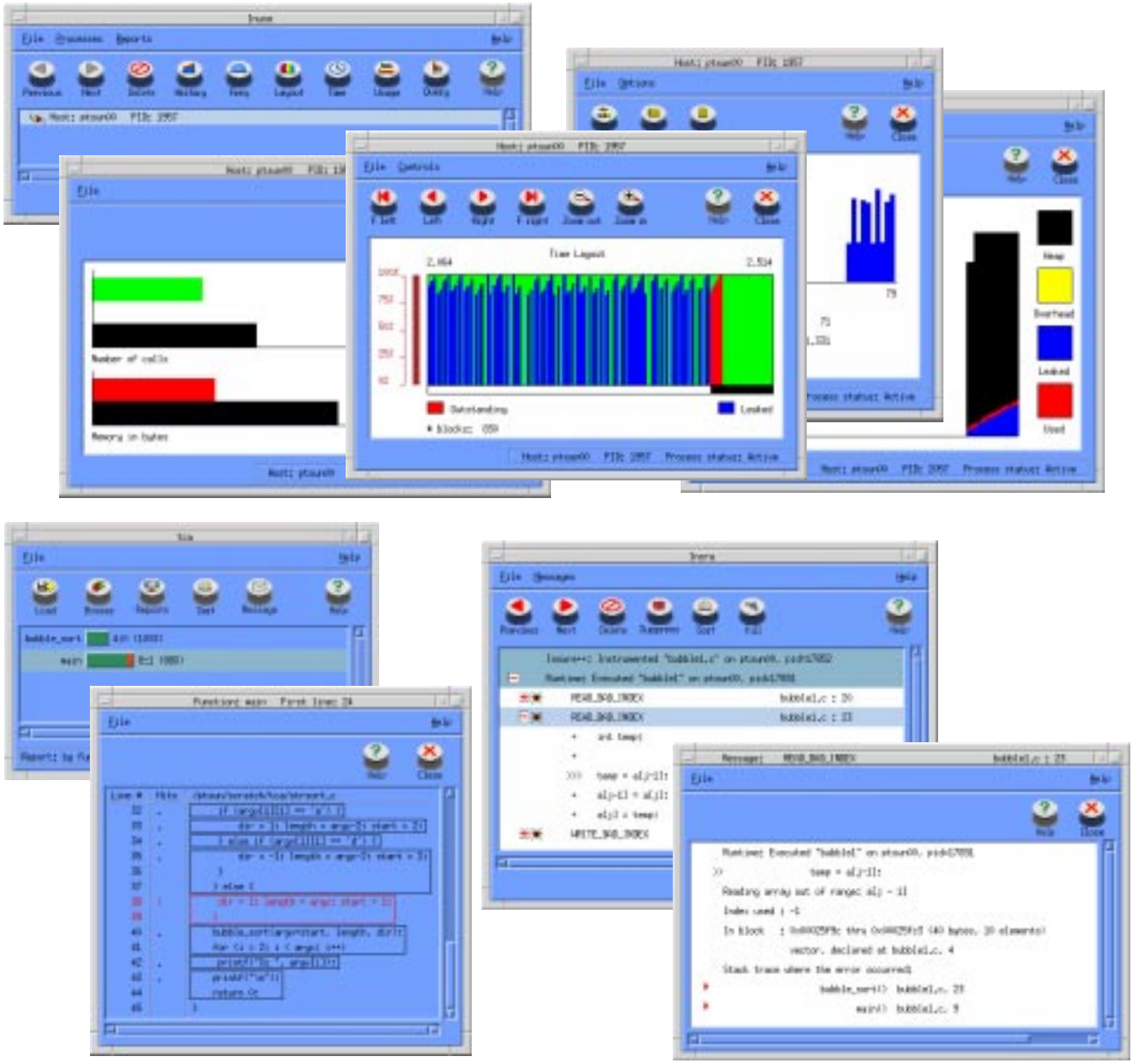
Insure/SRT Integration
User's Guide

Arash Khodabandeh



ATLAS DAQ Technical Note 085

Document Version 2.1, February 1999

URL: `http://atddoc.cern.ch/Atlas/Notes/085/Welcome.html`
`http://atddoc.cern.ch/Atlas/Notes/085/usersguide.ps`
`http://atddoc.cern.ch/Atlas/Notes/085/usersguide.pdf`

Copyright CERN, Geneva 1997 - Copyright and any other appropriate legal protection of this documentation and associated computer program reserved in all countries of the world.

Organisations collaborating with CERN may receive this program and documentation freely and without charge.

CERN undertakes no obligation for the maintenance of this program, nor responsibility for its correctness, and accepts no liability whatsoever resulting from its use.

Program and documentation are provided solely for the use of the organisation to which they are distributed.

This program may not be copied or otherwise distributed without permission. This message must be retained on this and any other authorised copies.

The material cannot be sold. CERN should be given credit in all references.

This document has been prepared with Release 5 of the Adobe FrameMaker® Technical Publishing System using the User's Guide template prepared by Mario Ruggier of the Information and Programming Techniques Group at CERN. Only widely available fonts have been used, with the principal ones being:

Running text:	Palatino 10.5 pt on 13.5 pt line spacing
Chapter numbers and titles:	AvantGarde DemiBold 36 and 24 pt
Section headings	AvantGarde DemiBold 20 pt
Subsection and subsubsection headings:	Helvetica Bold 12 and 10 pt
Captions:	Helvetica 9 pt
Listings:	Courier Bold 9 pt

Use of any trademark in this document is not intended in any way to infringe on the rights of the trademark holder.

