Software Design Specification

도구명 : TraceRecoviz

C++ 실행 추적 및 시퀀스 다이어그램 생성 도구

버전 : 1.0

작성일 : 2025-10-01

목차

1 Introduction

1.1 Purpose

본 문서는 C++/GoogleTest 테스트 실행 시 함수 호출/반환 및 Assertion을 로그로 수집하고, 이를 파싱하여 시퀀스 다이어그램(GoJS 버전)으로 변환하는 도구 모음(계측코드 삽입 도구, GTest 리스너, 로거/파서, 파일 유틸)의 요구사항을 정의한다. IEEE Std 830-1998이 권장하는 구조를 따른다.

본 문서는 TraceRecoviz 도구의 상세 설계 내용을 정의한다.

TraceRecoviz는 C++ 코드의 실행 추적 및 시퀀스 다이어그램 생성 도구로서, Clang LibTooling을 이용한 빌드 타임 계측, GoogleTest 리스너 기반의 실행 시 추적 로깅, 그리고 GoJS 모델 기반의 시퀀스 다이어그램 생성 기능을 통합 제공한다

1.2 Scope

- Instrumentation Tool: Clang LibTooling 기반으로 함수 호출, 반환, Assertion 구문을 탐지하고 trace 코드를 자동 삽입한다.
- Runtime Runner: GoogleTest 를 이용해 테스트 시작/종료 이벤트를 감지하고 로그 파일을 관리한다.
- Log Parser: [CALL], [RETURN], [ASSERTION_CALL] 라인을 정규식 기반으로 분석하여 호출 관계를 추출한다.
- Sequence Diagram Generator: 파싱된 호출 관계를 GoJS JSON 구조로 변환하여 시각적 시퀀스 다이어그램을 생성한다.
- .

1.3 Definitions, Acronyms, and Abbreviations

- 계측 삽입 : 소스에 추적 코드를 자동 삽입하는 과정
- TraceListener : GTest 이벤트 리스너 클래스
- CALL/RETURN/ASSERTION_CALL : 로그 라인 액션 구분자(정규식 그룹)
- GoJS GraphLinksModel : GoJS 시퀀스 다이어그램 데이터 모델 키 (nodeDataArray, LinkDataArray)

1.4 References

- IEEE Std 1016-1998 : 섹션 구성, 설계 엔티티 및 속성, 설계 뷰 (Decomposition, Dependency, Interface, Detailed) 정의 등 템플릿-가이드라인

1.5 Overview

SDS는 다음과 같은 구조로 구성된다.

- 2장**:** 설계 개요 – TraceRecoviz의 전체 구조 및 설계 접근 방법
- 3장: 모듈별 상세 설계 – Instrumentation, Runner, Parser, Diagram Generator 구조 및 인터페이스
- 4장: 설계 근거 및 추적성 – 요구사항(SRS)과 설계 단위 간의 매핑
- 5장: 부록 – 데이터 구조 정의, 클래스 다이어그램, 시퀀스 다이어그램 등

2 Overall description

2.1 Product perspective

본 소프트웨어는 소스 계측, 실행 시의 리스너/로거, 로그 파서로 구성된 모듈 묶음이다. 소스 계측은 Clang ASTFrontendAction/ASTConsumer 기반으로 동작하며, 수정된 코드를 표준 출력으로 제공한다. 실행 시에는 GTestEmptyTestEventListener를 통해 테스트 생명주기를 캡처한다.

2.2 시스템 구조 개요

- TraceRecoviz 는 다음 4 개 서브시스템으로 구성된다
- Instrumentation Tool
  Clang LibTooling 을 이용하여 C++ 코드 내 함수 시작부와 반환부를 탐지하고, trace_enter() 및 trace_return() 함수를 자동 삽입한다. EXPECT_*, ASSERT_* 구문은 대응되는 _LOG 형태로 치환되어 assertion 이벤트도 로깅된다. 모든 코드는 표준 출력으로 내보내져 빌드 과정에 통합된다.
- Trace Listener
  GoogleTest 의 이벤트 훅을 이용하여 테스트 실행 중의 각 단계(프로그램, 스위트, 테스트, Fixture 등)를 감시하고, 생명주기 이벤트를 포함한 로그 파일을 생성한다. 로그는

build/{TRACE_VARIANT}/<파일명>.log 경로에 저장되며, [CALL], [RETURN], [ASSERTION_CALL] 등의 액션 식별자를 포함한다.

