A Tale of BETA's Wayward Son

Michael I. Schwartzman
President, ValueSearch Capital Management, LLC
September 2004

"The true success of a project undertaken by an academic institution is in the legacy of ideas and students it created; if these are good, the project has succeeded."
Professor John Guttag[1], MIT

"The first million is always the hardest one to make."
American folk wisdom

"Any idiot can face a crisis – it's this day to day living that wears you out."
Anton Chekhov at the end of the 19th century, regarding the plight of humans in general, and perhaps the plight of managers of some complex engineering projects in the 20th century.

Introduction

By 2004, when I am writing this paper, I have spent exactly the same number of years in two very different fields: seventeen years, from 1970 to 1987, in Computer Science and Software Engineering; and seventeen years, from 1987 to 2004, in Investing and Money Management. This story will address events that occurred a long time ago, in the first seventeen years.

I participated in project BETA for three years, from its inception in 1970 until I left the project to emigrate to the United States at the end of 1973. I was in charge of development of the Internal Language of BETA, a task that was picked up by other members of the project after I left. First I will address only those aspects of Andrei Petrovich Ershov's legacy as they pertain to project BETA, and later I will address the non-scientific aspects of his legacy.

Very briefly, the goal of BETA was to produce a multi-language compiler that would translate every one of the source languages it accepted into its Internal Language, which

[1] John Guttag -- formerly the Head of Electrical Engineering and Computer Science Department at *MIT*, currently Director of Networks and Mobile Systems Group of MIT Computer Science and Artificial Intelligence Laboratory.

in turn would be translated into the (machine) language of a given family of computers. The compiler would perform intensive optimizations on the level of the Internal Language.

The source languages contemplated at the inception of BETA were

- Algol-68
- PL/1
- Simula-67
- Pascal

The BETA project, which wound down in 1982, twelve years after its inception, seems to be universally considered to have ended without producing a usable system. Of course, project BETA shed much light on the structure of the languages with which it dealt, and on various aspects of compiler design.

It is often assumed that the fate of BETA has befallen all other attempts to produce a practical multi-language compiler system. While BETA itself did indeed end without producing a working system, there is an intellectual offspring of BETA that achieved all the goals of BETA and perhaps far more.

This project is the LPI Multi-Language Family of compilers (LPI-MLF) produced by Language Processors, Inc. (LPI), a company that I founded in Massachusetts in 1980. Soon after I founded the company, John Ankcorn[2] joined me and until 1986 was in charge of all advanced research efforts at the company including the development of LPI-MLF.

The project, while widely known and discussed in the Research and Development departments of all US and quite a few foreign computer manufacturers, never caught the attention of the scientific community – just the reverse of project BETA. Of course, during the development and active selling of LPI-MLF, LPI did not encourage and probably would have prohibited publication of any materials by its employees that would give an edge to a competitor.

LPI-MLF was the first multi-language compiler system that succeeded in practice. It resulted in a set of compilers that were very competitive with stand alone compilers written for specific languages. The system was practical, usable, and had been actually retargeted and extensively used on several very different computer architectures -- in short, it was the first production quality multi-language compiler system.

After LPI-MLF became obsolete, the key participants moved on to other interests and duties, some very challenging: John Ankcorn moved to Japan and founded a hardware company there; I founded an investment firm in Massachusetts. As a result, not much

[2] John Ankcorn -- currently a Senior Researcher at *Hewlett-Packard* Research Laboratories, a specialist in Mobile Computing and Networking.

about LPI-MLF was ever published except, of course, sales and promotional literature, and articles in trade publications. (Please see Attachment A for an example of the many articles about LPI published at the time, and Attachment B, C, and D for samples of LPI's own marketing literature.)

Results Achieved by LPI

By 1984, when all components of LPI-MLF were completed and were shipping to commercial customers, LPI-MLF consisted of the following:

- Front-ends for
  - COBOL
  - RPG-II
  - FORTRAN
  - PL/1
  - Basic
  - Pascal
  - C

- An optimizer on the level of the Intermediate Language. The optimizer had several user-selectable levels of optimization.

- A large and growing collection of code generators for many computer architectures, including Motorola 68000, Intel x86, IBM 8100, ATT 3B2 and ATT 3B20.

- A source-code level debugger.

- A common run-time library.

- A collection of tools that would allow the production of table-driven code generators for new architectures. The generators themselves used expression tree reduction techniques.

- A sophisticated system for automatic testing of compilers, and

- Tools based on the most advanced research of syntax analysis for producing front-ends for new languages.

The customers were large and small computer manufacturers, start-up computer manufacturers and start-up divisions of companies venturing into computer manufacturing (AT&T Information System is a good example of this). The OEM agreements with the customers took a great deal of time to negotiate and competition with the internal R&D staff of many OEM customers was intense.

By 1987, when the system reached its peak of functionality and versatility, a code generator for a new architecture could be running as a prototype in a week's time, and a production quality code generator took about four months to produce. The technology of putting components together became so good that a release of a compiler could be built and fully tested in several hours, thus enabling a very quick cycle of new releases. This was at a time when the industry usually took six to twelve months between releases. At that time building a single language compiler for a new machine took several years, and the projects often failed. The LPI-MLF made it possible to obtain a new compiler in several months, and to add additional compilers every few weeks.

