FROM BINARY STRINGS TO VISUAL PROGRAMMING

A Nordic perspective on history of programming and programming languages

Knut Skog

NORUT Information Technologies, Science Park, Tromsø. Knut.Skog@itek.norut.no

Abstract: Starting with the events around the first von Neumann machine in 1945, this paper presents the some of the main steps in the development of programming and high level languages for instructing machinery to perform according need, ideas, or any form of stated requirements. The main emphasis is on Nordic contributions. The paper leads up to present time with the marked influence of the network and its web technology, a technology offering programming light for millions of computer owners and users. Some concern is raised as to the latest trend of regarding documents and programs as being the same notion.

Key words: Computer programming, Nordic history

1. PROGRAMMING AND LANGUAGE

Keeping in touch with the art of computer programming for more than four decades is like standing up against an intellectual hurricane - the most forceful and rapid changing technology ever in the history of humanity. Still there is more to come. The computer is a device instructed by man to perform operations according to his will. Mostly these operations have an immaterial effect but connected to mechanical devices the operations can also cause material effects. Let us not forget that when the device is a missile the effects of the programs are potentially highly devastating.

The instructions to a computer appear in lexical forms of some artificial, formally and carefully constructed language, a language never spoken, only written by a programmer, and read by him and the computer.

Strangely enough, it is my experience that reading other peoples programs is not at all appealing. In some cases, it feels like passing the border of privacy. We write to a computer using a programming language.

The focus of this paper is on programming languages during the last forty years, and an attempt to extract what may be an essence with lasting value. Making an evaluation of the close to 150 different recognised programming languages of this period is impossible. At this occasion, I narrow my focus to the Nordic scenes and its pioneers. The field of programming is so large that I can only ask forgiveness if the reader feels that I have ignored important events.

People have a natural talent for creating languages that reflects his environment. This became apparent when they tried to build a tower in Babel, reaching Heaven. We read in Genesis (11:1-9.) "But God confounded their tongue, so that they did not understand one another's speech". Like the people of Babel, computer programmers do not always understand each other. However, as long as computers do, a language is valuable. Some will say that all these languages have enriched the computer industry whereas other will claim it has been a curse.

The history of computing and computer languages is prevalent. A link to thumbnails of the history of programming languages appears in [CompHist].

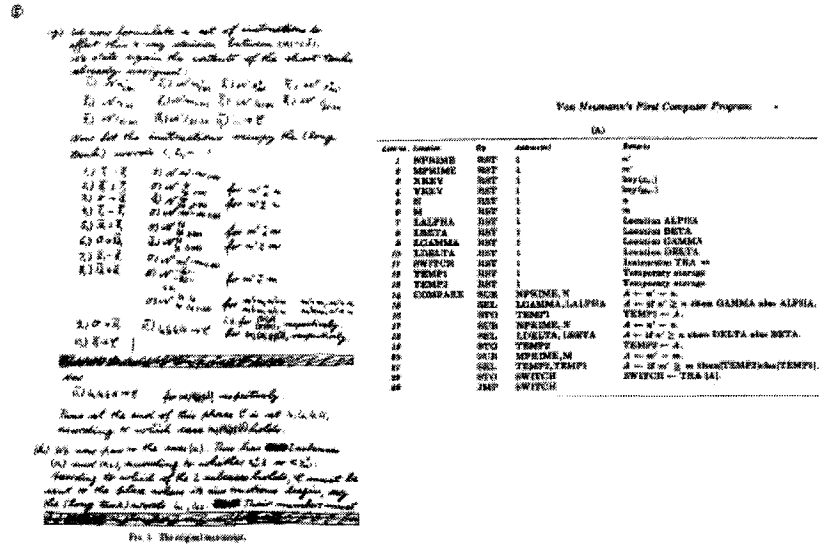
2. IN THE BEGINNING

Instructing mechanical counting machinery to perform their data processing task happened long before World War II. Lady Lovelace's program for calculating Bernoulli numbers on Babbage's Analytical Engine is a well-known example. Ten years before the first electronic computer with modifiable stored program was running, Allan Turing [Turing] presented many of the theoretical aspects of computing in his description of a universal finite state transition machine. However, the focus on programming - as we know it today - starts with von Neumann's work on the EDVAC (Electronic Discrete Variable Automatic Calculator) in 1945. We stress the word *variable* since it denotes the flexibility of this electronic device. It is also a reflection of the modern notion of a computer that in von Neumann's mind, *in itself should be able to modify its program.* His idea of heuristic aspects of computers appears in many of his later papers. (See his lectures on "Theory of self-reproducing automata" in [Burks], Chapter 11.)

