9: Edinburgh 0 mm, Munich 0 mm
WP10: Edinburgh 0 mm, Munich 0 mm
WP11: Edinburgh 0.5 mm, Munich 0 mm

I.1.b Tasks completed

There is one deliverable associated with each task, so it is enough to give the deliverables scheduled for 2002 together with an indication of which have been completed. Much more detail can be found in Section II.1 below.

Del. no.	Deliverable name	WP no.	Est. person-months	Del. type	Delivery (proj. month)	Complete?
D1a	Definition of virtual machine platform	1	2	report	0	yes
D1b	Cost model	1	6	report	2	yes
D1d	Comparison of JVML with .NET for project use	1	3	report	3	yes
D1f	Representative examples for project use	1	2	report	3	yes
D2a	Language of assertions for bytecode logic	2	2	report	4	yes
D2b	Proof rules for bytecode logic	2	3	report	6	no
D3a	Definition of experimental high-level language	3	1	report	1	yes
D3b	Compiler for high-level language	3	4	prototype	4	yes
D3e	Extend compiler with optimisations	3	3	prototype	10	no
D4a	Reasoning principles for resource usage	4	6	report	7	yes
D4b	Type system for space-like resources	4	4	report	11	no
D5a	Type system for expressing limits on parameter values	5	4	report	11	no
D10a	Practical effectiveness of mobile virtual machines	10	4	report	9	no
D11a	Project website	11	1	website	3	yes
D11b	Kickoff workshop	11	0	workshop	2	yes
D11c	Workshop at end of year 1	11	0	workshop	11	yes
D11f	Measurable criteria of progress/success	11	0	report	6	yes
D11g	Assessment of progress in year 1	11	0	report	12	yes
D11j	Dissemination and use plan	11	0	report	6	yes

Workpackage 1 is thereby completed, apart from work required to polish results for publication.

With the exception of the late delivery of deliverables planned for 2002 that have not yet been completed, no changes to the project workplan in the Technical Annex are foreseen.

I.1.c Partner cooperation and coordination

A one-day kickoff workshop took place in May 2002 at the National e-Science Centre in Edinburgh, attended by all MRG personnel from Edinburgh and Munich, two of the three additional researchers whose employment began later in the year (Amadio and Konečný), students associated with MRG in Edinburgh, and external experts Peter Lee from Carnegie Mellon University and Martín Abadi from the University of California at Santa Cruz, Lee's travel and expenses being paid by MRG.

A three-day internal workshop took place in September 2002 in Bad Wiessee near Munich, attended by all MRG personnel from Edinburgh and Munich and Antoine Galland from the Gemplus Software Research Lab in Marseille as invited speaker.

Additional research visits between the sites were as follows:

Beringer to Munich, 30 Sep – 3 Oct 2002
Loidl to Edinburgh, 10–19 Dec 2002
MacKenzie to Munich, 30 Sep – 2 Oct 2002
Sannella to Munich, 8–10 Dec 2002
Shkaravska to Edinburgh, 23–30 May 2002
Stark to Munich, 11–15 Nov 2002

Further travel on business with an MRG connection was as follows; most of this was funded from other sources:

Amadio (travel not funded by MRG): to Paris in Dec 2002 to attend an "Action Specifique du CNRS Methodes formelles pour la mobilite" to give a talk on his MRG work on max-plus quasi-interpretations.

Aspinall (travel not funded by MRG): to ETAPS in Grenoble in Apr 2002 to present a paper with Hofmann at ESOP on a type system for space-like resources.

Hofmann: to POPL in Portland in Jan 2002 to present a paper on the strength of non-size increasing computation, and to learn about MRG-relevant work on proof-carrying code.

Hofmann (travel not funded by MRG): to PPL 2002 in Beppu, Japan in March 2002 to give an invited talk "Functional programming without garbage-collection and resource certification".

Hofmann: to a GI workshop in Freiburg in Oct 2002 and (travel not funded by MRG) to an IFIP 1.3 workshop in Frauenchiemsee in Sep 2002, to present his MRG work with Jost on static prediction of heap space usage for first-order functional programs.

Hofmann (travel not funded by MRG): to CIRM in Marseille in March 2002 and MFO Oberwolfach in April 2002 to give tutorials on "Programming languages capturing complexity classes" which included a brief overview of MRG.

Sannella: to an IFIP 1.3 workshop in Alpe d'Huez in Jan 2002 to present an overview of MRG.

Sannella: to Global Computing workshop during ICALP 2002 in Malaga in Jul 2002 to give a talk on "Security and resource management" and participate in a panel discussion.

I.2 Major Technical Achievements

Work during the first year of MRG has been devoted mainly to establishing a solid infrastructure for the central scientific work of the project. We have drawn on the existing state of the art, with particular attention to quality as this basic platform is vital to the success of the whole project. In the course of this research, some unexpected research opportunities have come to light and we have diverted some effort into exploring these. Work on several tasks in the scientific core of the project has commenced and some interesting results have been obtained but work on these tasks is still underway.

Research on MRG involves languages on two levels. On one hand there is the *low level* of bytecode programs to be downloaded by the code consumer, to which certificates containing checkable proofs of resource consumption are to be attached. At this level we need a definition of the virtual machine on which the code is to be run, a cost model defining the resource consumed by such programs during execution, and finally an implemented logic in which the proofs of resource consumption are given, which makes reference both to programs and their cost. Soundness of the proof rules in this logic means that if there is a proof that a given program has a given cost, then (according to the cost model) running that program on the virtual machine will have *at most* that cost. Establishing this level is the goal of Workpackages 1 and 2.

On the other there is the *high level* of the language in which these programs will be written by the code producer. This language will be equipped with various novel resource-aware type systems in which programs are certified by virtue of their typing as satisfying certain resource bounds. This language compiles to code in the low-level language, and a proof that the resource bound indicated by its type is satisfied by the resulting low-level program is generated from the type derivation. Establishing the high-level language and compiling it is the goal of Workpackage 3, while resource-aware type systems are the business of Workpackages 4, 5 and 7.

For the sake of this research, it is important that both the low level framework and the high level language be small and simple enough to be studied in detail. At the same time they need to be powerful enough to provide a convincing demonstration that the vision behind MRG is valid.

Several existing bytecode languages are good candidates for the low level language. We originally selected a subset of the Java Virtual Machine Language (JVML) for this purpose, but after consideration of the possible alternatives (the JavaCard virtual machine language and Microsoft's .NET platform), we were inspired by [LTS01] to develop and implement a slightly higher-level language called *Grail*. This is a small functional language which incorporates JVM instructions as primitive operations while abstracting away from details of stack and register usage. Grail has several features that make it particularly suitable for our purposes:

- Grail can be viewed as a description of a particularly well-structured form of JVM bytecode. Bytecode derived from Grail programs has several pleasant properties which make the process of analysis and verification much simpler. In particular, Grail bytecode satisfies certain conditions which have been shown by Leroy [Ler02] to enable bytecode verification to take place in an environment with very limited memory (for instance, on a smart card).

- We may regard a Grail program as having two different semantic interpretations: one derived from the functional aspect of the program and another derived from the imperative semantics of the bytecode translation of the program. We have shown that these two semantics coincide, in the sense that a Grail program evaluates to a particular value with respect to the functional semantics if and only if the corresponding bytecode program terminates and yields the same value. This should allow us to prove properties of imperative bytecode programs using techniques from functional programming, which is usually much simpler than trying to prove properties of imperative programs directly.

