

# IT Profile

<b>Name</b>	Winfried Huber
<b>Place of residence</b>	Illerweg 6, 82140 Olching
<b>Phone</b>	+49 (0) 8142 13038
<b>Fax</b>	+49 (0) 8142 18612
<b>Mobile</b>	+49 (0) 172 9449946
<b>WWW</b>	<a href="mailto:Winfried.Huber@huber-und-boehm.de">Winfried.Huber@huber-und-boehm.de</a> <a href="http://www.huber-und-boehm.de/">http://www.huber-und-boehm.de/</a>
<b>Year of birth</b>	1957
<b>IT professional since</b>	1984
<b>Nationality</b>	German
<b>Foreign languages</b>	English
<b>Special skills</b>	system programming Unix/Windows database, data backup and recovery, versioning, linking unix/host/Vista/XP/NT/W2K integration, test
<b>Location</b>	Greater Munic
<b>Hardware</b>	Sparc, Intel, HP, Alpha, VAX, PDP
<b>Operating systems</b>	Solaris/Unix/Linux/HPUX/AIX, Windows, VMS, RTE
<b>Programming languages</b>	C/C++, various Assemblers, FORTRAN
<b>Scripting languages</b>	Perl, shell, sed, awk etc.
<b>Databases</b>	Oracle 7/8/9/10, SQL, ProC, OCI, BLOBs, Image/1000
<b>Data backup / archiving</b>	TSM
<b>Data communication</b>	TCP/IP, VoIP, SNA, LU6.2, Winsock
<b>Resume last updated on</b>	June 2010
<b>Lines</b>	banking, industry, power supply, defense

## **Special know-how:**

- High availability, reduction of down time
- Synchronization of QA environment to the production environment
- Connection of computers via ssh tunnels
  
- Application and system programming
- Design, development, test, rollout and operation
- Recovery of faulty software projects
  
- Automatically remote monitoring of and automatically software rollout on big, heterogeneous networks (involving several hosts, 650 Unix/NT/W2K-servers, 45,000 clients)
- Capturing, evaluation and analysis of SNA problems of 8,500 self-service stations aiming to make the SNA connection more robust
  
- simulators for education of ship crews (radar navigation, fighting of hostile vehicles)
- inter process communication, ssh tunnels, firewalls
- simulation of complex technical issues in nuclear power plants using cause/implication diagrams

## Some Projects:

- Project** 01.2010 – 03.2010 Defence  
Enhancement of Fleetwork Trainer
- Situation** A simulator developed by me needed enhancement
- Goal** The simulator needed porting to Linux and recent hardware and enhancement by radiotelephony (via VoIP) and flag signals
- Solution** Migration to Linux, separation of the ships in two groups (red and blue) implementing the possibility to show individual flag signals to the groups according to the international list of signals and the NATO extensions. The individual crews have to commit the flag signals to the instructor by a mouse click. All actions are recorded so they can be discussed in a later debriefing in a big room using a beamer. The instructor may skip parts of the exercise or increase the speed (“turbo mode”) to save time.  
The red and the blue group may communicate individually using head sets (maritime radio) while the instructor may join the red or the blue group at his choice.
- Environment** Linux, C, X11R7, VoIP, mumble
- 
- Project** 05.2009 – 12.2009 Bank  
Infrastructure monitoring system
- Situation** Staff members joining, leaving or changing their focus result in changes at different places (entries in LDAP, net groups, generation and distribution of ssh keys etc.).  
In the long run this results in inconsistencies.
- Goal** Those inconsistencies are to be avoided or at least they should be reported
- Solution** The automatically infrastructure monitoring system makes sure all entries are consistent (e.g. all staff members that should be allowed to log in to a given machine actually are a member of the appropriate net group, but no one else). Inconsistencies are to be reported directly on interactive runs, reported by mail and/or entered into a appropriate quality survey system.
- Environment** Solaris, LDAP, perl, shell
- 
- Project** 04.2009 – 08.2009 Bank  
automatic setup of QA and test environments
- Situation** The customer runs a big versioning system. While the real data is kept on disc the metadata is located in an oracle database. For development and test purposes test systems need to have consistent and recent data.
- Goal** automatic setup of test systems
- Solution** automatic jobs running after the backup window provide a TSM catalogue showing the state of the item library consistent to the database. This makes it possible to quickly set up test environments with consistent database (metadata) and item library (data).  
In addition one early integration system automatically gets the database cloned and the item library restored synchronized each weekend. So this early integration system holds consistent and recent data each Monday morning.
- Environment** TSM, Solaris, ORACLE, perl

