AUTOMATED GENERATION OF REAL-TIME SOFTWARE FROM DATASHEET-BASED INPUTS
THE PROCESS MODEL, THE PLATFORM AND THE FEEDBACK FROM THE MSL PROJECT ACTIVITIES

Rainer Gerlich

BSSE System and Software Engineering

Auf dem Ruhbuehl 181
D-88090 Immenstaad, Germany

Phone +49/7545/91.12.58
Mobile: +49/171/80.20.659
Fax +49/7545/91.12.40
e-mail: gerlich@t-online.de
URL:http://home.t-online.de/home/gerlich/

Michael Birk, Uwe Brammer, Michael Ziegler, Klaus Lattner

Kayser-Threde GmbH

Perchtinger Strasse 3
D- 81379 Muenchen, Germany

Phone: +49/89/72494 - 359

Fax: +49/89/72495 - 398
e-mail: la@kayser-threde.de
URL: http://www.kayser-threde.de

Abstract: In order to cope with varying, non-settled requirements, to cover the needs of the onboard target and the scientific ground platforms, and to make software development more efficient, an approach has been applied to the software of the MSL (Material Science Laboratory) project [1] which allows for automated generation and early validation of the distributed real-time software from inputs provided by datasheets. The platform, which is ISG (Instantaneous System and Software Generation), takes a formal definition of a system's properties, generates automatically the infrastructure software and a significant part of the application-specific software, the environment for execution and automated testing, and performs automatically formal checks at pre-run-time, run-time and post-run-time. This paper describes the ISG platform and the underlying formalisation, compares ISG with other development approaches and discusses the feedback as obtained from the MSL project.

1. INTRODUCTION

In the past a number of activities were executed to speed up the software development process and to allow for early system validation [2]. A first result of these activities was the EaSySim II [3] environment. Based on the feedback obtained by a number of projects to which EaSySim II was applied a new environment called ISG (Instantaneous System and Software Generation) [4] was established which completely automates the generation and testing of the software. ISG expects inputs from datasheets, generates executable code of a real-time application for Intel and Sparc platforms, provides information as needed for verification and validation (V&V), and evaluates this information automatically.

While [5] describes the use of ISG from the project's point of view this paper presents details about ISG, its capabilities, the difference to other development approaches and the feedback from the MSL project.

While at the beginning it was planned that ISG should generate the distributed real-time infrastructure, only, in the course of the discussions with the MSL project the database, the telemetry, the data and command processing software ("DTDC Software") were also identified as further subjects of ISG.

Chapters 2 and 3 describe the generation of the infrastructure and DTDC software and the Verification and Validation (V&V) activities. The differences between ISG and other software development approaches and the feedback from the project activities are discussed in chapters 4 and 5. Finally, an outlook is given on future activities and evolution.

2. THE ISG APPROACH

2.1 The ISG Process Model

ISG allows for incremental development and early and continuous system validation. Most of the application software is automatically generated from system engineering inputs which are (currently) provided by datasheets as described by [5]. Fig. 1 shows the ISG process model. The user needs to provide information about the partitioning of the system into processes and processors, the behaviour as expressed by Finite State Machines (FSM), the data (command) exchange between the processes, the toplogy and performance. This information is provided by the "Command Procedure Table" (CPT) and is maintained by an Excel [6] data sheet in case of MSL.

The generation process allows for a number of options. Together with the CPT these options are given as user inputs by files which are shown at the upper left corner of Fig. 1-1.

