Poseidon House
Castle Park
Cambridge CB3 0RD
United Kingdom

TELEPHONE: **Cambridge (0223) 323010**
INTERNATIONAL: **+44 223 323010**
FAX: **+44 223 359779**
E-MAIL: **apm@ansa.co.uk**

ANSA Phase III

ANSA and tools

Rob van der Linden

Abstract

This is a report on the results of a short survey of 20 programming, analysis, design and configuration tools. The survey was carried out to see whether ANSA transparencies can be provided by extending such tools. As a side effect, we discovered where ANSA principles are appropriate in the provision of software tools.

The results show that the way in which tool providers now hide system complexity from the system designer and programmer could potentially be used in the provision of automated transparencies.It also shows that ANSA engineering can improve tooling environments, and that ideas on federation and scaling are new to the tool community.

APM.1058.00.01 **Draft** 17 September 1993

Request for Comments (confidential to ANSA consortium for 2 years)

Distribution:
Supersedes:
Superseded by:

Contents

1 Introduction

The aim of the Automated Transparencies group in ANSA Phase III is to find ways in which distributed systems technology[1] can be provided to designers and programmers in a selectively transparent manner.

Technology is to be provided by encouraging abstraction and automation [TR.042]. The “Overview of ANSA” [APM.1000.01] describes the tool orientation as follows:

> The ANSA Architecture applies to a class of computing systems in which different parts of a distributed application work together in a unified and natural way. The goal of ANSA is to allow application interworking and portability to be achieved with a relatively small effort; application designers do not need to concern themselves with the diversity or physical distribution of different computers, operating systems and network protocols which comprise the technological basis of the underlying computing environment.
>
> The Architecture is tool oriented: application programmers use a programming language to state the application properties they desire. The tools automatically insert functionality required to achieve the desired application properties (such as atomicity, fault tolerance, quality of service).The tools so hide the technical details of the underlying mechanisms.
>
> By removing the burden of technological detail from application designers, they can concentrate on meeting the requirements of end users and on issues concerning the provision of applications that span organisational, political, and geographical boundaries.

The idea of automating parts of the software development process are not exclusive to ANSA: many design, analysis, programming, and configuration tools are based on the same premise.

Moreover, designers who use tools as an accepted way to increase their productivity, will not give up their tools or switch to less sophisticated tools, just so they can use ANSA technologies.

It is therefore incumbent on us to investigate the existing tools market, to compare the ANSA architectural principles [APM.1000.01] with the approach taken by tool suppliers, and to see how ANSA technologies can be provided in ways which are consistent with the expectations of systems designers and programmers.

1.1 Audience

This paper is aimed at those involved in setting the direction of the work on Automated Transparencies in the ANSA Phase III work plan. It is also useful to those working on programming abstractions in ANSA Phase III.

1. In Phase III, specifically technology which addresses performance and dependability.

1.2 Approach and initial results

We looked at a representative sample of some 20 design, analysis, programming, test, and configuration tools and compared the ANSA Architectural Principles [APM.1000.01] with the approach taken by tool suppliers[1].

Initial results demonstrate that:

1. there are areas of agreement between the ANSA community and the tool builder community, especially on late binding and early type checking;
2. the ANSA community can contribute to the development of tools for building distributed systems, in particular in the areas of federation and scalable[2] interworking technology;
3. the tool builder community can help us understand how to make distributed systems technology available to the system designer and application programmer.

The other finding is that tool suppliers do not advertise the strategic issues of tool selection to their customers. The resulting tool proliferation directly leads to a plethora of solutions all tackling the complexities introduced by multiple GUIs, database management systems, communications facilities and heterogeneous distribution in general. If issues of federation are continued to be ignored in this situation, then end users will once again bear the cost of the mistakes of their suppliers.

1.3 This document

We shall first examine the important features of some of the more advanced productivity increasing software development and construction tools. In doing this it will become clear where the tool builders and suppliers agree with the ANSA principles, where they disagree, and where they have not thought the consequences of their actions through with respect to distribution.

Secondly. we will examine some of the strategic issues of tool selection and show that ANSA principles can be used to explain the challenge and suggest a strategy to meet it.

The appendix lists the set of tools surveyed, points to ongoing work in the area of tool interoperability, and lists a set of object oriented or object based methodologies for reference.

Note: The results of the survey itself, in terms of short product descriptions may be added.

1. A list of the tools we looked at is provided in the appendix.

2. “Scalable” in this context means that the technology is valid independently of the size (number of nodes and networks) of the system.