Very few OEM customers bought the entire seven language compiler system. What LPI customers liked was the ability to obtain one or two compilers very quickly (often C or FORTRAN, and perhaps COBOL), use one of these compilers to port essential applications to their new computer architecture, and have the option of adding new compilers as their business required.

Thus, to customers, LPI's technological breakthrough was of value not for the ability to have a multi-language compiler system on their machine all at once, but the ability to add new compilers quickly and at a far lower cost than LPI competitors and the customers' internal development staff could provide.

The entire system was fairly large by the standards of the mid-80's. Its footprint on hard disk was 16 Megabytes, and considering that hard disks were around 30 to 50 Megabytes, this was one of the impediments for many customers to obtaining the entire system. Most of the source code was written in PL/1, so the customers who licensed the source code also needed the PL/1 compiler of LPI-MLF.

The system was competing against traditional compilers, produced one by one, and optimized for their specific function. Nevertheless, no significant sales would have been possible if the LPI-MLF compilers were inferior to competing ones. They were not, and FORTRAN, C, and COBOL consistently won various compile and run-time benchmarks. In fact, running benchmarks and tuning the code generator for a given architecture to compete with existing compilers for the same architecture took the lion's share of the cycle of producing the LPI-MLF for a given architecture.

Dealing with complaints from the OEM customers' pesky internal staff often bent on proving any acquired software as inferior was another major part of this cycle, but this would have occurred no matter what technology was used in the acquired software. The 80's were indeed different from the 90's and today in the attitude of internal staff of computer manufacturers.

That there are so few manufacturers left for system programmers to choose from may have something to do with this change of attitude. The fact that those manufacturers that were most recalcitrant in reigning in costs of system development were also the ones that came upon hard times and were acquired in distress or ceased to exist at all may also have instilled a different business approach in the 90's and today. But that is now – LPI

sold its LPI-MLF at the height of "make vs. buy" battles inside R&D departments of computer manufacturers.

It would take volumes to describe all of the difficulties, failures, victories, and the immense amount of work that went into designing, documenting, and debugging LPI-MLF. The frustration in R&D ran very high at times, and the efforts required to keep the company on course were monumental.

LPI-MLF was a successful system, licensed to a dozen leading computer manufacturers in the United States and Japan and to a fair number of start-up computer manufacturers. Its compilers not only competed successfully with traditional compilers, but surpassed many of them in the traditional measures of compiler performance such as compile and run-time benchmarks. LPI-MLF cut time-to-market and acquisition cost for compilers several fold.

LPI-MLF achieved all practical goals set out for BETA fifteen years earlier, did so for a very diverse set of languages, and made this achievement with a very small group of designers: at the height of a development process that took three to four years, LPI's R&D group had about thirty-five to forty people.

Roots of LPI-MLF and Reasons for its Success

LPI-MLF was built after many years of attempts by compiler designers, BETA included, to build a multi-language compiler system. Of course, the idea of a multi-language compiler system was not born in BETA. It had already been discussed for many years.

The intellectual influence of BETA on LPI-MLF is unmistakable; I was the founder, President and Chairman of the Board of LPI for its first seven years. The very idea of what became LPI-MLF came to me, in part, out of my experience at BETA. BETA and the follow-on experience of introducing common components for a family of compilers at Digital Equipment Corporation (DEC), as well as managing the compiler development department at Prime Computer were, among other influences, at the foundation of our approach at LPI.

It must immediately be observed that BETA was but one leg of the multi-legged foundation in LPI's approach. John Ankcorn, whose guidance of R&D was indispensable in achieving success, brought ideas and methods that were not rooted in BETA at all. My own experience was far from being limited to BETA by the time LPI was founded.

The R&D department at LPI contained a dozen world-class compiler designers, some with doctorates in Computer Science and all with their own strong ideas as to how to achieve success and with their own definitions of success. Day-to-day guiding of this group was Ankcorn's job. Overall management of the company, including setting the

pace for R&D, its goals and definitions of the product, sales, marketing, and raising capital were among my tasks.

BETA, similar projects elsewhere, and many projects of similar, extreme complexity died the death of a thousand bites, a thousand small failures. They died because the sheer number of many relatively small design problems would reach a tipping point and the project would fail irretrievably. In fact, explaining what it was that prevented success of a multi-language compiler idea in the past was by itself a frustrating experience. Every small problem, be that of a certain construct in the Intermediate Language, a memory limitation, etc., could easily be resolved on the spot by an expert – it was the sheer number of small interconnected problems that needed to be overcome.

A balance between pragmatism on one hand, and the desire of the scientists at LPI to apply all of their extremely good knowledge of the academic literature on the other hand, needed to be maintained in LPI's R&D department. LPI succeeded spectacularly in finding this balance and in harnessing the problem of solving a thousand small problems before they caused a catastrophic failure.