Preface

This document describes the Insure++ integration in the SRT based software development environment of ATLAS DAQ. It explains how the integration was implemented and gives some examples of usage. This document is not a replacement for Insure++ or SRT manuals.

How to Use This Document ii

How to Use This Document

You can use this manual differently depending on your role in the project.

- Chapter 1 gives a general overview of what Insure is and how it works.
- Chapter 2 gives an overview of how the integration is organized.
- Chapter 3 details some of the most common actions that one may want to perform with Insure under SRT.
- Appendix A lists the requirements for SRT in order to integrate Insure.
- Appendix B lists the known problems and workarounds if any.

As an End User of Insure/SRT

Start by Chapter 1 and Chapter 2 as introduction. They will help you to use the actions detailed in Chapter 3. Refer to Appendix B for a workaround to possible problems.

As the Project Quality Manager or the Project Manager

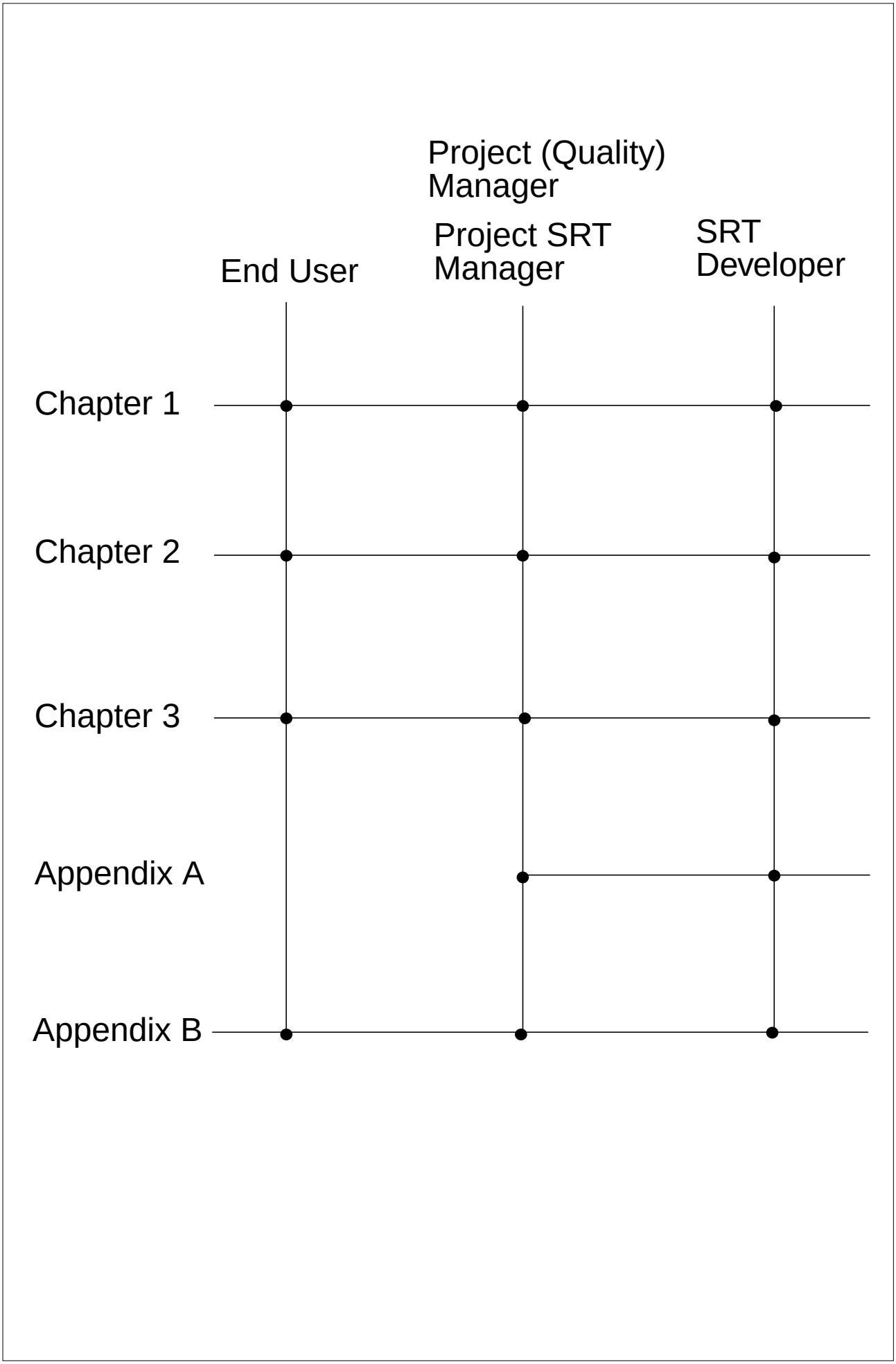
All chapters are relevant for you.

As the Project SRT Manager

As the person maintaining this Insure / SRT integration, all chapters are of interest to you.

As the SRT Developer

As the person implementing the requirements listed in Appendix A in SRT, read Chapter 1 and Chapter 2 to see what is this all about. The use cases in Chapter 3 can be turned into test cases. Appendix B lists outstanding problems some of which can be fixed in the implementation.