- It is easy to translate from Grail to bytecode and back. This means that we can transmit proofs relating to the functional form of a program, but transmit programs using the usual JVM class file format. When proof checking is required on a client machine it is easy to recover the functional version of the program from the class file. We have implemented translators from Grail to JVML and vice versa.

- The similarity between JVM and the .NET Common Intermediate Language means that it would be easy to modify a Grail compiler to target .NET; in fact, it should even be possible to implement a translator which would run on the client machine and would convert Java class files derived from Grail directly into .NET assemblies. This would allow programs to be distributed in a single format, while targeting both platforms.

- The relative abstractness of Grail will also admit straightforward modification to target other versions of the Java platform such as JavaCard.

The introduction of Grail achieves significantly more than was laid out in the original workplan, and we have put more work than foreseen into studying and perfecting Grail to ensure that it will provide a good basis for subsequent work and because it may be of interest independent of MRG. During a recent visit to Edinburgh, John Reynolds observed that Grail has many ideas in common with the "flat form imperative programs" that arise in his work on Separation Logic.

Work on the cost model was relatively straightforward, apart from the observation that since exact resource measures depend on implementation details of the virtual machine, it is appropriate to give two kinds of cost model. A *structural* cost model merely indicates the resource consuming entities, without detailing the costs down to bytes or microseconds; this is complemented by a more detailed *raw* model where the memory or time consumed is measured as bytes/microseconds and VM-dependencies occur as model parameters. The development of the bytecode logic is not yet complete, but work has so far explored specially adapted versions of a logic of object-oriented programs due to Abadi and Leino [AL97, Lei98a, Lei98b] and Reynolds' Separation Logic [Rey02].

Existing work on resource-aware type systems (e.g. [Hof00]) applies to simplified functional programming languages, so for the high-level we developed and implemented a functional language, which is called *Camelot*. This is a subset of OCaml with features that are restricted with respect to OCaml both for the sake of simplicity and in order to facilitate prediction of the resource use of programs in subsequent work. The most important restriction is to higher-order functions, where functional *arguments* are allowed but functional *results* are forbidden, and no syntax for lambda expressions is provided. As a result, *closures* are not required in the implementation; this avoids the seemingly difficult problem of predicting their size. The functional nature of Grail allows Camelot to be compiled to Grail in a fairly direct manner, and should make it relatively straightforward to translate assertions about Camelot programs into assertions about the corresponding Grail programs, and thence to assertions about JVM bytecode. Future work includes tasks on extending Camelot with additional features to take it further beyond what has been considered before in work on resource-aware type systems.

MRG participants have continued their established work on resource-aware type systems, and also developed further novel techniques. Most of the work begins from Hofmann's LFPL language [Hof00], which adds a special space resource type to an affine linear functional programming language. Under a translation scheme which uses in-place update when resource values are consumed, LFPL programs work in constant heap space (functions compute within the space occupied by their arguments). Hofmann gave a translation to C programs. As an alternative to C, [AC03] showed that a typed low-level language, with similarities to JVM bytecode, could be used as an alternative target language for Hofmann's compilation scheme. New investigations into more permissive languages than LFPL have been undertaken: the aim is to relax linearity (allowing sharing when possible), so that more algorithms can be type-checked, but retaining the heap usage guarantee. In [Kon02b], Konečný considered a more abstract formulation of the typing system for LFPL, elaborating the intricate correctness proof for several variants with relaxed linearity conditions. But still resource sharing was somewhat coarse: either some variable may share heap space with another or it may not. As a refinement, [Kon02a] introduces a type system which allows data-structures with complex sharing patterns: for example, a list where the cons cells occupy distinct memory regions, but their contents may occupy overlapping regions. A major step forward was achieved by [HJ03], which shows an inference mechanism for determining whether an ordinary functional program (without resource types) can be annotated with the resource values used in LFPL. The inference algorithm uses linear programming to solve constraints which determine the annotations. This algorithm brings us much closer to a resource type system for Camelot; adaptation for the current version of Camelot should be straightforward. Amadio has initiated a line of research on using max-plus algebras to bound values and resources during computation. He is investigating automatic synthesis of these *quasi-interpretations*, with the prospect of inferring detailed resource usage for high-level functional programs. Current examples cover insertion sort, common binary subsequence, and checking quantified boolean formulas.

There are so far no concrete links with other projects in the Global Computing initiative, apart from those stemming from involvement of specific MRG personnel in other projects: Stephen Gilmore is also a member of DEGAS and he attended a joint Agile-DEGAS meeting during 2002; Roberto Amadio is also a member of PROFUNDIS at the Sophia Antipolis site (on sabbatical during his employment on MRG); Don Sannella is on the Mikado advisory board. There are natural affinities with projects in which type systems play a central role (DART, Mikado, MyThS, PROFUNDIS) and with projects dealing with problems of security and control of resources (SECURE, MyThS, PROFUNDIS). The MRG project focuses on code mobility within a sequential model of programming, unlike the other projects which are mainly concerned with concurrency and distribution. In particular, in this sequential framework, it develops techniques to *extract* resource bounds and to *certify* these bounds. We expect that these techniques can be adapted and applied in a concurrent/distributed context. In particular:

- In [TZH02] and [GHS02], type systems have been proposed to bound the number of threads in an ambient. Inferring such resource bounds involves the solution of numerical constraints akin to those arising in the extraction of space bounds in the MRG project.
- Type systems have also been proposed to partially ensure the correct behaviour of *open* systems. These systems require some form of dynamic typing (e.g., when a new agent joins the system) which appears to relate to the MRG issue of production and verification of certificates and more generally to the proof-carrying code approach.

We have established good links with the ConCert project at Carnegie Mellon University [CCD+02], which is studying the problem of how the components of a distributed application may be disseminated to the greatest possible number of hosts, thereby leading to the optimum exploitation of their (otherwise unused) resources. To this end it is developing a distributed computing fabric to support the deployment of Internet-scale distributed applications. Because the Internet must be regarded as an effectively hostile computing environment containing both malicious applications and well-intentioned (but erroneous) applications, the ConCert project emphasises the *trustless dissemination of software*. The disseminated software is distributed as machine code binaries produced by certifying compilers. The attached certificates are proofs which guarantee concrete security and privacy properties of the software. There are many points of contact between ConCert and MRG but they consider different application areas and deploy different technologies (typed assembly language [MWCG99, MCGW02] in the case of ConCert, Grail bytecode with proofs of properties of code derived from type derivations in resource-aware type systems in the case of MRG). A further major point of difference is MRG's focus on guaranteeing resource bounds; resource consumption is considered in ConCert, but as a relatively minor component. Peter Lee, one of the principal investigators of ConCert and co-inventor of proof-carrying code, attended the MRG kickoff workshop in May 2002 in order to pursue these connections. There he made many helpful and encouraging comments about our plans and progress as well as providing useful information from his extensive experience with proof-carrying code and typed assembly language. He also supplied us with a copy of Special J, a certifying compiler for Java produced by his company Touchstone. Martín Abadi of the University of California at Santa Cruz, who was in Edinburgh for another event, also attended the MRG kickoff workshop, and made further useful comments. See Deliverable D11b (Kickoff Workshop) for a summary of their input and a copy of Lee's slides.

