Weiterentwicklung der GRAMOVIS-Werkzeuge zur interaktiven Anwendung von AC^2-Simulationscodes

GRS - 834

Gesellschaft für Anlagen-
und Reaktorsicherheit
(GRS) gGmbH

Weiterentwicklung der GRAMOVIS-Werkzeuge zur interaktiven Anwendung von AC²-Simulationscodes

Abschlussbericht

Daniel von der Cron
Tanja Eraerds
Jürgen Hartung
Joachim Herb
Volker Jacht
Josef Scheuer

September 2025

Anmerkung:

Das diesem Bericht zugrunde liegende Eigenforschungsvorhaben wurde mit Mitteln des Bundesministeriums für Umwelt, Klimaschutz, Naturschutz und nukleare Sicherheit (BMUKN) unter dem Förderkennzeichen UMRS1605 durchgeführt.

Die Verantwortung für den Inhalt dieser Veröffentlichung liegt bei der GRS.

Der Bericht gibt die Auffassung und Meinung der GRS wieder und muss nicht mit der Meinung des BMUKN übereinstimmen.

GRS - 834
ISBN 978-3-911727-27-3

Deskriptoren

AC^2, ADM, AGM, ATHLET, ATLASneo, ATM, GRAMOVIS, Reaktorsicherheit, Simulation, Visualisierung

Kurzfassung

Die computergestützte Modellierung und Simulation physikalischer Anlagen ist weiterhin zentral für das Verständnis komplexer Systemverhalten. Um der steigenden Komplexität bei Generierung, Ausführung und Analyse von Simulationen gerecht zu werden, mussten die vorhandenen Werkzeuge für Modellierung und Visualisierung im GRAMOVIS-Umfeld erneut modernisiert und erweitert werden. Im Rahmen des Projektes wurde die Weiterentwicklung der GRAMOVIS-Werkzeuge zur Nutzung der AC²-Rechencodes fortgeführt und vertieft. Dies umfasst sowohl den AC² Design Modeller (ADM) zur Eingabegenerierung als auch ATLASneo als zentrale Anwendung zur interaktiven Simulations-Steuerung und Ergebnisanalyse.

GRAMOVIS bietet domänenspezifische Werkzeuge und Methoden zur grafischen Modellbildung und Ergebnisauswertung. Diese unterstützen Fachanwender bei der Durchführung von Sicherheitsanalysen – insbesondere durch interaktive Eingabewerkzeuge für thermohydraulische und regelungstechnische Systeme, durch die Ausführung von Simulationen mit AC²-Rechencodes sowie durch moderne Visualisierungsfunktionen (Zeitreihenplots, dynamische Bilder, interaktive Datenanalyse).

Im aktuellen Projekt wurden die GRAMOVIS-Werkzeuge technisch und methodisch substanziell weiterentwickelt. Für ADM und die Eingabemodellierung wurden zentrale Wartungsarbeiten als auch Vorarbeiten zur möglichst umfangreichen Unterstützung von AC²-Rechencodes geleistet. Ein Schwerpunkt war dabei die Harmonisierung und Vereinheitlichung der Verarbeitung unterschiedlicher Eingabeformate, um dieselben Konzepte, internen Datenstrukturen und Verarbeitungsketten möglichst übergreifend für alle AC²-Codes nutzen zu können. Für ATLASneo wurden die Analyse- und Visualisierungsfunktionalitäten erweitert und modernisiert (u. a. neue Plot-Interaktionen, verbesserte Datensteuerung) und die Ausgabeformatunterstützung durch den Konverter x2hdf5 wurde weiter ausgebaut.

Darüber hinaus wurden wesentliche Grundlagen für nachhaltige Softwarequalität geschaffen: Die Migration auf aktuelle Python- und Qt-Versionen, strukturierte Environment-Definitionen, automatisierte Paketbereitstellung sowie die vollständige, automatisierte Ausführung AutoIt-basierter GUI-Tests in GitLab-CI-Umgebungen ohne grafische Oberfläche. Erst dadurch wird die konsequente Umsetzung kurzer Releasezyklen (*„release early, release often“*) realistisch umsetzbar und die Voraussetzung geschaffen, den regelmäßigen Austausch mit Anwendern und AC²-Teilprojekten zu gewährleisten.