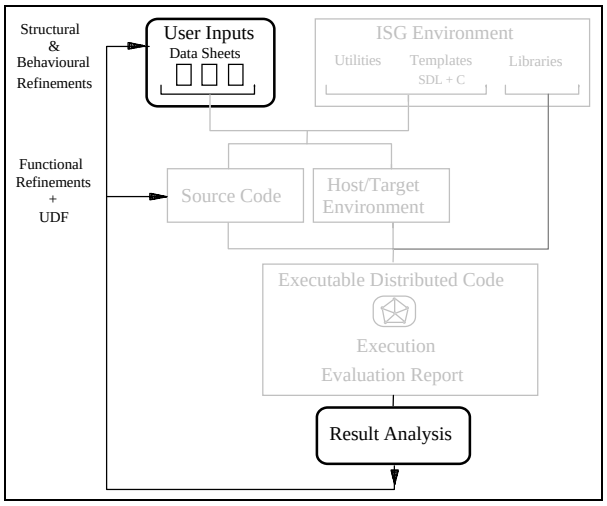

Fig.1 -1: The ISG Process Model - The User's Point of View

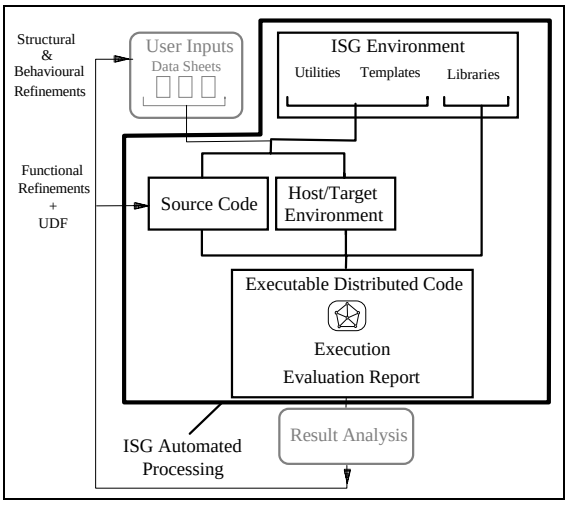

Fig. 1-2: The ISG Process Model - The Tool's Point of View

ISG takes these inputs and generates the source code, the data inputs needed for execution and the environment for compilation, execution and reporting automatically for each of the different platforms. This is shown by the middle part of Fig. 1-2. Such platforms may be Sparc or Intel with Solaris, Linux or VxWorks. To get a feedback from the next system version a user just needs to provide his engineering inputs and to wait until a window pops up with a report on the results of execution[1].

Although the input is based on datasheets and ASCII files, formal checks can be performed on such inputs because a meta-model exists which correlates the various inputs by rules. This is true for generation of the infrastructure and of the DTDC software.

The meta-model is directly related to the ISG construction rules which convert the user inputs into the executables and the V&V environment. Such construction rules imply standardisation of interfaces, inter-process communication and a process' structure. They generalise the class concept of the object-oriented paradigm, allow to cover a broad range of applications, and to automate system generation.

The formal approach which ISG is applying does not only support the generation of the source code, but everything what is needed to get immediately an executable system and an evaluation report.

2.2 The ISG Toolset

The ISG toolset consists of a set of utilities and Unix scripts which convert the user inputs into executable code, execute it within a network and evaluate the results as obtained by execution.

In case of MSL a complete generation process of the infrastructure from user inputs takes about 45 minutes (on an UltraSparc I/143), of which about 15 minutes are needed for compilation and linking. Another 30 minutes are needed until a coverage of 100% of the input domain is achieved and the evaluation report is available. The generation of the DTDC software takes about one hour and covers the generation of the database and about 650 functions for calibration and post-processing.

This yields a total turn-around time between about 1¼ hours and 2¼ hours. The turn-around time is reduced to ¾ and 1¾ hours respectively if the infrastructure is not changed, but software has to be recompiled only like the DTDC software, User-Defined Functions (UDF) or other software as provided by the user.

2.3 Automated Generation of the Infrastructure

In context of ISG a system's infrastructure is defined as software which is needed for system set-up, inter- and intra-process communication within a network, representation of behaviour by Finite State Machines (FSM), timed execution (periods, timeout conditions), and the implementation of exception handling and fault tolerance (Fig. 2).

ISG covers all aspects of real-time processing, communication and distribution. The remaining software like DTDC which is purely sequential and which does e.g. data processing can be plugged in by the concept of UDF's as described below in 2.4.

---

[1] For details about the organisational issues of ISG from the project's point of view see [5].

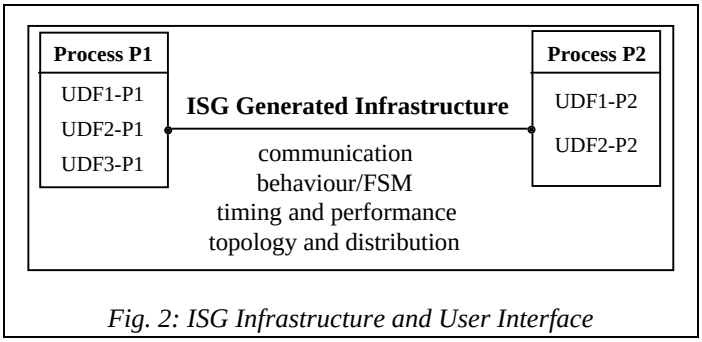

Fig. 2: ISG Infrastructure and User Interface

The infrastructure software of MSL is derived from the Command Procedure Table (CPT) and the files for definition of the configuration.