The survey work undertaken in Workpackage 10, on mobile virtual machines, has uncovered a related project with similar aims and objectives to that workpackage, namely the Virtual Virtual Machines project at INRIA [pro03]. Ideas and results from that project will be taken into account in work on this workpackage.

I.3 Dissemination of Results

We see the prime audience for this work as other researchers, and consequently the main route for dissemination will be technical reports, journal papers and conference presentations by the participants. This enables rapid and timely publicity among those who can draw most from the work. The following publications and pending submissions have been completed during 2002:

R. Amadio. Max-plus quasi-interpretations. Laboratoire d'Informatique Fondamentale de Marseille Research Report LIF 10-2002 (2002). Submitted for conference publication.

D. Aspinall and A. Compagnoni. Heap bounded assembly language. *Journal of Automated Reasoning*, to appear (2003).

M. Hofmann and S. Jost. Static prediction of heap space usage for first-order functional programs. *Proc. 30th ACM Symp. on Principles of Programming Languages*, New Orleans (2003).

M. Konečný. Typing with conditions and guarantees for functional in-place update. Submitted for conference publication.

M. Konečný. Functional in-place update with layered datatype sharing. Submitted for conference publication.

In addition, Martin Hofmann presented the following conference paper during 2002 with travel support from MRG. This paper was produced before MRG began but it provides part of the underpinnings of MRG and the talk made reference to the MRG project.

M. Hofmann. The strength of non-size increasing computation. *Proc. 29th ACM Symp. on Principles of Programming Languages*, Portland (2002).

Further talks on MRG research are listed in Section I.1.c. A poster on MRG was produced and displayed at two events including the review meeting for the National e-Science Centre in Edinburgh.

To complement these papers, we have produced the first in a series of prototype and proof-of-concept implementations. These are an effective way to demonstrate theoretical results in a practical context, and they are made available (with source code) over the web. They are:

K. MacKenzie. Grail compiler and decompiler. `http://www.lfcs.ed.ac.uk/mrg/publications/programs/Grail.tar.gz` (2002).

K. MacKenzie and N. Wolverson. Camelot compiler. `http://www.lfcs.ed.ac.uk/mrg/publications/programs/Camelot.tar.gz` (2002).

The following publications, which have no direct connection to specific MRG tasks but are still relevant, were produced or completed by MRG personnel during 2002:

D. Aspinall and D. Sannella. From specifications to code in CASL. *Proc. 9th Intl. Conf. on Algebraic Methodology and Software Technology*, St.-Gilles-les-Bains, Reunion. Springer LNCS 2422, 1–14 (2002).

M. KH. Aswad, H.-W. Loidl and P.W. Trinder. Developing functional programs for multiple parallel architectures. Submitted for conference publication.

L. Beringer. Asynchronous Queue Machines with Explicit Forwarding. Ph.D. thesis, Report ECS-LFCS-02-429, Laboratory for Foundations of Computer Science, University of Edinburgh (2002).

A. Rauber Du Bois, R. Pointon, H-W. Loidl and P. Trinder. A declarative parallel bottom-avoiding choice. *Proc. Symposium on Computer Architecture and High Performance Computing (SBAC-PAD 2002)*, Vitoria, Brazil (2002).

A. Rauber Du Bois, H.-W. Loidl and P. Trinder. Thread migration in a parallel graph reducer. *Proc. 14th International Workshop on the Implementation of Functional Languages*, Madrid. To appear (2003).

An industrial connection giving a path for potential future practical exploitation of the results of MRG has been made with the firm Gemplus, the world's largest maker of smart cards, which has shown a strong interest in MRG. Antoine Galland from the Gemplus Software Research Lab in Marseille attended an MRG workshop in September 2002. He said that MRG offers what seems to be ultimately the most promising approach to the problem of space resource control, which is a critical issue for smart card software because of the severe restrictions on memory size in smart cards imposed by the physical constraints, and that they will be watching our work with great interest.

Part II

Detailed Report

II.1 Work Carried Out

Below is a verbatim copy of relevant parts of the MRG workplan, augmented with a description of what work has been carried out under each workpackage and task and explaining significant departures from what was foreseen. Workpackages and tasks on which work was neither planned nor carried out during 2002 are not included.

II.1.a WP1: Virtual machine and cost model

Objectives: Definition of virtual machine platform, formalization of cost model, and collection of examples.

This workpackage provides the basis for the research in the remaining packages. The overall goal of the project is to build a framework for Java-style downloadable bytecode equipped with checkable certificates regarding their usage of resources. The executable part of these will be bytecode for a specific virtual machine, to be defined in 1a. Claims about resource usage will refer to the cost model defined in 1b, which specifies the amounts of various kinds of resource that are consumed during execution. For the sake of connecting with current practice, we will employ a version of the Java Virtual Machine Language. We plan to investigate the suitability of Microsoft's .NET intermediate language as an alternative platform at an early stage (1d). If JVML turns out to be too restrictive for our purposes then we could switch attention to .NET.

a. Define **virtual machine platform**. *(2 months)*
Deliverables: internal technical document
We will define a subset of JVML that is expressive enough to permit straightforward compilation of the high-level language to be defined in 3, yet small enough that to be tractable as an object of study, admitting proofs like those to be done in 4. Some modifications or simplifying assumptions may be required.

Work carried out: Completed and delivered in February 2002 as Deliverable D1a, a technical paper defining a JVML subset comprising about 30 instructions that subsumes the functionality of most of the full JVML instruction set. But see 1d and 1e below.

b. Formalise **cost model**. *(6 months)*
Deliverables: technical report *Prerequisites: 1a*
Here we will define the amount of resource consumed by running a unit of mobile code. This will take the form of a semantics for our VM that delivers a cost along with the computed result, using previous work on formalisation of JVM (e.g. [SA99], [CGQ00], [NvOP00] and/or [FM99]) as a basis. Ultimately, many different kinds of resource are of interest; we hope that by considering two different concrete examples — heap space and number of system calls — the elements of a general account will emerge. There may well be a division into time-like measures that are additive with respect to independent components versus space-like measures where the cost of performing two independent computations is the maximum of the costs of the components. In practice, costs depend on the implementation of the JVM with techniques such as just-in-time optimization potentially leading to dramatic improvements. Initially we will assume that no optimizations are done, but see 7b. Included in the deliverable will be a number of illustrative examples.

Work carried out: Completed and delivered in September 2002 as Deliverable D1b, a technical paper defining the memory and time usage of programs in our JVML subset. Since exact measures depend on implementation details of the virtual machine, two kinds of model are given: a *structural* cost model merely indicates the resource consuming entities, without detailing the costs down to bytes or microseconds; this is complemented by a more detailed *raw* model where the memory or time consumed is measured as bytes/microseconds and VM-dependencies occur as model parameters.

c. **Milestone: Completion of cost model.** *Prerequisites: 1b*

This has been successfully achieved by the delivery of Deliverable D1b.

d. Investigate **.NET as an alternative to JVML**. *(3 months)*
Deliverables: internal technical document *Prerequisites: 1b*
Microsoft's .NET platform provides a virtual machine and runtime environment that offers an alterative to JVM for supporting Java and other languages [GS00]. With the support of HP and Intel the .NET intermediate language is now being taken through the ECMA standardisation process. A benefit of JVML is that at present implementations are widely available, but it is designed specifically to support Java, and its built-in support for Java's type system may be a barrier to progress. Then .NET, which is designed to support a wide variety of languages and so offers extra flexibility, may turn out be more appropriate for our purposes. Here we plan to compare the suitability of .NET with JVML for the project.

