THE TEN15 PROJECT

Dr N E Peeling

Royal Signals and Radar Establishment, UK

INTRODUCTION

This paper is a description of the background and progress of the Ten15 project at RSRE. The emphasis of this paper centres on the potential advantages to the software engineering community of the widespread adoption of Ten15 as a kernel for software engineering applications.

Ten15 Foster (1) provides an algebraic basis for software development. It has three important aspects:

- First, it provides an interface that makes different computers compatible, thus ensuring the portability of all programs that are written on top of Ten15. The use of techniques developed for high level programming languages, such as compilation and strong type checking, means that this compatibility is achieved without compromising efficiency or integrity.

- The second major facet of Ten15 is that it provides a powerful set of facilities for implementing and integrating system components and databases within a distributed environment. Systems whose complexity or functionality makes them difficult to implement on conventional operating systems should be significantly easier to build on Ten15. Ten15 achieves this by providing facilities for dynamic and secure resource allocation in mainstore, filestore and over a network, within a comprehensive type system that has a sound mathematical basis. The sorts of system that Ten15 will support cover many important areas: high integrity systems, secure systems, IPSE developments, expert systems, heterogeneous networks, fine grain databases, and formal methods.

- Thirdly, Ten15 allows its users to preserve their existing investment in software. Ten15 runs efficiently on conventional computers. Ten15 can coexist with existing operating systems and can share data with them. Ten15 efficiently supports most existing programming languages and provides flexible mechanisms for mixed language working. This means that Ten15 provides an evolutionary advance in software engineering practice.

The paper starts with a description of the background that led up to the development of Ten15, showing how the different threads of the research programme in Computing Division, RSRE, came together to enable the design of Ten15 to be attempted. This section gives a description of Ten15 in a "bottom-up" manner and clearly shows the role serendipity has played in the development of Ten15. The next section gives a summary of what Ten15 is, in the sense of what it offers its users, with an analysis of the potential application areas that would benefit from the use of Ten15. This is a description of Ten15 in a "top-down" form that gives an idealised description of the requirements that Ten15 satisfies. The paper concludes with a summary of the current state of development of Ten15 and the plans for its introduction, demonstration and evaluation in collaboration with industry and academia.

BACKGROUND OF THE TEN15 PROJECT

Ten15 is a product of a long-term computing research programme at RSRE. The research team has in the past been responsible for the development of the first MoD standard language for real-time systems, CORAL66 (2); the first Algol68 compilation system, Algol68-R Currie et al (3); the Queen's Award winning Hardware Description Language, ELLA Morison et al (4); and the FLEX capability computer Currie et al (5) and advanced Programming Support Environment Stanley (6).

Ten15 is a part of the future support environments programme at RSRE which is aimed at reducing the costs and risks of software development for large complex distributed information systems for the MoD. Prime technical objectives of the research include support for new models of the design process (including prototyping), better software reuse, and better security and integrity.

Ten15 arose when two major strands of research within Computing Division came together in an unforeseen manner. The name Ten15 was first used for a formally defined intermediate language that was developed as part of the formal methods research programme. The intention was to run a new generation of algebraically based analysis tools on the Ten15 algebra, and to compile programs written in different languages into the Ten15 algebra prior to analysis.

The second major influence on the Ten15 project came from the FLEX project. FLEX is a computer architecture with the design aims of integrity, flexibility and support for highly interactive systems. FLEX is a tagged capability architecture with an instruction set orientated to high-level language compilers. It provides common, garbage-collected address spaces for programs and backing store. The FLEX view is that interactive programming is essentially the unanticipated combination of programs, and that a common address space is needed to allow data to be passed freely between programs while the capabilities provide control of access in order to maintain integrity. The garbage-collected common address space also allows FLEX to support dynamically creatable procedure values, which can be used to implement user-defined, high-level capabilities. Four hardware versions of FLEX have been implemented, in each case by microcoding; most recently on an ICL Perq2.

