



DELIVERABLE D2.1

PART I: GENERAL REQUIREMENTS AND DETAILED PLANNING FOR PROGRAMMING ENVIRONMENT

WP2 Grid Application Programming Environment

Document Filename:	CG-2-D2.1-0005-SRS.doc
Work package:	WP2 Grid Application Programming Environment
Partner(s):	CYFRONET, FZK, USTUTT, TUM, UCY, CSIC, U.S.C
Lead Partner:	FZK
Config ID:	CG-2-D2.1-0005-SRS-1.3
Document classification:	PUBLIC

Abstract: This report describes the basic requirements for the CrossGrid application programming environment. The environment consists of three major components:

- an MPI code debugging and verification tool, which detects and reports erroneous use of MPI routines,
- a Grid benchmark suite, which allows to investigate experimentally the performance of Grid configurations,
- a performance evaluation tool, which supports the measurement and the prediction of an application's performance on the Grid.

According to this structure, the present first part of this report on "general requirements and detailed planning" is complemented by three additional documents (parts II-IV) that provide a detailed requirements specification for the respective components of the programming environment.



Delivery Slip

	Name	Partner	Date	Signature
From	Roland Wismüller	TUM	10.05.2002	
Verified by	N. Meyer, P. Wolniewicz, R. M. de Llano	PSNC, U. Cantabria	21.05.2002	
Approved by				

Document Log

Version	Date	Summary of changes	Author
1.0-DRAFT_A	10.05.2002	First Draft	R. Wismüller
1.1	24.05.2002	Footers + changes suggested by Jose Marco and review team	R. Wismüller
1.2	29.05.2002	Added state-of-the-art section	W. Funika, R. Wismüller, F. Rivera, M. Müller
1.3	02.06.2002	Final layout	H. Marten

1. OVERVIEW

This document specifies the software requirements for a programming environment supporting the development of interactive Grid applications. Particular emphasis is placed on the specific requirements posed by the CrossGrid applications developed in WP1. In addition, the document already presents first design considerations for the programming environment.

The environment consists of three main components:

1. An MPI code debugging and verification tool. This tool will perform semantical checks on all MPI calls issued by an application during run-time. It will detect and report errors and possible portability problems as well as uncommon (and thus most probably incorrect) usage of MPI routines. The tool will be implemented as an additional library that has to be linked to the application.
2. A Grid benchmark suite. It will contain three different classes of benchmarks:
 - Micro-benchmarks examine single, very specific performance aspects of a Grid configuration (hardware and middleware),
 - Micro-kernels perform a kind of "stress-test", i.e. they determine how well several aspects of performance can be met at the same time,
 - Application kernels provide results representative for a given application or class of applications.

The results of these benchmarks will be used in various ways by different entities:

- By the application programmer (and/or user) to find out how well a given Grid configuration is suited for his application.
 - By the programming environment: Benchmarks provide reference data for high-level analysis as well as parameters for performance prediction.
 - By the Grid middleware: Resource managers and schedulers will use benchmark data to determine the best suited resources for (parts of) an application.
3. A performance evaluation tool. Like other state-of-the-art performance tools it will allow to measure a selection of predefined performance metrics in a running application. However, the metrics will be specifically adapted to the needs of the CrossGrid applications (e.g. by considering disk access and secondary storage). In addition, the performance evaluation tool will include components for
 - high-level analysis: Using a specification language for performance metrics, this component will allow an application programmer to define application specific metrics. In addition, there will be a standard library of metrics definitions, which helps detecting common performance problems, such as load imbalance.
 - performance prediction: This component is based on analytical models of application kernels and complements the measurement-based parts of the tool with the capability to answer "what-if" questions like "what would be the application's runtime if this network link were twice as fast?"

2. STATE OF THE ART

2.1. MPI DEBUGGING AND VERIFICATION

The only reasonable programming model for distributed simulation is message passing. A standard for message passing has been defined and adopted by industry (MPI). Due to the complexity of parallel programming there is a clear need for debugging of MPI programs that has been addressed in two different ways:

- Classical debuggers have been extended to address MPI programs, this is done by attaching the debugger to all processes of the MPI program. On the commercial market there is currently only one product available that is portable across many platforms (TotalView from Etnus [1]).
- Some MPI libraries include a debug version of the library (e.g. MPICH [2]). This version is used to catch internal errors in the MPI library. It also detects some incorrect usage of MPI by the user.