Work carried out: Completed and delivered in December 2002 as Deliverable D1d, a collection of technical papers defining the Grail intermediate language with translators to and from the JVML subset. This allows us to target both JVML and .NET simultaneously while providing an abstract format whose properties can be studied by means of well-established theoretical techniques.

The introduction of Grail in D1d achieves significantly more than was laid out in the original workplan, and there is overlap between developments here and what was anticipated under 1a. We have put considerable work into studying and perfecting Grail to ensure that it will provide a good basis for subsequent work. Some of the documents reporting on this work are unpolished draft notes but we have provided them as part of Deliverable D1d nevertheless.

Terminology: "Grail" is the name of the intermediate language, "jGrail" is the name of the corresponding JVML subset, and ".Grail" would be the name of the corresponding subset of the .NET Common Intermediate Language. In some early MRG documents, the name "λGrail" is used for the intermediate language (after the language λJVM of [LTS01]) and "Grail" for the JVML subset.

e. **Milestone: Decision to use JVML** as planned **or switch to .NET.** *Prerequisites: 1d*

This has been achieved. Consideration of .NET as an alternative virtual machine platform prompted the development of the Grail intermediate language, see 1d; this is now our virtual machine platform, replacing the JVML subset that was delivered earlier as Deliverable D1a. On one hand Grail can be seen as a more abstract version of our JVML subset with implemented translations to and from JVML. On the other it gives a degree of platform independence since it seems clear that translations to and from .NET or any similar bytecode language would not be difficult to provide.

f. Collect some **representative examples**. *(2 months)*
Deliverables: technical report
Here we will compile a set of examples to use as test cases throughout later stages of

the project. A good source of interesting algorithms and data structures is the LEDA project [Gmb]. The amount of memory available on a smart card is severely restricted, making this an area where resource guarantees like those we aim to supply will be of interest. This means that JavaCard [Mic] is another source of interesting examples, based, for instance, on the e-cash examples from the JavaCard documentation or medical information systems such as [GSF00].

Work carried out: Completed and delivered in January 2003 as Deliverable D1f, a technical paper giving Grail implementations of a collection of algorithms with different resource consumption characteristics, together with a description of their expected resource behaviour.

The research in this package provides the first part of the infrastructure on which later packages will build.

II.1.b WP2: Definition of bytecode logic

Objectives: Development of bytecode logic, including language of assertions and proof rules, and a proof checker.

The purpose of this workpackage is to provide a language (2a) for making assertions about the resource usage of bytecode programs with respect to the cost model defined in 1b, and a logic (2b) for proving such assertions. The certificates that will be attached to dowloadable bytecode will be proofs in this logic (but see 9). Certificates must be easily checkable by the recipient, and the implementation of the proof checker (2c) will be part of the trusted code base (TCB); thus the logic and its implementation must be simple and generally acceptable. To ensure the quality of the logic and its implementation, we have included related tasks (2e and 2f) whose main underlying aim is assessment of these components, and do not directly feed into later work.

a. Develop **language of assertions**. *(2 months)*
Deliverables: technical report *Prerequisites: 1be*
We will develop a logical language for asserting resource-related properties of bytecode programs. This will probably take the form of a Hoare-style logic, with assertions making explicit reference to code fragments. The expressiveness and convenience of the notation will be checked using examples from 1b and its semantics will be formally defined.

Work carried out: Completed and delivered in January 2003 as Deliverable D2a, a technical paper exploring specially adapted versions of Reynolds' Separation Logic [Rey02] and a logic of object-oriented programs due to Abadi and Leino [AL97, Lei98a, Lei98b]. This includes a comparison of the two logics and an assessment of advantages and disadvantages. These preliminary explorations go considerably beyond the mere definition of a language of assertions foreseen for this task.

b. Develop **proof rules**. *(3 months)*
Deliverables: conference/journal paper *Prerequisites: 2a*
We will develop rules for proving assertions expressed in the language of 2a. The goal is rules that are simple, without computationally intractable conditions, that enable a large class of interesting assertions to be proved in a relatively straightforward way. We aim to generate proofs automatically via the use of novel type systems (see 4) so these rules will not normally be used directly by users (although see 7a). Understandability is nevertheless important to instill confidence in the integrity of our framework, and the length of proofs is

also an important issue in practice (see 9). It will be essential to demonstrate that the rules are sound with respect to the cost model in 1b, and we will also attempt to check relative completeness of certain fragments.

Work carried out: This work is progressing well but is not yet complete. An Isabelle encoding of the operational semantics of Grail, which takes account of the resources used during computation, is close to completion, along with a report on this; this will constitute a component of D2b. This is already usable as a low-level logic. Higher-level proof rules are under development, including an Isabelle encoding, and we hope to have these complete during the first quarter of 2003. A draft report on a possible set of rules exists but more study is required.

Although work on this task is not yet complete, the decision to develop the proof rules in Isabelle and the work already completed on encoding the operational semantics of Grail goes beyond what was foreseen in the original workplan. A consequence of this is that subsequent work on 2c, 2e and 2f will be much easier to accomplish, see below, since these can build directly upon this Isabelle encoding. This technical decision has therefore had the effect of shifting work from these later tasks to this one.

c. Implement a **proof checker**. *(4 months)*
Deliverables: prototype implementation *Prerequisites: 2b*
Given an ostensible proof built using the rules in 2b, we need to be able to efficiently and automatically check that it is valid. For this purpose, Necula [Nec97] uses Edinburgh LF [HHP93], a generic type-theoretic framework for encoding and checking proofs. Depending on how complicated the logic is, we will take the same approach or alternatively handcraft a proof checker.

Later work (for example 6d) requires that a rudimentary metalanguage be provided, so that lemmas can be proved and then referred to, or for construction of proofs by primitive recursion. Basic facilities of this kind will be provided. The estimate of work required is under the assumption that this will be relatively straightforward. Some care is required to prevent denial of service attacks based on the creation of huge proofs.

Work carried out: Work on this task has not yet begun, but building a prototype proof checker should be straightforward once the Isabelle encoding of the proof rules in 2b is complete. At this point we still intend to build a more compact stand-alone proof checker, in order to reduce the size of the trusted code base.

e. Implement a **theorem prover**. *(3 months)*
Deliverables: experimental implementation; case studies *Prerequisites: 2b*
Here we plan to experiment with the use of an interactive theorem prover such as Isabelle [Pau94] for generating proofs. This will give access to the features provided by such provers, e.g. context management, goal-directed proof search, and built-in decision procedures. The main point of this work is to explore some of the ramifications of the choice of proof rules in 2b; in practice proofs will be generated by other means, see 4.

Work carried out: Work on this task has not yet begun, but building a basic theorem prover should be straightforward once the Isabelle encoding of the proof rules in 2b is complete. Further work will be required to develop Isabelle tactics and to conduct experiments.

f. Optional: **encode VM semantics** in theorem prover. *(1 month)*
Deliverables: experimental implementation *Prerequisites: 2b*

Here we would encode the entire virtual machine with its semantics in a general-purpose theorem prover such as PVS [ORS92], and then use this to *formally* establish that the proof rules in 2b are sound. The aim would be to explore some of the consequences of our earlier choices in order to validate these decisions. This could be carried out as a final year student project under the supervision of project personnel; this supervision is what is allowed for in the manpower estimate.