2 Important features of advanced application development and building tools

2.1 Construct rather than write

The emphasis in software development is slowly shifting from *writing* applications to *constructing* them. The important advance is that re-use of components is now a practical proposition.

It is generally known that no more than 20% of an application consists of code which is directly related to solving a particular problem or providing a particular functionality desired by an end-user. The other 80% or more is concerned with complexities which are independent of the particular application in hand. Examples are code to ensure inter-process communication, to drive a graphical user interface (GUI), code to access data in a database or file system, code to deal with different levels of functionality offered by operating systems, code to interface devices, such as printers, hand-held scanners, etc.

2.2 Late binding

The idea of not binding application specific code and “system” code until configuration (or system building) time clearly encourages re-use and increases productivity.

Advanced application development systems allow application development staff to specify the application specific aspects in detail, whilst tools automatically deal with the systems related aspects. For the GUI, for instance, a visual prototyping environment is provided, within which the right interface can be constructed from pre-defined components, by pointing, clicking, dragging and dropping. Binding to a particular GUI environment is only done just before system deployment. The same approach is adopted for database management systems. A range of database management system interfaces are provided and will be linked in once it has been decided which database management system will be used in a particular configuration.

2.3 Hide complexity

The currently emerging generation of tools hide complexities which are introduced as a result of heterogeneity.

The “system” code can remain invisible to the application development staff by hiding it in libraries. In environments such as that provided by SNAP, application developers simply state the properties they want their applications to possess and the tool will link in just those libraries and routines which are required to provide these properties.

Examples are Art*Enterprise, Magic CAPtm, DEC/Forte, Distributed Smalltalk, Genera, ObjectTeam, and SNAP. Complexities introduced by distribution of the application are not treated with the same urgency in all of these. According to a recent announcement by Template Software, SNAP is now integrated with OSF/DCE. In most other tools, 80's technology is employed. ANSA engineering principles are sound and can with benefit be introduced. Thus, distribution facilities can be offered to application developers through libraries.

2.4 Reduce (eliminate) 3rd generation programming effort

It is accepted that productivity gains can be achieved by programming in a 4th generation programming language rather than a 3GL. Many of the tools which we examined offer the opportunity to program in a rule based environment supported by an expert system (Art*Enterprise, Magic CAPtm, OOA/RD, Ptech, SNAP). This offers three important advantages:

- programming proceeds in close cooperation with the problem owner, who sees his problem translated in recognisable and comprehensible rules in the system;
- rapid prototyping, interspersed with many consultations with the end-users, is a reality, resulting in more satisfactory solutions;
- tools are used to turn rule based specifications into imperative programs, which can then be compiled to achieve specific performance characteristics, delivering results more quickly.

2.4.1 The ANSA Computational and Information models

The abstraction in the interaction model in the ANSA computational model is the procedure call. This is essentially a third generation programming concept, which does not meet the need of rule and constraint based programming.

This does not mean that the computational model is obsolete. It does mean that ANSA has to develop its information model such that distribution, and in particular modularity, is taken care of. Many of the tools we examined have severe limitations on the size of the system which can be designed “in one go” as it were.

The expert system can be used to support 3rd generation programming. It ensures early type checking so avoiding errors at a later stage of the design process. A visual style of application programming is adopted where the application programmer is guided through the “sensible” choices which can be made at each point, with the expert system suggesting defaults where appropriate (e.g. VisualBasic, VisualC). Structure editors are a further aid.

All this reduces the amount of information which designers need to carry around in their heads, thus freeing their creative powers, allowing them to concentrate on the application area itself.

2.4.2 Optimising rule based programs

The rule based specifications can be turned into imperative programs. This translation enables performance increases over the interpreted rule based system.

2.5 Support for methodologies

The expert system can implement a methodology. On the downside (at least for some people), the expert system is used in some tools to impose constraints on the development process and so force designers to stay within a particular methodology.

2.6 Browsing and iterative design

Some tools include browsers for symbols, macros, classes, variables, types, etc.(e.g. SNiFF+). These effectively make the structure of an application or system explicit. This information is no longer locked up in programmer's heads, freeing them from this burden, and making maintenance by others easier. Browsing technology is anenabler of iterative design and implementation with its recognised strengths.

In a distributed system, this information will itself be distributed. Current tools have no support for the distribution of this information. However, OMG object and interface repositories would offer such support. Browsing these could provide the same support as current browsing tools do in the local system. Such distributed browsing is akin to the functionality provided in the DEC/HyperBase proposal for a dynamic ORB in OMG.

