EFDA–JET–PR(12)08

D. Dodt, N. Cook, D. McDonald, D. Harting, S. Pamela
and JET EFDA contributors

Improved Framework for the Maintenance of the JET Intershot Analysis Chain

Improved Framework for the Maintenance of the JET Intershot Analysis Chain

D. Dodt[1,2], N. Cook[1,3], D. McDonald[4], D. Harting[1,5], S. Pamela[1,6]
and JET EFDA contributors*

JET-EFDA, Culham Science Centre, OX14 3DB, Abingdon, UK

[1]JET-EFDA, Culham Science Centre, Abingdon, OX14 3DB, UK
[2]Max Planck Institut fürPlasmaphysik, EURATOM Association, D-85748 Garching, Germany.
[3]Tessela plc, 26 The Quadrant, Abingdon Science Park, Abingdon, Oxon OX14 3YS, UK
[4]EURATOM-CCFE Fusion Association, Culham Science Centre, OX14 3DB, Abingdon, OXON, UK
[5]Institut für Energieforschung Plasmaphysik, Forschungszentrum Jülich, TEC, Association EURATOM -FZJ, D-52425 Jülich, Germany
[6]Association EURATOM-CEA, CEA/DSM/IRFM, Cadarache 13108 Saint Paul Lez Durance, France
** See annex of F. Romanelli et al, "Overview of JET Results", (23rd IAEA Fusion Energy Conference, Daejon, Republic of Korea (2010)).*

Preprint of Paper to be submitted for publication in
Fusion Engineering and Design

ABSTRACT

At the JET experiment data from routine diagnostics is analysed automatically by a suite of codes within minutes after operation. The maintenance of these interdependent codes and the provision of a consistent state of the physics database over many experimental campaigns against a backdrop of continuous hardware and software updates, requires well defined maintenance and validation procedures. In this paper, the development of a new generation of maintenance tools using distributed version control and a work-flow following the principle of continuous integration [1] is described.

2. INTRODUCTION

The operation and facilitation of the JET experiment relies on the automatic processing of diagnostic and plant data after each pulse. Upgrades to these systems are often provided within projects carried out by the participating associations, before they are handed over as finished products to the operator.

2.1. PUBLIC DATA PHILOSOPHY

In order to guarantee transparency and availability of physics data to all members of the collaboration, high value is set on the provision of automatic analysis codes that produce public data. Accordingly, after every JET pulse a suite (Chain1) of around 90 analysis programs is run to provide data from the routinely available diagnostics. These analysis codes (called steps) are used to process raw data collected from around 60 different systems in diagnostics and plant. The raw data arrives at variable times after the pulse depending on importance and volume. A task scheduler program monitors the availability of raw and processed data as well as computing resources in order to optimise the Chain1 performance. See also [2, 3] for a more detailed description of the Chain1 infrastructure and its historical development.

2.2. AVAILABILITY OF THE INTERSHOT ANALYSIS

The operation of the JET experiment relies on the timely availability of diagnostic data and physics results to facilitate the planing of ongoing experimental sessions. Although the intershot analysis is not strictly operations critical, the execution of all but the most straight forward experiments benefits greatly from the results of the intershot analysis. Therefore, the availability of the analysis chain during experimental campaigns needs to be assured at all times.

2.3. INTERDEPENDENCE OF INTERSHOT ANALYSIS CODES

Reliance of analysis codes on data produced by other codes leads to complex interdependencies. In Figure 1, the dependencies of 13 important Chain1 codes are shown as an example. These dependencies have to be taken into account not only in determining the correct running sequence of the analysis chain, but also to maintain the consistency of data produced by Chain1 when codes need to be modified, e.g. to update calibrations. Therefore, Chain1 steps may have to be rerun frequently, in post-pulse reprocessing. The necessity of reprocessing leads to strict requirements

on the robustness of the codes, as any failure during reprocessing has to be investigated manually, to avoid an inconsistent state of the diagnostic database.

2.4. COLLABORATIVE CODE DEVELOPMENT MODEL

The maintenance and development of the analysis codes together with the infrastructure of the intershot chain involve a number of individuals.

