Technische Universität Ilmenau

Rechnerarchitekturen und Eingebettete Systeme

Finales Review Document

RION Package-Manager

---

vorgelegt von: P. Augustin, J. Skopp, C. Juin,
G. Lehmann, A. Schmidt, R. Schöne

eingereicht am: 18. Juli 2022

Fachgebiet: Rechnerarchitekturen und Eingebettete Systeme
Institut für Mikroelektronik- und Mechatronik-System

Betreuer: Georg Gläser, Andreas Becher

Seminarleiter Prof. Armin Zimmermann

Inhaltsverzeichnis

Reviewdokument

1 Zweck des Systems

1.1 Anwendungsbereiche

Der vorliegende Package-Manager richtet sich an Entwickler, sowie Kunden und Entwickler der X-FAB.
RION soll die technischen Grundanforderungen zur PDK-Installation auf Kundenseite auf ein Minimum begrenzen. Durch entsprechende PDK-Paketinformationen und eine Plausibilitätsprüfung dieser, durch RION „(rion check)“, sollen Fehler bei der Installation von PDK-Paketen verhindert werden. Diese könnten im schlimmsten Fall zu einer Fehlfunktion des Halbleiterchips führen.

1.2 Zielgruppe

Die Zielgruppe umfasst alle Kunden der X-FAB.

2 Zielbestimmungen

RION ist ein Package-Manager zum suchen, installieren, aktualisieren und verwalten von Packages für das Process Design Kit (PDK). Die Pakete für RION werden von der Serverseite INOR verwaltet.

2.1 Problemstellung

Zur Zeit werden Packages manuell von einem Webinterface heruntergeladen. Sie werden manuell installiert und aktualisiert. Dieser jetzige Weg ist jedoch unübersichtlich, fehleranfällig, inkonsistent, kostet sehr viel Zeit und erfordert sehr viel manuelle Arbeit, sowohl server- als auch clientseitig. Daher soll dieser Prozess durch Verwendung von RION, clientseitig für X-Fab-Kunden, und INOR, serverseitig für X-FAB-Entwickler, vereinfacht und automatisiert werden.

2.2 Musskriterien

2.2.1 Funktionalität

- INOR muss die Packages serverseitig über eine Datenbank in Form von Archiven zur Verfügung stellen.
- RION muss eine Liste der auf dem Server vorliegenden Pakete herunterladen und durchsuchen können.
- RION muss Pakete installieren können. Dabei müssen auch ältere Versionen des Paketes zugänglich gemacht werden können.
- RION muss installierte Pakete aktualisieren können. In diesem Kontext bedeutet das, dass die neuere Version des Paketes parallel zur Alten zu installiert wird.
- RION muss installierte Pakete entfernen können.

2.2.2 Interaktion

- Dem Benutzer muss ein CLI zur Verfügung stehen.

2.3 Wunschkriterien

- INOR soll Pakete signieren.
- INOR soll Hashwerte von Paketen anfertigen und an RION übermitteln können.
- RION soll nach dem Herunterladen, aber vor der Installation, Pakete auf ihre Signatur, sowie auf ihren korrekten Hashwert überprüfen.

3 Grobentwurf

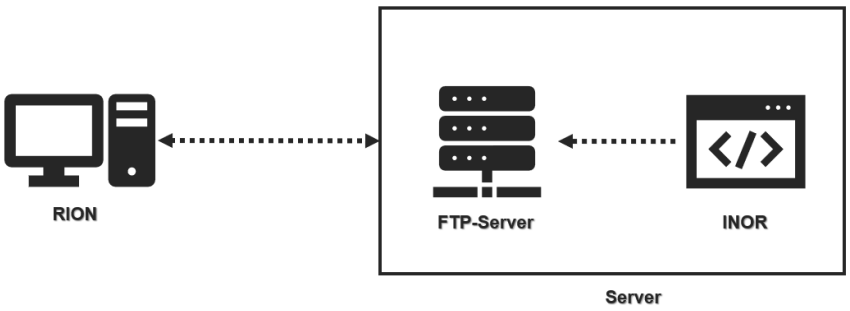

Abbildung 3.1: Use-Case-Diagramm

Das Programm arbeitet prinzipiell mit drei Komponenten. Zum einen gibt es den Packagemanger RION. Dieser läuft auf dem jeweiligen Endgerät (Client), der Pakete sucht, installiert, aktualisiert, entfernt und verwaltet. Hierzu werden mehrere Datenbanken verwendet. Darunter eine Datenbank, die eine Liste aller installierten Pakete enthält und eine weitere, mit einer Liste der Namen aller verfügbaren Pakete, sowie deren Metadaten und die Namen der notwendigen Abhängigkeiten.

