PROJECT SPERBER

BACKGROUND, STATUS, FUTURE PLANS[1]

Erhard Ploedereder, Ph.D.
Industrieanlagen-Betriebsgesellschaft mbH (IABG)
Ottobrunn, Fed. Rep. of Germany

Abstract

Project SPERBER is concerned with the development of a high-quality software environment facilitating the development of software written in Ada or Basic Pearl. It is to be used for the design, implementation, maintenance and enhancement of embedded system software. This presentation elaborates on the background that led to project SPERBER, presents its current status, and outlines future plans. It also provides an overview of the system structure from the viewpoint of a user.

1. Introduction

In the early seventies, the US Department of Defense (DoD) became concerned with its growing software expenditures, two thirds of which were spent on maintaining and enhancing existing software. It identified the proliferation of more than 100 programming languages and dialects in use as a major reason for this cost explosion. In 1975, it therefore created the High Order Language Working Group (HOLWG) and tasked it with the project of designating a single language to become the Common HOL for Embedded Systems. This project led to the the development of a new language [1] and later became known as the Ada[2]-project.

[1]Work reported herein was financed by the Bundesamt fuer Wehrtechnik und Beschaffung (BWB) under contract no. E/F61C/B0129/95031.

[2]Ada is a trademark of the US Department of Defense (AJPO).

2. Background for SPERBER

2.1. The German Involvement in the Ada Effort

In recognition of similar problems in the German armament sector, the German Bundesministerium der Verteidigung (BMVg)[3] decided in 1977 to join the US effort for a standard HOL and contracted with IABG to act as its technical representative. It appointed a voting member to HOLWG, participated in the early language evaluations and the Test and Evaluation Phase for Preliminary Ada (the first publicly available version of Ada), and provided a member of the initial group of Distinguished Reviewers for Ada, a small experts group advising the designers of Ada in the final stages of the language definition. Furthermore, it contracted with the University of Karlsruhe to produce a pilot implementation of a compiler front-end for Preliminary Ada.

Apart from the language development, the Ada effort is also, and more importantly, concerned with the creation of sophisticated software development environments into which the language is to be embedded.

The German MoD has been involved in the environment-related efforts from their very beginnings by providing substantial contributions to early versions of PEBBLEMAN [2], one of the requirement catalogues for Ada environments.

Probably the most important German contribution to the Ada effort was made in early 1981, when the then on-going designs for Ada Programming Support Environments (APSE) threatened to diverge on one of the most important tool interfaces, the intermediate language used to represent Ada programs internally after syntactic and semantic analysis. The two candidate languages were TCOL/Ada, produced at Carnegie Mellon University in Pittsburgh, USA, and AIDA, produced at the University of Karlsruhe, West-Germany. Upon IABG's and subsequently AJPO's urging, the designers of these languages combined their efforts and defined a common sibling language, DIANA [3]. Today, DIANA is used in all military and many commercial developments of Ada compilers.

DIANA was first presented to the public at the first Murnau Workshop in March 1981. The Murnau Workshops, sponsored by the German MoD and hosted

[3]German Ministry of Defense

by IABG, have, since then, become an annual forum for an exchange of ideas on tool interfaces among the designers of Ada environments.

In its concern for standard interfaces in APSEs, IABG has delegated a member to the KAPSE Interface Team - Industry and Academia (KITIA). This effort is headed by the US Navy under charter by the Ada Joint Program Office (AJPO) of the US DoD and tasked to define a standard KAPSE (Kernel APSE, *c.f.* STONEMAN [4]) to enhance portability and interoperability of APSE tools.

2.2. Germany's Commitment to Ada

In February 1982, the vice-president of the Bundesamt fuer Wehrtechnik und Beschaffung (BWB)[4] issued a directive stating that, for future implementations of embedded systems, only the languages Ada and Basic Pearl are approved, with exceptions granted to certain special purpose applications using ATLAS, CMS-2 or JOVIAL as implementation languages.