Most analysis steps are maintained by a Responsible Officer (RO) who is typically also taking care of the hardware of the respective diagnostic. The ROs of Chain1 steps are responsible for the correctness of the produced data. They provide codes that can be used during operation, as well as for post-pulse reprocessing. A team of physicists and software engineers is responsible for the infrastructure and the consistency of the data produced by Chain1. The Chain1 support team helps step ROs with advice regarding development of best practices for specific problems such as performance optimisation and data volumes. In Figure 2 a work flow diagram illustrates a typical example of an intervention required for the maintenance of a Chain1 step. If the RO of a step is not available the support team provide basic maintainance for the analysis codes, where possible.

2.5. TRACEABILITY OF PHYSICS DATA

In order to ensure the reproducibility and traceability of the scientific conclusions drawn using JET derived data, it is necessary to record the versions of all codes and all input data that was used during a particular processing run. All processed physics data at JET is written to a global experiment database allowing universal access to all data by everyone. It is policy that for each public entry in this physics database, the means to produce it have to be identifiable. This additional requirement adds to the complexity of the maintenance procedures. The use of automated tools for procedures such as releasing codes and testing allows the generation of detailed logs, which are more complete and less prone to error than manually updated log files.

2.6. STRUCTURE OF THE INTERSHOT ANALYSIS CHAIN BEFORE THE UPDATE

The aforementioned requirements led to the current implementation of Chain1 which was used since the late 1990s without major design modifications. The analysis codes are wrapped by so called control scripts written in Perl [4] which set up the environment required by the code and are called by the task scheduler using a standardized syntax. Copies of the code binaries, input files and control scripts used intershot were kept in specific directories, constituting the so called live chain. Updates to Chain1 codes were tested manually for correctness and archived partly in CVS repositories and partly using scripts before they were manually released to the live Chain1 directories and entries added to log files. A configuration file was used to specify the interdependencies between the intershot codes for the task scheduler. The same file was used to manually determine the run assemblage during data reprocessing.

2.7. GROWING DIVERSIFICATION OF THE INTERSHOT ANALYSIS

With time, the growing number of available diagnostics and increasing demand for more complex data analysis lead to an increase of the number of codes that are part of Chain1. Similarly, the number of utilized programming languages has increased, especially in analysis packages delivered to the JET collaboration as part of upgrade projects carried out by the EFDA associations. Often, the ROs of these codes have strong preferences for the programming environment they use in their home association. Therefore, in some cases the provision of codes has been accepted as black box, because a reimplementation in a language more established at JET would have required significant resources. For these codes the Chain1 support team can only provide limited support. Altogether, these developments have lead to a rising complexity of the code base which sometimes lead to issues during the maintenance procedures, because the analysis codes do not correspond to the program structure envisaged during the original design of the Chain1 system.

3. IMPLEMENTED MODIFICATIONS

In order to account for the changed requirements and to reduce the amount of manual intervention required by the Chain1 support team, a number of modifications have been implemented. The new development of the Chain1 framework, as described below, allows the automation of a number of activities in the maintenance procedure for the individual steps as well as the infrastructure of Chain1 (e.g. the task scheduler). At the same time, improved traceability and an increased possibility of performing formalized tests on the modified software is achieved. In particular, the Chain1 infrastructure is made aware of the modification history of the interdependencies between Chain1 steps allowing the creation of powerful tools for the reprocessing of data which are not restricted to the current configuration. In the following sections, the modifications applied to the Chain1 framework are described.

3.1. DATABASE FOR TIME DEPENDENT STEP CONFIGURATION INFORMATION

Analysis steps often use some quantities for the analysis of diagnostic and plant data that vary with time and accordingly need to be specified as a function of pulse number. Furthermore, sometimes the interdependencies between Chain1 codes change, e.g. because the diagnostic of choice for certain plasma parameters has been replaced. It has proven to be advantageous to use a mechanism for the specification of pulse dependent configuration data that is transparent to the Chain1 framework and to use the same mechanism for the configuration of step interdependencies. This way, implementation duplication can be avoided and the integration of the configuration of an analysis step and the surrounding framework reduces the risk of inconsistencies.

A database has been implemented, allowing Chain1 ROs to specify configuration information. In view of the storage of step interdependencies, the top level module is called DependendcyDB. It uses ASCII files in different conveniently human readable formats as a back end. These are stored together with other data input files used by a step. The database allows the storage of key-value pairs

as well as the specification of versions of input files containing larger datasets such as calibrations that vary with time. Certain configuration variable keys are interpreted by the wider Chain1 framework. Currently, the complete deactivation (keyword disabled), and the exclusion from intershot running (not intershot) or automatic reprocessing (not reprocess) are recognized by the tools for the release and reprocess procedure. In addition, the database stores all configuration information that is passed to the task scheduler via the so called dependencies file, e.g. the dependencies of a step on raw data sources and other Chain1 codes as well as some additional options. Using this, the correct pulse dependent versions of the dependencies file can be generated, as required for live intershot running as well as in post-pulse reprocessing. This means, depending on what is required, an assemblage for the historical configuration, a maintained contemporary configuration or the current configuration of Chain1 can be automatically generated.