Work carried out: Work on this is underway as a final year student project under the supervision of MRG personnel, to be completed in June 2003. Already complete is an Isabelle encoding of jGrail including the cost model of 1b, and an encoding of the translation from Grail to jGrail has begun.

The research in this package is a mixture of design, assessment, and implementation work. A working proof checker (2c) will be an essential component part of the final overall system, and the quality of the logic that it implements is vital to the success of the entire framework.

II.1.c WP3: Design of experimental high-level language

Objectives: Design and implementation of high-level programming language targeted at the bytecode language of WP1, to provide a test bed for WP4–7.

The preceding workpackages provide a grounding for resource guarantees on bytecode; but this is too low a level for practical programming. The objective of this workpackage is to write a compiler for a high-level programming language targeted at the bytecode language of 1a. This will provide a test bed for the higher-level developments of packages 4, 5, 6 and 7. The compiler should be small enough to allow for rapid progress, yet sufficiently expressive to demonstrate that scaling up to real languages is possible.

The most distinctive requirement on the compiler is *transparency*: it must be possible to calculate how a particular fragment of high-level code will transform into bytecode, and what its resulting resource usage will be according to the cost model of 1b. This is vital to support the language-level approach of following workpackages.

As this compiler is to provide a framework for later development, it should also be *extensible*. Other packages aim to add functionality, like resource types (4c) and generation of certificates (6a). This must be possible without disturbing the basic action of the compiler. To meet these requirements the compiler needs an open and well-documented architecture, in sufficient detail to support reasoning (formal and informal) about its behaviour (4e). Consequently, the technical documentation produced by this workpackage is of particular importance, providing reference material for later work.

a. Define a base **programming language**. *(1 month)*
   Deliverables: internal technical document *Prerequisites: 1a*
   The language should be strongly typed, providing at least functions and recursive datatypes. It needs an accompanying operational semantics, and information about the intended mapping to bytecodes.

   Work carried out: Completed and delivered in May 2002 as Deliverable D3a, a technical paper describing the Camelot language including design choices and rationale. Camelot is a subset of OCaml, with features that are restricted with respect to OCaml both for the sake of simplicity and in order to facilitate prediction of the resource use of programs. The most important restriction is to higher-order functions, where functional *arguments* are allowed but functional *results* are forbidden, and no syntax for lambda expressions is provided. As a result, *closures* are not required in the implementation; this avoids the seemingly difficult problem of predicting their size.

The Galahad compiler adds modular programming with ML-style signatures and functors, as well as special constructions for array iteration. Galahad compiles into Camelot so it adds convenience but no additional power (and so no new problems for resource prediction). (Tennyson's poem "Sir Galahad" includes a clear reference to the importance of functional languages in situations of limited resources:

> My strength is as the strength of ten,
> Because my heart is pure.

See `http://www.lib.rochester.edu/camelot/galahad.htm`.)

b. Write a **compiler** for the base language. *(4 months)*
Deliverables: prototype implementation; technical documentation *Prerequisites: 3a,1b*
The important issue here is not to produce a high-grade optimising compiler, but rather one that is predictable, clearly written, and well-documented, to support the work to follow. The documentation should include discussion of the anticipated resource usage of bytecodes generated by the compiler.

Work carried out: Completed and delivered in January 2003 as Deliverable D3b, a running implementation of a compiler for Camelot together with a technical paper describing its detailed operation.

c. **Milestone: Completion of implemented programming language.** *Prerequisites: 3b*

This has been successfully achieved by the delivery of Deliverables D3a and D3b. We have written high-level functional programs in Camelot, compiled these to Grail, and executed their translation to JVML as bytecode on a Java virtual machine.

d. Extend with **immutable objects** and **higher-order functions**. *(4 months)*
Deliverables: prototype implementation; technical documentation *Prerequisites: 3b*
These additional features make the language more expressive, and will assist more advanced parts of later packages, like 7c. Like the base language and compiler, these extensions need comprehensive documentation: covering both the semantics of the language features themselves, and their compilation to bytecode. This particularly important for higher-order functions, as they typically fit less well with existing object-style VM's.

Work carried out: Camelot already provides certain kinds of higher-order functions, see 3a above, and the Camelot compiler handles these correctly. Design work on extending Camelot with objects has begun. The minimum goal of such an extension would be to provide interoperability with the Java libraries.

e. Implement well-understood **optimisations** in the compiler. *(3 months)*
Deliverables: prototype implementation; technical report *Prerequisites: 3b*
It is sensible to extend the compiler with optimisations whose effect can be determined statically. Example are some uses of in-place update, or the replacement of tail recursion by iteration. Exactly what is possible will depend on the chosen bytecode and its cost model

— because of the need to keep track of resource usage in a provable way, this is a tighter constraint than for most compilers.

Work carried out: Work on this is underway as a final year student project under the supervision of MRG personnel, to be completed in June 2003. Already complete is an implementation of tail-call elimination with a proof of correctness. Work has begun on formally proving the effect of this optimisation on resource usage.

II.1.d WP4: From reasoning principles to high-level type systems

Objectives: Develop reasoning principles and type systems for characterising resource usage, including a typechecker and soundness proofs.

This workpackage builds on the foundational strands in the first three packages. Beginning from the experimental high-level language designed in 3a, we investigate ways of expressing resource constraints and proving that they are satisfied by the compiled program, according to the cost model of 1b. We begin from reasoning principles, perhaps related to the bytecode logic, and then move towards type systems. The notion of type is a very broad one: type-checking can be used to enforce simple consistency checks (that the addition operation + is always applied to two numeric arguments) but also rich semantic notions (such as interference [Rey78], presence of side-effects [TJ92], and resource usage as proposed here or studied in [Hof00]).

Our approach is to stick with type systems where the type-checking problem is *decidable*, whereas the problem of proving that a resource constraint is satisfied will generally be undecidable. This means that we accept an unavoidable gap (the "slack") between the set of programs which are typable in a resource type system and the larger set of programs which satisfy the resource property of interest. For many natural examples, though, the resource bounds are met for obvious reasons which are in the scope of our type systems. The craft of designing type systems lies in capturing these natural examples and minimising the slack, while retaining a practical notion of type and practical type-checking algorithms.

a. Investigate **reasoning principles for resource usage**. *(6 months)*
Deliverables: technical report; case studies *Prerequisites: 1b,3a,3b*
Here we will investigate the resource behaviour of the constructs in our high-level language from 3. This involves program analysis: we will identify natural examples of programs which meet different kinds of resource bounds, and try to establish this formally by explaining how the compiler from 3b compiles these examples to byte code, and proving how the resulting byte code meets the resource bounds, using the bytecode logic. In effect, we are looking for high-level derived rules and reasoning principles for the bytecode logic, which apply directly to the high-level language. The results of this work will suggest patterns and constructs that should be built into the type systems.

Work carried out: Completed and delivered in December 2002 as Deliverable D4a, a technical report studying quasi-interpretations as a tool for bounding the size of the values computed by a first-order functional program and thus as a means to extract bounds on its complexity.