**Project** 02.2009 – 04.2009 Bank  
Automatic, central capturing of test results

**Situation** Software Configuration Management produces distributed test results, by installation and by cyclic tests

**Goal** These test results should be gathered in a central place

**Solution** Using an interface these test results are grouped to the products and machines and stored in a central place, So the team leaders can look at one place to see the state of all products.

**Environment** Solaris, perl, shell

**Project** 02.2009 – 04.2009 Bank  
Automatic support to fix misconfiguration

**Situation** Building packages by an automatic build system driven by a versioning system raises the need to set the owner and group, permission flags etc. of the files that make up the newly created package. This may fail due to incorrect configuration of the build system, LDAP or wrong entries in the package configuration files.

**Goal** The reason of the failure should be detected automatically (e.g. wrong case in a group name or missing owner entry on the build system) and the way to fix this. This gets reported to both the person who is in charge of fixing this failure and the person that suffers from this failure.

**Solution** Failed package builds get analyzed for the reason of the failure. Information needed to fix and handle this problem is derived from various sources (e.g. LDAP) and sent by mail to the appropriate persons or mail boxes. This mail contains the commands to run so the admin does not have to collect all the parameters needed (e.g. numerical group id) while still actually deciding whether he wants to run the commands or not – they did not accept automatically modifications to the system done by the build system.  
So no one needs to take the effort to analyze the problem and to figure out whom to contact to fix this. All parties get the information needed for free and they know that this knowledge is available to the others as well. This way the person fixing the problem knows which person to contact “Ok, the problem got solved now, please restart your build”. This greatly reduces the effort of all persons affected, reduces the round trip time and avoids errors hard to find.

**Environment** Solaris, LDAP, perl, shell

<b>Project</b>	09.2008 – 11.2008 Bank Automatic creation of solaris patches
<b>Situation</b>	The customer runs a versioning system. This is used to create solaris packages.
<b>Goal</b>	Upgrading solaris machines by installing new solaris packages demands a down time that sometimes may hurt. So this down time should be reduced or avoided as possible. One way to achieve this is to install patches instead of packages.
<b>Solution</b>	The package creation system grabs the sources from the versioning system, throws them on the build server and builds the new package. This is inserted into the versioning system along with other build results to assure the reproducibility. I enhanced this system to compare the package just build to the version released recently or the version indicated by the package configuration files. This yields a patch that might be applied to bring the package already installed on a machine to the current state thus avoiding taking the effort to install the full package. The system is smart enough to distinct between differences that may be neglected (e.g. checkout time stamps in headers lines that do not affect the operation) and functional modifications that need to be inserted into the patch. This way the developer gets a new package containing everything and a patch to promote the last package to the current one without the need to tell what exactly got modified since the last release. This is determined automatically.
<b>Environment</b>	Solaris, perl, pkgadd/pkgmk family
<b>Project</b>	08.2008 – 09.2008 Bank Automatic setup of solaris zones
<b>Situation</b>	Many different development systems lead to high costs. Setting up virtual solaris machines (“zones”) manually is expensive and error prone. Manually analyzing misbehaviours later and fixing them is consuming time and money.
<b>Goal</b>	Consolidation into few physical servers running virtual machines (“zones”). Making it simple to throw away to and set up zones yields better and clean test environments.
<b>Solution</b>	Preparation of a job that tears down and recreates solaris zones in a reproducible way. Thus it is easy to get a brand new solaris zone containing all basic elements providing a clean test environment.
<b>Environment</b>	Solaris, shell, perl