The idea of extending the trading function into the design phase is not new to ANSA, but must be revived, since recent work on trading in the standards bodies seems to have de-emphasised this angle.

A strong link with the federation work is anticipated, as there will in general be little agreement on the semantics of the content of the object and interface repositories.

3 Strategic issues in tool selection

3.1 Summary

There exists a real potential for increased diversification of distributed systems technology, as tool builders begin to supply distributed application writing tools. In addition, tool incompatibility will increase the chaos which is already much in evidence in this market. Distribution technology will move out of the vendors control and into the control of the tool suppliers. End users will again remain shut out.

Recent history shows that it takes only a few years for tools to appear. Three years ago, GUI programming meant interfacing C code to X-libraries, much like distributed programming today means interfacing C or C++ code to distributed infrastructure APIs (e.g. OSF/DCE, DEC ACAS). If recent history is anything to go by then we will see many distributed systems writing tools become available within the next three years. The first to market we know about are Template Software, who have integrated SNAP with OSF/DCE, and HP, who have extended Parc Place Smalltalk with a class library for access to OSF/DCE.

Note: and what about ICL DAIS ?

Further evidence of this trend are many tool suppliers racing to market with tools which are claimed to be CORBA compatible (Orbix, ACAS, DOME). Some have interworking technology in place, some have not. There is little agreement on what technology is appropriate or what the consequences of particular choices will be on performance and dependability for instance. With tool builders trying to distinguish their products in the market place, there is little assurance that a set of common interworking technologies will even begin to emerge.

The principles we adopted in ANSA can be used to explain the challenge we face.

3.2 The tools market

There is an increasingly large set of analysis, design, progarmming, testing, and configuration tools. It is difficult to uniquely characterise the tools. Metrics which can be used for a comparison include:

- *functionality*, e.g. some tools cater only for particular parts of a design. There are too many GUI design and build tools to mention. Specific tools for database oriented applications include the many SQL based tools, and application generation tools. Some tools are targeted only at programming (SNiFF+), others at several activities in the design process. Some include support for (internal) database design (System Architect), design of transactions (DEC/Forte, ONTOS), configuration management (DEC/ Forte, SNAP), etc. Tools exist now to help in most areas where programming is either hard or tedious.

- *intelligence*, e.g. some tools include an expert system which can be used to state and interpret business rules and constraints (Art*Enterprise, Magic CAPtm, Ptech, SNAP, Tenet?). The expert system is the also used to support the design process or a analyais and design methodology.
- *methodology*, e.g. some tools enforce the use of a particular methodology (e.g. OOA/RD), some impose some methodological constraints (e.g. Ptech, ObjecTeam), or no constraints at all (e.g. SNAP). To suit the size of the tool builder community there are now in excess of 20 object oriented or object based design methodologies [SIGOAD 93], for which tool support is either present or announced[1].
- *object orientation*: almost every tool supplier claims support for object orientation. In some cases it involves a reworking of structured programming, sometimes it means C++ or Objective C. Adaptations to relational databases are said to make this technology object oriented. Object based or oriented languages only become usable once relatively comprehensive libraries are added: Smalltalk is now emerging with decently populated libraries for instance.
- *implementation*, e.g. some tools are based around a central database in which “the design” is kept (Ptech, Magic CAPtm). Sometimes this database is a closed system from the conceptual schema and/or the internal schema point of view (Tenet). The design can equally be kept in a set of files (SNAP). The “internal model” is often said to embody the essential aspect of a tool, which distinguishes it from others.
- *vertical market approach*, e.g. some tools are specially appropriate to construct applications in a particular application area, such as hardware design (e.g. Statemate), CAD/CAM, or training, marketing and communications (e.g. ONTOS).

Most tool suppliers are keen to point out the advantages of their products in only a few of the categories above. Making an informed choice as a user who is aware of all of the above categories is a much more difficult task.

3.3 Problems facing the tool user

From the tool user perspective the tool diversification offers increased choice but of course has a downside: where multiple tools are in use the usual and inevitable questions arise. This time it is about multiple tool sets in a distributed environment. Three classes of problems can arise. The first two are related to the realisation that in large systems, the design process is itself a distributed application, possibly involving many tools. The last issue is related to interworking problems between the systems which are created using different tools.

3.3.1 The tool as a distributed application

For the problems associated with the way a tool is implemented in a distributed environment we can use the Architecture to suggest design choices: it is as if the tool is an application which is to be distributed. This can ensure that the tool is portable, can be distributed, meets certain reliability requirements, scales, can be configured etc.