The work in Deliverable D2a concerning Separation Logic and the Abadi/Leino logic is also relevant. To begin, these logics have been considered primarily at the lower level of Grail, but because Grail is designed to be understood in a functional way, it is closer to Camelot than plain bytecode, and so similar reasoning principles should carry over.

b. Develop a **type system for space-like resources**. *(4 months)*
Deliverables: working design document *Prerequisites: 4a*
The first type system will be one for ensuring space-like resource bounds. There is some relevant recent research here, including [Hof00] which describes a type system with a special *resource type* which corresponds to a unit of reusable space. Hughes and Pareto [HP99] describe a type system for programming in bounded space. Crary and Weirich [CW00] have a type system which provides explicit bounds for time usage: the run time of a function can be expressed as a function of the input. We want to extend and adapt these systems, in particular investigating ways of attaching explicit bounds on space usage. Other resources, such as file handles, network connections, or hardware resources, could be treated in a similar way to space usage in these systems; this will be a novel approach and should help to eliminate a common class of program failures.

Work carried out: Work on a type system for Camelot is at an early stage. However there is a great deal of highly relevant work by MRG personnel which provides an excellent basis for the completion of this task, including [Hof00] and [Hof02]. The paper [HJ03] provides a type system for space-like resources that appears to be directly applicable to Camelot although the exact details in this context remain to be worked out and implemented. Also relevant for this task is [AH02], [Kon02a] and [Kon02b], and the unpublished ongoing PhD work of Robert Atkey in Edinburgh.

The research in this package is a combination of improving existing work and combining several different ideas, previously treated in separation and with different motivations. Drawing together these strands is a vital step towards our vision.

II.1.e WP5: Further high-level and low-level type systems

Objectives: Generalise type systems in WP4 to accommodate more general notions of resource, and develop type systems for expressing resource bounds at the byte-code level.

This workpackage continues and expands on the work begun in 4. The idea here is to begin to generalise the systems for space-like resources studied there to consider more general notions of resource. As a particular case, we will examine type systems for expressing limits on parameter values (5a).

This package also broadens the scope of the type systems to consider *low-level* type systems for expressing resource bounds at the level of the VM byte code. Although our main interest is in ensuring that resource bounds are met for programming in high-level languages, a way to help ensure this is to push the resource type information as far down as we can, and annotate the bytecode with additional typing information.

a. Develop a type system for expressing **limits on parameter values**. *(4 months)*
Deliverables: technical report; conference paper *Prerequisites: 4a*
As one of the potentially most useful concepts in our generalised concept of resource bounds, we want to have type systems which express restrictions on parameter values for functions. This generalises the most common case needed in programming: that of ensuring that an array access does not violate the bounds of the array. One particularly general way of achieving this is with so-called *dependent types* where the type expression can contain ordinary terms from the language. In practice, full dependent types lead to complex type systems which are almost always undecidable, but restricted forms of dependent types may be useful here. There is

interesting related recent research [XH01, Aug98] which may help here. Apart from dependent types, there are other novel type systems for programming languages which propose ideas we might adapt, such as the notion of *shape* [BM97].

Work carried out: This is in progress, building on the work with quasi-interpretations and max-plus algebras described in Deliverable D4a. There is a preliminary technical note treating the situation where resource-bounded functional programs are part of a larger Kahn network. The question being considered is what requirements must be imposed on the programs and on the network to obtain a system that runs with bounded resources. This note will be developed into a conference submission during the first half of 2003 that will constitute Deliverable 5a.

II.1.f WP7: Advances in high-level type systems

Objectives: Improve expressiveness, user-friendliness, and accuracy of the type systems developed in WP4 and WP5.

The goal of this workpackage is to improve expressiveness, user-friendliness, accuracy of the type systems developed under 4 and 5.

d. **Type inference** *(6 months)*
Deliverables: research paper; implementation (optional) *Prerequisites: 4b,5a,7a,7b*
The basic type system developed under 4 and the extensions developed under 7a and 7b may rely on any number of user-supplied type annotations, for instance, recursive functions might be annotated by suggested bounds on their resource usage which are merely certified, cf. [CW00].

The aim here is to develop ways to infer such annotations automatically to a certain degree based on decision procedures for arithmetic inequalities [HP99], automata-theoretic methods [PWO97], unification [Mil78], program analyses, fixpoint methods [PS91], etc.

Work carried out: Important progress towards this goal has been made by Hofmann and Jost in [HJ03]. In that paper a type inference algorithm for the language LFPL [Hof00] is presented which takes an arbitrary first-order functional program as input and computes an annotated version of that program in LFPL from which an upper bound on heap consumption can be directly read off. This is done by solving linear integer inequalities. It is shown that in important subcases the systems of inequalities arising in this way are solvable in polynomial time. It is also shown that assuming a memory model with freelist and garbage collection rather than the static memory allocation of LFPL one can remove the integrality restriction and thus feasibly obtain solutions in all cases. A prototype implementation of [HJ03] is currently underway.

All that remains to be done to complete this task is to adapt the algorithm in [HJ03] to Camelot and to the precise version of the type system from 4b once it is completed. We do not foresee any technical difficulties; in particular, the addition of higher-order functions as in Camelot should be straightforward.

II.1.g WP8: Integration with existing security model

Objectives: Implement resource manager and relate proof-checking infrastructure to present-day security management.

The preceding workpackages have detailed a proof-checking infrastructure which advances the state of the art in security management capabilities. This workpackage will enrich our understanding of this infrastructure by relating it to present-day security management.

b. **Experimental implementation** *(2 months)*
Deliverables: experimental prototype *Prerequisites: 2c,8a,5e,6a*

A prototype implementation of a certificate-led resource manager will be produced. This will provide a platform for further speculative research on developments in static security assessment. Recent work on Java-based agent models which work with the Java 2 security model [GP01] would provide the basis for further development here.

Work carried out: This task, which was scheduled for near the end of year 3, has been brought forward in order to support an early demonstration of what MRG intends to deliver as its main final product, to encourage implementation and integration of project components, and to enhance visibility of the project. At this point it will consist of an integrated web-based demonstration platform that will allow one to receive a program endowed with a certificate of resource usage, check the certificate, and run the program if the check succeeds. The program will be in Grail (in the final version, it will be in jGrail) and the certificate will be an Isabelle proof script referring to the operational semantics of Grail (in the final version, it will be a condensed proof in the bytecode logic). Work on this is nearly complete, and it should be ready for demonstration at the project review.

II.1.h WP10: Mobile virtual machines

Objectives: Investigate extension to support downloadable virtual machines.

This workpackage develops a thread of investigation into a mechanism to support mobility between computational environments. As with 9 this investigation is visionary and speculative. Here we are concerned with a promising technology which could provide a way to enable greatly increased interoperability of mobile software. The foundational technology is the *mobile virtual machine*, a bytecode interpreter which is itself downloaded before the bytecode application which is to be interpreted by it. Mobile virtual machines can be realised as *circlets* [Bre98]. One use of this technology would be to allow more advanced virtual machines to be installed between high-level language programs and the JVM. Another would be to perform upgrades on pre-installed micro virtual machines.

a. Understanding the **practical effectiveness** of the technology *(4 months)*
Deliverables: internal technical report *Prerequisites: 1b*

To add another layer of software interpretation to the static virtual machine model calls into question the practical usefulness of this technology in terms of system performance. To consider that the target platform of the mobile virtual machine might be a handheld device further heightens this concern. This task will investigate the use of re-configurable hardware as an implementation technology for this concept. Re-configurable hardware offers the promise of performance close to that of circuitry but without the same specificity.