**Project** 06.2008 – 07.2008 Bank  
Clearly arranged overview of the software versions

**Situation** It was hard to keep an overview of the various software versions installed on all the development, IT, QA and production environments

**Goal** Clearly arranged overview over the actually installed software versions pointing out any discrepancies to the expected versions or installation problems

**Solution** Development of a data aggregation tool to gather the needed information from the different servers and to show them on HTML pages. Any problems are marked using colours. So one glimpse is enough to see whether there are any problems: Unexpected software versions, installation problems, missing or misplaced packages.

**Environment** Solaris, perl, HTML

**Project** 06.2008 – 07.2008 Bank  
Trouble shooting

**Situation** Excessive resource consumption lead to crashes  
Modifications requested by another customer to the software of a third party manufacturer lead to serious problems for my customer: Excessive memory consumption of new created processes made the system slow due to swapping, later thrashing and finally resulted in a crash of the cluster.

**Goal** Diagnosis, emergency procedures to ensure smooth operation.

**Solution** Development of a tool to monitor the memory state of the machine and to identify processes that may be killed causing no harm. The processes got sorted according to their duration of inactivity. The processes with long inactivity that could be transparently restarted causing no harm got killed to ensure smooth operation of the system.  
The only effect to the users affected was a small delay when they resumed their work necessary to transparently restart their application process.  
The criteria used to sort the processes was the time stamps of all active file descriptors of the process thus sparing processes with network, disc or database activities to avoid killing long running processes.

**Environment** Solaris, PVCS Dimensions, perl

**Project** 02.2008 – 05.2008 Bank  
Trouble shooting

**Situation** dead locks blocked third party manufacturer software

**Goal** Diagnosis, remedy

**Solution** Creation of a tool to recognize dead locks and to combine the information derived from the process and various databases to obtain the information needed to track down the problem (Which user caused the problem? What did he try to do? Which software did he use to access the system?) and to get a clue what to do right now to solve the problem. You're better off killing one application process than to wait until the database gets blocked completely.

**Environment** Solaris, Oracle, PVCS Dimensions, SQL, perl

<b>Project</b>	03.2007 – 01.2008 Bank
<b>Situation</b>	Migration of (PVCS Dimensions 9, Solaris 9, Oracle 9) to (PVCS Dimensions 10, Solaris 10, Oracle 10), introduction of a Sun cluster, resource groups, zones PVCS Dimensions Version 9 running on a Sun Fire 440, the Oracle-9 database was running on a different Sun Fire 440, both machines running Solaris 9
<b>Goal</b>	Migration to PVCS Dimensions 10, Solaris 10, Oracle 10 and to make sure uninterrupted operation by using a Sun Cluster
<b>Solution</b>	<ul style="list-style-type: none"><li>● Make sure the robustness of the package building procedure by running it directly driven by the versioning system (PVCS Dimensions)</li><li>● Setup of the complete systems including the application and the database by installation of solaris packages</li><li>● automatic, parallelized migration of the data from PVCS Dimensions 9 to PVCS Dimensions 10</li><li>● Assure high availability by movability of the components within the cluster: the database runs using a resource group and the application runs within a solaris zone (virtual machine). This has big advances:<ul style="list-style-type: none"><li>● Distributing the cluster nodes to independent data centres and mirroring the SAN discs in the other data centre ensures further operation even on disasters destroying one of the data centres.</li><li>● Running at low load may be done within one cluster node avoiding network traffic between the application and the database while increasing the performance.</li><li>● Running at high load the database and the application are running on different cluster nodes. By using private TCP/IP connections the loss by network performance is low while both the application and the database have access to the full memory and CPU resources</li><li>● In case maintenance activities are needed at one cluster node this can be done by evacuating this cluster node. So the evacuated node may be shut down and maintained.</li><li>● In case of serious problems at one cluster node this may be evacuated assuring further availability with perhaps reduced performance. Adding one more cluster node to the cluster makes it possible to redistribute the load again to gain time to fix the problems of the sick cluster node.</li><li>● By adding further cluster nodes the performance may be increased further.</li></ul></li><li>● Providing of a QA environment that is regularly updated by cloning the productive database and synchronizing the item library using the TSM archive. This item library needs to be consistent to the database. Doing things this way does not put any additional load on the productive environment while keeping the effort to restore the item lib reasonable. Only the deltas to the last restored version of the item lib need to be restored and the items generated during the last test phase on the test system need to be deleted. A full restore would be way too slow.</li></ul>
<b>Environment</b>	PVCS Dimensions, Solaris, Oracle, TSM, perl, bash