On top of this architecture a sophisticated Programming Support Environment has been implemented, with editors, extensible graphics, compilers and other tools. Unfortunately the need to implement the FLEX architecture in microcode has cut this advanced PSE off from the mainstream of software engineers who are equipped with 'conventional' computer hardware.

During the period 1985-87 the experience and results of the FLEX work and the formal methods work began to come together in a way which showed the way forward towards a portable software system with the characteristics of the FLEX PSE. The FLEX PSE has a system of datatypes which is used as a guide to help users, rather than as a mandatory checker, although the type mechanism in the FLEX PSE was more extensive than in the Ten15 algebra, as it then was, because of the need to handle operating system values as well as just programming language values. As the FLEX type system developed further, it was realised that the capability checks in the microcode were being used only as a safety-net when type-checking had been abused in the system programming language, Algol 68. It was then obvious that if the type system could be made all embracing, and was rigidly enforced, then the microcode checks would become completely redundant. In this way an abstract machine could be designed which would serve for compilers, operating systems, editors, databases, networks and users' programs alike. This machine was based on the compiler target Ten15 algebra and perhaps unfortunately not renamed.

The concept of Ten15 is now a high-level abstraction of the machine (or more accurately an abstraction of the languages that are used to program the machine) with a comprehensive set of types and operators, extensive enough to describe the complete programming world. With the realisation that completeness, integrity and efficiency were simultaneously possible, came the idea that Ten15 could be the common intermediate language for all compilers, and the Ten15 translator which compiled Ten15 into machine-code could be the sole machine-code generator in a complete system. The universal type system would provide a more refined access control than that of capabilities, with the checking done during compilation rather than being interpreted at run-time. A Ten15 translator can, like FLEX, offer a single, garbage-collected address space so that data can be communicated in unanticipated ways. It thus provides all the advantages of FLEX, but without needing a non-standard architecture to support it; porting an implementation of a Ten15 system to a different machine is now a matter of rewriting the Ten15 translator to generate instructions for the new machine's order-code, together with the reimplementation of a small Ten15 kernel that handles those aspects of the machine not accessible through the order-code (network, backing-store etc.).

It can be seen that a Ten15 system not only solves the major problem with FLEX (the need for special-purpose hardware support): it also extends its virtues. In summary, these improvements are:

- A full type system providing better checks than the single word checks provided by capabilities.

- A Ten15 system can coexist with existing (conventional) systems, and can share data with them.

- New compilers are not needed for new machines.

- The Ten15 system types are those of the system programming language, called Ten15-notation, which is based directly on the Ten15 algebra.

- The Ten15 algebra provides a well defined interface for mixed language working.

OVERVIEW OF TEN15

This section concentrates simply on what Ten15 is and what it can do.

Ten15 is a kernel layer of software that sits between all programs built on top of Ten15 and the host computer system. This kernel hides the host machine completely in the sense that all user code obeyed by the computer system is generated by the Ten15 software. The definition of Ten15 contains constructs that can efficiently implement all the features of the common programming languages, which allows compilers for languages such as Ada, Pascal, C, ML etc. to compile through Ten15. Ten15 contains features for controlling all aspects of the computer system including those not normally reachable from conventional programming languages e.g. backing store, databases, Local Area Networks, and the loading and execution of programs.

Because Ten15 completely hides the underlying machine, it can be thought of as a machine in its own right. It is however an abstract machine in the sense that it is not tied to a particular hardware implementation. Any program, including compilers and other hitherto machine dependent tools, written on top of the Ten15 machine will run unchanged on any hardware on which the Ten15 system is implemented. The only limitation to the compatibility provided by Ten15 stems from the availability of special hardware on some systems e.g. signal processing chips and array processors.