A different path through the chapters can be followed depending on your role in the project.

Outline

		Preface	i
		Contents	vii
Chapter	1	Insure++	1
Chapter	2	SRT Integration Strategy	7
Chapter	3	Use Cases	13
Appendix	A	SRT Integration Requirements	23
Appendix	B	Known Problems	25
		References	27
		Index	29

Contents

Preface i
How to Use This Document ii
As an End User of Insure/SRT ii
As the Project Quality Manager or the Project Manager . . . ii
As the Project SRT Manager ii
As the SRT Developer ii

Outline v

Chapter 1
Insure++ 1
1.1 What is Insure++ 2
Insure 2
Inuse 2
TCA 2
1.2 Analysis Process 4
1.3 Insure++ at CERN 6

Chapter 2
SRT Integration Strategy 7
2.1 Definitions 8
2.2 Implementation per SRT Module 10
Compile Time Messages 10
Runtime Messages 10
Coverage Data 10
Memory Usage Reports 10
2.3 Going Further 11

Chapter 3
Use Cases 13
3.1 Prepare for Analysis 14
Configure the Project for Analysis 14
3.2 Interactive Analysis 15
Compile the Project for Memory Checking 15
Check Memory 16
Coverage Analysis 17
Compile the Project for Memory Usage Monitoring 18

Memory Usage Monitoring 19
3.3 Batch Mode Analysis 20
Compile the Project for Memory Checking 20
Check Memory 21
Coverage Analysis 22

Appendix A
SRT Integration Requirements 23
A.1 List of Requirements 24
General Requirements 24

Appendix B
Known Problems 25
B.1 List of Known Problems 26
Compilation Problems 26
Insra Problems 26
LeakTool Problems. 26
TCA Problems 26

References 27

Index 29

Chapter 1

Insure++

This chapter outlines the different components of Insure++ which have been integrated with SRT and the way they work.

1.1 What is Insure++ 2

1.2 Analysis Process. 4

1.3 Insure++ at CERN. 6

1.1 What is Insure++

Insure++ [3] from Parasoft [5] is a set of 3 tools addressing 3 of the aspects of code quality: memory management checking (Insure), memory usage monitoring (Inuse) and test coverage (TCA). All 3 tools have been integrated with SRT [2].

Insure

Insure helps identify run time memory related errors which can not be detected at compile and link time such as pointer equations errors, memory leaks, string / array manipulation errors, etc. It can performs those tests on third party libraries.

Insure can be used only at link time to instrument the binary code, but it is more powerful if it is also used at compile time.

Inuse

Inuse visualizes in real-time the memory manipulation of a program. It helps developers spot leaks, inefficiencies in memory handling, memory fragmentations, memory usage for a specific event. These suspicions must then be investigated further for example with Insure.

TCA

TCA is a small source code test coverage tool. It allows the programmer to identify which part of his source code was not covered during tests.

Note
When in this document a concept is explained referring to Insure++, it should be understood that the concept is applicable to all of the 3 above products.