Abstract

Computer-aided modelling and simulation of physical systems remains essential for understanding complex system behavior. In order to deal with the ever-increasing complexity in generating, executing, and analyzing simulations, the existing modelling and visualization tools in the GRAMOVIS toolchain had to be further modernized and expanded. Within this project, the ongoing enhancement of GRAMOVIS tools for the use of AC² computational codes was continued and enhanced. This includes both the AC² Design Modeller (ADM) for input generation and ATLASneo as the central application for interactive simulation control and result analysis.

GRAMOVIS provides domain-specific tools and methods for graphical model construction and result evaluation. These support specialist users in performing safety analyses – in particular through interactive input tools for thermohydraulic and control systems, through the execution of simulations with AC² computational codes, and through modern visualization capabilities (time-series plots, dynamic images, interactive data analysis).

In the present project, the GRAMOVIS tools were further developed on a substantial technical and methodological level. For ADM and input modelling, key maintenance work was carried out, as well as preparatory work towards broad support of AC² computational codes. A major focus was the harmonization and unification of processing for different input formats, in order to reuse the same concepts, internal data structures, and processing chains as comprehensively as possible for all AC² codes. For ATLASneo, analysis and visualization functionality were extended and modernized (including new plot interactions and improved data control), and output format support through the x2hdf5 converter was further expanded.

In addition, essential foundations for sustainable software quality were established: migration to current Python and Qt versions, structured environment definitions, automated package builds, as well as fully automated execution of AutoIt-based GUI tests in GitLab-CI environments without a graphical user interface. Only then the consistent implementation of short release cycles (“release early, release often”) become realistically feasible, what establishes the necessary preconditions for regular exchange with users and AC² sub-projects.

Inhaltsverzeichnis

Kurzfassung I

Abstract II

1 Einleitung 1

1.1 Gesamtziel 1

1.2 Wissenschaftliche und technische Arbeitsziele 3

1.2.1 Allgemeine Weiterentwicklung der Softwarekonzepte 3

1.2.2 Ausbau der Unterstützung von Rechencodes 4

1.2.3 Weiterentwicklung der Anwendungswerkzeuge 4

2 Stand von Wissenschaft und Technik 7

2.1 Internationale Entwicklungen 7

2.2 Qualifikationen und Referenzen der GRS 8

2.2.1 Modellierung und Input-Erstellung 9

2.2.2 Simulations-Durchführung 10

3 Arbeiten und Ergebnisse 13

3.1 AP 1: Modellierung und Input-Erstellung 15

3.1.1 Entwicklung Input-Parametrisierungs-System 16

3.1.2 Entwicklung Grammatik-basierter Eingabe-Analyse 19

3.1.3 Implementierung fortgeschrittener Eingabeverarbeitung 24

3.1.4 Vereinheitlichte Verarbeitung von Eingabeformaten 30

3.1.5 Verbesserung der Anwendbarkeit bisheriger Entwicklungen 31

3.1.6 Allgemeine Wartungsarbeiten 36

3.2 AP 2: Simulations-Durchführung 37

3.2.1 Auslagerung des Simulationsprozesses 38

3.2.2 Weiterentwicklung Simulations-Controller 42

3.2.3 Weiterentwicklung Monitoring und Simulationssteuerung 44

3.2.4 Verbesserung der Anwendbarkeit bisheriger Entwicklungen 47

3.2.5 Allgemeine Wartungsarbeiten 51

3.3 AP 3: Ergebnis-Analyse und Visualisierung 53
3.3.1 Erstellung Anwendungspaket 54
3.3.2 Ausbau Unterstützung Ausgabeformate 57
3.3.3 Anbindung an Jupyter-Notebooks 58
3.3.4 Weiterentwicklung generischer Datenvisualisierung 60
3.3.5 Verbesserung der Anwendbarkeit bisheriger Entwicklungen 65
3.3.6 Allgemeine Wartungsarbeiten 68
3.4 AP 4: Querschnittsaufgaben 70
3.4.1 Software-Infrastruktur 70
3.4.2 Projektmanagement 72