**Project** 09.2004 - 12.2005 Bank  
Design, development and doing of the consolidation of data that needs safekeeping by law to a TSM archive

**Situation** Over many years transactional data has been produced on a lot of servers distributed around Bavaria. This data has been archived to a lot of different cartridge types. This data needed consolidation in a central TSM long time archive while keeping them separated by client.  
The data needed to be checked for completeness, consistency and redundancy. New servers showed up at some times (e.g. new branch bank in a newly build commercial centre), old servers vanished because they got replaced by new ones or consolidated to a new, bigger server, and some of the clients fused.

**Solution** The cartridges got read on site by the individual admins and transferred to a filer. By running a perl script the data was checked for completeness, consistency and redundancy. It is not a real problem to have the same data twice as long as the contents really is identical. Just make sure the data is moved to the correct place and the superfluous duplicate gets removed and you're done. But having the same data twice with **different contents** requires more attention. This checking procedure perhaps was done once more, e.g. after reading more tapes that got forgotten in the first run, cleaning tape drives to avoid read errors, using a different tape drive etc.  
After the cleanup procedure has been finished the data was written to a TSM long time archive thus avoiding the need to keep a lot of different tape drives types, servers and databases for many, many years thus greatly reducing costs. And in addition the admins got an overview about the completeness of their data – seeking for lost cartridges gets more and more unpromising as years go by, staff members leave, etc.

**Environment** W2K, perl, TSM command line tools

**Project** 07.2004 - 10.2004 Bank  
Checking of database software to meet new requirements

**Situation** For many years software running on a lot of servers produced data and wrote them to oracle databases running on local sites.

**Goal** These databases should get consolidated to one database running in a data centre.

**Solution** A careful analysis showed potential risks that did not hurt as long as the databases were distributed. But consolidating them to one database would have caused serious problems.  
By redesigning the software in time this potential risk could be avoided.

**Environment** C, Oracle 9, Java, ProC, NT4.0, W2K, XP

**Project** 04.2003 - 09.2003 Bank  
Design and development of a password distribution service

**Situation** Passwords to access database accounts or network resources were hard coded or set at the installation time

**Goal** Passwords hard coded or set at installation time should be avoided and replaced by passwords that can be changed on a regular basis. Applications authorized should get this passwords automatically while applications not authorized or even manipulated (e.g. by a forged shared library) should be rejected.

**Solution** I developed a service that provides authorized applications with the current passwords transferring using encryption. So these passwords can be modified on a regular basis, either automatically or manually. They get distributed to the authorized applications automatically (and to those only). In addition these applications were protected against manipulations of sensitive parts of the application.

**Environment** C/C++, MFC, PGP/gpg, NT4.0, W2K, XP

**Project** 05.2003 Bank  
Long time archiving

**Situation** The client runs a high available payment transaction system running on a AIX HACMP cluster. Transactions initiated by staff members or corporate customers create data walking through various pools (directories).

**Goal** To make sure these transactions can be reproduced these data is – separated by client – to be archived for a long time in a TSM archives.