Die Pakete, sowie die zweite genannte Datenbank, können von einem FTP-Server über eine verschlüsselte ftps-Verbindung abgerufen werden. Hierzu wird das Programm pyftpdlib, welches unter anderem virtuelle Nutzer zur Verfügung stellt, die zum Rechtemanagement verwendet werden. Dieses Programm wurde nicht von uns entwickelt.

Die Daten für den FTP-Server, sowie die Konfiguration der Rechteverwaltung, werden durch die dritte Komponente INOR zur Verfügung gestellt. Durch diesen Vorgang werden einige Funktionen erleichtert bzw. automatisiert. INOR befindet sich dabei auf dem selben Server wie der FTP-Server.

3.1 Klassendiagramm

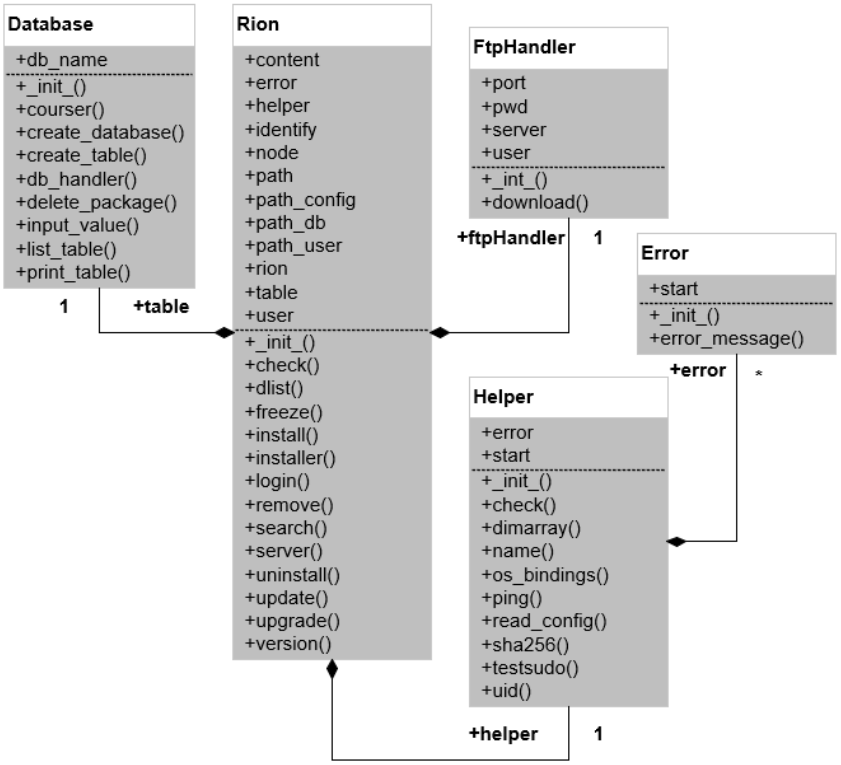

Abbildung 3.2: Klassendiagramm

3.2 Use-Case-Diagramm

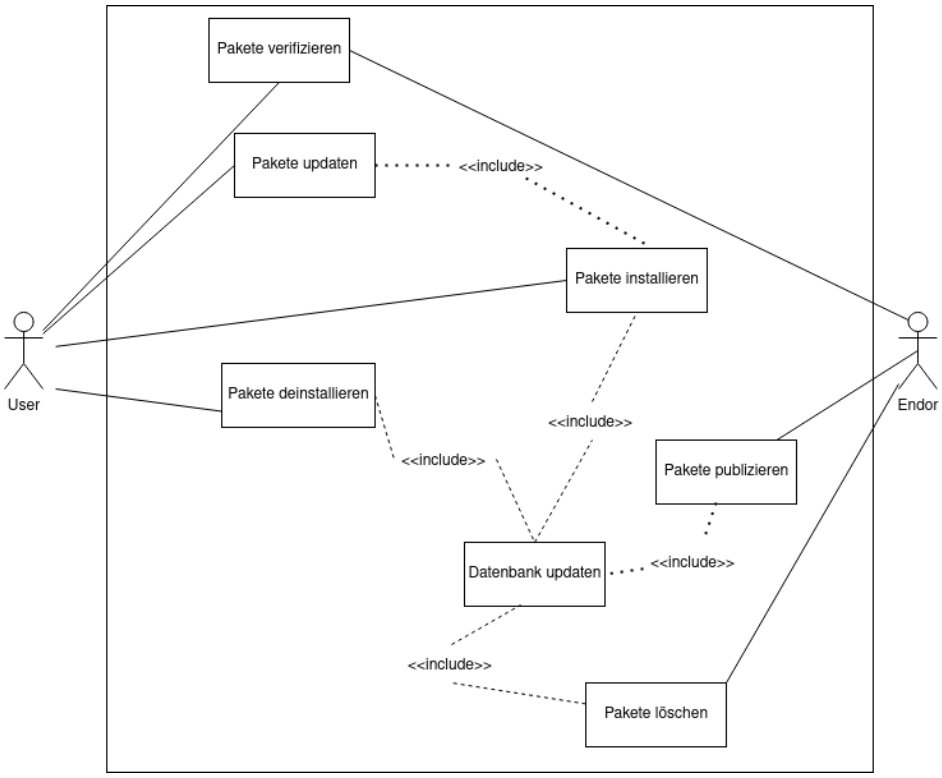

Abbildung 3.3: Use-Case-Diagramm

4 UML

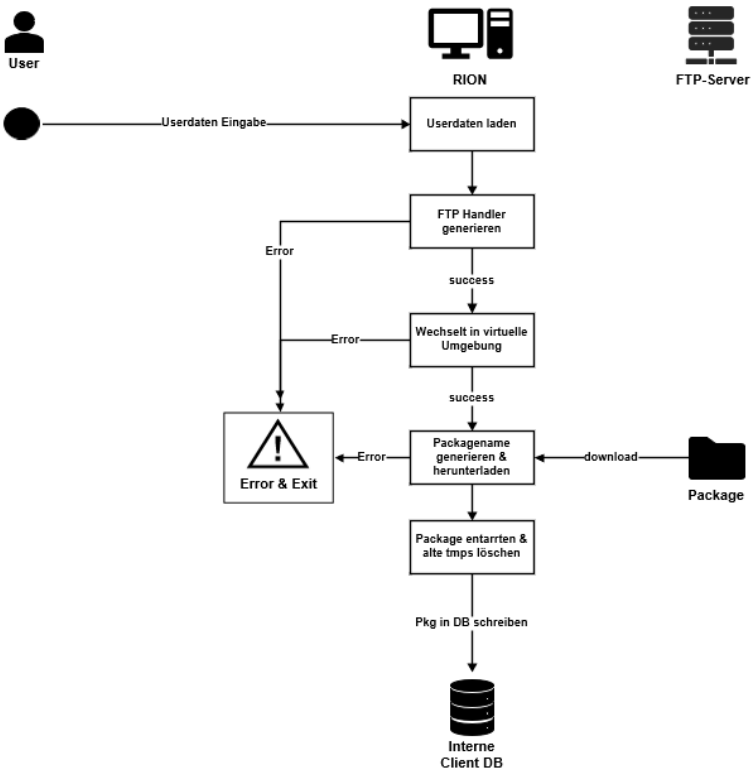

Abbildung 4.1: uml

4.1 Doxygen

Die Ausführliche von Doxygen generierte Dokumentation ist im Anhang zu finden.

5 Bug Report

5.1 Unclear error message

- Anpassen der Error Messages.
  - alt: Missing Input
  - neu: Missing login credentials. Please enter them in the config file or with "rion login"
- : GitHub Issue #23:`https://github.com/Riffecs/rion/issues/23`
- Status: Fixed, Closed

5.2 Wrong Syntax

- Erlauben von Sonderzeichen bei Passwörtern.
  - alt: Keine Sonderzeichen erlaube
  - neu: erlaubt Sonderzeichen
- : GitHub Issue #20:`https://github.com/Riffecs/rion/issues/20`
- Status: Fixed, Closed

5.3 Delete the name from the licence

- Entfernen der Namen.
  - alt: Namen stehen in der Lizenz
  - neu: Schaffung eines Templates zur Entfernung der Namen
- : GitHub Issue #16:`https://github.com/Riffecs/rion/issues/16`
- Status: Fixed, Closed

5.4 Error: Invalid Version

- Prüfen von Versionsnummern.
  - alt: Durch eine Diskrepanz in der Versionsnummer stürzte Rion immer ab
  - neu: Entfernen der Versionsprüfung
- : GitHub Issue #17:`https://github.com/Riffecs/rion/issues/17`
- Status: Fixed, Closed

Alle Issues können im Git im Reiter Issue [1] eingesehen werden.