1. See appendix for a list of OO methodologies.

ANSA need not address this issue as special since it is covered by our concern about system integration and federation in general. However, it is clear that all of the tools we looked at have limitations in terms of the size of the design they can manage and the extent to which the tool can be used in a large distributed design process.

3.3.2 Exchanging designs between tools[1]

Issues relating to exchanging designs between tools are more akin to multi-database problems, where schema integration / conversion is appropriate. The design itself is a complex of information held by the design tool. The information is organised according the “data model” which the tool builder considers the heart of the tool (and that which he believes gives it the competitive edge). The reluctance to be open about this leads to issues which are are perhaps more difficult to sort out. Specific questions include:

- can a design developed in one tool be further developed in another?
- what if my company who use tool set A merge with a company who use tool set B?
- what if I want to rationalise the use of tools in my company?

In ANSA we advocated to address this problem by choosing CORBA IDL as the common interface definition language at the scenario implementation level. Where multiple IDL’s are in use we suggest the use of Abstract Data Types (ADT) [TR.042]. By adopting a three stage process, consisting of a front end (FE), an abstract syntax tree (AST) and a back end (BE) we hope that mappings between other tools which adopt the same approach will be simplified.

There is little evidence however, that the tools market is either adopting IDL or ADT style service definitions. Current work in the area of federation further suggests that the information carried by an IDL is insufficient to decide whether a service will meet a particular need. The use of an AST as the “internal” representation of a design is common in principle. Of course many such ASTs exist: they are often secret as they are thought to give market edge.

3.3.3 Interworking between systems which are built with different tools

We have to recognise that the tool which assists designers build distributed applications will transparently insert engineering mechanisms to make the distribution come true. The nature of the distribution technology is hidden from the application programmer. This is deemed to be “a good thing” because it releases the programmer from having to worry about it. A tool builder may not support many engineering functions or options. In fact it is likely that most tools will initially be brought to market with just one fixed set (e.g. a tool which can generate applications on DCE 1.01 and OSF/1[2], c.f. SNAP on IBM RS6000).

The hidden problem is that application programmers no longer have any sight of the “engineering” and therefore cannot solve interoperability problems themselves. If the market behaves properly, then there may well be a push for common interconnection engineering or engineering which can cope with

1. The appendix lists various groups concerned with tool interoperability.

2. Current support for distribution is however typically limited to inclusion of drivers for TCP/IP for instance.

multiple technologies. This will not happen in the short term: it will take time for the end-user or tool user to realise what are the significance of engineering issues. The Architecture can be used to point out the issues early rather than later and hopefully induce a willingness to agree before it is too late.

3.3.4 Are you ready for automation?

There have been several reports of organisations who have introduced CASE tools to find a reduction in productivity of their staff and a reduction of the quality of the software they produce [INFO 93]. CASE users say that without careful preparation you end up with “an automated mess”, a “faster disaster”, or “paralysis by analysis”. The preparations concern the human and organisational dimensions within which the tools will be deployed.

Most complaints are said to come from CASE tools which encompass the analysis and design activities. In many cases it was the inflexibility of the tools with respect to methodology enforcement which affronted and insulted the users. The design of a class library and its use are of crucial importance to the success of object technology. Productivity gains were reported more often from so called programming workbenches.

4 Appendix

4.1 Tools and suppliers

Abbreviated list of tools and tool suppliers which were included in our short survey:

- Art*Enterprise (from Inference Corp)
- Magic CAPtm (from Magic Software Enterprises)
- DEC/Forte (from Forte Software Inc and Digital Equipment Corp)
- Distributed Smalltalk (from HP)
- Emeraude PCTE (from Emeraude)
- Genera (from Symbolics)
- Metis (from Digital and Metis)
- ObjectTeam (from CADRE)
- OMG IDL CFE 1.2 (from Sun)
- ONTOS (from ONTOS Inc)
- OOA/RD (from Kennedy Carter)
- Orbix (from IONA)
- Ptech (from Associated Design Technology)
- SNAP (from Template Software & Instrumatic UK)
- SNiFF+ (from PtS)
- Statemate (from i-Logix)
- System Architect (from Real Time Techniques and Methods)
- Tenet (from Tenet Systems)
- VisualWorks (from ParcPlace)

4.2 Work on tools and their interoperability

The Object Management Group have for some time included a task force which has listed and compared Object Oriented Analysis and Design Tools (OMG-SIGOAD). This group has now published a comprehensive review of the various methods. The group is about to disband or reform itself. The report can be used as a basis for investigating the raison d'etre of such tools.