4 Zusammenfassung und Ausblick 75
4.1 Ausblick 79

Literaturverzeichnis 83

Abbildungsverzeichnis 87

Abkürzungsverzeichnis 89

Glossar 91

1 Einleitung

Im Feld von Forschung und Sicherheitsbewertung von Reaktoranlagen stellt die Simulation physikalischen Anlageverhaltens seit langem eine unverzichtbare Methode für die Nachweisverfahren zur Beherrschung von Transienten, Stör- und Unfällen (gem. /BMWi 21/) dar. Die Möglichkeiten der dabei zum Einsatz kommenden Rechencodes haben sich seit jeher ständig erhöht und erreichen mit jedem Entwicklungsschritt erweiterte Möglichkeiten im räumlichen und zeitlichen Detaillierungsgrad der durchführbaren Simulationen. Die hierfür notwendige Erstellung von Eingabedaten, die Auswertung der zum Teil sehr hoch aufgelösten Ergebnisse sowie das Verstehen der aufgetretenen Phänomene sind ohne zusätzliche Werkzeuge praktisch unmöglich. Aus diesem Grund zielte dieses Vorhaben darauf ab, für Simulationsprogramme der Reaktorsicherheit unterstützende und soweit möglich auch zeitgemäße Werkzeuge bereitzustellen, die den Benutzer bei der Anwendung der Rechencodes unterstützen, fehleranfällige Arbeitsschritte weitgehend automatisieren und den Erkenntnisgewinn aus den durchgeführten Simulationen maximieren.

1.1 Gesamtziel

Das Gesamtziel der GRAMOVIS-Vorhaben besteht darin, für diese Simulationsprogramme eine zeitgemäße Anwendungsumgebung zur Modellierung, die Erstellung von Eingabedaten und die Auswertung von Simulationsergebnissen im GRS-eigenen Interesse weiterzuentwickeln. Die physikalischen Rechencodes, welche von den Benutzern zur Nachbildung und Simulation von Anlagenverhalten angewendet werden, sind einer steten Weiterentwicklung unterworfen. Um physikalische Phänomene immer genauer nachbilden zu können, werden die der Simulation zugrunde gelegten Modelle ständig erweitert sowie ursprünglich eigenständig entwickelte Rechencodes gekoppelt, wie z. B. ATHLET/-CD und COCOSYS. Die GRS-Rechencodes, welche verwendet werden, um Vorgänge in Kernkraftwerken und deren Verhalten im Betrieb sowie bei Stör- und Unfällen zu simulieren, wurden mittlerweile als Softwarepaket AC² zusammengefasst, welches den Anwendern ein breites Spektrum an Simulationsmöglichkeiten bietet.

Allerdings steigt durch diese Code-Kopplungen und Modellerweiterungen nicht nur die Anzahl an Möglichkeiten, sondern auch die Komplexität im Aufbau der Rechencodes und in deren Anwendung. Gerade letztere fordert vom Benutzer mehr und mehr spezielle Kenntnisse über die korrekte Spezifizierung der angesprochenen Modelle und deren

Verhalten im Zusammenspiel untereinander (vgl. Abb. 1.1). Konkret zeigt sich diese Komplexität bei der Modellierung von Anlagen, bei der sämtliche Randbedingungen in Betracht gezogen werden müssen, um einen ablauffähigen Eingabedatensatz zu erhalten. Des Weiteren kann auch das korrekte Starten von Simulationen, was mitunter eine ganze Reihe von Argumenten und Eingabedateien erfordert, für ungeübte Anwender zur Herausforderung werden.

Besonders die nach, oder auch schon während der Simulation, durchgeführten Arbeitsschritte zur Auswertung von Ergebnissen gestalten sich oft nicht nur durch die Menge an entstehenden Ergebnisdaten sehr anspruchsvoll. Die als Zeitreihen vorliegenden Ergebnisse müssen in Relation zueinander gesetzt oder in einem geometrischen Kontext betrachtet werden, damit diese richtig interpretiert werden können. Gerade die hierfür eingesetzten Methoden zur Visualisierung können dabei oft nicht ohne Informationen über die zu Grunde gelegten Modelle arbeiten, sondern müssen ein gewisses Maß an Kenntnis über die simulierte Situation mitbringen, damit die Daten aussagekräftig dargestellt werden können. Daraus entsteht auch für die Anwendungswerkzeuge der Ergebnisanalyse die Notwendigkeit, die zum Einsatz gekommenen Modelle und Rechencodes unterstützen zu müssen, um für die Benutzer wirklich hilfreich zu sein.