It is important to be clear about the relationship between Ten15 and an operating system kernel, such as UNIX (UNIX is a trademark of AT&T). Ten15 could be installed as the native operating system kernel on a computer system (referred to as a "bare machine" implementation). A less radical approach is to implement Ten15 on top of an existing operating system. The advantage of this approach is that Ten15 can be used on an existing computer configuration; Ten15 will be able to communicate with tools running on the host operating system, and the Ten15 implementation will be able to use the existing device drivers provided by the host. The performance degradation compared with the bare machine implementation is likely to be small and the trustworthiness of the system should be acceptable for all but the more stringent security and integrity requirements. The ability to implement Ten15 on top of an existing operating system and hence access tools running on the host, together with the provision of compilers for existing programming languages, greatly enhances the migration route to Ten15.

As previous efforts to define a "universal" abstract machine have foundered on the grounds of unacceptable run-time performance, Ten15's claim to provide acceptable efficiency has to be justified. The early abstract machines were attempts to abstract machine architectures and as a result were low-level and machine-like. The mapping of the abstract machine to concrete architectures was simple, and hence cheap, but allowed little scope for optimal use of the features of a particular machine, and hence performance was very inefficient compared with direct use of the underlying computer architectures. Ten15, on the other hand, is an abstraction of high-level programming languages and consequently is at a much higher level than the early abstract machines. Ten15 preserves most of the high-level structure of the programs that compile into it. This information can be used to drive all conventional code optimisation strategies in the implementation of a compiler/translator from Ten15 to a computer's order-code. This means that the run-time performance of a program compiled through Ten15 can be as good as that achieved with a native compiler.

The attitude to performance is perhaps best summed up by a target the designers of Ten15 set themselves: that no program running on a Ten15 system should take more than 20% longer to run than an equivalent program running on the same machine under a conventional operating system. Current tests indicate that this target is easily achievable and research is concentrating on removing the overhead altogether. Ten15 has the added advantage that many programs which take advantage of Ten15's more advanced features will run very much faster when implemented on a Ten15 system.

Another potential problem for a abstract machine that tries to be complete is that it might grow like Topsy, ending up both large and complex. There are a number of obvious advantages in keeping Ten15 as compact and simple as possible. It not only keeps the overhead of the system code size small; it also reduces the cost of rehosting a Ten15 implementation; and given the possibility (discussed later) of Ten15 being a suitable basis for satisfying the most stringent security and/or safety requirements, it helps to bring the Ten15 implementation within reach of the current capabilities for formal software proof. The code size of a typical Ten15 implementation gives some idea of the success that the designers of Ten15 have had in containing the size of Ten15. The Ten15 implementation consists of two main components - a translator which compiles Ten15 constructs to the machine-code of the host computer, and a run-time kernel that looks after peripherals, storage allocation and the like. The translator on VAX/VMS (VAX and VMS are trademarks of digital equipment corporation) occupies less than 100 Kbytes and the kernel is less than 200 Kbytes.

One of the principles used to keep Ten15 small was to include only general purpose primitives. New primitives were only considered for inclusion if they could not be built out of existing Ten15 features, or if they could be implemented much more efficiently by building them into the definition of Ten15.

The provision of integrity and security is built on this same principle. The definition of integrity which Ten15 implements is very low-level; integrity is taken to mean the ability of the system to limit the access that a program has to the resources of the computing system: in particular, to limit the amount of damage that program can do, no matter how maliciously it behaves. This kind of integrity is unarguably a property that all systems will want to share. However, it does not preclude the design of a system that allows users to access data that offends some higher level notions of privacy, such as those needed in a military secure system. The Ten15 viewpoint is that there is no universal higher level definition of privacy/integrity that will satisfy all the requirements of different user communities, and so the job of Ten15 is to provide a soundly based toolkit which can be used to implement easily any particular notion of privacy/integrity. Certainly the low-level definition of integrity enforced by Ten15 is essential for the user to be able to assure himself that any program he writes to police access to an item of data cannot be circumvented by another potentially malicious program. For the implementor of a secure system, Ten15 provides a mechanism for hiding a resource behind a procedural interface. The procedures in the interface can then implement a policy for limiting access to that resource. Ten15 guarantees that the only route to that resource is through the procedural interface.