A dry run, so to speak, of attempting to introduce common components in a compiler system was made by me at DEC. There, under the grueling schedule of designing a new computer (VAX), a new operating system (VMS), and a series of new compilers, FORTRAN being the first one, an attempt was made to introduce a Common Run-Time Library that would serve this compiler and future compilers and provide a library of routines for other VAX sub-systems to use. The intense schedule pressure of the entire effort resulted in the remarkable achievement of going from a paper specification of a new architecture to the shipment of the first VAX computer in just three years.

The battles were memorable, to say the least. The enclosed memo (Attachment E), from Dave Cutler[3] who was in charge of a relatively small team of twenty five designing the future VMS, to Richard Grove[4], who worked for me and was in charge of FORTAN compiler development, represents some of the arguments and issues discussed when new concepts such as a Common Run-Time Library and associated dependencies were introduced. Their introduction affected the linker (a sub-system of the operating system) designed by Trev Porter[5], who worked for Dave Cutler. At the time Dave wrote the enclosed memo, the design of the linker was already being overwhelmed by conflicting requirements and was falling behind the schedule. Introducing the Common-Run Time Library was not helping the situation.

[3] Dave Cutler -- currently manages a large team of system programmers developing several Windows versions at *Microsoft Corporation.*

[4] Richard Grove -- a specialist in compiler development, currently a senior researcher and a Fellow at *Intel*.

[5] Trevor Porter -- formerly, in charge of developing the linker and several other components during the initial development of VAX/VMS.

Roger Gourd[6] was Cutler's boss, so to speak, although at that time at DEC many things got done by individuals who reported, in truth, to nobody except their own conscience and the boss of all of us, Gordon Bell[7]. Dave was one of them, and my group contained one or two of these free spirits too. It was quite a challenge for managers like Roger and me to know when to put our foot down, when to cheer and lead, and when to get out of the way. Roger was a very good practitioner of this art, and I was still learning but learning fast.

I have a small collection of memos, some from me to Dave Cutler, which I hand-wrote and hand delivered to avoid involvement of anybody including secretaries who typed memos in those pre-email times. These memos, whose language and strong expressions were remarkable for their clarity, to say the least, and which required ultimate confidentiality to avoid unnecessary embarrassment of the recipient, carry the imprint of the battles for turf, ideas, and scheduling of the all important first shipment of VAX/VMS.

I also have a collection of such memos written to me or to my subordinates and bosses. Reading them no longer hurts. It is sufficient to observe that at DEC during VAX development the weak ones got crushed but the strong ones, after trashing each other mercilessly, went out for a beer and, if they did not become friends, retained a life-long respect for each other. I count myself among those survivors.

Ron Ham[8] was one of my bosses in the multi-dimensional management matrix at DEC during these battles. After one particularly contentious meeting chaired by me, when he received a flurry of phone calls asking him to intervene and overrule one of my decisions, Ron Ham told me that he sometimes felt like the man with the broom standing in the middle of Main Street after a circus (with elephants) passed through town. My decisions survived in no small measure because Ron's wisdom and management experience allowed him to withstand pressures to intervene.

The idea of the Common Run-Time Library for VAX finally survived all battles, and the Library became a part of VAX/VMS.

In the recent words of Rich Grove, 'The Common Run-Time Library was a great success with many features that we now take for granted:

- Common calling conventions,

[6] Roger Gourd -- formerly, a manager in charge of developing the initial version of VAX/VMS at *DEC.*

[7] Gordon Bell -- formerly, Vice-President of Advanced Research at *DEC*, in charge of all, hardware and software, VAX-related efforts during its initial development. Currently a senior researcher at *Microsoft Corporation.*

[8] Ron Ham – formerly, a manager in charge of compilers and filing systems development efforts for the late stage PDP-11 and the initial stage of VAX.

- Multi-language run-time interoperability,
- Exception handling system that supports [languages as complex as] PL/1, Ada, etc. and that is used for all message and error reporting in VMS,
- State-of-the-art multi-language source debugger, and
- Integrated network file system.'

All participants of this dust-up and other, similar battles went on to most illustrious careers.

Dave Cutler, who already was a towering figure at DEC, went on to Microsoft and was in charge of development of several key versions of Windows, including NT. He surely deserves inclusion in the Guinness Book of Records as the only person on Earth who was ever in charge of designing widely used operating systems for at least three generations of computer systems: PDP-11, VAX, and a succession of Intel-based Windows computers. Microsoft did not wait for a cue from the Guinness Book and has already made Dave a Senior Distinguished Engineer. Dave is still a key influence in the future of Windows.

Richard Grove is now a Fellow at Intel, one of very few at Intel who achieved this status.

There were valuable lessons learned at DEC, such as developing great products on a frugal budget while combining advanced concepts from academia with ruthless pragmatism. These lessons and skills developed at DEC were carried to LPI by John Ankcorn, myself, and people we later hired away from DEC.

I was the initial designer of the Intermediate Language at BETA, called "Internal Language" in project BETA. After I left the project in 1973, my design, clearly too abstract and formal for a project aiming at a working compiler, was replaced by a different design of which I knew nothing during my LPI years and about which I still do not have much knowledge.