[Burks] has well-documented von Neumann's contributions to computer programming. This reference also contains a remarkable paper, "von Neumann's first computer program", written by Donald E. Knuth

[Burks, chapter 2]. Von Neumann wrote this sorting and merging programs in 1945 as part of design of the EDVAC computer, a machine that at that stage had an auxiliary memory of 8192 words of 32 bits. Note however that it was not possible to access randomly the memory of EDVAC. The kind of programming challenges von Neumann was facing we know today as micro coding. Despite many low-level type of operations, his work for the first time introduced the basic programming notion of a subroutine using relative addresses and symbolic coding with mnemonic names of variables and instructions. He was a frequent user of flow diagrams in planning the structure of his programs.

Deeply rooted in the world of mathematics his systems analysis was in mathematical notation and the resulting program in a form that we call *assembly coding* as can be seen in the extract below from Donald E. Knuth's paper.



It is fair to say that programming at this stage was regarded as a type of work requiring deep insight in the architecture and behaviour of the computer and its binary nature. People considered it a craft for a person trained in logic and mathematics. Von Neumann himself was, however, one of the first that suggested key board shortcuts for denoting patterns in the binary string of a instruction instead of writing sequences of 1 and 0. He started the era of programming in the form of *assembly coding*.

3. THE FIRST PROGRAMMING LANGUAGES

In the middle of the fifties, IBM had made its Naval Ordnance Research Calculator (NORC), the most powerful computer in existence at the time. It was a vacuum tube machine with 3600 words of main memory and binary coded decimal arithmetic operations. This machine was part of the environment in which John Backus and his team of programmers invented FORTRAN (FORmula TRANslator), the first programming language in which the human or problem domain was the prime focus and where the computer itself translated the scripts of the language into binary instructions and numbers. This translation required a program of a new kind, compilers as they where called. The FORTRAN compiler was able to produce code almost as efficient as assembly code and they reduced programming time considerably. People say that when John Backus presented the idea of a compiled programming language to von Neumann, his reaction was not at all enthusiastic. In his mind, a computer should do calculations and not such clerical tasks as compiling.

The FORTRAN was targeted for scientific and numeric calculations and loved by scientists who could write formulas the way they where used to. It offered two name scopes (local and global) for variables, familiar mathematical ways of writing expressions and intuitively natural statements for loops and conditional branching. Later separately compiled subroutines were added. The extensive growth in the programmer population in the sixties and the seventies is mainly due to FORTRAN. As member of the committee that created ALGOL a few years later, John Backus introduced the syntax notation that, slightly modified by Naur, to day is known as Backus Naur Form (BNF).

It is symptomatic that John Backus and his work did not come out of a university research environment. His own background was highly irregular. IBM management of his time deserves credit for supporting his diverse and eclectic bunch of people that was outsiders to the academic establishments and hackers of its time. John Backus and his team of programmers new from hard work what was needed and created the first and for a long time the most used higher level programming language.

The need for programming tools closer to the problem domain was generally acknowledge in the late fifties and in 1958, the first version of the Algorithmic Language ALGOL was borne. Two years later the language was the target of international cooperation and an agreement on ALGOL60 was a fact much due to the work at Dansk Regnecentral and with Peter Naur as the supreme editor putting all pieces together. ALGOL was the first programming language of a scientific flavour. It was an orthogonal language, meaning it has a relatively small number of basic constructs and a

set of rules for combining those constructs. It introduced nested (block-structured) scopes of names, strongly typed variables, procedure parameters by name or reference, and recursive procedures and dynamic arrays. In close cooperation with the Dutch ALGOL group (Dijkstra, van der Pool, van Wijngaarden), Naur, Jørn Jensen and the Danish group pioneered the stack for dynamic memory management and its addressing schema for recursive execution.

Although ALGOL never reached the level of commercial popularity of FORTRAN, it is considered the most important language in terms of its influence on later language development. ALGOL's lexical and syntactic structures became so popular that virtually all languages designed since have been referred to as "ALGOL–like". On the Nordic scene, Peter Naur and Jørn Jensen (in the compiler team of 1962, they where called the Doctor and the Master) made substantial contribution to our conception of high level programming languages and the associated compiling techniques.