3.2. STEP PACKAGE INFORMATION AND CONTROL SCRIPTS

The control scripts wrapping each binary of the analysis steps serve a number of functions. The control scripts. . .

- accept a standardized syntax used to pass basic information like the pulse number and the ,userid used to write the JET diagnostic database.
- create a run directory, in which links are used to point to input files needed by the analysis step and where files are created to which the output (stdout and stderr) of the analysis programs is forwarded.
- prepare the shell environment (e.g. the LD LIBRARY PATH) for the analysis program and may pass command line options depending on pulse number and other conditions.
- finally create the rc-file summarizing the environment that was set up, as well as containing the time spent to run to the step and its return code indicating the success or failure of the analysis.

Before the current update to the Chain1 framework, the control scripts did not use a dedicated configuration database but the information, e.g. names of binaries and input files, was hard coded into the scripts. As part of the described development effort, the functionality of the control scripts has been reimplemented using Perl modules, greatly improving maintainability by avoiding a large amount of code duplications. The configuration information formerly contained within the control scripts has been stored into a database which is provided by a set of Perl modules called RepositoryDB using an xml file as the back end. In contrast to the DependendcyDB, here information is stored that generally does not change over time. The term repository refers to a set of source and input files that are stored in a git repository (see section 3.4) and provide parts of, one, or even several steps. A step in this narrower sense, is a piece of software that is called using a control script and that produces specified entries to the experiment database. The RepositoryDB is also used by tools facilitating the maintenance of Chain1, by e.g. allowing automatic cloning and building from

the git repository. Both Perl packages are provided with automatic units tests and Perl POD [5] documentation to ensure good maintainability.

3.3. NEW DEPLOYMENT MODEL

The Deployment model of Chain1 was modified in several aspects, primarily to allow the use of parallel versions of the analysis chain for testing, development and reprocessing. To enable this, all instances of absolute path names addressing components (input files, libraries, executables) had to be replaced by relative paths. Where necessary, the build scripts were supplemented to provide all compilations 'ready to run'.

To enable the automated cloning and building of Chain1 assemblages for development and reprocessing, build dependencies which are stored in the RepositoryDB. are used to ensure the correct sequence of the builds e.g. in cases, in which libraries are used across different git repositories, see Figure 3 for an example.

3.4. COLLABORATIVE DEVELOPMENT USING DISTRIBUTED VERSION CONTROL

The development history of all Chain1 steps has been migrated from CVS [6] to the distributed version control system git [7]. In contrast to CVS, the distributed approach allows fine grained well documented commits to be performed directly by the developer, improving traceability and avoiding the manual application of patches by the Chain1 support team. This is especially useful for codes which are delivered as finished products and not usually modified by members of the chain1 support team. Although the distributed approach provides the capability for the step ROs to use version control for their private development, the experience was made that some encouraging and training is needed before the provision of updates via git is widely accepted and preferred to manual patches. In any case, the manual provision of code updates is still possible in a similar way as before.

4. IMPROVED TOOLS RATIONALISING THE MAINTENANCE OF CHAIN1

The provision of transparent configuration information, as well as the modified deployment model enables the rationalisation of existing maintenance procedures, as well as the provision of new procedures, which aim at improving the reliability of the Chain1 operation.

4.1. TEST AND RELEASE PROCEDURE

Before modifications of steps provided by the ROs are released to the live intershot chain, their correct integration and execution needs to be tested. The use of an independent clone of the processing chain greatly simplifies this procedure and removes several historical failure causes from the test procedure. Modifications affecting more than a single step can be tested in a straight forward and consistent way. The key point is the usage of identical mechanisms to obtain and build the codes from the central code repository during testing, development and deployment. The use of

an automated tool for testing allows us to rationalise further steps in the release procedure, log files about the testing are created automatically, and additional cross checks like the validation of the consistency between git repository and local code versions have been added to the automatic tool. By making the code release procedure resource efficient a continous integration [1] of changes is enabled.