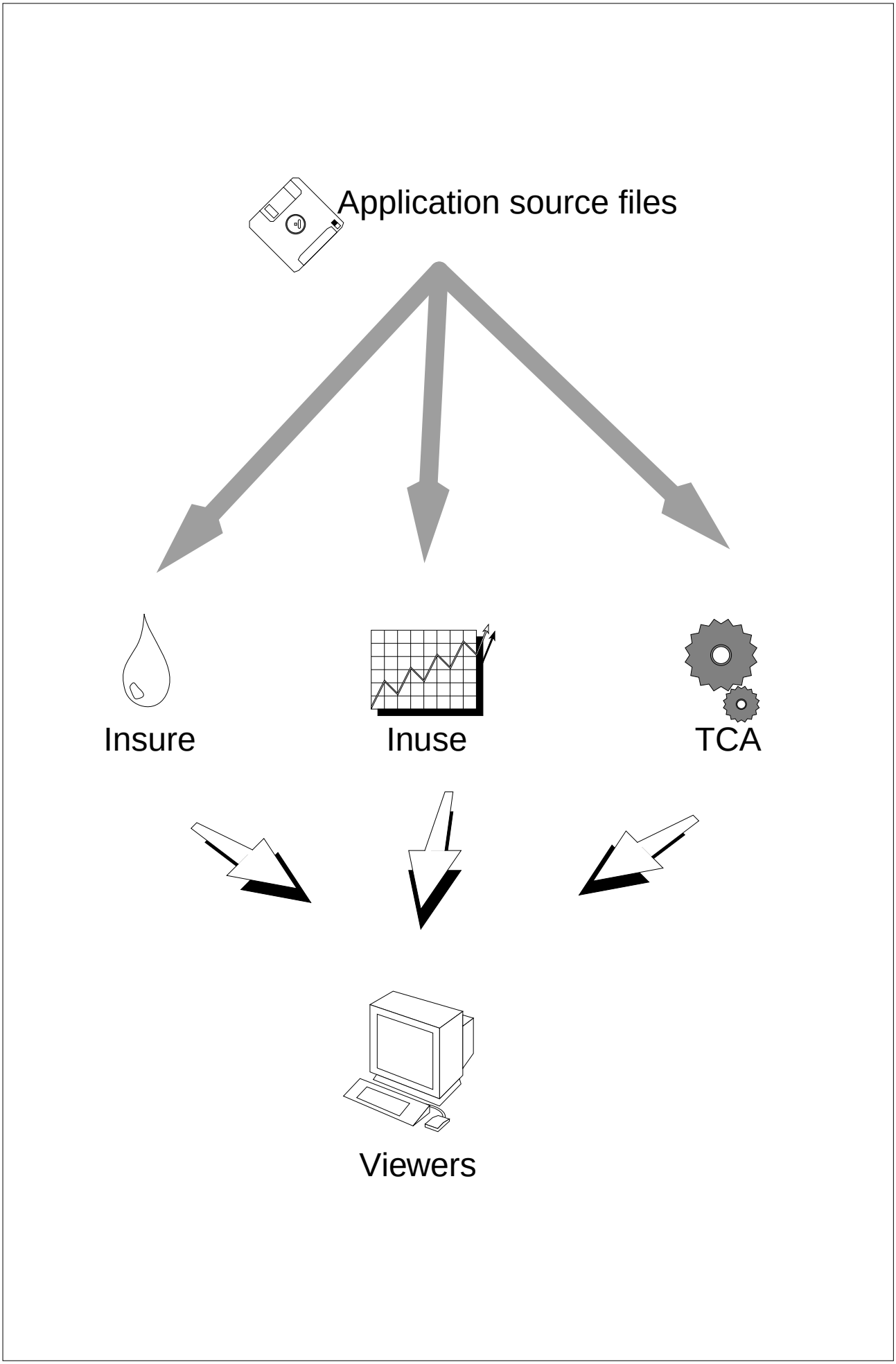


Illustration 1.1 Insure, Inuse and TCA instrument the source files or object code. The results are displayed in various viewers depending on the information gathered.

1.2 Analysis Process

As mentioned in the previous section 3 different analysis can be performed on the application following a similar analysis chain:

1. Compilation and link

   At this stage an instrumented version of the application is generated using Insure. Depending on the options passed to Insure the instrumented application can be used for memory usage monitoring, memory management checking or test coverage.

   In fact it is sufficient to just link with Insure, but for memory management checking Insure will be more powerful if the application is also compiled with it.

   At compile time Insure can already detect some errors which can be either sent to standard output, to a file or to the Insure error reporting interface Insra. The compile time messages file can be used for batch mode reporting or viewed offline with Insra.

2. Memory usage monitoring

   When starting the application which was compiled for memory usage monitoring, it will report via an Inuse interface its current memory activities. The reporting is done in real-time and can be interacted with.

3. Memory management checking

   Similarly when starting the application which was compiled for memory usage checking all runtime errors which are detected are send to standard output, to a file or to Insra. The runtime messages file can be used for batch reporting or viewed offline with Insra.

4. Test coverage

   Usually test coverage is performed automatically with the memory management checks, but it can be decoupled if needed. While the application is executing it writes to a coverage log file which part of the code it is in. The coverage log file can then be either viewed offline with the TCA GUI or used for batch reporting.