Work carried out: This work has progressed but is not yet complete. A more general study of other technologies necessary to implement mobile virtual machines has been undertaken, and a technical report is in preparation. This survey has included both practical VMs in development (including the Project Monty virtual machine [Mic02]) and research (including the Aegis virtual machine [FC00]). The latter incorporates a useful concept of "proof linking" which provides a structured way of separating bytecode verification from other safety and security analyses performed by the virtual machine.

II.1.i WP11: Project management, dissemination and evaluation

Objectives: Project management, dissemination and evaluation.

The project requires close collaboration between the two sites and provides many opportunities for dissemination. The purpose of this workpackage is to ensure that collaboration proceeds effectively and with attention to internal and external evaluation, while being able to take advantage of a wide variety of forms of dissemination for the results.

The small size of the project enables decisions about the overall technical direction of the project to be taken in close consultation with all of the people involved. Milestones and periodic meetings provide checkpoints where progress can be reviewed and plans adjusted if necessary. Meetings for technical coordination will be as follows:

- A kickoff workshop in month 2 plus workshops in months 11, 23 and 35. These will be attended by all project personnel, insofar as possible. One prominent non-EU expert will be invited to speak at each of the first three workshops at the project's expense; this will give a useful source of comment and advice without the need for project staff to visit these people individually. All of these workshops will be open to people from outside the project, with the final workshop being publicized more widely.

- Internal project meetings in months 7, 17 and 29. Each of these will include technical meetings on all active workpackages, and will be attended by all project personnel involved with those workpackages.

Individual visits are also planned for collaborative technical work.

The results of the project will be made available to all through a website set up at the beginning of the project and maintained under the direction of the Project Coordinator for the duration of the work. This is intended to give interested parties a view of the results as they accumulate. It will include at least the following:

- An introduction to the project including title, partners, and summary, with links to appropriate European Commission websites (GC, FET, IST and/or FP5).

- All of the deliverables and other publications produced by the project, as they are produced. Those that are most appropriate for external consumption will be given special prominence.

- A section (protected from access by non-project personnel) for working drafts, internal project documents, etc.

The results of the project will also be presented at appropriate conferences and published in academic journals. Many of these conferences take place outside the EU and this is taken into account in the travel budget.

The project workshops provide an opportunity for external participants to learn about the project's progress and to contribute their views. The final workshop will be associated with an established international conference for increased visibility; this event is intended more for disseminating the results of the project than for technical coordination and a proceedings is planned.

Linking workshops with the annual project evaluation meetings will allow more efficient use of the travel budget.

For self-assessment, each Workpackage Coordinator will supply in advance measurable criteria of progress/success for the different stages of the workpackage which will later be used to assess progress. This assessment of progress will take place in connection with each of the end-of-year project workshops.

a. **Project website** *(1 month)*
Deliverables: website

Work carried out: Completed and delivered in February 2002 as Deliverable D11a, see `http://www.lfcs.ed.ac.uk/mrg/`. The structure of the website is as described above, with content increasing as progress on the project is made. It is linked to a CVS-maintained repository with protected access where all project documents and software are stored. The internal part of the website and the repository have proved to be vital resources for collaborative work.

b. **Kickoff workshop** *(0 months)*
Deliverables: workshop

Work carried out: The workshop took place in May 2002; copies of slides and a summary of conclusions are Deliverable D11b. Peter Lee of CMU and Martín Abadi of the University of California at Santa Cruz attended as external experts, Lee's travel and expenses being paid by MRG.

c. **Workshop at end of year 1** *(0 months)*
Deliverables: workshop

Work carried out: An workshop involving all the projects in the "Analysis of Systems and Security" and "Languages and Programming Environments" clusters is taking place at Rovereto in February 2003, incorporating reviews of all these projects including MRG; we regard this as completing Deliverable 11c. Although this will give valuable opportunities for cross-fertilization between projects, we do not expect to have sufficient time there for in-depth technical discussion and collaboration between MRG project participants. We are therefore considering adding an additional internal workshop sometime in Spring 2003.

f. **Measurable criteria of progress/success** *(0 months)*
Deliverables: report

Work carried out: Completed and delivered in August 2002 as Deliverable D11f.

g. **Assessment of progress in year 1** *(0 months)*
Deliverables: report

Work carried out: Completed and delivered in January 2003 as Deliverable D11g.

j. **Dissemination and use plan** *(0 months)*
Deliverables: report

Work carried out: Completed and delivered in June 2002 as Deliverable D11j.

Most of the deliverables are allocated 0 person-months because this work will be done by the main investigators rather than the researchers who are employed by the project.

II.2 Self-Assessment

Deliverable D11g (Assessment of Progress in Year 1) contains a detailed assessment of progress in year 1 against the targets specified in Deliverable D11f (Self Evaluation Plan).

The self-assessment targets, list of deliverables and milestones strongly emphasize the *completion* of tasks. On that basis the project is somewhat behind schedule; reasons for this, as explained in Deliverable D11g, include unforeseen technical challenges and research opportunities, technical decisions that have had the effect of shifting some work from later tasks to earlier ones, and lack of manpower. It is therefore worth noting that good progress has also been made on many tasks that are not yet complete, including a number that are not scheduled for work in year 1, as detailed in Section II.1 above.

Bibliography

[AC03] David Aspinall and Adriana Compagnoni. Heap bounded assembly language. *Journal of Automated Reasoning*, 2003. To appear.

[AH02] David Aspinall and Martin Hofmann. Another type system for in-place update. In *Proc. 11th European Symposium on Programming, Grenoble*, volume 2305 of *Lecture Notes in Computer Science*. Springer, 2002.

[AL97] Martin Abadi and Rustan Leino. A logic of object-oriented programs. In Michel Bidoit and Max Dauchet, editors, *TAPSOFT '97: Theory and Practice of Software Development, 7th International Joint Conference CAAP/FASE, Lille, France*, volume 1214, pages 682–696. Springer-Verlag, New York, N.Y., 1997.

[Aug98] Lennart Augustsson. Cayenne — a language with dependent types. In *ICFP '98: Proceedings of the Third ACM SIGPLAN International Conference on Functional Programming*, pages 239–250. ACM Press, 1998.

[BM97] G. Bellè and E. Moggi. Type intermediate languages for shape-analysis. In *Typed Lambda Calculi and Applications: Proceedings of the Third International Conference TLCA '97*, number 1210 in Lecture Notes in Computer Science, pages 11–29. Springer-Verlag, April 1997.

[Bre98] Gordon Brebner. Circlets: Circuits as applets. In *IEEE Symposium on FPGAs for Custom Computing Machines*, April 1998. Napa Valley, California.

[CCD+02] B.-Y. E. Chang, K. Crary, M. DeLap, R. Harper, J. Liszka, T. Murphy VII, and F. Pfenning. Trustless grid computing in ConCert. In *Proceedings of the GRID 2002 Workshop*, volume LNCS 2536, pages 112–125, Baltimore, USA, November 2002. Springer-Verlag.

[CGQ00] A. Coglio, A. Goldberg, and Z. Qian. Toward a provably-correct implementation of the JVM bytecode verifier. In *Proc. DARPA Information Survivability Conference and Exposition (DISCEX'00), Vol. 2*, pages 403–410. IEEE Computer Society, 2000.

[CW00] K. Crary and S. Weirich. Resource bound certification. In *Proc. 27th Symp. Principles of Prog. Lang. (POPL)*, pages 184–198. ACM, 2000.