There are now voices in OMG which encourage the forming of a group which is to look into the tool interoperability problem. There are however also people in OMG who say that yet another group to do this would not help and that those interested in such issues should join one or more of:

- ANSI and ISO IRDS,

- CASE Communique,
- CASE Interoperability Alliance,
- EIA/CDIF (CASE Data Interchange Fromat),
- Exchange,
- IEEE/P1175,
- PCTE,
- X3H6 (CASE Tool Integration Models - Messaging Subgroup)

Below is what information I could find on some of these initiatives. Thanks to everyone who helped collect it.

4.2.1 ISO and ANSI IRDS

Note: Information Resource Dictionary Systsem has been a part of the standardisation activities in ISO for a very long time. It's early aims were to generate meta models which could be used to characterise information in independently developed database management systems. There were ideas of a sort of trading function for information which could be provided through IRDS.
IRDS has been mentioned in connection with tool interoperability work possibly because the meta models could be used as a point of reference for interworking. Details on this are not yet available.

4.2.2 CASE Communique and the CASE Interoperability Alliance[1]

There are discussions between CASE Communique and the CASE Interoperability Alliance, to present jointly a specification for abstract messages to the NIC (National Integration Center) in the USA.

4.2.2.1 CASE Communique

CASE Communique is a group, sponsored by Hewlett-Packard Company, IBM Corporation, Informix Software, Inc., and Control Data Corporation. The group was formed in October 1991. HP provided the base technology of SoftBench/BMS, which is licensed by the others. All four companies (the "framework providers") aim to provide a structured forum for the cooperative development of standard CASE tool interface specifications for application in BMS-based framework environments. Their vision is to "enable BMS-based CASE framework environments to be built from a variety of tools that provide the necessary level of control integration to automate processes and thus improve the productivity of individuals and software engineering teams". It's key goal from our perspective is "to enable inter-tool communication within BMS-based framework environments". They have a tinge of standardisation as well, since they seek to "encourage industry acceptance and use of the operation specifications developed and approved by CASE Communique".

4.2.2.2 CASE Interoperability Alliance

The CASE Interoperability Alliance appears to be Sun's equivalent to CASE Communique.

1. Thanks to Gary Levin (Bellcore) for supplying some of this information.

4.2.3 EIA/CDIF (CASE Data Interchange Fromat)[1]

The Electronic Industry Association (EIA) (best known from the RS-232 standard) is an ANSI accredited organization like IEEE or X3. The CASE Data Interchange Format (CDIF) Technical Committee is part of EIA. The CDIF Technical Committee mission is to provide a single interchange format usable by any CASE tool.

There is considerable international involvement, particularly from the U.K. (40-60% of members). Most active members are Boeing, Cadre, CCTA, Digital, DuPont, IBM, ICL, INTERSOLV, LBMS, MITRE, Oracle, Sybase, Unisys, and NIST. Most participants are CASE tool vendors. Oracle sells a CASE tool and Sybase has acquired a Canadian CASE tool company. Boeing and Mitre are user organizations.

The aim of CDIF apears to be to provide a basis which enables the exchange of specifications of systems, independent of methodology, specific tool, or application area.

1. The *CDIF Meta-meta-model* is a fixed notation for building a Meta-model. A Model is generated from the Meta-model that is universal between tools, and Data (instances of objects in the real world) are generated from the Model. The CDIF Architecture takes an approach which directly parallels the IRDS four-layer architecture and enables separation of syntax and semantic components. The goal is to minimize coupling and maximize reuse of individual components.
2. The *CDIF Semantic Model* distinguishes a Foundation (root objects) and Subject Areas for different application areas). Several subject areas have been completed and others are being worked on.
3. The *CDIF Presentation Model* is based on nodes and edges and is independent of any specific notation and can be used to relate pictures to underlying semantics. This allows transfer of graphical information, as well as the underlying semantic information. Or a graphical representation can be passed to a tool for display without any underlying semantics. The tool could display the information but would not know how to manipulate it.
4. The *CDIF Transfer Format* separates the syntax (the grammar, which is metadata) from the encoding (which could be in any language

CDIF are looking for input from Committees such as X3H7 for support in the area of object-oriented methodologies.

EXPRESS and X3H4 IRDS will use transport formats provided by CDIF. PCTE (Portable Common Tool Environment) and IDEF (high level analysis technique for business processes) may also use CDIF transport formats.

4.2.4 Exchange

Note: No information available yet.