During the last years of the Soviet Union, a pre-Internet and pre-email era that now seems like the distant past, my contacts with project BETA were completely cut off. This was largely my doing, in order to avoid tainting my former colleagues at BETA with association with, shall we say, a less than favorite individual of the Soviet authorities. I left the country when such a step was rarely taken, was discouraged by all available means, and was often severely punished by the authorities. I settled in the "citadel of capitalism," the United States. One could not blame BETA team members for not making heroic efforts to keep in touch with me; their careers and their very employment as scientists would have been at stake.

A small number of semi-clandestine meetings (sometimes with a whiff of Inspector Clouseau) did occur with an occasional visitor from BETA or an associated project, including with Andrei Petrovich Ershov, but we did not go into details of BETA at those brief and rare encounters.

Very little written contact occurred between BETA participants and me. I still remember how in 1986, when I found out about Ershov's illness, writing to him caused considerable wavering on my part between the desire to offer a warm word to him, and to protect him from being tainted by contact with me. In the letter I finally sent him (Attachment F), I intentionally chose a stuffy and officious "Dear Professor Ershov" on my company's letterhead instead of the customary "Andrei Petrovich" and a handwritten personal letter more appropriate for this occasion. This was done to put some distance between me and Ershov – a distance that Ershov would have been smart enough to ignore, while others who would have read the letter would hopefully have been fooled. To this day I do not know whether Andrei Petrovich ever received the letter.

Thus, contacts were effectively non-existent. BETA and LPI-MLF, started at LPI ten years after BETA's inception, proceeded completely independently of each other, except for the connection that I represented and which, for better or worse, captured the status of BETA in 1973.

I did not develop the Intermediate Language at LPI. By then I was doing what LPI's corporate marketing materials described as "providing strategic direction for the company." The Intermediate Language at LPI was designed largely by John Ankcorn and his group. It had among its intellectual precursors the internal language of PL/1 compilers designed by Robert Freiburghouse for MIT MULTICS and at his own company, Translation Systems, Inc. Freiburghouse later became a co-founder of Stratus, a successful manufacturer of non-interruptible computer systems.

As to my job of providing "the strategic direction," I have found an old memo (Attachment G) I wrote probably after one particular intense meeting with R&D where the strategic "directions," so to speak, were being provided by both sides. The memo is self-explanatory and will be understood well by anyone who has run advanced R&D departments.

The optimizer that worked on the level of the Intermediate Language and the front-ends had a multitude of practical and intellectual precursors, among them the outstanding FORTRAN optimizers at DEC, a robust PL/1 front-end designed by Freiburghouse and intensive academic research on compiler optimization.

The code generator technology evolved from the ad-hoc design of code generators translating the Intermediate Language into the language of a particular computer architecture to an outstanding system based on academic research that allowed table-driven definition of the target machine and production of a new code generator in a few weeks. This system was called 'The Generator of Code Generators' in LPI's marketing literature.

There was, of course, more to LPI than solving the elusive technological problem of designing a working multi-language compiler system. There was also a need for raising capital, attracting talent, dealing with large computer manufacturers whose internal

development staff could turn either hostile or friendly and not always rationally, and the execution of an occasional cloak-and-dagger maneuver. On one such occasion an LPI lawyer and I cornered in a bankruptcy courthouse a particularly nasty and underhanded competitor who was driven to bankruptcy not in a small measure by his own deeds, and had him sign off to LPI those assets of his company that could be used to impede LPI's success. Thus yet another problem out of the thousand on the path to LPI-MLF was solved.

LPI succeeded in producing LPI-MLF because of the clear understanding by its founders of the trade-offs involved in solving one thousand small problems at once without losing the big picture, and because of our ability to maintain the discipline and incentives required when running an R&D staff that is independent by its nature. For the reader who might not be familiar with the difficulties of running an R&D organization, "herding cats" would adequately describe this occupation.

BETA may well have failed as a project aiming at producing a family of usable compilers because the incentives inherent in the Soviet Academy of Sciences were at variance with the solutions that were needed to achieve BETA's success. These incentives caused the material and academic rewards to go to those with the most publications and advanced degrees, with the exception of scientists who worked on armaments and related problems. This warped incentive system affected all involved in project BETA, from the most junior researcher to A.P. Ershov and his academic superiors.

This may be a problem inherent in many academic institutions, including university research staffs in the United States, but in the Soviet Union the problem was exacerbated by many factors specific to a command economy. These factors further interconnected into often unbreakable riddles, such as lack of mobility of researchers, the extreme degree of correlation between academic titles and access to housing, and so on.

I was heartened to notice that very recently a former participant of BETA, Sergey Pokrovsky, made an observation very similar to my observation above. He wrote with the gentlemanly non-accusatory detachment so characteristic of him, that "…perhaps the [academic] inertia of project [BETA], or perhaps [Schwartzman's scientific] upbringing, or perhaps his orientation toward pure science (natural in light of his plans for advanced academic degrees) – something must have changed the direction of his research and from the moment his Internal Language was chosen for BETA, the language became more and more academic…" While I am not sure what Pokrovsky meant by "upbringing," the rest of his diagnosis, even put as gingerly as Pokrovsky is famous for, is right on the target.