Neither of these approaches address portability or reproducibility, two of the major problems when programming MPI.

2.2. GRID BENCHMARKS

Benchmarking is by no means a new concept. In fact, it has at least been around since the late 70's with some FORTRAN loops and pseudo applications [Dongara] that tried to measure how good a computer was at performing *similar* tasks. The methodologies and metrics currently employed are not that radically different, although they may be more accurate.

There is an abundance of benchmarking proposals, such as the NAS Parallel Benchmarks and implementations but few, if any, are targeted at Grids. This abundance of code, even of non-grid-targeted benchmarking tools, is by no means useless. In fact, some of these will surely form the starting point for the development of Grid-targeted benchmarks. Thus, most pertinent and available (open-source) benchmarks are in the process of being investigated. A very high-level summary of the results of this investigation forms this “*benchmarking state-of-the-art*” section.

The NAS Parallel Benchmarks (NPB) are a set of 8 programs designed to help evaluate the performance of parallel supercomputers [3]. They consist of several parallel kernels and several simulated application benchmarks. Some of the kernels employed include embarrassingly parallel benchmarks, simple 3D multigrid benchmarks, solving an unstructured sparse linear system by the conjugate gradient method, 3D fast-Fourier transform partial differential equation benchmarks and parallel sort over small integers. Also included in the benchmarks are, an LU solver, a pentadiagonal solver, and a block diagonal solver. The benchmarks are based on MPI. The integer sorting benchmark is written in C and the rest in FORTRAN.

The SPLASH parallel benchmarks are made up of kernels and applications. The latest version SPLASH-2 has more applications and enhanced versions of several of the original SPLASH benchmarks. The SPLASH-2 suite of parallel applications has recently been released to facilitate the study of centralized and distributed shared-address-space multiprocessors of the original SPLASH programs [4]. They are all written in C apart from one FORTRAN application. The applicability of this type of benchmarks (i.e. shared-memory based) to the CG testbed relies on the extent to which software based shared memory architectures (such as PVM) will be used.

LLCBench (Low Level Characterization Benchmarks - <http://icl.cs.utk.edu/projects/llcbench/>) is comprised of MPBench, CacheBench and BlasBench.

MPBench is a benchmark to evaluate the performance of MPI and PVM on MPPs and clusters of workstations. It uses a flexible and portable framework to allow benchmarking of any message passing

layer with similar send and receive semantics [6]. MPBench generates reports with raw data and Postscript graphs. MPBench currently tests several different communication aspects: Bandwidth, Roundtrip, Latency, Broadcast, Reduce and All-Reduce. These are measured for both MPI and PVM (even though PVM may not provide explicit calls for these). MPBench contains very simple, understandable and essential tools for evaluating the performance of MPI.

CacheBench is designed to evaluate the performance of the memory hierarchy of computer systems. [5] Its specific focus is to parameterize the performance of possibly multiple levels of cache present. Since CacheBench measures memory bandwidth of microprocessors it may only be applicable for the measurement of Computing Elements.

BlasBench is designed to evaluate the performance of some kernel operations of different implementations of the BLAS or Basic Linear Algebra Subroutines [7]. The BLAS were initially developed as part of LINPACK (see below). The BLAS routines form the basis of many scientific applications therefore benchmarks based on BLAS will prove useful.

The Linpack/ScaLAPACK benchmarks are quite well known (especially Linpack). Original Linpack was based on fixed-size matrix transformations, which is one of the reasons why it has been adopted as *the* benchmark for the TOP500 rating for supercomputers. The distributed memory version of it (High Performance Linpack) is actually what ranks the TOP500. It has been successfully run on grid infrastructures using MPICH-G2. ScaLAPACK (or Scalable LAPACK) library includes a subset of LAPACK routines redesigned for distributed memory MIMD parallel computers [9]. ScaLAPACK is designed for heterogeneous computing and is portable on any computer that supports MPI or PVM.

