Panoramica tecnica del workspace — GrifoAutomaticTestEnv

Questo documento fornisce una descrizione tecnica e operativa del progetto, pensata come base per gli sviluppi futuri.

1. Scopo generale

- Ambiente di test automatico Python per il sistema radar GRIFO-F/TH. Fornisce accesso al bus 1553, comunicazione seriale, controllo alimentazione, raccolta/logging dei risultati e generazione di report in PDF.

2. Struttura principale (cartelle rilevanti)

- `PlatformSimulator/` — componenti nativi e binding SWIG per 1553.
  - `PlatformSimulator/bin/interpreter.py` e `_interpreter.pyd` (driver nativo). Questi espongono l'API a basso livello per la comunicazione 1553.
- `TestEnvironment/env/` — librerie Python condivise tra gli script di test.
  - `leo_grifo_1553.py` — wrapper Python per 1553 (classe `GrifoInstrumentInterface` e singleton `theGrifo1553`).
  - `leo_grifo_terminal.py` — gestione terminale seriale e parsing messaggi (`%%E`, `%%F`, `RECYCLE`).
  - `leo_grifo_io_box.py` — interfaccia BrainBox per controllo alimentazione (`theBrainBox`).
  - `leo_grifo_core.py` — recorder (`theRecorder`) per tracciare step e sessioni.
  - `leo_grifo_test_report.py` / `leo_grifo_pdf2report.py` — generazione PDF a partire da template JSON.
  - `test_common_function.py` — funzioni di alto livello usate dagli script (`check`, `getValue`, `setValue`, `generate_pdf_report`).
  - `site-packages/` — dipendenze locali usate dall'ambiente (fpdf2, pyserial, PIL, ecc.).
- `TestEnvironment/scripts/` — script utente e test runner.
  - `GRIFO_M_PBIT.py` — test principale (PBIT). Rileva `--simulate` per usare i mock.
  - `GRIFO_M_PBIT_mock.py` — implementazione mock (monkey-patching dei singoli oggetti globali per simulazione).
  - `__init__.py` — bootstrap dei path (aggiunge `../env` e `env/site-packages` a `sys.path`).
- `TestEnvironment/json/` — template e configurazioni (es. `default_template.json`).
- `TestEnvironment/pdf_reports/` e `TestEnvironment/LOG/` — output di esecuzione.

3. Pattern e convenzioni di progetto (importanti per gli agenti)

- Singleton globali: `theGrifo1553`, `theBrainBox`, `theRecorder`. Non rinominare questi oggetti senza aggiornare tutti i call-site.
- Interfaccia comune: classi di livello dispositivo implementano `check(expected, *fields, **kwargs)`, `get(*fields, **kwargs)`, `set(value, *fields, **kwargs)` (vedi `TestCommonInterface` in `leo_grifo_common.py`). Usare le funzioni wrapper in `test_common_function.py` per tracciare i risultati.
- Recorder: usare `theRecorder.add_step()` / `close_session()` per registrare lo storico del test; i report PDF aggregheranno questi dati.

- Path precedence: gli script si aspettano che `env/site-packages` sia prioritario rispetto alla Python di sistema. Il bootstrap in `TestEnvironment/scripts/__init__.py` gestisce questo.

4. Interazione con il codice nativo 1553

- `PlatformSimulator/bin/_interpreter.pyd` è la libreria nativa (funziona solo con la versione di Python con cui è stata compilata). Per questo motivo è obbligatorio usare la versione di Python contenuta in `PlatformSimulator/bin` (embedded Python) per eseguire i test in modo affidabile.
- Evitare di eseguire gli script con la Python di sistema: il binding nativo potrebbe non essere caricabile o provocare errori di ABI.

5. Modalità di esecuzione raccomandate

- Rapida (consigliata per sviluppo):

```
cd <repo-root>
.\run_simulate_simple.bat
```

oppure (esplicito):

```
PlatformSimulator\bin\python.exe TestEnvironment\scripts\GRIFO_M_PBIT.py --
simulate
```

- Produzione-like (usa l'ambiente embedded):

```
cd <repo-root>
.\run_simulate.bat
```

- Wrapper manuale utile per CI/debug:

```
python python_simulate_wrapper.py
```

6. Esecuzione in CI

- Nel job CI bisogna utilizzare l'interprete Python presente in `PlatformSimulator/bin` o garantire che `_interpreter.pyd` sia compatibile con la Python della runner. Consiglio pratico per GitHub Actions: copiare `PlatformSimulator/bin` nel runner e lanciare `PlatformSimulator/bin/python.exe python_simulate_wrapper.py`.

7. Cosa cercare quando si modifica codice critico

- Non cambiare le firme `check/get/set` né i nomi dei singletons senza aggiornare tutti gli script sotto `TestEnvironment/scripts/`.

- Se modifichi report PDF o template JSON aggiorna `leo_grifo_test_report.py` e controlla che `generate_pdf_report()` sia ancora invocato con i parametri attesi.
- Aggiunte a `site-packages` locali devono essere testate con l'interprete embedded.

8. Raccomandazioni di sviluppo immediate

- Usare esclusivamente l'interprete in `PlatformSimulator/bin` per sviluppare e testare.
- Eseguire `GRIFO_M_PBIT.py --simulate` per validare le modifiche senza hardware reale.
- Aggiungere test di smoke che: lancino lo script in modalità `--simulate`, verifichino che esca con codice 0 e che venga generato un PDF in `TestEnvironment/pdf_reports/`.

9. File chiave per iniziare (ordine suggerito di lettura)

- `TestEnvironment/scripts/__init__.py` (bootstrap dei path)
- `TestEnvironment/env/leo_grifo_1553.py` (wrapper 1553)
- `TestEnvironment/env/leo_grifo_common.py` (interfaccia comune)
- `TestEnvironment/env/test_common_function.py` (API helper per test)
- `TestEnvironment/scripts/GRIFO_M_PBIT.py` (flow test principale)
- `TestEnvironment/scripts/GRIFO_M_PBIT_mock.py` (implementazione simulazione)

10. Esempio rapido di comando per sviluppo locale (embedded Python)

```
# dalla radice del repository
PlatformSimulator\bin\python.exe TestEnvironment\scripts\GRIFO_M_PBIT.py --
simulate
```

11. Note finali e prossimi passi consigliati

- Primo passo: aggiungere uno smoke-test script sotto `TestEnvironment/scripts/` che esegua il flusso `--simulate` e verifichi la presenza del PDF; questo renderà immediato il feedback per ogni modifica.
- Se vuoi, implemento subito lo smoke-test e un esempio di job CI che usa l'interprete embedded.

Documento generato automaticamente come base per i futuri sviluppi; chiedimi di espandere sezioni specifiche o di creare gli script di smoke-test e CI.