1. The text is based on notes taken by Jeff Sutherland (ODB/Intellitic Int'l) during a X3H7 meeting, at which Mike Imber, Principal Consultant, Product Management Group, LMBS, and Chair, CDIF Technical Committee, made a presentation. The text is courtesy of the OMG SIGOAD mailing list. Any misrepresentations are entirely the responsibility of the author(s) of this Request for Comments document.

4.2.5 IEEE/P1175

Note: No details available yet.

4.2.6 PCTE

Note: Portable Common Tool Environment details to be provided (we've got some of these).

4.3 List of OO methodologies

The following list resulted from a survey of methodologies by OMG SIGOAD.

Note: I don't think I've got them all!

1. Booch
2. Class-Centred Modelling (CCM)
3. Coad & Yourdon
4. Demeter
5. Fresco
6. Fusion
7. Graham/SOMA
8. Information Engineering with Objects
9. Martin
10. Marketing to Design
11. OBA
12. Object oriented SDL
13. OGROUP
14. OORAM
15. Objectory
16. OSMOSYS
17. RDD
18. Rumbaugh
19. Schlear/Mellor
20. Software Engineering for Object Technology
21. SSADM
22. Z++

References

[TR.042]

Otway, D.J., “Abstract and Automate”, APM Ltd., Cambridge U.K., February 1993.

[APM.1000.01]

van der Linden, R.J., “An Overview of ANSA”, APM Ltd., Cambridge U.K., May 1993.

[INFO 93]

“New horizons dawn for CASE”, Infomatics, January 1993, pp.22-26.

TOOLS and ANSA

The Automated Transparency Topic Group

(in association with the Federation Topic Group)

Motivation

- **AT group is about providing distributed systems technology to designers and programmers**
- **Selective transparency and transformation tools**
- **ANSA is tool based: abstract and automate**

- **Automation is not unique to ANSA**
- **Many design, analysis, programming and configuration tools provide automation of (parts of) the design process**
- **Tools users will not give up productivity aids they use now**

- **Investigate tools market and compare ideas**

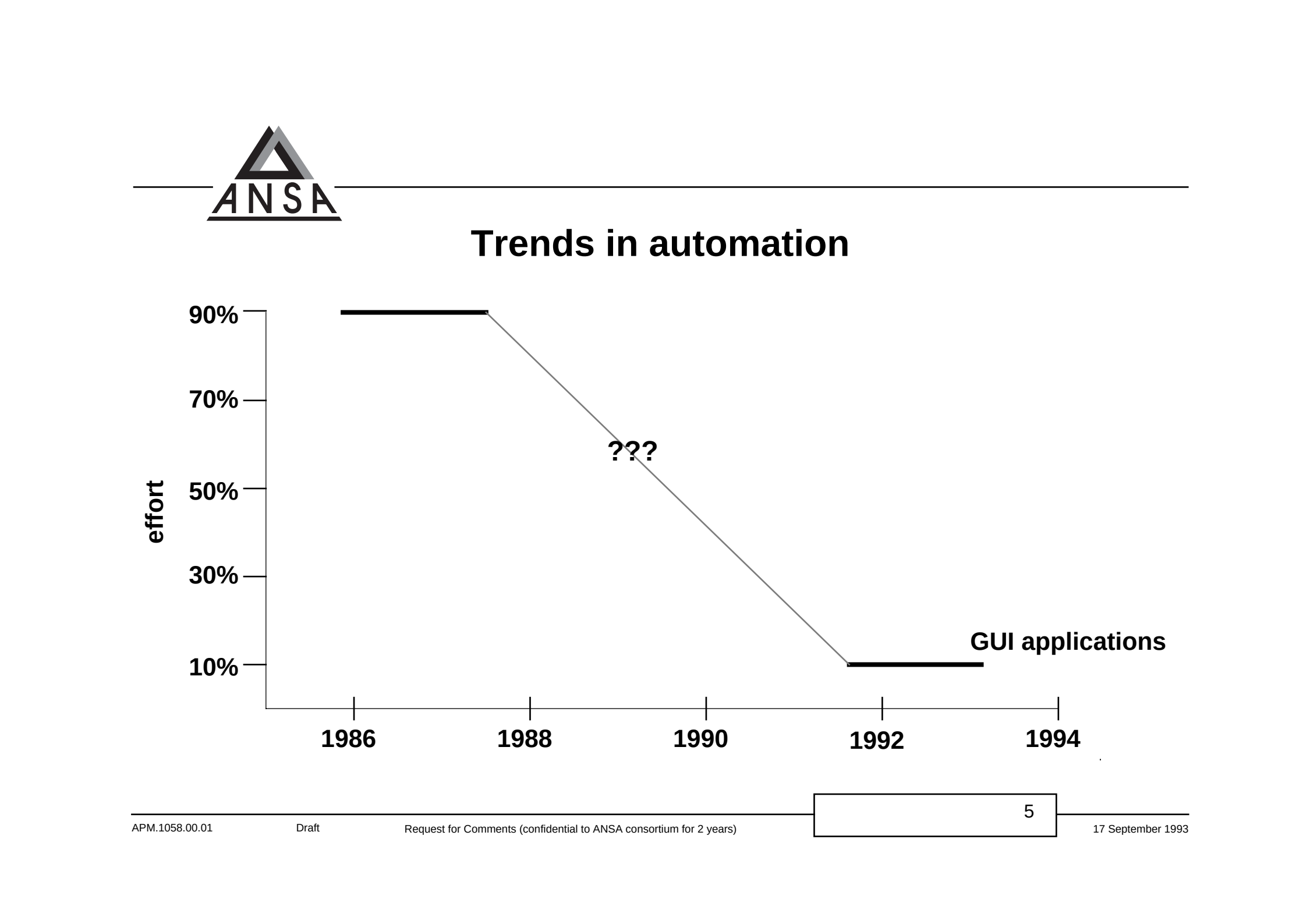
- **Survey of 20 design, analysis, programming, and configuration tools**