5.5 Fehlende Anforderungen

- Abhänigkeitenmanagement

Das Abhänigkeitenmanagement wurde nicht implementiert, weil wir keinen für uns voraussichtlich, im Rahmen unserer Fähigkeiten, umsetzbaren Lösungsansatz gefunden haben.

[1] `https://github.com/Riffecs/rion/issues?q=is%3Aissue`

6 Testzenarien

T0110 *Installation:* Der Nutzer installiert mit „rion install *Packagename*“ ein Paket. Das Paket wird, inklusive eventueller Abhängigkeiten, installiert. Skripte (Post-Install-Prozessierung) werden erfolgreich ausgeführt.

Anforderung kann laut Blackboxtest nur teilweise erfüllt werden. Das Paket wird installier, es existiert aber keine Abhänigkeitsbehandlung

T0120 *Suchen:* Der Nutzer sucht mit „rion search *text*“ ein Paket und bekommt eine Liste von passenden Paketen, inklusive einer kurzen Beschreibung derer, ausgeben.

Anforderung wurde laut Blackboxtest erfüllt, wobei bei den Textpaketen keine Beschreibung vorliegt.

T0130 *Information:* Der Nutzer sucht mit „rion info *Packagename*“ Informationen zu einem Paket und bekommt diese detailliert zu einem Paket ausgeben.

Anforderung wurde noch nicht implementiert.

T0140 *Aktualisieren:* Der Nutzer aktualisiert mit „rion update“ alle installierten Pakete. Diese werden aktualisiert. Er aktualisiert mit „rion upgrade *Packagename*“ ein bestimmtes Paket, welches nun aktualisiert wird.

Anforderung wurde noch nicht implementiert, da das Abhängigkeitenmanagement fehlt.

T0150 *Entfernen:* Der Nutzer entfernt mit „rion remove *Packagename/s*“ ein oder mehrere Pakete, welche dadurch deinstalliert werden.

textbfAnforderung wurde laut Blackboxtest erfüllt.

T0160 *Anleitung:* Der Nutzer schreibt „man rion“, worauf ihm die Manpage zu RION angezeigt wird.

textbfAnforderung wurde laut Blackboxtest erfüllt.

T0170 *Installierte Pakete listen:* Der Nutzer gibt „*rion list (package_name_pattern)*“ ein. Er erhält Liste aller installierten Pakete, bzw. aller installierten Funktionen eines Paketes.

Anforderung wurde noch nicht implementiert.

T0180 Die Repositories, auf die RION zugreift, können mit einem config file angepasst werden.

Anforderung wurde laut Blackboxtest erfüllt.

T0190 RION kann mehrere virtuellen Umgebungen verwalten.

Anforderung wurde laut Blackboxtest erfüllt.

T0111 RION kann mit „rion check (*Packagename (Packageversion)*)“ überprüfen, ob bestimmte oder alle installierten Pakete noch korrekt installiert sind.

Anforderung wurde noch nich erfüllt.

T0121 RION kann mit „rion update“ die lokale Datenbank aktualisieren.

Anforderung wurde laut Blackboxtest erfüllt.

T0131 RION kann sich mit „rion auth -u »USER« -p »PWD«“ anmelden. RION kann sich auch über einen config file (nur für den Ersteller lesbar) anmelden. Sollte keine Anmeldung vorliegen, fragt RION beim ersten Aufruf nach den Anmeldedaten.

Anforderung wurde laut Blackboxtest erfüllt. Allerdings wurde die Syntax leicht angepasst.