On the highest abstraction level process types are defined by the CPT. From the process types the desired number of process instances ("processes") is derived. All instances of a process type share the same code. For each instance data are separately allocated by ISG, but a user may add additonal data, which also may be shared by the instances of a process type. A CPU is assigned to each process instance or to a process type in case all instances execute on the same processor in order to measure the consumption of resources when the real target configuration is not available

The behaviour of a process type is defined by an extended FSM concept which allows for cross-checks between process types regarding inputs and outputs. A state transition is triggered by an incoming command. A number of "alternative actions" may be defined for each state transition. Each alternative action consists of a number of "atomic actions". The principal structure of an atomic action is "input/state - processing - output/state". A UDF (see 2.4) is associated with each atomic action and does the data processing. The amount of expected CPU consumption, the priority of processing, and the expected amount of issued data are also required to allow for representative execution right from the beginning when the real target might not be available.

The configuration files define the type of the platform, the degree of instrumentation, the processes which shall be subject of automated testing, the type of test stimulation, optimisation options and environmental dependencies like paths. Such information may be given by lists for a number of host and target configurations.

The CPT and the configuration options are passed to ISG which then starts the generation process, the (distributed) execution of the system and the evaluation of the results.

2.4 UDF - The User and Application Interface of ISG

A "User-Defined Function" is the standard interface through which a user can plug-in sequential code like algorithms for data processing. A UDF is correlated with each atomic action of a state transition and just its name is defined by the CPT together with the mode of its generation: to be generated by the system or by the user, or a mix of both. In any case, it is ensured that user code is not lost if a new version is generated automatically by the ISG toolset.

On request, ISG generates an instrumented stub which allows for a complete and executable system right from the beginning. Alternatively, a user may provide UDF's as stubs or they may include the full functionality. Also, a user may take an automatically generated stub and refine it towards the desired functionality.

2.5 Automated Generation of the DTDC Software

The DTDC software is automatically derived from three more datasheets (see [5]) which

1. describe the data provided by the hardware
   e.g. interfaces, the derived housekeeping data and their grouping into telemetry frames,
2. define the allocation of data
   e.g. data type, amount of data, memory type (RAM, NVRAM, ROM)
3. define the templates for data calibration and data post-processing, and
4. some more properties.

From this information the database structure, the software for database access, update and information retrieval, the software for generation of the telemetry frames, the data processing functions, and the interface software for data acquisition (like generation of lookup tables) are automatically generated. The databases of the two processors are automatically updated.

2.6 The ISG Concept of Automated Reporting

ISG does not only address automated code generation from high-level engineering information, but also automated verification and validation. This implies automated instrumentation of the source code such that sufficient information can be provided for automated evaluation.

ISG provides the following reports:

- graphical diagrams on data flow
  MSC, timing diagram

- coverage reports
  -- coverage of atomic actions and a list of non-covered atomic actions
  -- coverage of states and a list of non-covered states
  -- executed state transitions
  -- exception report
  -- error injection report
- performance reports
  -- CPU utilisation for all allocated nodes / CPU's
  -- timer report, load calibration report
  -- network utilisation report
  -- response time report
  -- buffer utilisation report
- information on sizing and timing budgets e.g.
  -- usage of malloc
  -- CPU utilisation per process instance, overall utilisation etc.

After each run this information is immediately available and allows for an analysis right from the beginning whether the required system properties are met or not.

There are cases for which an application-independent answer can be given like for coverage of states or actions, or robustness against faults. E.g. a figure of 100% state coverage tells an engineer in any case that all states have been executed, this is a property by which the evaluation facility can automatically conclude on whether the test was successful or not.