**Solution** The transaction data is gathered and separated by client. This is easy for some of the files while others require database queries to determine the appropriate client. Log files are filtered per client and archived along with the transactional data to meet the lawful demands.  
The initial versions of the log files are archived along with log files that can not be separated per client in a “rag picker” archive.  
This way retention periods specific for each client may be realized to avoid wasting money.

**Environment** perl, shell, TSM

<b>Project</b>	09.2003      Bank Optimization of the backup-strategy
<b>Situation</b>	The client runs a high available payment transaction system running on an AIX HACMP cluster. Every night during the low activity period a TSM backup is triggered while the system remains operable, the application is not shut down.
<b>Goal</b>	This backup strategy was to be checked for weak points and cost cutting opportunities.
<b>Solution</b>	<p>The analysis showed up some weak points causing unnecessary costs and resulting in dangerous points, e.g.</p> <ul style="list-style-type: none"><li>● The ORACLE online backup was started after the file system backup leading to the consequence that the database files produced one day earlier were saved. This makes it necessary to roll the database one full day ahead by applying the redo logs to bring the database to the state of the nightly backup. This is not necessary and time consuming. In addition to the full last day the redo logs of the current day up to the crash needed to applied anyway.</li><li>● The retention periods of the online backups and the data pools selected via TSM management classes did not match. This generated unnecessary costs without producing any useful effects.</li><li>● In addition to the online backup files and the redo logs of the ORACLE database the “living” oracle data files got saved as well. This produces unnecessary costs without producing any useful effect. Contrary, this is dangerous: Restoring the file system data in parallel (to save time) while rolling ahead the ORACLE database destroys the just painful reconstructed database!</li><li>● To make sure a fast recovery after a crash several generations of the online backup files were kept on disc as well as saving them to TSM. Due to the rapidly growing amount of data this lead to the need to buy new discs soon. By compressing the online backup files after the TSM run this could be avoided. As ORACLE database files compress very well this greatly reduces the amount of disc storage needed for this – only one generation of this files need to be kept on disc without compression.</li></ul>

All of these weak points could be remedied completely.

**Environment** perl, shell, TSM

<b>Project</b>	04.1999 - 10.2002 Bank Remote Service Agent
<b>Situation</b>	<p>The client, an umbrella organization, provides software to about 100 autonomous organisations. I will call them “customer” here. The installation and operation of the software is done by the admins of the customers.</p> <p>The staff of the client is able to access the customers machines via an internal network as long as the customers admin enables this so called “remote service”.</p> <p>This cumbersome procedure (call the customer's admin and ask him to enable remote service on a certain machine for a while) is OK for singular actions. But in rare cases where access to a lot of servers is required this way is not possible.</p>
<b>Solution</b>	<p>Development of an automatically, unattended, host based remote service system for about 650 application servers running Unix, NT and W2K.</p> <p>The jobs are prepared on a specific server of the client called the “master server”:</p> <p>Provide all files necessary to do the job (programs, scripts, files ...)</p> <ul style="list-style-type: none"><li>● Provide one or more AutoAction scripts: This AutoAction script does the real work. In case multiple AutoAction scripts are provided the correct one will be selected automatically.</li><li>● Pack all theProvide one orse file in a compressed archive</li><li>● Sign this archive with a special key</li></ul> <p>Now this job needs to be distributed to the customer machines. This is done by uploading them from the master server to a specific host database. At this place is determined for which servers this job is provided and when it should be executed.</p> <p>The customer machines regularly check the host database whether there are any jobs provided for this machine.</p> <p>In this case these jobs are loaded down on the customer machine and executed there. Messages and logs are sent back to the master server. So the jobs are executed under “full visibility”: It can be determined whether the job ran fine on all systems thus identifying the systems that had trouble: Only these few machines need manual maintenance with the need to ring up the customers admin and ask him to enable the remote service access.</p> <p>Besides the AutoAction script one or more FailureRevert scripts may be provided as well to undo actions that got stuck in the middle of the way for reason whatever. It is better to revert to the old version than to completely screw up the machine.</p> <p>My part of the job was everything besides the implementation of the host parts.</p> <p>The system proved to be very useful for jobs that need to run on a specific day on all systems or for “fire fighting” operations. Or to make pretty sure that really <b>all</b> 45,000 clients got a new software version installed before you can dare to install a new software version on the host...</p> <p>One more point is that this system can provide servers that got set up newly with the latest patches.</p>
<b>Environment</b>	C, perl, SNA