PARKBENCH with its 21 provided benchmark applications was intended to close the gap that existed for Massively Parallel Machines [8] although it does contain some serial codes. Low-level benchmarks include Timer resolution of individual machines, basic arithmetic operations, memory bottleneck benchmarks. On the network side it provides communication benchmarks (ping-pong, "duplex transfers"), total saturation bandwidth, communication bottleneck and synchronization benchmarks (barriers etc.). PARKBENCH then provides several Matrix-kernel benchmarks, as well as 3D FFT and other benchmarks some of which originate from the already mentioned NPB. Most benchmarks are written in FORTRAN and their current release utilizes MPI.

Genesis is a suite of distributed-memory benchmarks that aim to evaluate distributed-memory MIMD systems [10]. Based on previous versions of *Genesis* the latest versions of this benchmark suite were augmented and modified in order to evaluate the specific computers.

From the review of existing benchmarking suites one quickly concludes that there is a distinct lack of Grid-targeted benchmarking. An exception is a Linpack port to MPICH-G2, although it aims to compare MPICH to MPICH-G2 rather than evaluate Grid performance. One can also see the opportunity to build on top of the existing benchmark suites adapting the code as required and apply them to Grids. Benchmarks based on computations included with NPB and PARKBENCH will be quite useful, as some WP1 applications will have similar computations. An MPICH-G2 port of selected MPBench benchmarks will prove quite valuable (first indications show the porting will be quite straightforward) for evaluating the effectiveness of Grid configurations with regard to MPI. Tools utilizing other-than-MPI technologies (such as Genesis that uses the PARAMACS macros) are in the process of being ported to MPI and others can be ported. There is definitely room for a well-structured, concise and easy to use benchmark suite that captures the performance of a Grid.

2.3. PERFORMANCE MEASUREMENT AND EVALUATION

Performance evaluation, instrumentation, prediction and visualization of parallel codes has been found to be a complex multidimensional problem [11] in parallel and distributed systems. This situation is specially critical in GRID environments. Tuning the performance of codes on distributed memory systems has been a high time-consuming task for users. When programming these systems, the reasons

for poor performance of parallel message-passing and data parallel codes can be varied and complex, and users need to be able to understand and correct performance problems in order to achieve good results. This is especially relevant when high level libraries and programming languages are used to implement parallel codes. A performance data collection, analysis, prediction and visualization environment is needed to detect the effects of architectural and system software variations.

Since there already exist several recent reviews of existing **performance analysis** tools, prepared in the framework of the DataGrid project [12], the American NHSE initiative [13], and the APART project [14] [15], this section will not repeat these reviews, but rather provide a relatively short summary, focusing on features relevant within the CrossGrid project.

There is already a reasonable number of performance measurement tools that support MPI. However, as [13] clearly shows, none of them actually supports Grid applications at the moment. Although such a support might be added with reasonable effort, the existing tools (with the exception of *Paradyn*) are not suited for the interactive Grid applications developed in CrossGrid, since they work only in off-line mode, i.e. they cannot provide performance information while an application is still running. However, this is a vital requirement for the CrossGrid applications. E.g. in the medical application (Task 1.1), the simulation in total may run for a very long time, however, the developer will not be interested in performance data summarizing the whole run-time, but rather in the performance of the part of the simulation, which has been requested by the current user interaction. Relating these user interactions with performance data is only possible with on-line analysis and visualization. A second drawback of off-line tools is that the user of the tool cannot change the type of data to be measured during the application's run-time. Thus, typically much more data is recorded than is actually needed. This can result in huge tracefiles generated on each node used by the application, which must then be collected on a single workstation in order to be analyzed. For large and long-running Grid application this procedure gets impractical.