4.2. REPROCESSING AND VALIDATION

When an analysis code needs to be modified the first JET pulse number from which the new code version is to be applied needs to be determined. Frequently, this means reprocessing existing data produced using the now outdated version of the code. Because of the interdependencies between analysis codes, this may in turn require the reprocessing of other codes, which are reading the output of the modified code. Traditionally, the affected codes and their dependency state was obtained using the current dependency configuration of the intershot analysis chain, but because this also changes over time, error prone manual intervention was needed to obtain historical or contemporary assemblages. The new system allows to automatically generate an instance of the intershot chain using the correct run configuration for old pulses.

4.3. LIMITATIONS OF THE CURRENT SYSTEM AND IDEAS FOR FUTURE DEVELOPMENT

Chain1 operated very successfully without major disruptions for more than ten years in its old form. The described updates were incorporated at the beginning of the 2011 campaign and havebeen used successfully, ever since. Nevertheless there are some limitations, partly caused by the gradual nature in which modifications to the highly available and traceable system need to be restricted. Ultimately, a Chain1 infra structure that allows diagnosticians to upload code updates and new calibrations using an automatic procedure would be desirable. In the following sections, a list of ideas about how the current framework could be further developed are given.

4.3.1. Interchangeability of Physics Data from different Diagnostics

Some analysis codes need plasma parameters as input which can be provided by different diagnostics. Usually these codes fall back to an alternative measurement in case of problems with the preferred diagnostic. Currently, this is not transparent to the Chain1 framework but is implemented within each analysis code and may therefore be inconsistent across different codes. This could be addressed by introducing analysis steps providing recommended physics parameters using a well defined algorithm to combine or select information from different diagnostics. By this, the processing of derived quantities would be more independent from individual diagnostics. In addition, the creation of recommended physics profiles would allow to combine and cross calibrate information from different diagnostics in a well defined way.

4.3.2. Storage of Configuration Information

Step and interdependency configuration information is currently stored in two places: static

information is stored in the RepositoryDB, while pulse dependent information is stored in a number of step-specific DependendcyDBs. The implementation of the DependendcyDB is based on flat files stored with other input data of analysis steps. This implementation is grown with the needs of steps for flexibility and the need for transparency from the framework, but has its limitations, e.g. it requires the cloning of a a large number of repositories in order to work out the configuration of the intershot chain for a given pulse. A more stringent solution would be the use of a database and user friendly tools for the diagnosticians to provide this information.

4.3.3. Wider applicability

At JET, there are several other systems providing data for the collaboration. Notably, there is a suite of anlysis codes which are only run on request performing more resource intensive computation than usually done intershot, Chain2. It seems worth assessing whether parts of the Chain1 infrastructure can be reused here efficiently, or if an integration of chain2 analysis within the intershot chain is feasible.

CONCLUSIONS

We present an update to the existing framework used for the JET intershot analysis allowing the rationalisation of maintenance procedures. A new deployment model making use of automatic tools achieves greater transparency and reliability while requiring less manual intervention. The new framework is geared to a continuous integration development model. It has been used successfully for the latest JET campaign using the new ITER like wall, which involved significant changes to the chain analysis codes and their interdependencies. Thanks to the update, it was possible to perform reprocessing of data from previous campaigns without major manual intervention. This way the efforts of the Chain1 support team could be focused on new developments and supporting the work of the step ROs.

ACKNOWLEDGEMENTS

This work was supported by EURATOM and carried out within the framework of the European Fusion Development Agreement. The views and opinions expressed herein do not necessarily reflect those of the European Commission.

REFERENCES

[1]. M. Fowler, Continuous integration (2006). URL http://martinfowler.com/articles/continuousIntegration.html

[2]. R. Layne, N. Cook, D. Harting, D. C. McDonald, C. Tidy, The jet intershot analysis: Current infrastructure and future plans, Fusion Engineering and Fesign **85** (3-4) (2010) 403–409. doi:10.1016/j.fusengdes.2009.12.012. URL http://tinyurl.sfx.mpg.de/sus4

[3]. J.P, Christiansen, Integrated analysis of data from jet, Journal of Computational Physics **75** (1) (1987) 85 – 106. doi:10.1016/0021-9991(87)90107-0. URL http://www.sciencedirect.com/science/article/pii/0021999187901070