- Project** 02.1998 - 07.2002 Bank  
Service Control Monitor (monitoring and automatic trouble shooting on servers)
- Goal** Make sure all servers run smoothly
- Solution** Design, development and operation of a monitoring system that monitors the software components installed on a server and autonomously takes actions if something goes wrong or at least generates alarms.  
A sophisticated timing is provided; some of the components are to be monitored during the office time only while other problems are negligible if they disappear within a given period on their own. To achieve this the service control monitor (scmon) dynamically reconfigures itself to adapt to the current state.  
Especially for rare error conditions the log files generated by scmon proved to be an excellent source of information to analyze the problems.  
One main goal of the design of scmon was efficiency to avoid increasing resource problems. Common checks therefore are integrated in scmon and thus run very efficiently. Exotic checks may be provided as shell scripts or programs that are called by scmon.
- Environment** C, Unix, SINIX, NT, W2K, perl
- Project** 10.1997 (extended later)  
LogFile() library
- Goal** provide simple and efficient log file writing to scripts and programs
- Solution** Design and development of a library to be used by scripts or programs that provides simple and efficient logging facilities, e.g. adding formatted time stamps and limiting the size of the log files. Once the log file reaches its limit it gets aged by adding a time stamp to its name and a new log file is started. When aging a log file the system considers discarding previously aged versions that exceeded the retention period.  
If disc space is short the systems starts dropping old log files by reducing the retention period to ensure smooth operation. But it won't discard the current log file and its youngest aged version to avoid wiping the logs needed to analyze what caused this problem.  
Several scripts or programs may write to the same log file. They will discover aging done by other processes ensuring consistent logs.
- Environment** C
- Project** 03.1996 - 12.2001 Bank  
Software distribution, automatic setup and configuration
- Goal** Smooth software installation in spite of heterogeneous environment
- Solution** Design and operation of a integration test environment to make sure the software packages install and operate fine. Development of a setup script that gathers all data needed for the setup, preferred automatically, and statically verifies this information for completeness and consistency and checks for other potential risks (e.g. incompatible software versions on other servers or ORACLE databases).  
The main goal was to reduce the down time and to make the process reproducible.  
Some of the packages to be integrated were mine, for some others I just wrote the setup, while others were provided completely.
- Environment** Unix, Sinix, NT, W2K, C, perl, shell, pkgadd family

**Project** 03.2001 - 10.2002 Bank  
Capturing down times of cash terminals (SNA problems)

**Situation** The client runs a big net with about 8,500 automatic teller machines. These write log files showing the state of the SNA connection needed for proper operation.

**Solution** Parsing of the log files to gather information and send this to a central place for analysis:

- detection of network problems (e.g. router overloaded)
- analysis of error recovery problems following network failures or maintenance down times
- detection of load problems (SNA time outs)
- detection of sick servers
- detection of faulty configurations / device combinations

The results are provided in various formats, tabular and graphical.

**Environment** C, perl

**Project** 10.1988 - 05.1990 (multiple sub projects) Defence  
Fleetwork Trainer

**Goal** provide a simulator to train ship crews to interpret radar screens

**Solution** Training ship crews with respect to radar navigation is fairly complex and expensive, so providing a simulator to do this is a good choice. Every crew sees its own radar screen and may select some options, e.g. radar range, range rings, north up etc. Every crew has a own cubicle and runs its own ship. All Crews have to do a common fleetwork task, e.g. various formation changes within a given time.