The number of existing on-line tools for application-oriented performance analysis in distributed systems is very small. *Paradyn*, which has been mentioned above, can measure the performance of large-scale parallel and distributed applications on SMPs and clusters of workstations. The unique features of *Paradyn* are its ability to instrument a program during run-time and an automatic facility for searching performance bottlenecks. However, the dynamic instrumentation technique used by *Paradyn* exhibits a very poor portability, since it modifies the application's machine code. Therefore, *Paradyn* is available only for selected platforms and is not a good option for use on the Grid. *Paradyn* also is the only tool we are aware of that makes use a metrics definition language [16]. However, it is hidden inside the tool and cannot be used to define application-specific metrics. A second on-line performance measurement tool is PATOP [18], which is used as a basis for the performance measurement and user interface components of the performance measurement tool developed in the CrossGrid project.

The current approaches to Grid performance monitoring (which also include analysis and visualization) are largely limited to the monitoring of the infrastructure, rather than the applications. The report [12] provides a detailed overview, which is briefly summarized here: *NetLogger* is a distributed logger that enables performance and bottleneck analysis, selecting hardware components for upgrade, and real-time and post-mortem analysis of applications. The *Heartbeat Monitor* is intended to monitor the states of processes on Globus, to notify process status exception events for recovery actions to be taken. *Network Weather Service* is a distributed system that periodically forecasts the performance of various networks and computational resources. *Autopilot* is a distributed performance measurement and resource control system complemented by *Virtue* that is an environment which accepts real-time data from Autopilot and allows the user to steer software behavior and resource policies. When used on the Grid, Virtue defines a visualization hierarchy for managing performance details. As already mentioned above, none of these systems provides support for performance analysis for Grid applications.

Within the DataGrid project, a performance analysis tool will be developed, that also supports application-oriented performance analysis [17]. This tool will be based on the R-GMA monitoring architecture, the GRM tracing library and the PROVE visualization tool. Although PROVE will operate in a semi on-line mode, i.e. it will read the trace files during the application's run-time, it can still create an impractical overhead when detailed measurements have to be done, since measurements cannot be defined or changed at run-time. In comparison to the performance tool that will be developed in CrossGrid, PROVE also offers rather limited functionality:

- It is focused on communication between processes and does not provide information on CPU usage, I/O etc.
- It does not provide performance information on the Grid infrastructure, which is necessary in order to assess the application's performance data.
- Although PROVE allows for application specific trace events, it can only visualize them in a process / time diagram, but cannot use them to derive any application-specific performance metrics.
- Finally, PROVE does neither include user-definable performance metrics nor performance prediction capabilities.

Although the CrossGrid performance analysis tool does not include an automatic search for performance bottlenecks, it will use some techniques developed in this context, most notably a specification language to define metrics. For a review of the state of the art in this field, the reader is referred to the report prepared by the APART working group [14][15].

To our best knowledge, there is no existing performance analysis tool that allows the user to define his own application-specific metrics. This feature is unique to the tool developed in CrossGrid and is vital for the support of applications as complex as the ones developed in Workpackage 1 of the project.

The a priori **performance prediction** of parallel and distributed codes becomes a hard problem. Sophisticated performance prediction tools are being developed by a number of groups. In particular we cite the PACE toolset [19], PERFORM and LEBEP [20], DIMEMAS [21], INSPIRE [22], Carnival [23], ALPES [24], P3T [25] and PerFore [26]. Other works used prediction as part of a software-aided approach for particular applications, e.g. solvers - we cite SPEED [27], SciRUN [28] or PARAISO [29]. A long-term project is the development of AlpStone [30], that simulates parallel performance using information from a database of kernel benchmarks and underlying hardware parameters using a "skeleton application" approach. The kernel benchmarks are representative of "algorithm classes", and a "synthetic program" is built from these to simulate the real application.

The application of these techniques has so far been largely restricted to the accurate performance prediction of code kernels with the goal of automatic parallelization or execution steering. And all of them focus on specific parallel systems, at most on clusters of workstations, like DIMEMAS or Carnival, but to our knowledge, no one is currently adapted to GRID environments. DAMIEN [31] is a project to do that for DIMEMAS. On the other hand, most of these tools require simulation of the codes, that takes much time, also they are not application-dependent, and some of them are for specific programming platforms, P3T is for High Performance Fortran programs, AlpStone and Carnival are for PVM programs, PERFORM is just for sequential systems, and PerFore is integrated in a specific compiler.