Since usage of Ada requires the availability of an Ada compiler and suitable environment software, the BWB has initiated the project SPERBER (Standardisiertes Programm-Erstellungssystem fuer den Ruestungsbereich[5]).

SPERBER is intended to provide a sophisticated software development environment supporting the full software life-cycle for the languages Ada and Basic Pearl.

SPERBER is designed to be an integrated environment, in which individual tools communicate with each other via a central data base. SPERBER subscribes to the STONEMAN principles, such as modularity, granularity, open-endedness, and so on.

Individual components of SPERBER are produced by different software companies in Germany under contract with BWB, thus ensuring the minimality and precise documentation of tool interfaces. IABG is tasked to provide the global system design, coordinate the development of the components, and provide technical advice to the implementors, as well as to perform acceptance testing and integration of components as they are delivered by the contractors.

SPERBER does not yet contain a KAPSE. A KAPSE will be integrated into the system when international agreements on a KAPSE standard have been reached.

3. Status of SPERBER

Work on SPERBER started in 1979 as an extension of the original pilot Ada compiler project. Today, a variety of components is already available or under contract to be produced.

The host system of SPERBER are the Siemens computers of the 7.xxx series under the operating system BS2000. Targets are initially the host systems and a small process control system. The architecture of SPERBER guarantees minimal effort in retargeting the system.

Components currently available or under contract are:

1. A compiler front-end for Ada-80 has been completed in February 82.
2. An extended I/O-Package, embedding Pearl-I/O and COBOL-I/O concepts, has been completed.
3. Development of two back-ends for Ada-80 is currently in its final stages.
4. A symbolic debug system, designed to be highly code generation independent, is being developed. This system, which allows both interpretive and compiled execution of the tested program, is scheduled for delivery in late 1984.
5. The upgrade of the compiler to ANSI-Ada is being made; the availability of validated compilers is expected by mid-84.
6. Programs for compiler validation have been developed at IABG.

In 1983, tenders for bids are planned for the implementation of the central data base and its interfaces, as well as for a version, configuration and project management system.

[4]Federal Office for Procurement and Military Technology

[5]Standardized Program Development System for the Armament Sector

The near term goal of the SPERBER development is to obtain a minimal language environment (*i.e.*, a STONEMAN MAPSE).

As a medium term goal, SPERBER is envisaged to provide a systematic framework for a comprehensive environment in support of the full software life-cycle. Individual methods and tools can then be embedded into this framework.

The long-term goal is a comprehensive, integrated environment (*i.e.*, a STONEMAN APSE) supporting many of the activities in the various phases of the software development.

4. The Minimal Language Environment

The minimal language environment is primarily targeted to support the implementation and testing phase of producing Ada software.

This section provides a brief overview of the minimal language environment as an educated user may view the internal structure of the system. This view takes a very global approach, omitting many of the possibilities for technical granularity of the system. It follows activity trails that typically must be traversed in order to obtain executable code and identifies the most important tools and intermediate products along these trails. Activity trails are shown as horizontal arcs linking tools to major input and output products. Secondary inputs to tools are indicated in the figures by vertical arcs.

Figure 1 shows an overall view and identifies three main activity trails, leading to compiled code, interpretive execution, and debugging sessions, respectively. The five tool areas along these lines will be detailed in the subsections below.

The user communicates with the system through a command language interpreter which invokes the requested (sequence of) tools. A single user command (*e.g.*, "compile") may involve the invocation of a string of individual tools along the respective activity trail.

All tools communicate with each other by storing and retrieving intermediate products in the Central Data Base. Access to the Central Data Base is controlled by the domain manager, responsible for access, version, and configuration control.