3. EARLY VALIDATION AND V&V

The following principal verification and validation steps were performed for MSL:
- visual checking of the CPT contents
- check of consistency and completeness (check of CPT regarding syntax and semantics) before generation of the next version
- checks during (normal) execution of the generated software by automated stimulation
- automated stimulation of all processes of the system (generation of high CPU load = stress testing mode 1)
- automated error injection for all processes of the system (= stress testing mode 2)

The potential errors are classified into five categories:
- errors detected by visual checks and tool support at pre-run-time
- errors detected by execution due to error messages at run-time
- errors detected at post-run-time by coverage analysis
- potential problems identified by high CPU load
- potential problems identified by fault injection.

Regarding the high complexity of the MSL software the number of identified errors in the CPT was relatively low.

The type and number of errors found at pre-run-time, run-time and post-run-time should be an indication of how well the set up of the system definition can really be done. Actually, no very serious errors were detected. About 60% of the errors were related to copy-paste-modify operations on CPT lines. A minor part was related to the complexity of the required behaviour, e.g. when more than one process level was involved, when a process receiving a command has to involve another process before it can respond. Other sources of errors have been: it was just forgotten to respond or a wrong destination (process, instance) was specified.

The V&V process started when the first complete version of the CPT was received from the MSL project. By visual checks its contents was analysed and suggestions were made to improve it. Then all errors and warnings which were recognised by the checking utility were removed and a first version of MSL software was automatically generated and executed in the Unix environment. The detected faults were discussed with the MSL project and the CPT was updated accordingly until a full coverage of 100% of CPT lines, states and the input domain was achieved and first figures for data trafiic, CPU utilisation and timing constrainsts were available.

Then the MSL software for the SPLC target was generated. The V&V steps were repeated and first figures of the target budget were obtained which showed that it should be possible to meet the performance constraints. However, it also became clear that no big margin is available and that the budget has carefully to be monitored during the following refinement steps.

Based on the first feedback for the budgets and further optimisation during the following refinement, the current CPT looks rather different from its first version. The initialisation procedure was completely changed and six processes were

removed. Actually, it is expected that the sizing budget will be met. For the timing budget no final conclusion is currently possible, although there is a good chance to succeed.

The development of hardware-related software and firmware was performed in parallel to the activities described above. This software was integrated stepwise into the overall environment as generated by ISG replacing the stubs by the real UDFs.

4. COMPARISON OF ISG WITH OTHER DEVELOPMENT APPROACHES

Compared with other software development approaches ISG needs a minimum of inputs to provide an executable, the environment for execution of a distributed system, and the facilities for reporting as required for V&V. This implies the use of standard components, templates and interfaces which allow to cover the variety of process structures, toplogies and fault-tolerant properties.

To achieve this high degree of automation and the large ratio between generated outputs and user-provided inputs[2] an extension of the object-oriented paradigm was necessary. ISG basically operates on construction rules rather than on classes. In the context of ISG a class is a specific case which is covered by the ISG construction rules. Hence, the ISG concept allows to cover structural differences between classes by a unified approach not requiring manual intervention for specialisation. Moreover, ISG addresses the whole lifecycle by a harmonised and automated, incremental approach.

4.1 ISG vs. the V-Model

The lifecycle of ISG differs significantly from that of the well-known V-model (Fig. 3) for the following reasons:

In the V-model the software is implemented by the lifecycle phases "specification", "design", "coding", "testing", and "integration". The acceptance of the software by validation takes place at the very end.

In case of ISG the first three phases are replaced by a number of small refinement steps and the remaining steps of the V-model are integrated into these steps. This integration implies an automation of the manually executed steps as needed for the V-model.

Consequently, an immediate feedback from the executable system is available right from the beginning while in case of the V-model a first feedback is not available before start of the testing phase (see Fig. 3, "feedback").