Automated data processing was an established industry of its own at the time the first variable (stored program) automatic calculator was born, made for numeric calculations, not for data processing. However, when they connected the calculator to punched card equipment and magnetic tapes, the "calculator" was ready for *data processing*. For this task, the quest for non-mathematical programming tools came at the end of the fifties.

From the home page of COBOL we quote:

> In 1959, an industry-wide team was assembled to formulate a common business programming language. The Conference on Data System Languages (CODASYL) developed a new language, and created the first standardized business computer programming language. COBOL (Common Business Oriented Language) was developed under the auspices of the U.S. Department of Defence in cooperation with computer manufactures, users, and universities. The initial specifications for COBOL were presented in a report of the executive committee of CODASYL committee in April of 1960. It was designed to be a business problem oriented, machine independent and capable of continuous change and development.

COBOL with its highly verbose lexical style was the first effort in creating exact computer programs with statements that intuitively could be understood by humans. The idea of instructing the computer the same way you instruct people is ill founded. However, as with native languages, people who once have learned them tend to love them and COBOL is still a living programming language in many commercial applications with roots back to the late sixties and the seventies.

4. OBJECT ORIENTATION

With SIMULA 67, a new dimension – the dimension of object orientation (OO) – was added to the art of programming. SIMULA was originally designed as a language for discrete event simulation, but was soon regarded as a general purpose programming language. ALGOL like and like ALOGOL SIMULA never became widely used, however the language has been highly influential on programming methodology and modern systems engineering. It implemented the mechanics of OO and paved the way for a new systems design philosophy. It invented the notion and mechanics of the class concept with inheritance and polymorphism, object instantiations and dynamic heap allocation. In this it broke the traditional sequential way of reasoning on how the computer programs should be organized and demonstrated how to "think and write" in terms of parallel event-driven tasks. It formed what to day is the foundation for modelling of computer applications at large.

[Holmvik] thoroughly documents the history of the work of Kristen Nygaard and Ole Johan Dahl and their colleagues at Norsk Regnesentral on SIMULA 67. Being myself affiliated with the activity in the period from the fall of 1965 to 1968, I feel inclined to present some of my recollections. One is that Bjørn Myrhaug, a junior member of the SIMULA team, deserves recognition for his pioneering implementation work on heap allocation and "garbage" collection.

In the fall of 1965, I was working at the Computer Laboratory at the Technical University of Norway in Trondheim. At that time, like Norsk Regnesentral, their mainframe compute was a UNIVAC machine. That machine replaced the GIER ALGOL computer that left a large community of scientists and students, trained as ALGOL programmers, in the FORTRAN desert. Based upon a strong request for an ALGOL service, my own scientific interests and experience in compiler writing from Dansk Regnesentral and Control Data Corp., an implementation of ALGOL on the UNIVAC machine started in early 1966. Since SIMULA was ALGOL++, it was agreed that our compiler should be expanded in due time to include the SIMULA capabilities.

As it turned out this effort was like shooting at a moving target. The SIMULA specifications were constantly changing and not ready when we had to make important decisions on the compiler side. However, the compiler was organized in a very open table driven manner in order to cope with the not yet seen capabilities, hoping that whatever the requirements where, they could be handled by adding new entries to the tables. In Trondheim, we completed the regular ALGOL implementation and shipped the compiler source code to the SIMULA team for additions. That this

created some pain is rather obvious. The date for the completion of the SIMULA 67 implementation on the UNIVAC computer is not quite clear but my guess is late 1970 early 1971.

Alan Kay first recognized the real substance and importance of the new model and associated programming concepts that SIMULA brought in the outside world (US). The history of science is full of cases where the implication of scientific research is not fully recognized in its time. I dear say that even Kristen Nygaard, who was a man not known for muting his opinions, did not realize the importance and impact of their work at the time it was done.

Alan Kay took the SIMULA concepts into his Smalltalk language at XEROX Park and this again had a substantial influence on the development in programming and programming languages in the states. James Gosling at SUN picked up many of Smalltalk's capabilities when he developed the NeWS (network-extensible window system) in the late 1980s. This work was the foundation for the JAVA language in the early 1990s.

Bjarne Stroustrup at BELL Labs, an exiled Dane, created C++ by adding the object oriented language elements of SIMULA to the C language of the UNIX world, very much a creation of Kernighan and Ritchie. C was born early in the 1970s and was probably the most used systems programming language in the early 1980s. By extending it, Bjarne Stroustrup brought object-orientation to systems programming. C++ is by far the most used low-level systems programming language of to day. Being an unsafe language with many low level characteristics like loose typing and dynamic memory allocation in the hand of the programmer with the risk of creating memory leaks, the cost of risky programming is no longer justified in terms of faster code. One may wonder why maturity sometimes comes slowly as we see the newly born C#, an offspring of Microsoft, finally offer automatic garbage collection and strong typing to its users.