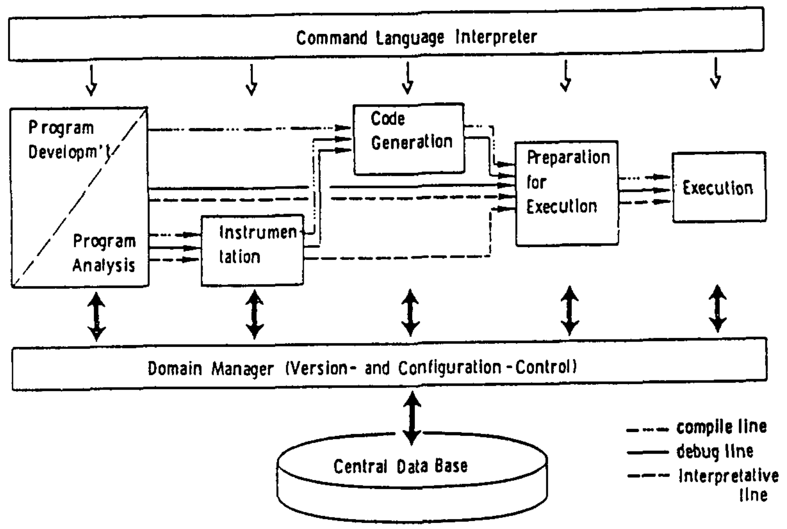

Figure 1: The Minimal Language Environment

4.1. Program Development and Analysis

Figure 2 shows the overall structure of the activity area "Program Development and Analysis".

Programs are entered into the system by means of a **Text Editor (ED)**. They are syntactically analyzed by a **Parser (PA)**, producing an Abstract Syntax Tree (AST). The AST is submitted to the **Semantic Analyzer (SA)** for static semantics analysis, resulting in the DIANA representation of the program. (For separate compilation, the Semantic Analyzer requires as secondary input the DIANA representation of previously compiled units.) Parser and Semantic Analyzer constitute the compiler front-end. Incorrect programs are rejected by these analyzing components. The intermediate representations are produced only for correct programs.
The tools **AST-Regenerator (ASR)** and **Pretty-Printer (PP)** reproduce the AST and a nicely formatted textual version of the program, respectively.

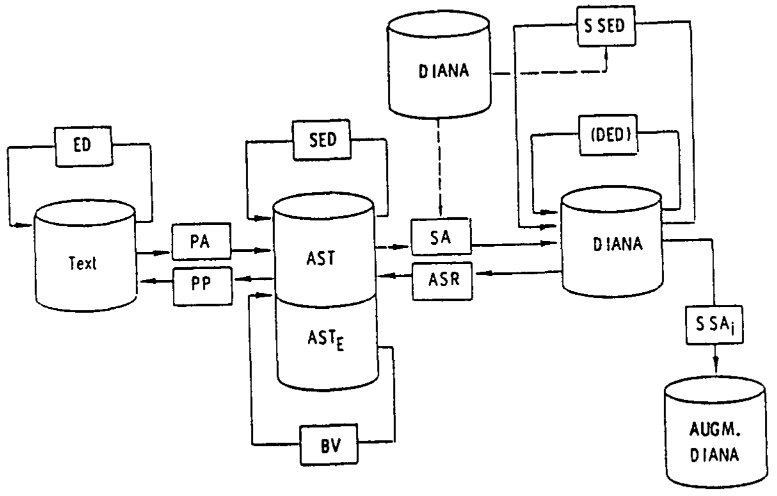

Figure 2: Program Development and Analysis

The tools mentioned above are available in SPERBER today. As further extensions, additional tools can be considered, such as

- a Syntax-oriented Editor (SED) that operates on the AST and helps in, as well as guarantees, entering only syntactically correct programs;
- a Syntax- and Semantics-oriented Editor (SSED) operating on DIANA and helping the user in entering only syntactically and semantically correct programs;
- a DIANA-Editor (DED) for debugging SPERBER implementations;
- support for potentially compilable program design languages or wide spectrum languages (BV) with tools operating on an appropriately extended AST; and
- a variety of tools (Special Analyzers (SSA)) performing semantic analysis deriving information beyond what is needed for code generation but extremely useful for the programmer (*e.g.*, cross references, set-use lists, flow-graphs, partial verification, *etc.*). The results of such analysis, which may be very expensive to perform, can be recorded in non-standard attributes added to DIANA (AUGM. DIANA) for later use to prevent the excessive cost of rederivation.