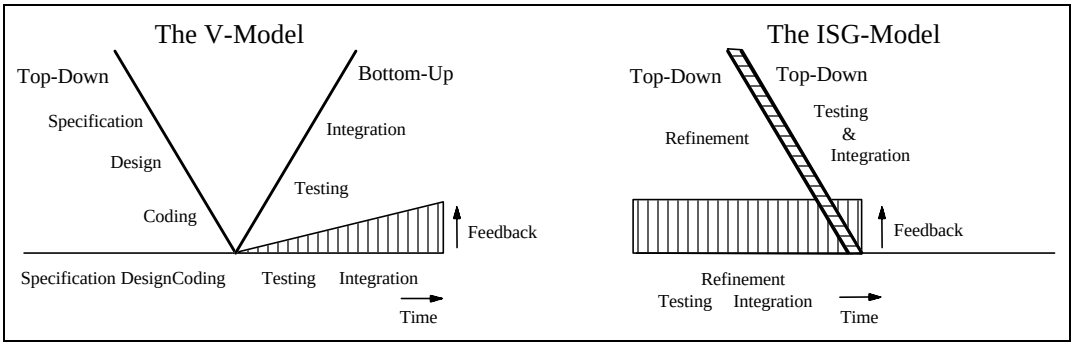

Fig. 3: ISG vs. the V-Model

The "top-down" - "bottom-up" implementation procedure of the V-model is replaced by "top-down" refinement of the software in case of ISG. A major advantage is that integration can start right from the beginning leading to "top-down" integration. A simple prototype can smoothly be extended towards the final version, the difference between "(rapid) prototyping" and "real implementation" disappears. With ISG there is "rapid prototyping" at the beginning and the prototype is refined towards the final version when acceptable. If not, a prototype can easily be dropped as it does not cost very much.

ISG combines behaviour with performance constraints, design and coding with V&V. This requires a different kind of thinking when starting with ISG: (1) ISG requires everything (but not more) what is needed to guarantee that the user inputs specify something which is really executable, (2) ISG guides the user to provide what is needed and rejects what is erroneous and inconsistent.

4.2 UML and ISG

UML [7] embeds the current various software development techniques into a unified approach. However, it only concentrates on the object-oriented aspects of software development, neglecting other important features like performance, verification and validation.

---

[2] Currently a preliminary figure of about 100 was observed.

The main intention of UML is to provide a complete environment together with guidelines by which an engineer can develop the software by the well-known lifecycle phases. However, this implies late system validation.

The intention of ISG is different. It addresses early validation, and hence it aims to get an executable as soon as possible to be able to confirm the required properties. This is the reason why most of the steps and activities as recommended by UML are not required by ISG because such steps are automated and are silently done by the ISG toolset. UML tools give an engineer a feedback by documentation while ISG provides a feedback by execution.

As UML does not support provision of all the inputs as required to generate an executable and to validate all its properties, it is currently impossible to use UML toolsets as front-end for ISG. However, solutions to this problem are possible and are currently discussed with potential customers.

4.3 ISG and N-Version Programming

ISG offers an alternative to "N-version programming" which is called "N-platform V&V" (see also 5.3.2 below). This relates to observations that hidden and dormant bugs may be a matter of the hardware and the operating system (OS) platform. So they may be not detected on one platform while they will on another one, because each platform acts as a filter which is (usually) only sensitive for certain types of bugs but not for all.

With ISG a change of the platform is very easy because just an option in a configuration file needs to be changed. The probability to detect hidden or dormant faults is significantly increased when the hardware architectures (like for Intel and Sparc) or OS concepts are different (like for Unix and VxWorks).

The aim of N-version programming is to identify programming bugs and specification and design errors by comparing the results of N equivalent programs. In context of ISG the immediate feedback and visualisation of results ease the detection of specification and design errors. In addition, the capability of "N-platform V&V" provides the capability to identify dormant bugs, a feature which is not covered by N-version programming. With ISG executables can easily be generated for different platforms, getting a reasonable chance that a dormant bug will wake up.

5. LESSONS LEARNED

A number of unexpected problems had to be solved during the activities for the MSL project.

5.1 Add-On's for the SPLC/VxWorks Platform

It turned out that the effort and time for porting of ISG from the Sparc/Solaris [11] to the SPLC/VxWorks platform was underestimated (eight weeks instead of planned two weeks). Most of the problems were related to missing support for visualisation of what the software does and for high-resolution timers.

A number of add-on's were needed in order to get the same visibility for debugging and reporting as a Unix platform provides by default. Such add-on's are now provided by the ITAP (Instrumentation, Tracing and Analysis) package [12] which is independent of the ISG environment.

E.g. for accurate performance analysis a higher time resolution than the tick of 1/60 s is needed. Firstly, for accurate measurement of the CPU consumption, secondly, to generate scheduling delays (phase shifts) smaller than the time tick.