It is only fair to observe that BETA was a difficult project and that similar undertakings failed elsewhere, where the extreme limitations of the command economy did not exist.

Of course, as I already discussed, the legacy of ideas and insights into compliers and programming languages created by BETA is an entirely different and a far more successful story, and LPI-MLF is a part of that story.

The Sunset of LPI-MLF

By 1987, threats to LPI-MLF as they are described below had become very palpable. I had made a plan to diversify away from LPI-MLF by investing in development of a new promising product -- a replacement for the then becoming obsolete Microsoft's MS-DOS operating system and its still primitive Windows extension. However the required investment for this undertaking was hard to obtain because the Board of Directors of LPI was split as to the validity and degree of threats faced by LPI-MLF. A part of the Board did not support such investment. I sold my shares in LPI in 1987 and left the company.

LPI-MLF sold well for a period of a few short years, 1983 to 1990. By 1989, the Board finally saw the threats to LPI-MLF, decided to diversify away from this system by acquisitions and promptly did so. LPI merged with another company and later took on a new name, Liant Software Corporation. (Please see Attachment H for a glowing letter from my successor, Roy Finney.) By then, the old management was gone (John Ankcorn left in 1986) and the threats to LPI-MLF detected in 1987 had finally materialized.

Liant has narrowed its focus to COBOL interpreters and associated tools, and in 2004 had sixty people in several countries. Its COBOL offering, an interpreter, is not based on LPI-MLF technology and came from the acquired firm. LPI-MLF, to which a C++ compiler was added, is still being offered but is labeled by Liant as a "legacy" product and probably is not selling well.

Liant's larger and older competitor, Microfocus Corporation, also concentrates only on COBOL and associated tools. Both Liant and Microfocus serve a large legacy of installed COBOL-based applications. The market for compilers and interpreters is now fairly small and these two COBOL-dependent companies generate annual revenue of approximately $140M.

LPI-MLF, outstanding technological and marketing achievement though it was, lasted only several years. The assault on its existence and propagation came from several directions and caused its eventual demise without the system morphing into a next generation product.

In other words, LPI-MLF, while itself having achieved success, took the compiler technology as far as it could be taken. To use an analogy, LPI-MLF was similar to those huge steam locomotives of the 1940s that took the steam engine as far as it could go, resulting in the heaviest locomotives ever manufactured, before this technology was replaced by much lighter diesel and electric engines. Or using another analogy, it is like today's heavy 21" cathode ray tube (CRT) computer monitors taken to the extreme that CRT technology allows before being replaced by much lighter LCD and plasma screens. The examples are many. LPI-MLF was the giant steam locomotive at the end of a long line of compiler technology development.

The assault on LPI-MLF was relatively fast and deadly, as if it had been planned, but of course it had not been.

- FORTRAN, Pascal, and PL/1 were quickly being supplanted by C, and later by C++ and eventually by Java. While C++ was swiftly added to LPI's family of languages, the availability from universities and related entities of free or almost free compilers for C and C++ check-mated this move.

- COBOL, RPG-II, and PL/1 were being supplanted by database systems and associated query languages. These systems were organically different from LPI-MLF, and LPI-MLF would bring little benefit to their users. The number of programmers proficient in COBOL, RPG-II, and Pl/1 quickly dwindled. The need for compilers for these three languages became limited to customers interested in legacy applications with the idiosyncrasies and deviations from language standards so characteristic of these applications.

- The remarkably fast increase in the speed of processors and in memory capacities obviated the need for a finely tuned LPI-MLF, which became expensive to buy and maintain, that is, expensive compared to free software. Competitors' shortcomings in run-time and compile-time speed and memory requirements became hardly noticeable on the generation of computers where speed and memory began to be measured in Mega units. LPI-MLF compiler offerings, for the few languages that still mattered, e.g. C and C++ for new applications, and COBOL and FORTRAN for legacy applications, were being quickly supplanted by easy to design, simple interpreters, many of which were also available for free. These interpreters, while woefully inadequate on slow computers, were quite acceptable on the new, lightning fast computers, except for niche applications, such as weather prediction and weapons design, for which even today no computer is fast enough.

- And finally, except for niche and specialized areas, the multitude of competing operating systems and computer architectures relatively quickly solidified into one architecture: a succession of the ever-faster backward compatible Intel chips, and a succession of backward compatible (more or less) versions of Microsoft's Windows operating system. The need for a system that would allow a quick porting of itself to a radically different architecture, i.e. the need for this aspect of LPI-MLF, has disappeared. There were no new architectures to port to. LPI-MLF became a "legacy" offering in the line up of Liant's (LPI's successor) products.

An observation that might help to understand the fate of LPI-MLF, is that a similar process is happening right now, in 2004. Microsoft's Windows is being assaulted by a bevy of free or almost free software products such as Linux and associated tools, sub-systems and office productivity suites.