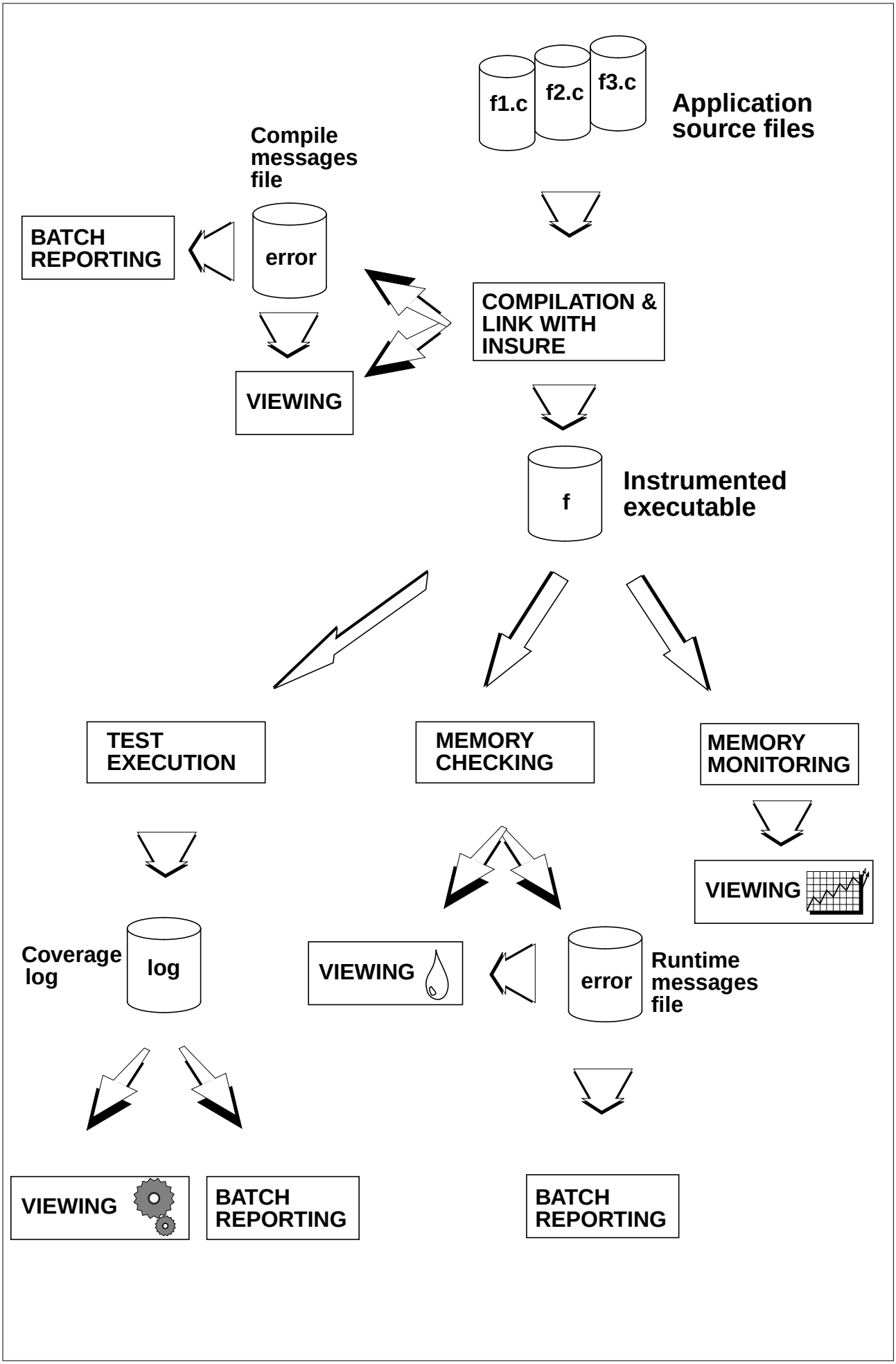


Illustration 1.2 The application source files are compiled and/or linked with Insure to generate instrumented executables. Depending on the instrumentation options the new executables can be used to perform memory usage monitoring, memory management checking or source line test coverage.

1.3 Insure++ at CERN

Insure++ is available through the Software Development Tools Service [3]. A number of licenses are purchased and can be used concurrently by a certain number of users.

Note
Insure++ uses floating licenses. To guarantee access to other users, please make sure you quit the application when you are not working with it.

In order to have access to Insure++, you will need to update your environment. Open `http://www.cern.ch/PTTOOLS/Insure/How.html`, go to the page corresponding to the architecture of the computer where you will use Insure++ on and follow the instructions.

The full set of manuals are available online [4].

Chapter 2

SRT Integration Strategy

This chapter outlines the Insure++ aspects specific to the SRT integration. It is recommended to read Chapter 1 for more information on what is Insure++ and how it works before starting with this chapter.

2.1 Definitions 8

2.2 Implementation per SRT Module 10

2.3 Going Further 11

2.1 Definitions

1-Instrumented target When compiling the code with Insure an instrumented version of it is generated. This includes instrumented object files which will be archived or linked together into instrumented libraries or instrumented executables.

The instrumented code contains added directives which will allow Insure++ to perform the different tests (memory checking, memory monitoring, test coverage).

2-Compile time message A compile time message is generated by Insure when reporting a problem (e.g. compilation warning) during the compilation of an instrumented object file.

3-Session compile time messages Collection of all the compile time messages generated for all the compilations. The session compile time messages can be generated on stdout, in files or sent to Insra GUI.

4-Runtime message A runtime messages is generated when reporting a problem (e.g. a memory leak detected) during the execution of an instrumented executable.

5-Session runtime messages Collection of all the runtime messages generated for all the runs of all the instrumented executables. The session runtime messages can be generated on stdout, in files or sent to Insra GUI.

6-Execution memory usage report There is one such report per execution of each instrumented executable. It comes in the form of interactive graphical report through Inuse.

7-Coverage data Each time an instrumented executable is run a set of coverage data are generated which track which line of the code was executed. There is one such data collection per execution of an instrumented executable.

8-Session coverage data Collection of all the coverage data generated for all the runs of all the instrumented executables.