The instructor may enable or disable various aids, e.g. history blips showing the position of the other objects in the past or displaying icons representing the type of the other ships (e.g. frigate) as opposed to simple blips.

The instructor may interrupt the exercise to gain time to explain problems as well as electronically “entering” other ships to check the settings selected by the individual crews thus avoiding the need to walk around to visit the cubicles. In addition to this the instructor is able to control ships not allocated to any cubicle (“orphaned ships”) or to cubicles currently not manned.

All actions taken by any crew are recorded. This way the exercise may be replayed later for debriefing purposes using a beamer, all crews in one room. The instructor may skip parts of the exercise, or increase the replay speed to save time (“turbo mode”).

Part of the system is a coast line editor to create pseudo realistic coast lines to be displayed at the radar screen as well as an exercise editor used by the instructor to set up the various exercise initial scenarios, e.g. placing the ships, assigning them to cubicles, selecting ship types, initial speed and course and more.

**Environment** C, X10, X11R3 - X11R5

<b>Project</b>	01.1994 - 11.1994    Defence Actual speed tactical trainer
<b>Situation</b>	The initial situation was a failed software project. The client ordered a simulator to train the use of weapon systems on ships from a foreign company that failed in the mean time. A central component of this simulator was the so called “postal service” that took care of the message distribution within the simulator (consisting of several computers), e.g. controlling the ship, pushing buttons to select or arm weapons, assigning fire channels, picking targets and so on. This postal service worked fine in periods of low activity. But it blocked as soon as the amount of messages increased a lot due to a threatening situation coming up.
<b>Solution</b>	The initial development team was gone, the documentation was rudimental. The first part of the project was statically analyzing this postal service to determine the weak points: Design mistakes lead to bottle necks resulting in dead locks. Result of this part was a documentation of the problems showing a way to overcome these problems. The second part of the project was to redesign the postal service to make it robust against any amount of messages coming up. This was done by smart intercommunication techniques, shared memory segments and semaphores to avoid dead locks.
<b>Environment</b>	C, Unix, IPC

<b>Project</b>	10.1986 - 06.1988    nuclear power STEP-Graph
<b>Situation</b>	<p>In order to simulate the behaviour of nuclear power plants, especially dealing with failures, the propagation of failures is described by reason/impact diagrams provided by engineers in advance.</p> <p>In order to process these diagrams automatically and to feed the system with the current data derived by the plant these diagrams need translation into a source text satisfying a specific grammar.</p> <p>Due to the uncommon grammar this procedure is tedious and error prone resulting in many round trips (edit, compile, fix errors, recompile, ...).</p>
<b>Solution</b>	<p>In order to assist this tedious activity a syntax driven editor was created. This reads the grammar at start up time and accepts only text as input that is compliant with the underlying grammar. If you can move the cursor ahead to a certain point of the text this text is syntactically correct up to this point. A menu shows all terminal symbols that may appear at the cursor position. So you don't have to enter all characters making up a terminal symbol; just type until it is unambiguous and the rest will be completed automatically.</p> <p>If you can move the cursor to the end of the text and a special terminal symbol "End of text" is possible there this means the text both is correct and complete. This solved the problem dealing with syntax errors, but it did not make sure the diagram was entered correctly. For example selecting an "or" gate instead of the intended "and" gate.</p> <p>So in a second step the underlying grammar was enhanced by graphical information with no effects to the syntax but holding all the necessary information to reconstruct the graphic diagram on a second screen while typing the text. If the graphic diagram derived this way is identical to the diagram used as source this proves the diagram has been converted to the text form needed for automatically processing has been done correctly.</p> <p>These automatically derived diagrams were printed as well and used for documentation purposes for the surveillance authority (TÜV).</p>
<b>Software:</b>	C, FORTRAN, GKS, X11