  - **Art*Enterprise (from Inference Corp)**
  - **Magic CAPtm (from Magic Software Enterprises)**
  - **DEC/Forte (from Forte Software Inc and Digital Equipment Corp)**
  - **Distributed Smalltalk (from HP)**
  - **Emeraude PCTE (from Emeraude)**
  - **Genera (from Symbolics)**
  - **Metis (from Digital and Metis)**
  - **ObjectTeam (from CADRE)**
  - **OMG IDL CFE 1.2 (from Sun)**
  - **ONTOS (from ONTOS Inc)**
  - **OOA/RD (from Kennedy Carter)**
  - **Orbix (from IONA)**
  - **Ptech (from Associated Design Technology)**
  - **SNAP (from Template Software & Instrumatic UK)**
  - **SNiFF+ (from PtS)**
  - **Statemate (from i-Logix)**
  - **System Architect (from Real Time Techniques and Methods)**
  - **Tenet (from Tenet Systems)**
  - **VisualWorks (from ParcPlace)**

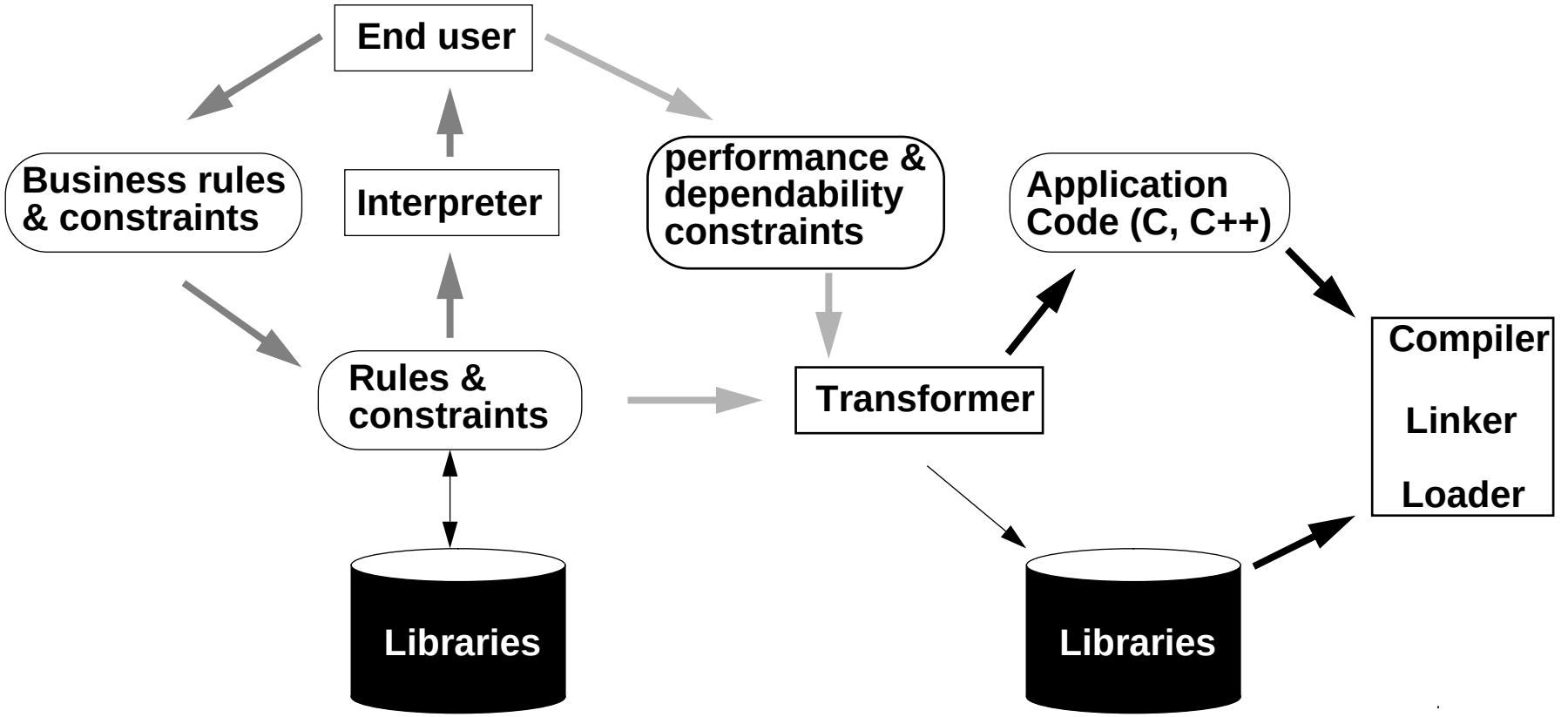
Initial results:

- **Agreement on late binding and early type checking**
- **Tools are closed systems of limited size**
- **Tool builders do understand how to make complexities transparent**

- **Tool suppliers do not advertise strategic issues of tool selection to their customers**
  - **ANSA principles can help explain the challenge**
  - **meeting that challenge requires agreement amongst tool suppliers (with user pressure?)**


ANSA
Trends in automation
90%
70%
50%
30%
10%
effort
???
GUI applications
1986
1988
1990
1992
1994

Simplified tool environment



Important features of advanced tools (1)

Rule and constraint based programming
- programming style fits business rules - close to end user
- rapid prototyping, with end-user involvement
- ANSA Information Model needs developing (rather than Computational Model)

Tool based optimisation
- rules can be turned into imperative programs (C/C++)

Construct applications rather than write them
- 20/80 rule: 20% application & 80% system code (GUI, data access, comms)
- code reuse now practical

Use libraries to hide complex systems code

Configure as late as possible
- late binding of application code to systems code
- late binding of application components (late configuration)

So far in ANSA we have restricted our work to that part of the libraries and systems code which deal with distribution of any application. This is really a small part of the whole problem space (the bottom right hand corner of the slide, cutting through the libraries and the systems code boxes, including the compile, link, load boxes.

Important features of advanced tools (2)

- **Other uses of expert system shell in rule & constraint based programming**
  - **ensure early type checking**
  - **support visual programming, e.g. GUI construction, VisualBasic, VisualC**
  - **guide browsers etc.**
  - **enforce methodologies**

- **Problems / shortcomings / extensions**
  - **tools limit size of application**
  - **comms libraries often based on 80s technology (but DCE support emerging)**
  - **IDL not visible as an agreed way to describe interfaces**
  - **stating properties ANSA-style not yet accepted**
  - **link browsing tools to OMG repositories and trading**

Strategic issues when “tooling up”

- **Do tools support distribution of the design process in time and location?**
  - **No**
- **Do current tools permit exchange of specification between tools?**
  - **No**
- **Is there any agreement on common libraries and/or class hierarchies?**
  - **No (tools hinder info exchange)**
- **Will the applications built with different tools interwork?**
  - **No (portability vs interworking)**

cgt example: legal issue of using one another’s proprietary protocols

Conclusion and direction of work

- **ANSA distribution technologies must be delivered to the programmer / system builder**
  - **How: look at modern tooling environments:**
    - buy one ?
    - get someone to buy one for us (new sponsor, MCI) ?
    - build (part of) one is NOT practical
  - **What: coordinate programming abstraction work in other groups**
- **ANSA engineering can be “exported”**
  - **If we obtain an existing environment, it must be extendable with**
    - ANSA engineering
    - OMG object & interface repositories
    - attribute based transformation technologies
- **ANSA can be used as a thinking model to help end users understand the strategic issues asociated with tooling up**
  - **A short briefing paper is in preparation**