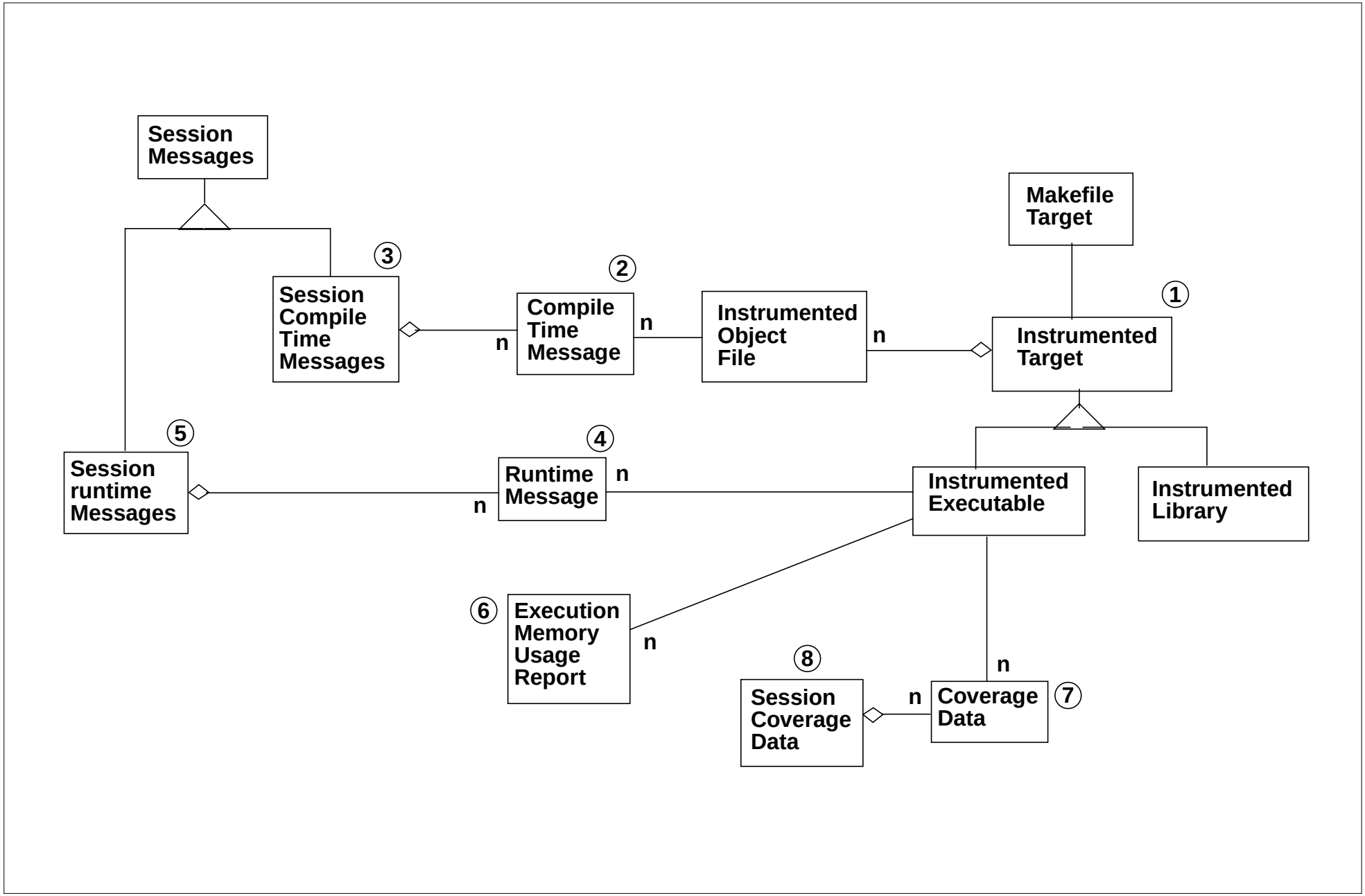


Illustration 2.1 Object diagram of the different files and messages of Insure++.

2.2 Implementation per SRT Module

Compile Time Messages

For interactive analysis the compile time messages are sent to Insra GUI.

By default, in batch mode, the process described in Chapter 2 saves all compile time messages of all source files in one ASCII compile time message file (`compile.txt`) located in the `.srt/$SRT_TARGET` directory of the SRT module. This behavior can be easily changed if needed.

It is possible to load the ASCII compile time messages file into Insra GUI and vice versa.

Runtime Messages

For interactive analysis the runtime messages are sent to Insra GUI.

By default, in batch mode, the process described in Chapter 2 saves all runtime messages of all executions of all instrumented executables in one ASCII runtime message file (`runtime.txt`) located in the `.srt/$SRT_TARGET` directory of the SRT module. This behavior can be easily changed if needed.

It is possible to load the ASCII runtime message file into Insra GUI and vice versa.

Coverage Data

Coverage data is always saved into a file for later analysis with TCA. By default the process described in Chapter 2 generates all coverage data of all executions of all instrumented executables in one file (`tca.log`) located in the `.srt/$SRT_TARGET` directory of the SRT module. This behavior can be easily changed if needed.

Memory Usage Reports

The execution memory usage reports are only viewed interactively and can not be saved into files. Each time an instrumented executable is started its sends its data to a running Inuse GUI.

2.3 Going Further

Insure comes with various other features which can be used directly from within your code. They are documented with examples in the Insure++ User's Guide sections:

Selective Checking You can turn off memory checking for selected portion.

Interacting with Debuggers You can work with your debugger and Insight at the same time.

Tracing Helps tracing the function call sequence without having to use conditional `printf` statements in your code.

Signals Helps debugging signal handling.

Code Insertions You can define assertions that are then checked for you by Insight. If your assertion raises an error, it is handled and reported by Insight as the errors checked for by default by Insight.

Interfaces An API that helps you to extend the Insight functionalities.

Chapter 3

Use Cases

This chapter describes some of the common actions the user may want to perform. It is not a replacement of the manual which should be consulted for more details.

Note
The Insure++ manuals [4] come with very interesting tutorials on Insure, Inuse and TCA. It is highly recommended to perform those tutorials as they will take you through the various features of the tools.

The example used throughout this chapter is the MRS package [6] [7].