- Log Parser
  Python 기반 모듈로, 표준화된 정규식 패턴을 이용해 로그를 파싱한다. 각 로그 라인을 분석하여 호출자-피호출자 관계를 매핑하고, 반환 타입·값, Assertion 이벤트 등을 구조화된 데이터로 변환한다. 파싱 결과는 GraphLinksModel JSON 포맷으로 저장된다. Diagram Generator 파서가 생성한 JSON 데이터를 기반으로 GoJS 렌더러를 호출한다. 각 노드(객체 인스턴스)와 링크(함수 호출)를 매핑하여 실행 흐름을 시각화하며, 생성자와 소멸자 이벤트를 별도의 라이프라인 시작/종료로 구분한다.
- 이 네 모듈은 독립적으로 실행 가능하지만, 전체 시스템은 "계측 → 실행 → 파싱 → 시각화"의 파이프라인으로 동작한다.

2.3 데이터 흐름

- 데이터 처리는 순차적이며, 다음의 흐름으로 구성된다.
- 계측 단계에서 원본 C++ 코드가 Clang을 통해 AST로 분석되고, 각 함수 진입/반환 위치에 추적 함수가 삽입된다. 결과는 계측된 C++ 코드로 출력된다.
- 실행 단계에서 GoogleTest 리스너가 활성화되어 테스트의 시작, 종료, Assertion 등을 감시하고 로그 파일을 생성한다.
- 파싱 단계에서는 생성된 로그를 정규식으로 분석하여 호출 관계를 식별하고, 각 객체 인스턴스의 고유 ID를 추적한다.
- 시각화 단계에서는 파싱 결과를 JSON 모델로 변환한 후, GoJS를 이용해 시퀀스 다이어그램 형태로 표시한다.
-

2.4 설계 관점별 구성

- dddddd
- dddd

2.5 설계 제약사항

- 언어 및 빌드 환경: C++17, Python 3.11, GNU Make 4.3

- 플랫폼: Linux 환경에서의 동작을 우선 보장하며, Windows는 제한적 지원
- 종속성: Clang LibTooling 18.x, GoogleTest 1.15.0, PySide6, FastAPI, GoJS

2.6 시스템 상호작용

- 사용자는 PySide6 기반의 GUI를 통해 프로젝트 경로를 설정하고 "빌드 및 실행"을 수행한다. 이 명령은 내부적으로 계측 → 빌드 → 테스트 실행을 순차적으로 호출하며, 생성된 로그는 자동으로 Parser에 전달된다. FastAPI 서버는 파싱 결과를 JSON으로 반환하며, UI는 이를 시각화한다

  전체 과정은 사용자의 추가 개입 없이 자동으로 수행되며, GUI는 진행 상태와 로그 메시지를 실시간으로 표시한다.
-

3 상세 설계

3.1 전체 아키텍처 개요

- 전체 아키텍처는 정적 코드 분석(Instrumentation), 테스트 실행(Runner),로그 파싱(Parser), 그리고 시각화(Diagram Generator) 로 이어지는 단방향 파이프라인 구조를 가진다. 이 네 개의 계층은 서로 독립된 책임을 가지며, 각 단계의 산출물이 다음 단계의 입력으로 전달된다.

3.2 계측 코드 삽입(Instrumentation)

- 프로그램 동작중 함수 단위의 호출, 반환등 이벤트를 수집하도록 계측 코드를 삽입한다.

- **계측 코드 삽입:** src/generator/inject_trace_tool.cpp (Clang LibTooling 기반)
- **런타임 계측:** trace.h, src/generator/trace.cpp
- **테스트 리스너:** trace_listener.h (GoogleTest Event Listener)
-
- **대상 선택 규칙**
- 시스템 헤더 제외(isInSystemHeader), testing:: 네임스페이스 내부 제외.