Just as Ten15 provides general purpose primitives to synthesise larger constructions, so programmers are encouraged to create general purpose components within their field of interest. This implies that the composition of general purpose components must be very efficient. To achieve this the Ten15 system provides very fast data communication between the parts of a system. The mechanism chosen is to run all programs in the same mainstore address space, and to provide sophisticated control of access, and storage allocation/deallocation. This mechanism looks potentially risky from the point of view of integrity, because a mistake in storage access in one program might give access to another program's data. Ten15 actually provides very good mainstore integrity and it can do this as a consequence of the fact that it is a complete abstract machine. By denying the user any access to the native machine instructions it keeps total control over memory allocation and deallocation. It backs this up with an unbreakable type checking system that ensures that no Ten15 operation can be applied to inappropriate data.

The use of mathematics has played a fundamental part in the design of Ten15 - particularly Goguen et al (7) and Martin-Löf (8). First, mathematics has been used to help understand the structure and semantics of Ten15, and has proved invaluable in exposing which were the most general purpose primitives to include. As a result it should be possible to produce a comprehensible formal definition of Ten15. Secondly, Ten15 is itself a mathematical entity, having an algebraic structure. An algebraic structure was chosen because it produces a representation of programs that is easy for programs to create, analyse and manipulate. The Ten15 representation of programs output by compilers is a language independent form on which automatic analysis and transformation can be performed. The role of Ten15 as a syntax independent representation of program is likely to become increasingly important. One of the earliest benefits will be the ease with which systems can be created from Ten15 components that were compiled from a number of different languages.

Potential Benefits of Ten15

Ten15 has an important part to play in improving the software engineering process "in the large". Admittedly, this view somewhat contradicts the perceived wisdom that the basic techniques of programming are now a mature technology and there is little prospect of any radical improvement, such as occurred in the move from assemblers to high-level programming languages. In terms of the writing of free standing programs this may be true. There is however great scope for improvement in the production of larger systems comprising many cooperating programs, possibly running on many machines in a distributed environment, with one or more significant sized, complex databases. To achieve inter-process communication, database manipulation and distribution one requires separate programs to communicate using the features provided by the operating system. Many of the system programming facilities offered by the widely available commercial operating systems are, by programming language standards, still at the assembler stage of development. A good example of this is provided by the extensive use of byte streams in UNIX for the contents of files and pipes. Those system programming facilities that have advanced from this stage (e.g. 4GLs) have unfortunately developed notions of data structuring that are incompatible with accepted standards in programming languages. The consequence is that the effort of implementing larger systems is now dominated by overcoming the restrictions imposed by the operating system and by the mismatch between the ways the operating system and programming languages structure, communicate and store data. Ten15 provides a framework that unifies the notions of program and data between programming languages and operating systems. As a result, the effort of designing large integrated systems can be significantly reduced. This effect has already been demonstrated in single language special purpose environments such as LISP and SMALLTALK systems.

Ten15 not only increases productivity by providing a uniform programming environment free from the petty restrictions, reliance on low level data types, and black magic of current system programming practice, but also makes it much easier to design reusable tools and tool fragments. The ability to pass typed data between tools via mainstore or filestore, instead of having to flatten the data into a byte stream for passing down a pipe or storing in a file, makes input and output to tools as good as for high-level language procedures. Experience with systems such as FLEX, LISP and SMALLTALK have shown that this encourages the development of reusable tools.

Although Ten15 can reduce software production costs across the board - and in the long term this may be seen as its most important property - initial uses of Ten15 are liable to be concentrated in application areas where it provides more specialised benefits. This section concludes with an analysis of the niche markets where Ten15 will be most attractive.