3.1 Prepare for Analysis 14
3.2 Interactive Analysis 15
3.3 Batch Mode Analysis 20

3.1 Prepare for Analysis

Configure the Project for Analysis

1. Setup SRT
2. Go to your working directory: `cd ~`
3. Check out MRS: `cvs co -d mrs backend/mrs`
4. `cd mrs`
5. Configure for Logiscope: `configure --with-tool=insure`
6. Go to the target host directory: `cd .srt/$SRT_TARGET`

3.2 Interactive Analysis

Compile the Project for Memory Checking

1. If you have performed other compilations before clean your directory if needed:

   ```
   gmake clean
   ```

2. Compile the code to perform memory checking with:

   ```
   gmake CXX="insure -Zoi\"insure++.compiler_cpp g++\"
   -Zoi\"insure++.compiler_default cpp\"
   -Zoi\"insure++.coverage_map_file `pwd`/tca.map\" -g"
   ```

   This will generate the instrumented version of the code and at the same time pop-up the Insra for compile time message reporting.

Note 1

See in the Insure++ Getting Started the Using Insure section and in the Insure++ Reference Manual the Configuration Files section on how to use Insure with other compilers and languages. For example for C compiled with `gcc` one would have typed: `gmake CC="insure -Zoi\"insure++.compiler_c gcc\" -Zoi\"insure++.coverage_map_file `pwd`/tca.map\" -g"` .

Note 2

You can save the compile time messages reported in Insra either for future usage or for batch mode processing. See in the Insure++ User's Guide the section on Insra.

Note 3

When more than one Insure user work with the tool on the same host they should select a different port number via which Insure communicates with Insra. In order to do this put the line `insra.port nnnn` (nnnn > 1024) in the `.psrc` file under your home directory.

Check Memory

1. Check that you do not have a local `.psrc` file or if you do it **does not** set the report file (`insure++.report_file`) to something else than insra, and it **does not** contain:

   `insure++.inuse on`

2. Run your programs as usual.

Caution 1
The various binaries should be started with the relative path. (e.g. to start `MRSserver` you should type `examples/MRSserver`. This is to ensure you get the instrumented version and not the public version from the SRT installed directories.

Caution 2
In order to use the instrumented library update your `LD_LIBRARY_PATH` to point to local directories and not to the public version from the SRT installed directories. For example in this case you should point to `~/mrs/.srt/$SRT_TARGET/src` in order to use the instrumented `libmrs.so`.

Every program will connect to the current Insra instance (or start one) and report any runtime memory problem via the GUI using the default port or the port you specified in the `.psrc` file under your home directory.

Note
You can save the runtime messages reported in Insra either for future usage or for batch mode processing. See in the Insure++ User's Guide the section on Insra.

3. Quit Insra with:

   `File->Exit`.

Coverage Analysis

When you run your programs for memory check, Insure merges all coverage data in the `tca.log` coverage data file.

1. Start the TCA tool with

   ```
   TCA -X tca.log
   ```

2. Perform your analysis.
3. Quit TCA with

   ```
   File->Exit
   ```

Note
You do not have to have one coverage data file for all your programs. See in the TCA Getting Started the Using TCA section for details on various ways to generate coverage data files.

Compile the Project for Memory Usage Monitoring

1. If you have performed other compilations before clean your directory if needed:

   ```
   gmake clean
   ```

2. Compile the code to perform memory usage monitoring with:

   ```
   gmake CXX="insure -Zoi\"insure++.compiler_cpp g++\"
   -Zoi\"insure++.compiler_default cpp\" -g -Zuse"
   ```

   This will generate the instrumented version of the code.

Note
See in the Insure++ Getting Started the Using Insure section and in the Insure++ Reference Manual the Configuration Files section on how to use Insure with other compilers and languages. For example for C compiled with `gcc` one would have typed: `gmake CC="insure -Zoi\"insure++.compiler_c gcc\" -g -Zuse" .`

Memory Usage Monitoring

1. Start the Inuse program in the background if no other Inuse is running

   `inuse&`

2. Create a local `.psrc` file which contains the line:

   `insure++.inuse on`

3. Run your programs as usual.

Caution 1

The various binaries should be started with the relative path. (e.g. to start `MRSserver` you should type `examples/MRSserver`. This is to ensure you get the instrumented version and not the public version from the SRT installed directories.

Caution 2

In order to use the instrumented library update your `LD_LIBRARY_PATH` to point to local directories and not to the public version from the SRT installed directories. For example in this case you should point to `~/mrs/.srt/$SRT_TARGET/src` in order to use the instrumented `libmrs.so`.

   Every program will connect to the running Inuse process and its PID will be listed in the main window. In order to display memory information on a process, first select it in the list, then click on any of the action bottons.

4. When you are finished monitoring the memory usage exit the Inuse GUI with

   `File->Quit`

3.3 Batch Mode Analysis

Compile the Project for Memory Checking

1. If you have performed other compilations before clean your directory if needed:

   ```
   gmake clean
   ```

2. Compile the code to perform memory checking with:

   ```
   gmake CXX="insure -Zoi\"insure++.compiler_cpp g++\"
   -Zoi\"insure++.compiler_default cpp\"
   -Zoi\"insure++.report_file ‘pwd‘/compile.txt\"
   -Zoi\"insure++.coverage_map_file `pwd`/tca.map\" -g"
   ```

   This will generate the instrumented version of the code and at the same time merge all compilation messages into the ASCII compile time messages file `compile.txt`.

Note 1
You do not have to have one ASCII compile time message file for all compilations messages. See in the Insure++ User’s Guide the Insure Reports section for details on various ways to generate report files.