Another feature supported by ITAP is the provision of independent output channels which allows tracing of messages up to the very last statement executed before a crash (see also 5.3.2).

5.2 Resource Utilisation

A first analysis of the sizing budget showed that the size of the memory allocated by the application has to be reduced. This problem was raised by the high number of process tpyes (about 40), which acts as a large multiplier for each instanciated byte of code, and the limited size of the memory on SPLC (8 MB). Hence, the ISG code generation was tuned towards a higher number of generic functions and about 1 MB could be saved. However, this feature is only available for the pure C version and not for the SDL/C version because full control of code generation is needed for this optimisation. Similarly, it is for the stacks and the communication buffers. However, only by reduction of the number of process types memory can be saved and it has been reduced now to about 30 process types.

The timing budget is still considered as critical and saving of CPU consumption has to be discussed from a "high-level" operational point of view. The potential contribution from the infrastructure is marginal. Since integration of the database and data acquisition functions and running of a test scenario the budget is exceeded. The interesting question is how "representative" is the current test scenario? It is known that the test scenario does more than the later operational scenario, but also some functionality is not yet integrated. It seems that some functionality planned for onboard execution like data processing prior to generation of telemetry frames needes to be moved to ground.

While the sizing budget stabilised rather soon at a non-critical level, the timing budget did not. A first experience is that more effort must be spent to define a scenario which is sufficiently representative and allows for more checks and deeper analysis of performance impacts.

5.3 Detection of Bugs

"Compared with Ada bug removal in a C environment is still exciting and challenging!"

5.3.1 The Impact of a Shared Address Space

In the Unix environment most of the ISG software could already be tested. Only such parts which were directly related to VxWorks had to be tested on the VxWorks/SPLC platform separately.

However, these "few" additional tests took more resources than testing of the complete ISG environment on Unix due to limited visibility, higher complexity of the software structure and system crashes which prevented post-mortem analysis. In case of VxWorks most bugs cause such a crash because the address space is shared between the kernel and the application processes, while it is not in case of Unix. Hence a corruption of memory usually causes a crash of the whole system.

The shared address space also requires modifications for initialisation, memory allocation, termination, commanding, instrumentation and reporting.

5.3.2 The Benefit of Porting Software

The ISG source code was carefully tested on each of the platforms (Intel, Sparc, Solaris, Linux, VxWorks) and compilation was done with all the relevant checks (errors and warnings) switched on. Nevertheless, not all bugs (mainly segmentation faults) could be detected, because some bugs were hidden or dormant on a certain platform.

The reason, why mostly crashes related to "segmentation fault" occur, is that all other bugs can be well detected by a C compiler at pre-run-time. But bugs related to memory access are not detected.

When the code was ported from Unix/Solaris to VxWorks, some bugs were detected on the VxWorks platform which were also of relevance for the Unix/Solaris platform. When porting the software from Solaris to Linux still some more bugs were detected which were common to all platforms. This happened although the same code was executed on each of the platforms and always a GNU compiler was used.

The bugs could not be detected because there was no impact on the results. E.g. overwriting of a leading byte of a pointer of value zero by zero does not cause any problem. However, it causes a problem when the partitioning of the address space is different and the contents of the leading byte changes e.g. to 0xff.

Therefore a remarkable result is that porting of code to different platforms and V&V on different platforms ("N-platform V&V") is very helpful to identify hidden or dormant bugs and to achieve a higher detection rate of bugs. ISG is inherently supporting such an approach because just by changing a configuration option another executable is provided for the desired platform within a rather short time (just by changing literals of the processors).

5.3.3 And the Drawbacks ...

A major - unavoidable - drawback of above observation is that a reasonable chance exists that software tested on a certain target platform will not run on another platform immediately. E.g. it seems that the probability of detecting a bug related to memory access is higher for a small ratio of memory size to process size. When a large amount of the memory is not used the chance to corrupt memory which is actually used by the application is smaller. Hence, a number of bugs were detected on the SPLC platform (ratio of about 2), but not during previous tests on the other platforms (ratio about 30).

As it is rather expensive to identify such bugs BSSE will provide further means for easier detection of the location where the corruption of data occured in case of wrong indexing or wrong pointers.