Bjarne Stroustrup's achievement was to give systems programmer an object-oriented tool within their "native language" framework C. It was an effort right in time and on the right spot. It is difficult to envision modern commercial software without C++.

There is a straight line from SIMULA to the unified modelling language (UML) of today. However, probably the most fundamental effect of the language is realizing that it led us to the most fundamental notion of computer science *Abstract Data Types* (ADT), the basic building block of computer programming. The SIMULA class is the concrete implementation of ADT and the truest reflection of the formal definition of the word data. The Object Management Group is now the most vivid forum of the SIMULA heritage.

A breakthrough of a philosophical nature – how to reason and how to support this reasoning with programming constructs – is not or will never be

recognized by commercial measures. The conceptual and philosophical contribution to computer science made by Nygaard and Dahl in the middle of the 1960s is in my opinion equally important as von Neumann's in the middle of the 1940s.

5. MODERN TIMES

A new area in computing started when Transmission Control Protocol/Internet Protocol (TCP/IP) became the de-facto standard for networked computing. From local area computing to world wide web (www) computing is a mayor brake from closely related development steps starting with von Neumann "variable calculator" ending with timeshared mainframe computers in the late 1980s. With hundreds of millions of computers on the net, reaching into schools, offices, homes, cars, almost anywhere within reach of even modest communication services, computers are effecting the population of the world in ways no one could imagine ten to fifteen years ago. Before approaching the network impacts on programming, let us split the notion of programming in non-professional programming and commercial professional programming.

The use of modern devices like mobile telephones, television sets, kitchen stoves, etc. require people without training to perform simple programming in operating these computerized gadgets. The combination of complexity and flexibility of things spread in quantities of hundreds of millions throughout the world is unparalleled in our history. Closest comes the car costing 100 times more with a much simpler task (move from here to there) to perform. In its simplest form, millions of people unconsciously do non-professional programming simply by operating menus and setting options.

The trend of reaching out to the novice programmer with lightweight programming tools like Basic, Visual Basic, and XBasic continues and is a consequence of a highly flexible device being a mass-market commodity. There are simply not enough highly trained people for adapting computer technology to applications. Providing tools for adaptation in so-called user-friendly programming manners is not to be underestimated. For certain applications, tailored light programming environments has proven to be extremely successful.

Turning our attention to professional programming, the role of the programming language as such is becoming more and more integrated with its environment. It is simply not enough to type and edit statements in your favourite plain text editor and turn it to its compiler. The reason is that we can hardly make any application without extensive use of components.

Managing these component libraries in the development process is necessary for a reliable and cost effective development. Component based application frequently involves very large quantities of code and requires tools for maneuvering, displaying, and versioning as important mechanics of the programming environment.

6. PROGRAMMING OF OUR TIME

To navigate within all of the source code of a modern application the code itself must be kept in html like document format in which links to components and modules is the prime navigation tool. The transition from plain program text to an html-formatted form is far more than "pretty printing". It is an indispensable function of program manipulation and reading. Of the total effort spent in applications programming, 80% goes into maintenance, frequently handled by other persons than those who initially wrote the programs. The readability of a program frequently depends on comments in the code itself. The problem of keeping commentaries consistent with the code is well known. The best remedy here is to stay away from low level programming styles letting the code speak for itself.

It is my opinion that the programming language issue of our time gives us only to alternatives for commercial developers: Java and C#. To my knowledge, these are the only fully commercially supported languages and environments for professional development. C# is the latest on the arena, though not the forerunner. Major developments on the environment side of these systems are likely to come. This change all relates to managing component growth and integration by sharing concentrated and distributed resources through the network.

6.1 Document program and data

Keeping track of the latest trends in network application development is an interesting challenge. There is a tendency to regard document and program as equivalent notions. The simple reasoning seems to be that a program is data, document is data, and therefore, a program is a document. This seems very harmless, but it has some serious consequences.

First, let us recall that data is a "representation of ideas or facts in a manner suitable for some process" (IFIP definition). For a document, the process for which the representation shall be suitable is ultimately a human reader, or listener for that sake, where as for the program the suitable representation is ultimately for the computer to execute. However, the

matter becomes complicated by the fact that humans also read a program and for that purpose, it should be well documented. Here the computer is an assistant.