Note 2
See in the Insure++ Getting Started the Using Insure section and in the Insure++ Reference Manual the Configuration Files section on how to use Insure with other compilers and languages. For example for C compiled with `gcc` one would have typed: `gmake CC="insure -Zoi\"insure++.compiler_c gcc\" -Zoi\"insure++.report_file ‘pwd‘/compile.txt\" -Zoi\"insure++.coverage_map_file `pwd`/tca.map\" -g" .`

Note 3
You can load the ASCII compile time message file `compile.txt` in Insra for interactive analysis. See in the Insure++ User’s Guide the section on Insra.

Check Memory

1. Check that your local `.psrc` file contains:

   ```
   insure++.report_file ~/mrs/.srt/$SRT_TARGET/runtime.txt
   insure++.report_overwrite off
   insure++.inuse off
   ```

2. Run your programs as usual.

Caution 1

The various binaries should be started with the relative path. (e.g. to start `MRSserver` you should type `examples/MRSserver`. This is to ensure you get the instrumented version and not the public version from the SRT installed directories.

Caution 2

In order to use the instrumented library update your `LD_LIBRARY_PATH` to point to local directories and not to the public version from the SRT installed directories. For example in this case you should point to `~/mrs/.srt/$SRT_TARGET/src` in order to use the instrumented `libmrs.so`.

Every program will merge its runtime messages to the ASCII runtime message file `runtime.txt` file next to the previous ASCII compile time message file `compile.txt` file.

Note 1

You do not have to have one ASCII runtime message file for all runtime messages. See in the Insure++ User's Guide the Insure Reports section for details on various ways to generate report files.

Note 2

You can load the ASCII runtime message file `runtime.txt` in Insra for interactive analysis. See in the Insure++ User's Guide the section on Insra.

Coverage Analysis

When you run your programs for memory check, Insure merges all coverage data in the `tca.log` coverage data file.

1. Batch mode results can be generated from `tca.log` with TCA.
2. You can also conduct interactive analysis with:

   ```
   tca -X tca.log
   ```

3. Perform your analysis.
4. Quit TCA with

   ```
   File->Exit
   ```

Note
You do not have to have one coverage data file for all your programs. See in the TCA Getting Started the Using TCA section for details on various ways to generate coverage data files.

Appendix A

SRT Integration Requirements

This chapter lists the requirements for the integration of Logiscope in SRT. This completes the description in Chapter 2.

A.1 List of Requirements 24

A.1 List of Requirements

General Requirements

UR-1. The source file names used in the compilation directives when using Insure++ must be given in full path.

Appendix B

Known Problems

This chapter lists the known problems with the integration. For each problem a description and a workaround if any is provided.

B.1 List of Known Problems 26

B.1 List of Known Problems

Compilation Problems

KP-1 **`.d` files not generated**

Description: Sometimes, when compiling with `insure` the `.d` files are not generated. This causes the `sed` which follows a compilation and the general `gmake` to fails.

Workaround: Run `gmake` (with no arguments) before any gmake with `insure`.

Action: See if when `configure --with-tool=insure` is implemented this problem goes away or not.

Insra Problems

KP-2 **Source file not found by Insra**

Description: When clicking on an error to view the source code, Insra does not find the original file and pops-up a file browser instead.

Workaround: Browse by hand up the directory structure till you reach the source locations directories and double click on the source file you are looking for. Insra will then position itself at the right source line.

Action: `configure --with-tool=insure` would configure SRT makefiles so that when compiling with Insure it uses full path to source code instead of relative path.

LeakTool Problems

~~**KP-3**~~ ~~**Files converted with LeakTool are not correct anymore**~~

Description: LeakTool does not convert correctly Insight ASCII error files to Insra report files. Vice versa it does not convert correctly Insra report files to ASCII error files.

Action: This is a known problem with Insure++ 3.1 and should be fixed in Insure++ 4.0.

Status: In 4.0, this conversion is done by Insra.

TCA Problems

KP-4 **Executables compiled with `insure` does not generate the `tca.log` file**

Description: When compiling with `insure` the `tca.map` is generated, but at run time no `tca.log` file is created and the TCA banner is not echoed on STDOUT.

Action: Fasrcase support is looking into this problem.

References

1 **ATLAS DAQ Software DEvelopment Environment.**
http://atddoc.cern.ch/Atlas/DaqSoft/sde/Welcome.html

2 **SRT - Software Release Tool**
http://wwwcn.cern.ch/~lat/exports/srt/manual.ps.gz

3 **Insure at CERN**
http://www.cern.ch/PTTOOL/Insure/

4 **Insure HTML manuals**
http://www.cern.ch/PTTOOL/Insure/Paper.html
or
Insure: http://www.parasoft.com/insure/manuals/home.htm
TCA: http://www.parasoft.com/tca/manual.htm
Inuse: http://www.parasoft.com/inuse/manual.htm

5 **ParaSoft**
http://www.parasoft.com/

6 **Design of the Message Reporting System for the ATLAS DAQ Prototype-1,**
D.Burckhart et al., ATLAS DAQ Technical Note 032,
http://atddoc.cern.ch/Atlas/Notes/032/Note032-1.html

7 **Users Guide and Implementation of the Message Reporting System for the ATLAS DAQ Prototype-1,**
D.Burckhart et al., ATLAS DAQ Technical Note 059,
http://atddoc.cern.ch/Atlas/Notes/059/Note059-1.html

Index