[4]. The perl programming language. URL http://www.perl.org/

[5]. The plain old documentation format. URL http://perldoc.perl.org

[6]. Cvs - open source version control. URL http://www.nongnu.org/cvs

[7]. Git - fast version control system. URL http://git-scm.com/

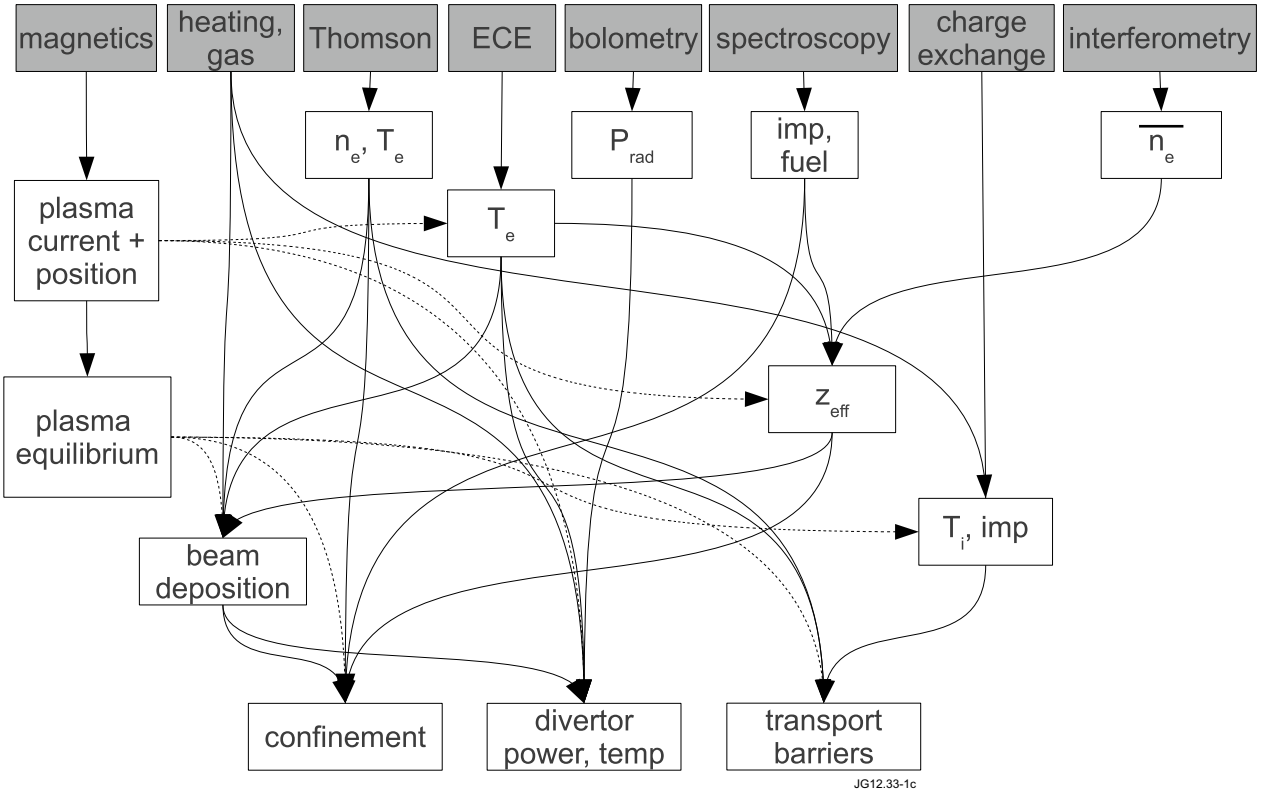

Figure 1: Flow of information of the most important diagnostics of the JET intershot analysis as used in the 2008/2009 experimental campaigns. In the top row of the diagram, diagnostic systems providing raw data are shown, the white boxes correspond to physics parameters obtained by the respective analysis codes. The arrows indicate the data that is taken into account. The use of shape information derived from the magnetics is shown by the dashed lines to aid the eye.