4.2. Instrumentation

Figure 3 displays the activity area of "Instrumentation". Instrumentation is done to change the execution characteristics of the program for debugging or statistics gathering purposes.

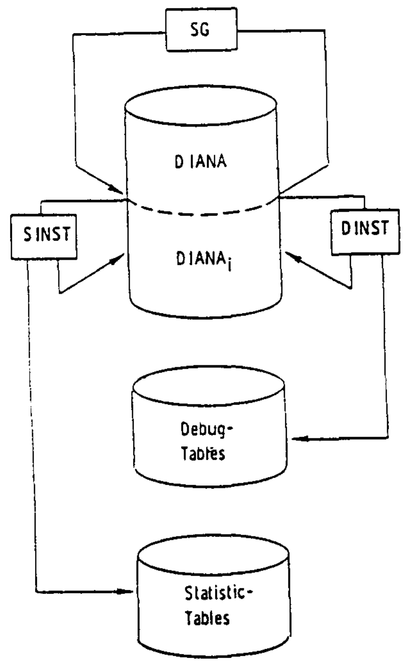

Figure 3: Instrumentation

The Debug-Instrumenter (DINST) adds debug-related instructions to the DIANA-representation of the program. The Statistics-Instrumenter (SINST) adds instructions for gathering run-time statistics during subsequent executions. The Stub-Generator (SG) provides defaulted bodies for program units whose specifications are referenced but whose bodies have not been provided yet by the programmer.

The DIANA-representation produced by these instrumenting tools is legal DIANA in the sense that it is re-translatable to legal Ada and therefore must be accepted by any code generator conforming to the DIANA standard.

These instrumenting tools are expected to be available in SPERBER by the end of 1984.

4.3. Code Generation

Figure 4 shows the activity area "Code Generation".

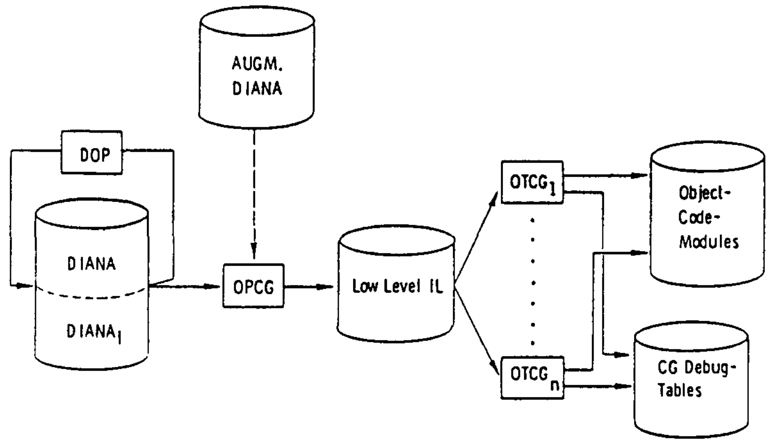

Figure 4: Code Generation

DIANA is translated by an **Optimizing Code Generator (OPCG)**, also referred to as Compiler Middle-End, to a representation in a low level intermediate language common for a large class of target machines. This tool is still largely target-independent and is primarily concerned with an analysis of the dynamic semantics of the program.
Heavy target-dependency sets in with the translation of the low level intermediate language to the object code of the respective target system by means of **Optimizing Target-Code Generators (OTCG)**.

The choice of the program representation on which optimization is performed by the OPCG is a design decision (*e.g.*, on DIANA, Low Level IL, or some hidden level between these two representations); Figure 4 nevertheless identifies the tool **DIANA-Optimizer (DOP)** as a separate component. While, initially, this tool is of restricted usefulness due to very limited optimization possibilities at the DIANA-level, it may gain in importance when programs are not entirely hand-written but partially generated by other tools, producing trivially optimizable Ada code.

The code generators for SPERBER are under contract. It is expected that by early 1984 these tools will be available.