7 Testfälle

7.1 Komponenten Test

- ___init___.py : erstellt einen Handler bleibt danach inaktiv bis Beendigung
- ___main___.py : erstellt einen Handler bleibt danach inaktiv bis Beendigung
- database.py : erstellt einen Handler bleibt danach inaktiv bis Beendigung
- errors.py : erstellt einen Handler bleibt danach inaktiv bis Beendigung
- ftp.py : erstellt einen Handler bleibt danach inaktiv bis Beendigung
- helper.py : erstellt einen Handler bleibt danach inaktiv bis Beendigung
- rion.py : erstellt einen Handler bleibt danach inaktiv bis Beendigung

7.2 Blackbox

- Uninstall funktion rion → deinstalliert rion
- Man rion → lädt und zeigt die man page
- rion Login funktion → user eingeloggt
- rion Update funktion → rion aktualisiert
- rion remove funktion → packet entfernt
- rion upgrade funktion → packet aktualisiert
- ion seach funktion → zeigt packete mit dem eingegebenen Namen/Flags

8 Kritische Betrachtung

Abschließend ergaben sich aus der Arbeit am Projekt „RION“ vielschichtige Erkenntnisse, die retrospektiv betrachtet werden müssen. Idealer Weise sollten dem Projekt gewisse Parameter in der Vorgehensweise vorausgesetzt sein, die für den Erfolg der Durchführung unseres Erachtens unabdingbar sind. Dazu gehören als Mindestvoraussetzung die termingerechte Gruppenbildung und eine grundsätzliche Themen-Affinität der Gruppenmitglieder. Einerseits, um den vorgeschriebenen Zeitrahmen optimal nutzen zu können und andererseits das Potenzial an Motivation und Kompetenzen in der höchsten Grade ausschöpfen zu können. Also braucht es eine gewisse sinnhafte Kompatibilität des Teams untereinander. Hierzu zählt jedenfalls ein annähernd gleicher Wissensstand bei grundlegenden projektimmanenten Inhalten. Je weniger fachlicher Aufholbedarf untereinander ausgeglichen werden muss, desto effizienter der Fortschritt der Gemeinschaftsarbeit. Hierbei wird das gegenseitige profitieren und dazulernen auf fachfremden Gebieten der Team-Mitglieder aus unterschiedlichen Bereichen nicht grundsätzlich in Abrede gestellt. Wohlgemerkt gilt die Kritik einem mangelnden gemeinsamen Grundwissen. Fachlich betrachtet meint dies bei vorliegendem Projekt, dass sich alle Teilnehmer mit der jeweils angewandten Programmiersprache, der UML, dem Ziel-Betriebssystem, dem genutzten Vorgehensmodell und sowie grundlegenden Programmierparadigmen wie der OOP auskennen müssten. Von größter Bedeutung für Erfolg oder Misserfolg des Projektes sind auch soziale Komponenten. Eine unbedingte Kommunikations- und Kooperations-Bereitschaft muss zwingend kleinster gemeinsamer Nenner im Team sein. Selbst fruchtbringende Alleingänge müssen dem Team transparent und diskutierbar gemacht werden. Teamwork muss Priorität haben, um gemeinsame Fortschritte zu generieren. Dazu bedarf es des unumstößlichen Willens eigene Fähigkeiten in den Dienst des Teams zu stellen und sich nicht nur selbst vordergründig profilieren zu wollen. Dass das Team nach außen hin als solches auftritt, muss dabei selbstverständlich sein. Zur Organisation der Teilnehmer ist es zwingend angeraten, dass zentrale Plattformen gewählt, bedient und regelmäßig und zuverlässig besucht werden – eine Grundkomponente effektiver Kommunikation! Ideal sollte schließlich das Ergebnis zielorientiert zu den gewünschten Ergebnissen führen.

In erster Linie müssen die Kundenanforderungen bedarfsgerecht erfüllt worden sein. Das heißt, das Ergebnis des Projektes muss voll funktionsfähig sein, was in diesem Fall scheiterte. Wünschenswert waren auch mögliche zusätzlich eingebrachte Funktionen, die im Verlauf des Projektes ebenfalls angedacht worden waren. Termintreu hätten abschließend alle Tests bestanden werden müssen und non-funktionale Anforderungen übertroffen werden können. Damit sind zum Beispiel Fragen von Sicherheit, Leistungsfähigkeit, Robustheit u.a. gemeint. In vorliegendem Fall konnten viele dieser Ansprüche aus verschiedensten Gründen nicht realisiert werden, unter anderem deshalb, da voraussetzende Parameter nicht gegeben waren.