- 생성자/소멸자, 연산자 오버로드도 포함(단, delete/~ 소멸에 특수 라벨).
- **삽입 지점**
- 함수 본문 **시작**에 trace_enter(file,line,func_sig,this_ptr,args...)를 삽입하고 모든 return 문과 암시적 끝(범위 종료)에 trace_return(file,line,func_sig,retval_meta)를 삽입한다.
-
- **GoogleTest Assert문 치환**
- EXPECT_*/ASSERT_*를 대응하는 EXPECT_*_LOG/ASSERT_*_LOG로 치환(소스 리라이터)하여 계측 코드를 삽입한다.
- 치환 매크로는 내부에서 원래 단언을 호출하고, 별도로 ASSERTION_CALL 로그를 남기도록 한다.
- **출력/오류**
- 변환 소스는 stdout으로 출력 → 빌드 파이프라인에서 리다이렉션해 파일화한다.
-

3.3 테스트 실행 로그 수집(Runner)

- 계측코드가 삽입된 소스코드를 찾아 빌드 후 실행한다.
- 프로젝트 2개에 대해서 수행하며 실행 결과를 각각 new, old 폴더에 저장한다. build/<old or new>/*.log로 분리 수집.
- 테스트 실행시 gtest_main을 사용하여 작성된 모든 googletest를 실행하여 로그를 수집한다.

3.4 로그 파싱 및 분석 (Parser)

- Python 기반 파서는 src/parser/utils/trace_parser.py, src/parser/utils/extract.py, src/parser/utils/files.py로 구성된다.
- 로그 파일은 main.py를 통해 일괄 처리된다.
- 삽입되는 로그 형태
  trace_enter(__FILE__, __LINE__, __PRETTY_FUNCTION__, this, args...);
  trace_return(__FILE__, __LINE__, __PRETTY_FUNCTION__, ret);

- main.py는 difflib 기반 라인 단위 비교를 수행한다.
- 각 라인은 _normalize()로 포인터 주소(0x[0-9a-fA-F]+)를 @ADDR로 치환하고 공백을 정리한다.
- SequenceMatcher를 이용해 equal / insert / delete / replace 구간을 산출하고, 결과는 접두 마커(/ + / -)로 구분되어 build/result/*.log에 저장된다.
- TraceParser 클래스는 각 로그를 읽고 GraphLinksModel 형태의 데이터(JSON)를 생성한다.
- 정규식 _LOG_PATTERN은 [CALL], [RETURN], [ASSERTION_CALL] 행을 파싱한다.
- parse_caller_sig(), parse_callee_sig()는 시그니처를 (반환형, 클래스명, 함수명, 인자)로 분리한다.
- 익명 네임스페이스는 "익명"으로 치환된다.
- _process_line()은 색상 구분자(+, -, !,)를 판별하여 변경 유형을 설정한다.
- CALL 시 "from"=caller_ptr, "to"=callee_ptr,
- RETURN 시 반대 방향으로 링크를 추가한다.

3.5 시각화 (Diagram Generator)

- Python main.py는 세 폴더(build/result, build/new, build/old)를 순회하며 각 .log를 TraceParser로 변환한다.
- 출력 파일명은 {테스트명}.json, {테스트명}_old.json, {테스트명}_new.json 형태로 저장된다.
- 각 JSON은 GoJS GraphLinksModel 형식으로 구성되며,
- 노드(nodeDataArray)는 클래스·객체 라인,
- 링크(linkDataArray)는 함수 호출·반환·단언을 의미한다.
- 색상은 Diff 상태를 반영한다.
- GoJS 기반 HTML 뷰어(index.html, viewer.html)는 FastAPI 서버로부터 /build/sequence_diagram/{file}을 fetch하여 다이어그램을 렌더링한다.
- /api/sequence-diagrams는 존재하는 JSON 파일 목록을 반환한다.

3.6 GUI 설계

- 전체 인터페이스는 상단 바, 좌측 제어 패널, 우측 작업 영역의 세 영역으로 구성되며, 각 영역은 MainWindow 클래스 내부에서 QSplitter 구조를 통

해 분할 배치되어 있다.