CASE tools. The support offered for the software engineering process will make Ten15 an ideal base for many CASE tools. In particular, any new operating system or Integrated Project Support Environment could benefit enormously from the use of Ten15 as its machine independent kernel.

Heterogeneous networks. The compatibility between different computers that the Ten15 abstract machine provides not only serves as a portability interface for systems built on Ten15, it also allows dissimilar computers to cooperate with one another in a distributed environment. As well as providing a common level for communication between different computers, Ten15 includes advanced features for interworking within a loosely coupled (over a LAN) distributed system, such as dynamically creatable remote procedure calls. The problems of communication within a tightly coupled, massively parallel computer system (e.g. an array of transputers) is the subject of an ongoing research project.

High integrity/security systems. The features Ten15 offers to the builders of high integrity and/or high security systems will make Ten15 a very good kernel for the design of systems whose failure cannot be tolerated on safety, security or commercial grounds. This is a fast growing market: for example, the recent publicity surrounding fly-by-wire avionics systems, hackers, viruses and computer crime has done a lot to increase the market awareness of the value of safe and secure systems. One of the attractions of Ten15 is that it can be used to build a portable system that can then be sold with varying levels of cost/security: from the most expensive/most secure system implemented on a special purpose computer with hardware support for security (in military terms, capable of offering beyond A1 levels of assurance), through a bare machine implementation of Ten15 (possibly capable of A1 assurance), to cheaper and still quite safe implementations based on secure commercial operating systems, ending up with the cheapest system built on top of a standard operating system which offers significant improvements to integrity while still being susceptible to a determined attack from the untrusted host operating system.

Support for modern programming styles. Ten15 contains many features that support modern programming paradigms. These features include a common, garbage-collected address space, support for first class procedure values, dynamically creatable remote procedure calls, an intelligent demand loading system for the code of procedures, a persistent heap filestore, and a polymorphic type system. This means that Ten15 will support object oriented programming languages, object-oriented databases, persistent programming languages such as PS-Algol Atkinson et al (9), polymorphic programming languages such as ML and functional programming languages. Ten15 may be a means of integrating a number of the good ideas in these different styles. Ten15 also provides automatic garbage collection for conventional languages such as Ada, Pascal and C. Many of the more advanced systems written in these languages expend a great deal of effort doing manual storage deallocation, a process which is very difficult to get right, very difficult to de-bug when it is incorrect and, even when correct, often incurs significant run-time overheads.

Databases. The concept of filestore in Ten15 extends programming language data structuring in an obvious way onto the backing store. The approach adopted is to provide general purpose primitives which separate the idea of mainstore pointers from disc pointers. Bringing a data structure from disc into mainstore, i.e. turning a disc pointer into a mainstore pointer, has to be done explicitly. The mechanisms underlying the Ten15 filestore are capable of supporting complex databases where individual items can, if necessary, be very small (the so-called "fine grain" database). The Ten15 filestore is a potentially more flexible and efficient implementation mechanism for building databases than are existing technologies such as relational and entity/relationship/ attribute databases, and will be much easier to integrate with the programming languages used to implement the system.

There is a lot of interest at the moment in hypertext interfaces to databases, in which the hierarchical nature of a database is explicitly visible on the computer screen.

The RSRE FLEX computer system (which has a filestore similar to Ten15's) supports an advanced hypertext user interface. The ease with which this interface was implemented suggests that the Ten15 filestore will be ideal for supporting such interfaces.

Formal methods. The mathematical definition of Ten15 makes it a suitable framework in which to analyse the properties of programs automatically. Ten15 provides features which will allow the results of such analysis to be communicated to the user in terms of the language in which the program was originally written. Because the definition of Ten15 covers those features that support system programming, Ten15 also offers the possibility of analysing complete systems as well as free-standing programs.

Ten15 was specifically conceived to support design by transformation. In this approach, an algorithm is expressed initially in high level constructs in a clear way that can be seen to be correct. It is then transformed by methods which are known to produce programs of equivalent effect into a version that will run with adequate efficiency.