The same is true for a modern digital document that a computer reads, or rather processes, to make it neatly readable and easy maneuverable. The only problem with our equation is that it makes a computer and a human identical in terms of representational requirements for their processing needs. This is in my opinion fundamentally wrong. So where does this philosophical reasoning bring us. What impact does it make and where?

Librarians are professional document workers. In the early 1970s, they introduced the Standard Generalized Markup Language (SGML), the first markup language, now an ISO 8879 standard. They developed it for adding computer detectable content structures to a document.

The development of web servers and network readers/browsers adapted the markup method to apply to documents that are typeset and linked. The Markup Language (ML) of what we call e-documents is HTML where HT stands for Hyper Text. Together with the *Multipurpose Internet Mail Extensions* (MIME) standard, HTML and TCP/IP (with domain name services) are the cornerstones of the Web and its grate success. Hypertext is a multimedia network integration facility that offers interactive (active server pages) capabilities in which the reader may deliver (write) data for processing. This e-document is far more potent than ordinary paper documents. No wonder a blur exists between program and document distinctions.

To follow up this success the www Consortium (W3C) has generalised html into eXtendable Markup Language (XML). In this language the low-level extensions occurs in a document type definition (DTD) notation and at higher level by XMLSchemas. The XMLSchemas offer a data structuring capability in which primitive built-in data types such as date, time, string, decimal, integer, and float become building blocks for composed data structures. There is no means for defining operations. The basic idea is that a XML document is data structured by markups. We do not state the processing aspect of this data, however; the marks receive mnemonic names to indicate intuitively the meaning for a human reader. By lookup in name-servers at URLs, we can inspect and interpret the XMLSchema for tags.

6.2 Reflections

XML is the promised new way of exchanging data among web processes using the hypertext transport protocol (http). It is a character-encoded vehicle for interoperability and is a working well for small volumes of data. However, we should not take its scalability for granted.

There are three reasons for my deeply founded skepticisms. First, methods for the computer process itself are left out. By that, we are defining a document primarily for human interpretation. This human is faced with the task of using his tag interpretation in order to make the exchange understood. Second, with increasing volume and complexity, the XML exchange will hardly be looked at with a bare eye. The reader will most likely use a tag interpretational service anyway. Third, the volume of the verbose XML format, even when compressed, is substantially larger than for anything seen so far. Even with constantly increasing transmission speeds, there will always be bottlenecks, in particular in mobile communication.

Hence, the rapidly growing and extremely potent web community does not seem to recognise the basic difference between document and program. In particular, they do not adhere to the object-oriented philosophy. Consequently, the golden rule of striving for orthogonal design and conceptual economy seems to be dropped. In addition, the tendency of modelling by defining exchange formats seems to be the new trend. A most striking example is the vocabulary for the exchange of geo spatial data define in the XML dialect called Geography Markup Language (GML3.0). The number of tags introduced for this application has passed 1000. The appendix of this paper gives an example of part of a GML exchange packet.

7. VISUAL PROGRAMMING

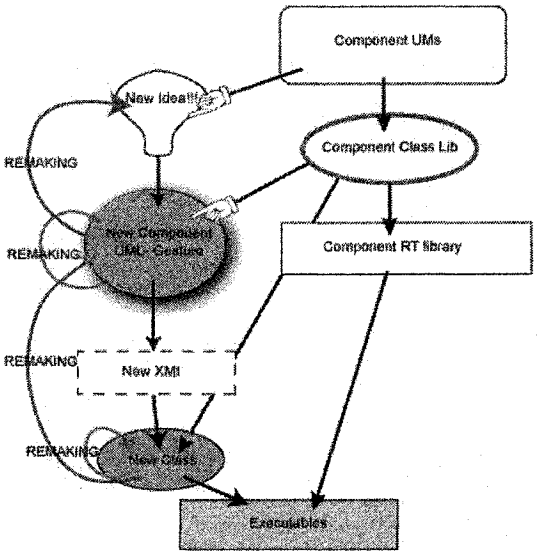
At the end of this jumpy tour through the first fifty years of computer programming, I would like to return to the "programming light" issue. The volume of new code created by non-professional programmers today by far out-weights that of the professionals. In my notion of programming, I include the craft of "html blacksmiths" – also called web-masters and web-typographers. Tools available for web publishing in multimedia forms, with active animated and interactive page areas, offer a kind of visual select and paste type of programming by which they build many networked applications. They may even give the reader (or should we say the client) means of data input (menu selection, radio button, text-fields, etc.) which when sent to application servers, interacts with databases and depositories of various kinds, and returns requested information to the client.