4.4. Preparation for Execution

Figure 5 shows the activity area "Preparation for Execution", more commonly known as linkage phase. Support for Ada, however, requires special modifications of this area.

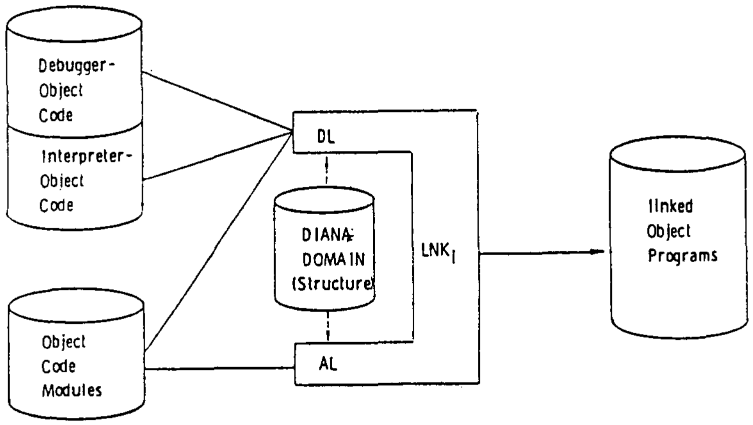

Figure 5: Preparation for Execution

The **Ada Linker (AL)** is responsible for determining the correct elaboration order (*i.e.*, linkage sequence) of the object modules prescribed by the elaboration rules of Ada. It uses the structure of the DIANA domain to derive this information. It is also responsible for recognizing the need for recompilations of program units in accordance with the dependency rules prescribed by Ada. It then delegates the linkage process to the **Target-Linker (LNK)** of the target system.

The **Debug Linker (DL)** is a special application of the Ada Linker; in accordance with user instructions, the debug system and/or the interpreter code are linked into the object program and appropriate interfaces are generated.

These components of SPERBER are currently under contract.

4.5. Execution of Programs

Figure 6 shows the activity area of "Execution of Programs".

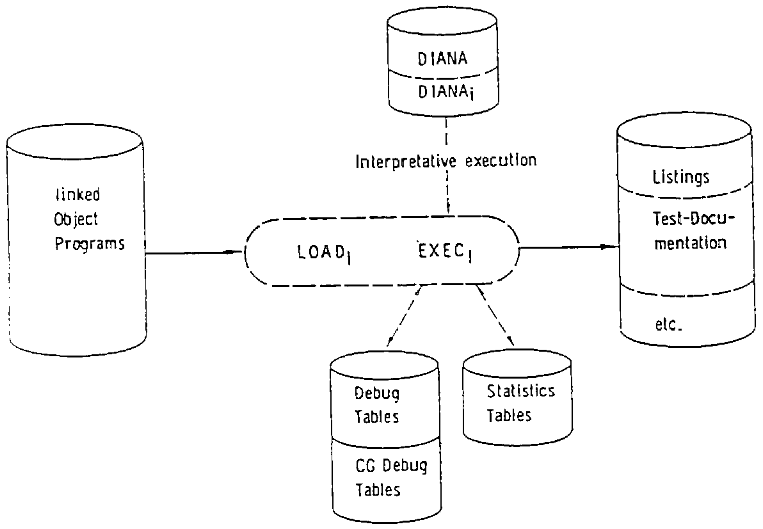

Figure 6: Execution of Programs

The linked object program is loaded into the target system by the Target-Loader (LOAD). Program execution is initiated either explicitly by an Executor (EXEC) or implicitly by the Loader.

If the executed object program is the interpreter code, it takes the DIANA-representation of the program to be interpreted as data input; if the executed object program contains debug directives, appropriate user interaction with the debug system is expected. Depending on the previously performed instrumentation, debug and statistics tables are read and updated. Program output listings and possibly information to a test harness are delivered.

The tools of the execution phase are usually the standard tools available on the target system.