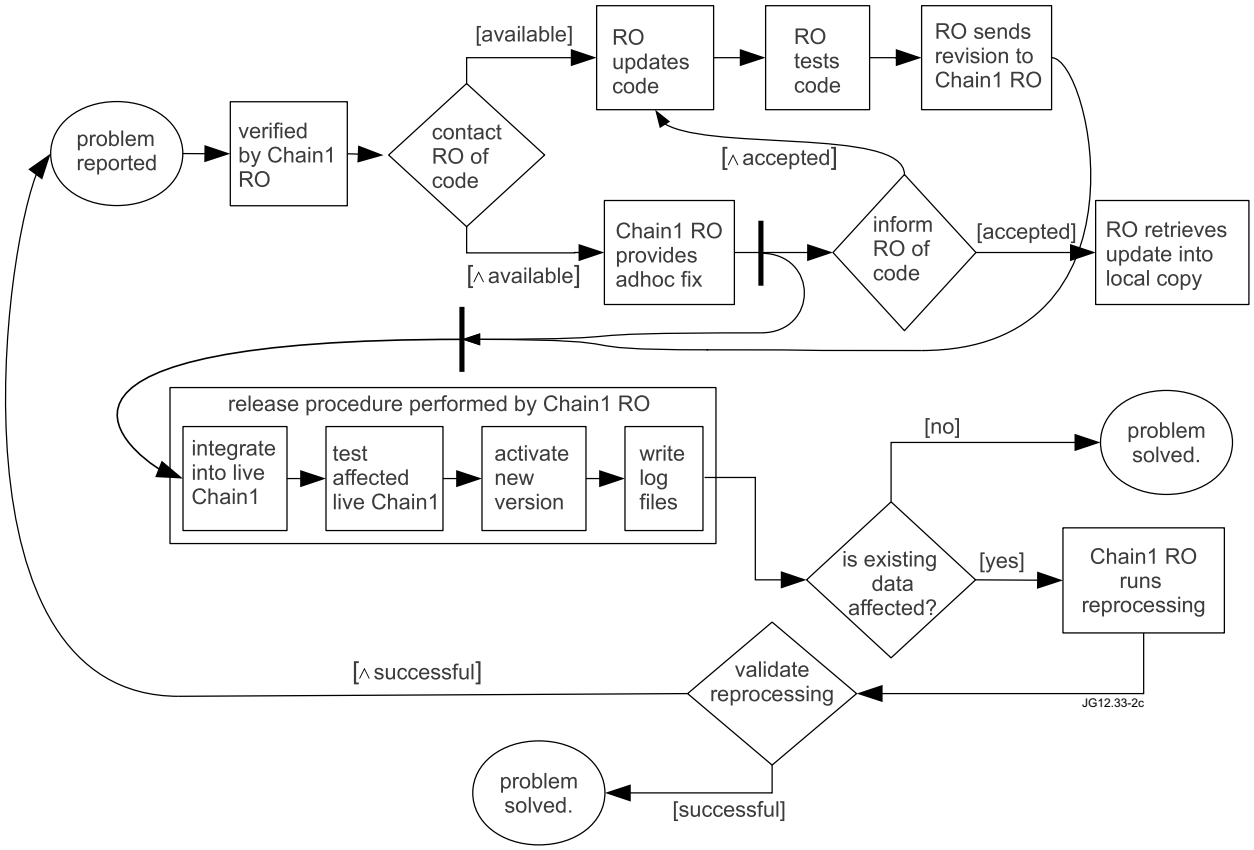

Figure 2: Flow diagram of a typical maintenance case within the Chain1. The member of the Chain1 support team that deals with the case is refered to as Chain1 RO.

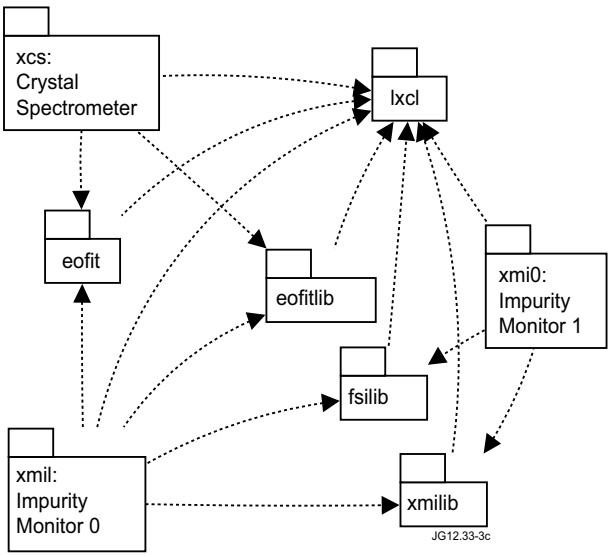

Figure 3: Example of build dependencies of the Chain1 steps for the analysis of different X-ray diagnostics. The steps xcs, xmi0 and xmi1 depend on a number of libraries to share analysis routines, which need to be build before any of the depending steps.