A similar problem as for the targets was identified for the host platforms. ISG implements a high degree of automation and therefore it depends heavily on the host configuration. Deviations from the expected/requested system configuration may cause a complete stop of the ISG generation chain. Therefore additional effort was needed to master such problems. Examples are: (1) only one compiler path may be included in the system path (either host or native compiler), but both compilers are needed during a continuously executed generation stream, (2) the format of the ps-command changes and therefore the required information about a process cannot be obtained (access of the wrong piece of information).

5.3.4 The User Interface

The user interface was subject of several improvements as identified by the MSL project. The MSL users suggested to improve naming conventions, to introduce pre-defined configuration sets, and to support more automation and shorter turn-around cycles. Also, an analysis of user errors was done to identify potential weakness of the ISG user interface and - when reasonable- corrective actions were taken.

6. CONCLUSIONS

Having performed a first exercise with ISG for the MSL project the following conclusions are drawn.

6.1 Current Achievements

The exercise performed in the context of the MSL project demonstrates that an automated approach can be successfully applied to such a rather complex system like MSL.

An input scheme was defined which formalises the system definition It allows to apply a process model which completely automates the generation and evaluation steps and allows for incremental development and early validation.. The formalisation is a pre-condition for the achieved degree of automation. Another necessary condition for automation is the extension of the object-oriented class concept to a concept which is based on construction rules.

A remarkable result is that errors in the system definition could be identified without being familiar with the engineering and operational aspects of MSL due to the formalisation of the software definition.

The feedback obtained from MSL and the discussions with potential users identified a number of desired improvements. Also, some bugs of the ISG implementation were found by the users, when they fully applied all the available features because it was the first test of ISG in practice.

6.2 Outlook

6.2.1 Extensions of the ISG Functionality

The MSL project identified further needs for high-resolution and synchronised scheduling within the processor network and the required features will be provided by ISG during the next months. These features address pre-defined scheduling events at time periods smaller than the tick of VxWorks and fixed correlation of task scheduling between tasks running either on the same or on different processors.

6.2.2 Higher Degree of Automation of V&V

During the first activities with ISG it was observed that a number of checks of system properties can be put in a standard form which is application-independent. This allows for automatic checks whether the required properties are preserved or not. Therefore a future goal is to transform all checks into a standard form so that an automaton can conclude on a system's properties.

6.2.3 Extended User Interface

Potential users have asked whether the system engineering inputs can be provided by other data formats. This is possible and there might be customised extensions which allow to exract the inputs e.g. from ECSS documents directly, from UML tools or from other representations a user is familiar with.

REFERENCES

[1] The Material Science Laboratory (MSL), Kayser-Threde GmbH, Munich, Germany, ESTEC contract

[2] R.Gerlich, V.Debus, Ch.Schaffer, Y.Tanurhan: EaSyVaDe: Early Validation of System Design by Behavioural Simulation, ESTEC 3rd Workshop on "Simulators for European Space Programmes" Noordwijk, November 15-17, 1994

[3] EaSySim II, 1996-1999, Rainer Gerlich BSSE System and Software Engineering, Auf dem Ruhbuehl 181, D-88090 Immenstaad, Germany

[4] ISG, 1999-2000, Rainer Gerlich BSSE System and Software Engineering

[5] M. Birk, U.Brammer, M.Ziegler, K. Lattner, R. Gerlich, Software Development for MSL on ISS by Automated Generation of Real-Time Software from Datasheet-based Inputs (this conference, same session)

[6] Excel and Access are trademarks of Microsoft

[7] Unified Modelling Language (UML), http://www.rational.com/uml

[8] Rhapsody, i-LOGIX Inc., 3 Riverside Drive, Anover research Park, Andover, MA 01810, USA

[9] Behavioural Validation of MSL, ESTEC contract no. 13309/98/NL/MV, Noordwijk, The Netherlands

[10] TORNADO / VxWorks, WindRiver Systems, Inc. 1010 Atlantic Avenue, Alameda, CA 94501-1153, USA

[11] Solaris is a trademark of SunSoft Inc. 2550 Garcia Avenue, Mountain View, CA 94043, USA

[12] ITAP (Instrumentation, Tracing and Analysis Package), 1999-2000, Rainer Gerlich BSSE System and Software Engineering

The implementation of ISG was funded in part by CRISYS (Critical Instrumentation and Control System) ESPRIT project EP 25514, 1997-2000