5. The Framework of the Life-Cycle Support

The support by SPERBER is intended to extend beyond the implementation phase to cover the full life-cycle of the software development process.
It is, however, unlikely that a single methodology can be imposed and supported by SPERBER fit to be used for the development of a wide variety of embedded system types, ranging from process control programs to complex command and control information systems. To support a variety of individual methods used in these developments and to account for the possible inclusion of existing tools, a framework needs to be established into which these individual tools can be integrated.

An examination of the software life-cycle activities reveals that certain activities are life-cycle embracing, while others are restricted to individual life-cycle phases.

These embracing activities are of central importance to a well-organized software development process. They can be categorized as belonging to one of five areas:

- Project Management
- Data Administration
- Quality Assurance
- Configuration Management
- Documentation

The framework supports these life-cycle embracing activities by the Vertical Support Methodology of SPERBER, while being largely indiscriminate with respect to specific purpose methods in the individual phases as long as those fit into the life-cycle model supported by the vertical methodology.

It is of course often the case that individual phase-dependent methods include activities interfacing to one or more of the life-cycle embracing areas. As part of the integration of phase-dependent tools, this interface must be matched to the requirements imposed by the Vertical Support Methodology, as shown by the intersection area in Figure 7.

As the medium term goal of the SPERBER development, tools will be

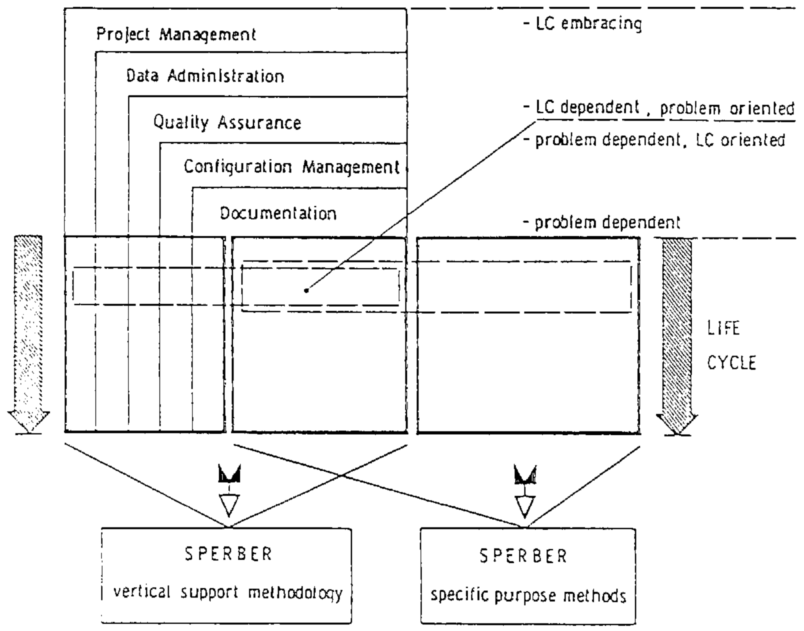

Figure 7: Framework of the Life-Cycle Support

aevcioped that implement the Vertical Support Methodology and provide the appropriate interfaces to which special purpose tools in the individual life-cycle phases can connect. Common denominator to these tools is the central data base through which all tool communication is to be accomplished.

Initially, it is expected that users of SPERBER will Integrate available phase-dependent tools suitable for their needs into the standard framework of SPERBER. As our knowledge about the suitability and versatility of individual requirement analysis and software design tools increases, it is hoped that many of these tools will migrate into the standard configuration of SPERBER, thus creating an integrated, standard but nevertheless flexible, highly sophisticated software production environment supporting all phases of the software life-cycle.

BIBLIOGRAPHY

[1] Reference Manual for the Ada Programming Language. United States Department of Defense. January 1983

[2] PEBBLEMAN revised – Requirements for the Programming Environment for the Common High Order Language. United States Department of Defense. January 1979

[3] DIANA Reference Manual. Revision 3 Technical Report TL-83-4. Tartan Laboratories Inc., February 1983

[4] STONEMAN – Requirements for Ada Programming Support Environments United States Department of Defense. February 1980