The proposal of our prediction model is application-dependent, and focuses on some kernels from the applications of WP1. In this way, the result will be more accurate, and as the performance will be modeled by simple analytical expressions, it is very fast. We will obtain a model for each MPI-coded kernel on the GRID.

Once the performance prediction models are established, we will develop a visualization tool according to them. An important feature is that the information will be shown into this interactive visual tool to be user-friendly for users or developers of the application, using the analytical models,

no trace files are used, as the performance prediction does not generate them. In addition, this tool can include detailed information about specific parameters of the codes, as well as the predicted information about the execution of these codes. In fact, it should be an interface that allows the user to interact in the analysis of the behaviour of the codes under different conditions (number of processors, different distributions, different input data, ...).

2.4. REFERENCES

- [1] Etnus: TotalView. <http://www.etnus.com/Products/TotalView>
- [2] Argonne National Labs: MPICH – A Portable MPI Implementation.
<http://www-unix.mcs.anl.gov/mpi/mpich>
- [3] Bailey, D. H., E. Barszcz, J. Barton, D. Browning, R. Carter, L. Dagum, R. Fatoohi, S. Fineberg, P. Frederickson, T. Lasinski, R. Schreiber, H. Simon, V. Venkatakrishnan, and S. Weeratunga, "The NAS Parallel Benchmarks," Technical Report RNR-94-007, NASA Ames Research Center, (March 1994)
- [4] Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop Gupta. The SPLASH-2 Programs: Characterization and Methodological Considerations. In Proceedings of the 22nd International Symposium on Computer Architecture, pages 24--36, Santa Margherita Ligure, Italy, June 1995.
- [5] Phillip J. Mucci and Kevin London, "The CacheBench Report", 1998.
- [6] P. J. Mucci and K. London, "The MPBench Report." March 1998.
- [7] Phillip Mucci Kevin, "The BLASBench Report", 1998.
- [8] R. Hockney and M. Berry, Public International benchmarks for parallel computers report-1. Tech. Rep., Parkbench Committee, 1994.
- [9] L. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. Whaley, ScaLAPACK: A linear algebra library for message-passing computers, in "Proceedings of SIAM Conference on Parallel Processing" (1997).
- [10] C.A. Addison, V.S. Getov, A.J.G. Hey, R.W. Hockney, and I.C. Wolton. The Genesis Distributed-Memory Benchmarks. In J. Dongarra and W. Gentsch, editors, Computer Benchmarks, pages 257--271. Elsevier Science B.V., North-Holland, Amsterdam, 1993.
- [11] M. Simmons and R. Koskela. Performance Instrumentation and Visualization. ACM Press. 1990.
- [12] Z. Balaton, P. Kacsuk, N. Podhorsky, and F. Vajda. *Comparison of Representative Grid Monitoring Tools*. Computer and Automation Research Institute of the Hungarian Academy of Sciences, Budapest, Technical Report LPDS-2/2000.
<http://hepunix.rl.ac.uk/edg/wp3/meetings/budapest-jan01/NP%20Grid%20Tools.pdf>
- [13] S. Moore, D. Cronk, K. London, and J. Dongarra. Review of Performance Analysis Tools for MPI Parallel Programs. In Y. Cotronis and J. Dongarra (Eds.): *EuroPVM/MPI 2001*, LNCS 2131, pp. 241-248, 2001. Springer Verlag.
- [14] APART Working Group: Index of Base Modules and Related Work. Sections 9 and 11.
<http://www.fz-juelich.de/apart-1/wp3/modules.html>
- [15] APART Deliverable: Report on State-of-the-art in Automatic Performance Analysis. Working Group No. 29488
<http://www.fz-juelich.de/apart-1/internal/state-of-the-art.ps>