3.6.1 상단 바 (Top Bar)

- 상단 바는 QToolBar로 구현되며, 프로그램명 TraceRecoViz이 윈도우 타이틀로 표시된다.
- 툴바에는 다크 모드 토글 버튼, 검색 필드, 경로 브레드크럼이 포함되어 있다. 다크 모드 전환은 QAction 객체(action_dark)로 제공되며, 사용자가 토글할 때 toggle_theme() 메서드가 호출되어 QSS 스타일시트를 전환한다.
- 검색 필드는 QLineEdit으로 구현되어 있으며, 입력된 문자열은 apply_filter()를 통해 파일 트리와 테이블에 실시간 필터링된다.
- 현재 탐색 중인 경로는 QLabel 형태의 브레드크럼(breadcrumb)에 표시된다.
- 종료 버튼은 기본 OS 창 프레임의 닫기 버튼을 사용하며, 별도의 사용자 정의 종료 버튼은 포함되지 않는다
-

3.6.2 좌측 제어 패널 (Control Panel)

- 좌측 제어 패널은 ControlPanel 클래스에 의해 구성되며, 세 개의 세션으로 나뉜다. 전체는 QVBoxLayout 기반으로 수직 정렬되어 있다.
- **(1) 프로젝트 패널**
- "수정 전 프로젝트 경로"와 "수정 후 프로젝트 경로" 입력란은 QLineEdit으로 구성되며 setReadOnly(True)로 설정되어 사용자가 직접 수정할 수 없다.
- 각 경로 필드 오른쪽에는 "폴더 선택" 버튼(QPushButton)이 있으며, 클릭 시 QFileDialog.getExistingDirectory()를 호출하여 폴더 선택 대화상자를 띄운다.
- 선택된 폴더는 내부적으로 target_old, target_new 디렉터리로 복사되며, QSettings에 경로가 저장된다.
- **(2) 작업 패널**
- "계측 코드 삽입 및 시퀀스 다이어그램 생성" 버튼(btn_build) 클릭 시, make clean → make 순으로 빌드 명령을 실행한다.
- 빌드 결과는 별도의 BuildLogDialog 팝업창에 실시간 로그로 출력된다.
- "웹서버 ON/OFF" 버튼은 QPushButton의 체크 가능 상태(setCheckable(True))로 구현되어 있다.

- 클릭 시 FastAPI 서버를 백그라운드 스레드에서 실행/중지하며, 버튼 라벨이 "ON/OFF"로 토글된다.
- **(3) 편집 패널**
- "Makefile 편집" 버튼 클릭 시 MakefileEditorDialog가 실행된다.
- 이 창은 프로젝트 디렉터리 내 Makefile을 읽어 텍스트 편집기로 표시하며, "저장" 및 "다른 이름으로 저장" 기능을 제공한다.

-

3.6.3 우측 작업 영역 (Main Workspace)

- 우측 영역은 QSplitter를 이용해 상·하로 나뉜다. 상단은 "폴더 트리 + 파일 목록", 하단은 "미리보기 + 로그 콘솔" 구조로 되어 있다.
- **(1) 폴더 트리**
- 폴더 트리는 QFileSystemModel과 QTreeView로 구성되며, 선택한 디렉터리를 기준으로 계층적 구조를 표시한다. 트리에서 폴더를 클릭하면 해당 경로가 파일 목록(QTableView)의 루트로 설정된다.
- 트리의 선택 변화는 on_dir_changed() 시그널을 통해 처리된다.
- **(2) 파일 목록**
- 파일 목록은 QFileSystemModel 기반 QTableView로 구현되며, 파일명, 크기, 형식, 수정일 등의 정보를 컬럼으로 표시한다.
- 테이블은 정렬 가능(setSortingEnabled(True))하며, 파일을 클릭하면 미리보기 창이 갱신된다.
- 검색창 입력 내용은 FilesProxy 모델을 통해 트리와 함께 필터링되어 표시된다.
- **(3) 미리보기 창**
- PreviewPanel 클래스가 담당하며, 선택된 파일 유형에 따라 자동으로 표시 모드가 전환된다.
- 텍스트 파일은 QPlainTextEdit을 이용해 읽기 전용으로 표시된다.
- 이미지 파일은 QImageReader로 로드하여 비율 유지 축소로 표시된다.
- 기타 파일은 이름, 경로, 크기 등의 기본 정보만 텍스트로 표시한다.
- 폴더 선택 시에는 폴더 정보(파일 개수, 총 용량 등)가 출력된다.
- **(4) 로그 콘솔**

- 하단 우측에는 QPlainTextEdit으로 구현된 로그 콘솔이 있으며,
- 빌드·웹서버·폴더 복사 등 주요 이벤트의 상태 메시지가 실시간으로 표시된다. 내부 메서드 _append_log()가 이를 수행한다.