Windows has finally achieved the reliability and functionality that long ago could only be found elsewhere. It may also be at the end of a certain approach to designing operating systems. The fog of war prevents the participants and observers of this Linux vs. Windows battle from a crisp knowledge of the outcome, the movements of the opposition, and from formulating exactly where each opponent stands. The opponents themselves do not necessarily know where they stand. It is not a foregone conclusion that Windows is indeed the end of a given technology. It is not always the case that a technologically superior product, i.e. the vastly improved Windows, must succeed in the end. Microsoft knows this cruel dictum better than most, and has benefited mightily from it. Changing hardware and changing markets add to the fog while affecting the outcome of the war, and determine, to a significant degree, which approach will win. To make it more complex, the winner may be neither of the two approaches or an unexpected combination of both.

My conclusions as to what caused the demise of LPI-MLF are being written fifteen years after the events discussed. The fog of compiler wars in the 1980's prevented me from having as clear a picture of LPI-MLF's fate as presented here, even though in 1987 I sensed the coming demise of LPI-MLF without necessarily knowing all factors as well as I do now.

Bill Gates, Dave Cutler, and the Linux people have a hell of a problem on their hands. Stay tuned; in 2020 someone will explain everything about Windows and Linux very clearly.

Related Multi-Language Compiler Systems

In this section I will not try to provide an exhaustive review of all attempts, successful and unsuccessful, to design a multi-language compiler system. This is a task for another day. I will review only the efforts closely related to LPI-MLF and its roots.

It is probably a universal law governing human endeavors: proof that a certain achievement is possible makes it far easier for other humans to attain the same achievement. LPI-MLF was such a trail blazer and its breakthroughs, while not copied directly, were to a large degree responsible for others achieving milestones that were not possible just a year or two earlier. For example, LPI-MLF successfully incorporated the irregular and non-formal language COBOL into its multi-language compiler family, thus spurring the efforts of others to succeed in the same endeavor.

Translation Systems, Inc. This company, founded by Bob Freiburghouse, had the greatest potential to have one of the first multi-language families of compilers but never quite came to realize it because its business model did not require such a family at the time. In the late 1970's Translation Systems built several code generators for its Internal Language but the family consisted only of a relatively limited set of two languages, PL/1 and FORTRAN; although some computer manufacturers who were licensees of Translation Systems built their own code generators for the Internal Language, they did

not extend the set of languages. The only licensee who extended the family of languages, Stratus, did so without retargeting the family to other architectures because it was a captive property of Stratus and served only Stratus's evolving needs.

Stratus Multi-Language Compiler Family. After Bob Freiburghouse left Translation Systems, he was in charge of software development at a start-up, Stratus Computer that he co-founded. At Stratus, starting with 1980, Bob managed the design and successful gradual release of several compilers of a multi-language compiler family that between 1980 and 1985 grew to contain FORTRAN, COBOL, PL/1 Subset-G, C, and Pascal. This multi-language compiler family with a common Intermediate Language was not often retargeted to new architectures. Stratus's business model did not require a family of compilers targeted for multiple computer architectures until several years after the Stratus family was released on a single architecture.

LPI-MLF and the Stratus multi-language family share an important root: the genius of Bob's intuition captured in the overall design of the Internal Language of his PL/1 compiler released in late 1970s. This internal language was used by the designers of the Intermediate Languages at both LPI and Stratus as the base for their efforts, and they arrived independently to the final Intermediate Languages used by their respective compiler families. Bob's intuition and experience in designing compilers gave these two teams a base that was a great boost on the road to success of a multi-language family of compilers, a road that until then was covered only by failures.

GEM Multi-Language Compiler Family. Richard Grove, a veteran of Common-Run Time Library battles during the development of VAX, went on to become the architect from 1984 to 2002 of a multi-language family of compilers that DEC began planning and designing in the mid-1980s.

This family of compilers, the GEM family, which over the years eventually grew to include Ada, Basic, C, C++, COBOL, FORTRAN, Pascal, PL/1, and BLISS, was ported by DEC to its Alpha architecture machines. The complexity of the project of porting GEM to Alpha was a pre-cursor of the full blown range of the difficulties that the retargeting of GEM for the Itanium would experience a few years later. GEM for Alpha had to satisfy many requirements, among them incorporation of many small, incompatible extensions to C and C++, a requirement for the compilers to process hooks into Alpha machine resources far beyond the hooks allowed by the standards for C and C++, and a demand for a tight executable code with interface to irregular run-time routines.

Having had enough difficulties, both technological and managerial, with C and C++, the project for porting other GEM languages to Alpha has never quite achieved the stage of these languages becoming established product offerings, and the demand for these languages became a moot point because of the relatively short life of the Alpha architecture.

The GEM family was also retargeted for MIPS and x86 machines but did not become a noticeable offering for these architectures.

An instructive lesson on factors affecting the fate of a complex engineering product such as the GEM family of compilers can be learned now; this process is occurring just as I am writing this paper. The factors affecting the success or failure of the GEM multi-language family as it is being retargeted for the Itanium architecture are complex, dynamic and far from being limited only to technology.