Formal specification languages such as Z and VDM are gaining currency. Having completed a specification in such a language, the next step is to produce an implementation that satisfies the specification. The implementation may be produced independently or it might be developed by a refinement process from the specification. Ten15 provides a formal description of the implementation which can be a basis for a formal comparison with the specification.

CURRENT STATUS AND FUTURE PLANS

There is already a prototype translator and kernel for VAX/VMS. A kernel for UNIX is being produced by PRAXIS plc and a translator for Motorola 68000 is being produced at the University of York. RSRE are developing an evaluation system for Ten15. This system would comprise:

- A simple but powerful Human/Computer Interface which will be based on the experience gained with the editor from the RSRE FLEX PSE. This will use an advanced hypertext format and will be user-extensible.

- Compilers for Pascal, Algol68 and Ten15-notation. Ten15-notation is the main implementation language of the evaluation system and serves as both an assembler for Ten15 (in that it allows text to be written that will generate exactly any piece of Ten15 required) and as a high-level system programming language.

- A symbolic Ten15 debugger which the compilers for the different languages tailor to their individual syntax.

- A separate compilation system.

- A framework of tools that allows the algebraic structure of Ten15 programs to be manipulated by user-written programs.

- A PostScript (PostScript is a registered trademark of Adobe Systems inc) output for driving laser printers.

A version of this evaluation system that runs on a standalone workstation (initially VAX/VMS) could be available in the second half of 1990. Further releases that run initially on homogeneous, and later on heterogeneous, networks of VAXs and SUNs, could be available in the following twelve months.

Planned enhancements to the evaluation system include compilers for Ada and Standard ML, and retargetting to RISC architectures. These enhancements should start becoming available in 1991.

The MoD will be using the evaluation system to undertake demonstrator projects in the area of secure systems. RSRE are however keen for other organisations outside MoD to use the evaluation system for demonstrators in the wider context of the UK software industry and advanced software engineering research. This paper should provide enough information on the most likely areas where Ten15 could provide considerable extra leverage. Any organisation, academic or commercial, wishing to discuss this opportunity further should contact the author.

ACKNOWLEDGEMENTS

Ten15 was designed by P W Core, I F Currie, and J M Foster.

REFERENCES

1. Foster, J.M., 1989, "The Algebraic Specification of a target machine: Ten15", High-integrity software, ed Sennett, C.T., Pitman.
2. Ministry of Defence, "The Official Coral Definition", HMSO London.
3. Currie, I.F., Bond, S.G. and Morison, J.D., 1971, "ALGOL 68-R", Algol 68 implementation, ed Peck, J.E.L., North-Holland, Amsterdam, pp 21-37.
4. Morison, J.D., Peeling, N.E. and Thorp, T.L., 1985, "The Design Rationale of ELLA, A Hardware Design and Description Language", Proc CHDL 85, Tokyo, Japan, pp 303-320.
5. Currie, I.F., Edwards, P.W., and Foster, J.M., 1982 "Flex: A working computer with an architecture based on procedure values", RSRE Memorandum 3500.
6. Stanley, M., 1986, "An evaluation of the Flex Programming Support Environment", RSRE Report 86003.
7. Goguen, J.A.,Thatcher, J.W., Wagner, E.G. and Wright, J.B., 1976, "An initial approach to the specification, correctness and implementation of abstract datatypes", Current Trends in Programming Methodology.
8. Martin-Löf, P., 1975, "An intuitionist theory of types: predicative part", Logic Colloquium, 1973, ed Rose and Sheperson, North Holland, Amsterdam, pp 73-113.
9. Atkinson, M.P., Chisholm, K.J. and Cockshott, W.P., 1981, "PS-algol: An Algol with a Persistent Heap", ACM SIGPLAN Notices Vol 18, No 7, pp 24-31.