-
- [16] J.R. Hollingsworth, B.P. Miller, M.J.R. Concalves, Z.Xu, O.Nai, and L.Zheng. MDL: A Language and Compiler for Dynamic Program Instrumentation. In *Proc. International Conference on Parallel Architectures and Compilation Techniques*. San Francisco, USA, Nov. 1997.
ftp://grilled.wisc.edu/technical_papers/mdl.ps.gz
- [17] DataGrid. Information and Monitoring (WP3) Architecture Report. DataGrid Deliverable D3.2
<http://150.146.1.82/grid/internal/Deliverables/D3.2/DataGrid-03-D3.2-0101-1-0.pdf>
- [18] M. Bubak, W. Funika, B. Balis, and R. Wismüller. On-line OCM-Based Tool Support for Parallel Applications. In Y.C. Kwong (Ed.): *Annual Report of Scalable Computing*, Chapter 2, pp. 32-62, 2001. World Scientific Publishing.
- [19] D.J. Kerbyson, E. Papaefstathiou, J.S. Harper, S.C. Perry and G.R. Nudd. Is Predictive Tracing too late for HPC Users? In R.J. Allan, M.F. Guest, D.S. Henty, D. Nicole and A.D. Simpson (eds.): *High Performance Computing -- Proc. HPCI'98 Conference 1998*. pp 57-67. 1999. Plenum/Kluwer Publishing.
- [20] T. Hey, A. Dunlop and E. Hernandez. Realistic Parallel Performance Estimation. *Parallel Computing* 23. 1997. pp. 5.21.
- [21] DIMEMAS. <http://www.pallas.de/pages/dimemas.htm>
- [22] K. Kubota, K. Itakura, M. Sato and T. Boku. Practical Simulation of large-scale Parallel Programs and its Performance Analysis of the NAS Parallel Benchmarks. *Lecture Notes in Comp. Sci.* 1470 1998. pp. 244-54.
- [23] Carnival. <http://www.cs.rochester.edu/u/leblanc/prediction.html>
- [24] J.P. Kitajima, C. Tron and B. Plateau. ALPES: a Tool for Performance Evaluation of Parallel Programs. In J.J. Dongarra and B. Tourancheau (eds.): *Environments and Tools for Parallel Scientific Computing*. North-Holland. 1993. pp 213-28.
- [25] T. Fahringer. Estimating and Optimising Performance from Parallel Programs. *Special issue IEEE Computer* 28. 1995. pp. 47-56.
- [26] Perfore. <http://ParaMount.www.ecn.purdue.edu>
- [27] C.-C. Hui, M. Hamdi and I. Ahmad. SPEED: A Parallel Platform for Solving and Predicting the Performance of PDEs on Distributed Systems Concurrency: Practice and Experience 9. 1996. pp. 537-568.
- [28] M. Miller, C.D. Hansen and C.R. Johnson. Simulated Steering with SCIRun in a Distributed Environment. In *Applied Parallel Computing -- Proc. 4th International Workshop PARA'98*. LNCS 1541. Springer. 1998. pp366-376.
- [29] PARASO. <http://www.ac.usc.es/~paraiso>
- [30] AlpStone. <http://www.ifi.unibas.ch/generate.doc/English/Research/ParProg/alpstone/doc.html>
- [31] DAMIEN. <http://www.hlrs.de/organization/pds/projects/damien/>

3. INTEGRATION

The components of the Grid programming environment presented in Section 1 should be integrated as far as possible. In the following, we outline the mutual inter-relations between each pair of components:

- MPI debugging - Benchmarks

The MPI debugging tool will be used to ensure the correctness (with respect to MPI) and the portability of the benchmarks. Vice versa, the benchmarks can be used as a testcase for the MPI debugging tool. They will also be used to assess the tool's impact on an application's performance, although this is not critical, since the tool will only be used during application development, not at production run time.

- Benchmarks - Performance evaluation

The performance tool's measurement and high-level analysis components will be used to extract performance data (i.e. the benchmarks' results) from the running benchmark code. On the other hand, benchmark results are used as input for both the high-level analysis and the performance prediction components.