Obviously, there will never be enough professionals to create all the home pages on the net. It is a fact that people trained as web designers, with no programming education, are capable of producing impressive active documents, in which some kind of interpretative programming (scripting) activated through the common gateway interface (CGI) is giving great flexibility on the client side. Hence, for the new generation of light

programmers, the time when programming was writing compiled code is gone. The masses of light programmers claim their rights and they will most likely grow in capabilities with the advancements of supporting development environments. These environments are as themselves visual.

8. FROM UML TO EXECUTABLES

To deal with the server side challenges with interfaces to databases which themselves could be part of a networked service, the developer are still required to master a complete general purpose heavy duty programming language as the integration method for components used. More than anything else, the network technology has created a major change in our conception and understanding of the notion of both programs and documents and even the computer itself. "The network is the computer" was a saying at SUN late in the 1980s. Today it is reality. We may call it by different names, but system design and implementation of today is network oriented. Adding this dimension to the object-oriented approach making what we could call networked object orientation and tying it to our overall modelling work, is the requirements of heavy-duty programming of our time.



The previous diagram is just to stress that programming as the task of creating executable code for applications is confined to input from a modelling activity and from component libraries. To implement any new

idea of real substance, the developer must make his selection of supporting component libraries that at the final stage merge with his own programs. From gestures or diagrams for which the current gospel is named Unified Modelling Language (UML), formalized specifications are transferred to the programming stage through the XML Metadata Interchange (XMI) format. However, the process remains to be a chain of iterations and remaking. Maintaining these iterations by the environments of system development is another great challenge of our trade.

REFERENCES

[CompHist] http://www.levenez.com/lang/

[Turing] "On computable numbers, with an application to the Entscheidungsproblem", Allan Turing, London Mathematical Society, ser. 2. vol. 42 (1936-7), pp.230-265.

[Burks]: W. Aspray and A. Burks (ed): *Papers of John von Neumann on computing and computer theory*, Charles Babbage Institute, reprint series for the history of computing, volume 12. The MIT Press, Cambridge Massachusetts, London.

[Holmvik]: http://java.sun.com/people/jag/SimulaHistory.html

APPENDIX

The following is an example of geographic data represented in the Geography ML format, an application of XML with application specific data structures define as XMLSchemas at specific name server URLs. The genuine data values represented in this packet is given in bold for ease of readability. Colon prefixed names refer to namespace assigned to the prefix (gml) in the head of this package extending the XML namespace (xmlns:gml="http://www.opengis.net/gml").

```
<gml:featureMember>
  <SchoolDistrict>
    <gml:name>District 28</gml:name>
      <gml:boundedBy>
        <gml:Box srsName="http://www.opengis.net/gml/srs/epsg.xml#4326">
          <gml:coord><gml:X>0</gml:X><gml:Y>0</gml:Y></gml:coord>
          <gml:coord><gml:X>50</gml:X><gml:Y>40</gml:Y></gml:coord>
        </gml:Box>
      </gml:boundedBy>

      <schoolMember>
        <School>
          <gml:name>Alpha</gml:name>
          <address>100 Cypress Ave.</address>
          <gml:location>
            <gml:Point srsName="http://www.opengis.net/gml/srs/epsg.xml#4326">
              <gml:coord><gml:X>20.0</gml:X><gml:Y>5.0</gml:Y></gml:coord>
            </gml:Point>
          </gml:location>
        </School>
      </schoolMember>

      <gml:extentOf>
        <gml:Polygon srsName="http://www.opengis.net/gml/srs/epsg.xml#4326">
          <gml:outerBoundaryIs>
            <gml:LinearRing>
              <gml:coord><gml:X>0</gml:X><gml:Y>0</gml:Y></gml:coord>
              <gml:coord><gml:X>50</gml:X><gml:Y>0</gml:Y></gml:coord>
              <gml:coord><gml:X>50</gml:X><gml:Y>40</gml:Y></gml:coord>
              <gml:coord><gml:X>0</gml:X><gml:Y>0</gml:Y></gml:coord>
            </gml:LinearRing>
          </gml:outerBoundaryIs>
        </gml:Polygon>
      </gml:extentOf>
  </SchoolDistrict>
</gml:featureMember>
```