Consider this:

- In 1998 DEC was acquired by Compaq, Compaq was subsequently acquired by Hewlett-Packard, and Hewlett-Packard later transferred the GEM development team to Intel, along with significant rights to GEM technology. Four owners of complex technology in six years, with the resulting uncertainties in development plans and personnel, can be a significant factor potentially destabilizing the technology and affecting this technology's plans for the future.

- Now that the Itanium architecture is a possible target for the GEM family of compilers, the decision making process and the process of retargeting is being affected by the following factors. Over many years, Intel has accumulated compiler technology that by now contains several intermediate languages. There is also a sizable set of front-ends, optimizers, debuggers, code generator development technologies, and various tools that interact with compilers and that help to build compilers. All these came from different sources, either acquired by Intel or developed in-house. The issue of which intermediate language to use or whether to retrofit GEM to another intermediate language is not a trivial one from a technological and management point of view, nor are the issues of which one of the available components to use, including GEM components. Put more simply, there are quite a few technological teams competing for the project of equipping Itanium with *their* product, *their* intermediate language, *their* front-ends, *their* optimizers, their debuggers, *their* tools, *their* ideas, and *their* technology. The GEM development team is only one of these teams, probably the newest and youngest addition to the Intel compiler development contingent, and naturally may be having a hard time advancing its ideas. In addition, passing through four owners in six years may not have strengthened the team.

- Remember the "herding cats" comparison? Well, in the case of Itanium there are several herds of cats and they are in different geographical locations. The "herding," the management and decision making process for the compiler project for Itanium must be very complex. Slow decision making, split decisions that satisfy nobody, and reversals of course as one or another compiler development team gains the upper hand are probably the order of the day.

- The Itanium architecture itself has to go through a proving phase and the conclusions of this phase are not yet clear. This will be another factor affecting the fate of GEM compilers retargeted for Itanium.

The fate of GEM technology for Itanium has become a function of many factors, most of them non-technological; luck will play a great role, and the technological soundness of the GEM framework will only be one factor. The ease of retargetability of the GEM family, a factor that could have been decisive in a simpler situation, is no longer the single or even the most important factor. The other factors listed above and their unpredictable permutations will determine whether the GEM family for Itanium will succeed in becoming a tool preferred by users. The picture described above is missing yet another very important variable: competition from compilers offered by developers besides those at Intel, including compilers that will be almost free of charge.

Challenges Faced by Multi-Language Compiler Systems Today

The complexity of the environment in which a daring project of the 70's, BETA, was trying to succeed was far lower than that faced by the GEM family of compilers today. Competition from other developers, competition by other technologies, and competition by other computer architectures play a much greater role in determining the success of this multi-language complier family. Product transition and survival are far more dynamic and unpredictable than in the planned economy and the planned research and development at the Academy of Sciences in the old Soviet Union.

One of the greatest obstacles to the success of a multi-language compiler family today is the diminished market share occupied by most of the languages in a large multi-language compiler family. These languages, such as FORTRAN, PL/1, Ada, COBOL, Pascal, and Basic, are in demand by narrow markets. Examples include PL/1 and FORTRAN for weapons design and oil exploration, Ada and FORTRAN for defense projects and space research, and COBOL for older, legacy commercial applications.

All of these markets have in common large installed bases of existing applications written in versions of these languages that deviate slightly and sometimes significantly from the standard versions. These deviations must be accepted by the multi-language compiler system for it to succeed. Retrofitting a compiler system for this purpose is relatively simple technologically but an extremely time-consuming effort. This effort must be repeated every time the multi-language family is retargeted to a new architecture because of additional deviations from the language standards that occurred since the last retrofit effort. In addition, the nature of the markets for these languages almost requires that the languages interface with various non-standard sub-systems, such as those used for inducing or detecting parallelism in programs. These new and experimental sub-systems do not stay stable, but change over the course of only a few years. Their constant changes require corresponding, equally constant changes to the appropriate interface extensions in the languages.

If a multi-language compiler system gets caught in a quagmire of constant updates its very potent advantage, short time-to-market, is neutralized. Thus, a multi-language compiler system is far better suited to a family of standard, widely used languages for

users with rigorous adherence to language standards. At this time such a family is limited to C and C++.

As I observed earlier, a full history of all multi-language compiler projects, successful and unsuccessful, is a task for another day. However, the five projects reviewed in this paper, BETA, LPI-MLF, Translation Systems, Stratus, and GEM, show how the complexity of the environment in which the projects were executed has increased with time, and how factors well beyond technological ones affect the success, breadth, and time-to-market of these products.

The Stratus family's retargetability was not tested for many years because Stratus simply did not have multiple architectures to serve; Translation Systems had a narrow family of compilers (FORTRAN and PL/1), and the GEM family's retargetability is largely a very complex management issue.

LPI, because the success of its business depended on this, was the first one to offer a wide multi-language family of compilers that was actually retargeted to quite a few very different computer architectures.

Epilogue

Ershov's legacies in Computer Science, through BETA and LPI-MLF as described here, and other legacies that no doubt will be described by his other pupils and colleagues, do not fully describe his influence on posterity. Another legacy, one that is very important to me, is also one of the most lasting legacies of Andrei Petrovich Ershov. That is the community of researchers and practitioners in Computer Science that is still alive today.