[FC00] Philip W. L. Fong and Robert D. Cameron. Proof linking: Modular verification of mobile programs in the presence of lazy, dynamic linking. *ACM Transactions on Software Engineering and Methodology*, 9(4), October 2000.

[FM99] S. Freund and J. Mitchell. A formal framework for the Java bytecode language and verifier. In *Proc. ACM Conf. on Object-Oriented Programming, Systems, Languages, and Applications*, pages 147–166. ACM Press, 1999.

[GHS02] J. Godskesen, T. Hildebrandt, and V. Sassone. A calculus of mobile resources. In *Proc. 13th Intl. Conf. on Concurrency Theory (CONCUR 2002), Brno*, volume 2421 of *Lecture Notes in Computer Science*, pages 272–287. Springer, 2002.

[Gmb] Algorithmic Solutions Software GmbH. LEDA — Library of Efficient Data types and Algorithms. A C++ class library originally developed at the Max-Planck-Institut für Informatik. See `http://www.algorithmic-solutions.com/as_html/products/products.html`.

[GP01] Stephen Gilmore and Marco Palomino. BabylonLite: Improvements to a Java-based distributed object system. In *Proc. 4th CaberNet Plenary Workshop, Pisa*, 2001.

[GS00] A. Gordon and D. Syme. Typing a multi-language intermediate code. Technical Report MSR TR 2000-106, Microsoft Research, 2000. A shorter version appears in Proceedings of Symposium on Principles of Programming Languages, (POPL2001), London, 2001.

[GSF00] GSF. The diabcard project. `http://www-mi.gsf.de/diabcard/`, 2000.

[HHP93] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *JACM*, 40(1):143–184, 1993.

[HJ03] Martin Hofmann and Steffen Jost. Static prediction of heap space usage for first-order functional programs. In *Proc. 30th ACM Symp. on Principles of Programming Languages, New Orleans*, 2003.

[Hof00] Martin Hofmann. A type system for bounded space and functional in-place update. *Nordic Journal of Computing*, 7(4):258–289, 2000.

[Hof02] Martin Hofmann. The strength of non-size increasing computation. In *Proc. 29th ACM Symp. on Principles of Programming Languages, Portland*, 2002.

[HP99] J. Hughes and L. Pareto. Recursion and dynamic data structures in bounded space: towards embedded ML programming. In *Proc. International Conference on Functional Programming (ACM). Paris, September '99.*, pages 70–81, 1999.

[Kon02a] Michal Konečný. LFPL with types for deep sharing. Technical Report EDI-INF-RR-0157, Laboratory for Foundations of Computer Science, Division of Informatics, University of Edinburgh, 2002. Shorter version with the title "Functional in-place update with layered datatype sharing" submitted for conference publication.

[Kon02b] Michal Konečný. Typing with conditions and guarantees in LFPL. Technical Report EDI-INF-RR-0151, Laboratory for Foundations of Computer Science, Division of Informatics, University of Edinburgh, 2002. Shorter version with the title "Typing with conditions and guarantees for functional in-place update" submitted for conference publication.

[Lei98a] K. Rustan M. Leino. Recursive object types in a logic of object-oriented programs. Technical Note 1997-025a, Digital Systems Research Center, Palo Alto, CA, January 1998. Superseded by [Lei98b], but with complete operational semantics and soundness proof.

[Lei98b] K. Rustan M. Leino. Recursive object types in a logic of object-oriented programs. *Nordic Journal of Computing*, 5(4):330–360, 1998.

[Ler02] Xavier Leroy. Bytecode verification for Java smart cards. *Software Practice & Experience*, 32:319–340, 2002.

[LTS01] Christopher League, Valery Trifonov, and Zhong Shao. Functional Java bytecode. In *Proc. 5th World Conf. on Systemics, Cybernetics, and Informatics*, July 2001. Workshop on Intermediate Representation Engineering for the Java Virtual Machine.

[MCGW02] G. Morrisett, K. Crary, N. Glew, and D. Walker. Stack-based typed assembly language. *Journal of Functional Programming*, 12(1):43–88, January 2002.

[Mic] Sun Microsystems. JavaCard software distribution and manuals. See `http://java.sun.com/products/javacard/`.

[Mic02] Sun Microsystems. White paper: The Project Monty virtual machine. Technical report, Sun Microsystems, Inc, 2002.

[Mil78] Robin Milner. A theory of type polymorphism in programming. *Journal of Computer and System Sciences*, 17:348–375, August 1978.

[MWCG99] G. Morrisett, D. Walker, K. Crary, and N. Glew. From System F to Typed Assembly Language. *ACM Transactions on Programming Languages and Systems*, 21(3):528–569, May 1999.

[Nec97] George Necula. Proof-carrying code. In *Proceedings of the ACM Symposium on Principles of Programming Languages*, 1997.

[NvOP00] T. Nipkow, D. von Oheimb, and C. Pusch. μJava: Embedding a programming language in a theorem prover. In F.L. Bauer and R. Steinbrüggen, editors, *Foundations of Secure Computation. Proc. Int. Summer School Marktoberdorf 1999*, pages 117–144. IOS Press, 2000.

[ORS92] S. Owre, J. Rushby, and N. Shankar. PVS: a prototype verification system. In *Proc. 11th Intl. Conf. on Automated Deduction, Springer LNCS vol. 607*, pages 748–752, 1992.

[Pau94] Lawrence C. Paulson. *Isabelle — A Generic Theorem Prover*. Lecture Notes in Computer Science 828. Springer-Verlag, 1994.

[pro03] Virtual Virtual Machine project. Web site at SOR, INRIA, France, January 2003.

[PS91] Jens Palsberg and Michael Schwartzbach. Object-oriented type inference. In *Proc. ACM Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*, pages 246–161, 1991.

[PWO97] Jens Palsberg, Mitchell Wand, and Patrick O'Keefe. Type inference with non-structural subtyping. *Formal Aspects of Computing*, 9:49–67, 1997.

[Rey78] J. C. Reynolds. Syntactic control of interference. In *Proc. Fifth ACM Symp. on Princ. of Prog. Lang. (POPL)*, 1978.

[Rey02] J. Reynolds. Separation Logic: A Logic for Shared Mutable Data Structures. In *LICS'02 — Symposium on Logic in Computer Science*, Copenhagen, Denmark, July 22–25, 2002.

[SA99] Raymie Stata and Martin Abadi. A type system for Java bytecode subroutines. In *ACM Transactions on Programming Languages and Systems 21*, volume 21(1), January 1999.

[TJ92] Jean-Pierre Talpin and Pierre Jouvelot. The type and effect discipline. In *Seventh Annual IEEE Symposium on Logic in Computer Science, Santa Cruz, California*, pages 162–173, Los Alamitos, California, 1992. IEEE Computer Society Press.

[TZH02] D. Teller, P. Zimmer, and D. Hirschkoff. Using ambients to control resources. In *Proc. 13th Intl. Conf. on Concurrency Theory (CONCUR 2002), Brno*, volume 2421 of *Lecture Notes in Computer Science*, pages 288–303. Springer, 2002.

[XH01] Hongwei Xi and Robert Harper. A dependently typed assembly language. In *Proc. 6th ACM SIGPLAN Intl. Conf. on Functional Programming*, pages 169–180, Florence, September 2001.