- Performance evaluation -- MPI debugging

These tools serve independent goals, but nevertheless they should be usable concurrently. This allows an application programmer to have only one development version, which can be used with both of these tools. Since the tools are independent, the performance evaluation tool can actually be used to optimize the MPI debugging tool, if there should be a need.

The technical process of integration and testing is visualized in Fig. 1. All prototypes developed in WP2 will first be installed at the local testbed at FZK. Here, the tools will be tested using the High Energy Physics application from WP1. This application has been chosen, since FZK is involved in its development. Severe errors will be passed back to the developers and should be corrected immediately. When the prototypes pass this test, they are packaged and installed in the whole CrossGrid testbed. Application developers from WP1 will then use the tools and provide feedback on suggested improvements, which will be collected, revised, and passed on to the tool developers for inclusion in the next prototypes. Fig. 1 also shows the timing (Mxx = project month xx) for the first prototypes. The timing for the next two prototypes will be as follows:

- Second prototype:

- **M22**: installation at development testbed. Since this prototype is no longer restricted to local environments, it can no longer be fully tested only at the FZK site. Instead, as suggested in deliverable D4.1, FZK will serve as the main site of a small development testbed.
- **M24**: release and report.
- **M26**: feedback collection.
- **M27**: progress report.

- Final prototype:

- **M34**: installation at development testbed.
- **M36**: demonstration, release and report

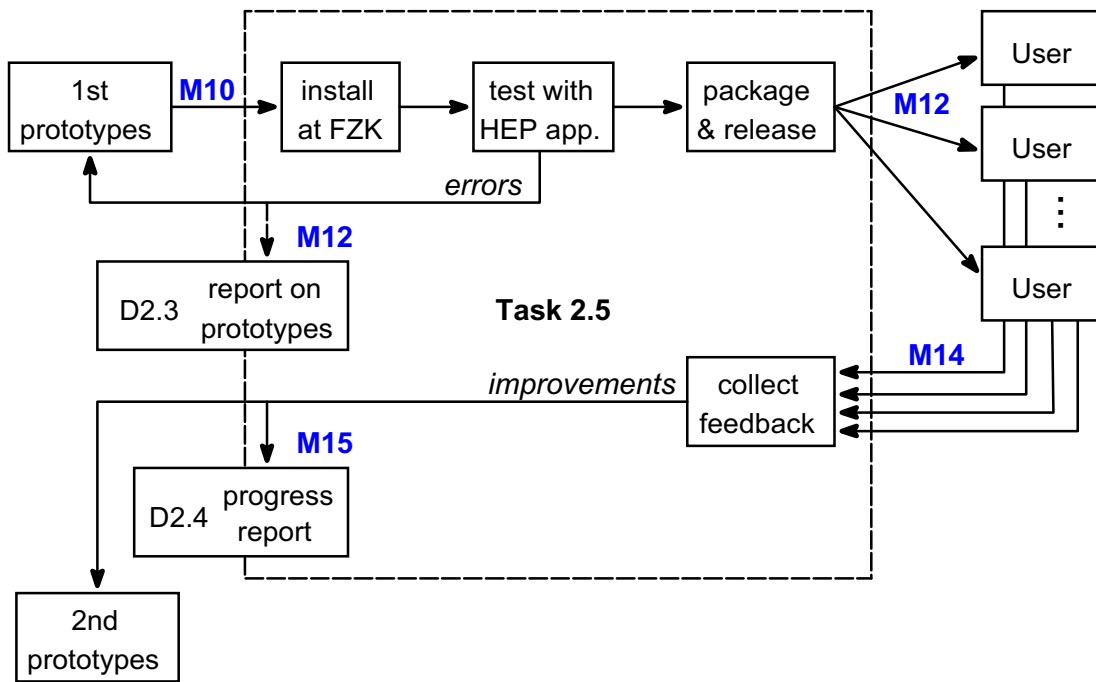


Figure 1: Integration and testing procedure

4. REQUIREMENTS SPECIFICATION

In order to clearly show their assignment to the different components of the programming environment, the software requirements have been specified separately for each of the three component in the following parts of this report. This procedure also made it easier to collect the requirements in a distributed way, thus reducing the management overhead in favor of the technical work.