Ershov had an uncanny ability to recognize, attract, and retain good, morally upright, noble people. These people had different levels of talent, they did squabble between themselves, they were all very different but for one common trait – decency. Exceptions, noticeable for their scarcity, could mostly be attributed to the youth of Ershov's pupils and their hopefully temporary folly.

One cannot forget that maintaining a large group of morally upright people, year to year, generation after generation, was a great achievement on its own, an achievement better understood by those who worked in the command economies of the USSR and its satellites, who could compare the "decency quotient" of one scientific family, like Ershov's, to another.

To this observer and member of Ershov's school, his school always won this contest hands down. Moreover, this observation comes from a member of this school whose relationships with some of Ershov's pupils were not the smoothest.

This legacy of Ershov, never quite discussed in scientific papers and the stuffy reviews of his academic work, is the most important achievement of Andrei Petrovich, the achievement that he did not live long enough to truly observe and enjoy.

Acknowledgments

This paper was reviewed by John Ankcorn, Joe Carchidi[9], Dave Cutler, Bob Freiburghouse, Richard Grove, Vadim Kotov[10], and David Kuck[11]. Their encouragement, praise, and occasional disagreements with me were straightforward and frank. I was not in agreement with all of their suggestions, and bear full responsibility for all of the shortcomings found in this paper.

The frankness of the exchanges with these reviewers reminded me again how blessed I was to work with each one of them at some point in the past thirty years, and how blessed I was to cross paths with these ultimate professionals for whom the old-fashioned values of the software engineering profession remain as valid today as they were when we worked together:

- never leaving the project until it is finished or cancelled no matter how high the salary offer by a competitor and no matter how maddening the current management structure is,
- allegiance to one's current employer and honorable behavior in protecting current and past employers' proprietary information,
- and dedication to the software engineering profession requiring one's emotions be set aside in order to achieve the best engineering product.

I am grateful to these reviewers for their comments and I am glad I had the chance to cross paths with them again.

---

[9] Joe Carchidi -- formerly, a manager of the group at *DEC* charged with maintenance and development of the follow-on versions of the VAX/VMS operating system.

[10] Vadim Kotov – Corresponding Member of the Russian Academy of Sciences, a specialist in concurrency theory and its applications to the system design and analysis. Formerly, Director of the Institute of Informatics Systems of the Siberian Branch of the Russian Academy of Sciences. Currently, Director of Engineering, High Dependability Computing Program, School of Computer Science, *Carnegie Mellon University, Moffett Field, CA*

[11] David Kuck -- currently, Intel Fellow, Software and Solutions Group Director, Parallel and Distributed Solutions Division, *Intel Corporation.* Also, an emeritus faculty member of the Computer Science and Electrical and Computer Engineering departments of *the University of Illinois at Urbana-Champaign.* Formerly, served as director of the Center for Supercomputing Research and Development at the same University, later a co-founder of *KAI Software Lab*, a leading provider of performance-oriented compilers and programming tools used in the development of multithreaded applications.

My opinions on the state of affairs of corporations mentioned in this paper and which employ some of my reviewers are my own opinions and not the opinions of my reviewers.

Ron Ham and Jeffrey Schriesheim[12] were generous with their observations and recollections related to various compiler development projects. I am indebted to them.

My son and my editor for the past ten years, Anthony Schwartzman[13], whose own writing talent is exceeded only by his patience in editing my work and his ability to preserve my style while removing my mistakes and oversights, has a range of knowledge that allows him to edit all my writings, from investment philosophy and practice to computer science and engineering. His help is invaluable.

My old colleagues from the Russian Academy of Sciences, Sergey Pokrovsky, Alexander Rar and Alexander Zamulin have translated the paper into Russian and edited the translation. I appreciate the speed and professionalism of their approach to the task.

Ivan Tcherkes[14] has read through the Russian translation and made many valuable suggestions.

I value the calm and organized effort of Natalie Cheremnykh[15] in coordinating the submission, translation, editing, and publishing of this paper in Russian.

Michael Schwartzman is President of ValueSearch Capital Management, LLC, an investment management firm in Swampscott, Massachusetts where he manages his family's and his clients' long-term investments in publicly held companies. He made his first million by founding and running LPI, and it was certainly the hardest one to make.

[12] Jeffrey Schriesheim -- formerly, in charge of porting VAX software to Alpha architecture at *DEC*. Currently, a senior manager in charge of several Advanced Research projects at *EMC Corp., Hopkinton, Massachusetts.*

[13] Anthony Schwartzman -- currently, a playwright and a theatre director; the founder of *Oblivion Productions* theatre company, *Boston, Massachusetts.*

[14] Ivan Tcherkes – currently, a senior consultant at a Swiss technology consulting firm *TG Consultancy Services SA.*

[15] Natalie Cheremnykh – currently, manager of Information Technologies Department, Institute of Informatics Systems, Siberian Division of *the Russian Academy of Sciences, Akademgorodok, Novosibirsk, Russia*