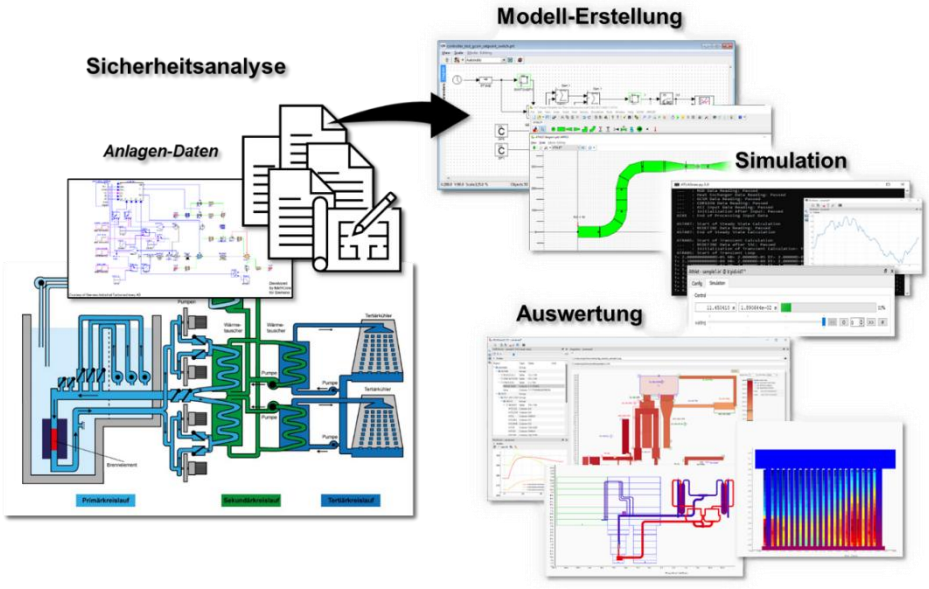

Abb. 1.1 Übersicht zum Einsatz von GRAMOVIS-Werkzeugen bei Sicherheitsanalysen

Um der Komplexität bei der Durchführung und Analyse von AC²-basierten Simulationen zu begegnen, müssen die vorhandenen Werkzeuge gepflegt, erweitert und an die jeweils aktuellen Anforderungen angepasst werden. Nach der Bereitstellung der im Rahmen des

Vorgängervorhabens /SCH 22/ entwickelten Werkzeuge, ergeben sich durch deren praktischen Einsatz neue Benutzeranforderungen und Verbesserungswünsche. Erst bei der Anwendung kristallisieren sich neue Workflows und Anwendungsfälle (Use Cases) heraus, welche bei der Planung wie des hier vorgestellten Vorhabens nur schwer vorherzusehen sind. Trotzdem musste der Aufwand auch für solche Arbeiten bereits von Anfang an in das Projekt miteinkalkuliert werden.

Darüber hinaus musste in den Projektzielen die allgemeine Weiterentwicklung in Konzepten der Softwareentwicklung berücksichtigt werden, welche eine stete Wartung und Anpassung von Schnittstellen und Maßnahmen zur Gewährleistung der Softwarequalität erfordern, um mit der allgemeinen Entwicklung schritthalten zu können. Nur so kann die Anwendbarkeit der Werkzeuge auch unter den sich kontinuierlich weiterentwickelnden Bedingungen erreicht werden, welche zum einen durch die sich verändernden Anforderungen von Rechencodes und Anwendern, zum anderen schon durch den steten Fortschritt von Betriebssystemen und Rechnerausstattung vorgegeben werden.

1.2 Wissenschaftliche und technische Arbeitsziele

Durch die breit gestreuten Anforderungen an das Projekt war es sinnvoll, die geplanten Entwicklungsarbeiten in die im Folgenden definierten Arbeitsziele zu unterteilen.

1.2.1 Allgemeine Weiterentwicklung der Softwarekonzepte

Das Projekt basiert auf bereits vorher bestehenden, meist unabhängig voneinander entwickelten Softwareteilen, welche weiterentwickelt werden mussten, um deren Zusammenarbeit zu gewährleisten oder zu verbessern. Gerade durch ihre in der Vergangenheit voneinander unabhängige Entwicklungsgeschichte mussten oft deren Schnittstellen besser aufeinander abgestimmt werden, damit eine einheitliche Anwendung möglich wurde. Durch die Erarbeitung einheitlicher Vorgehensweisen und generischer Schnittstellen ermöglichte dieses Projekt die Schaffung einer flexiblen Softwarebasis zur einfacheren Unterstützung alter Bestandscodes und Rechenprogramme, welche zukünftig durch die Bereitstellung von Bedienelementen einfacher in den typischen Arbeitsablauf der Anwender eingebunden werden können.

Im Rahmen dieser Weiterentwicklung musste auch die Wartbarkeit der Bestandscodes sichergestellt werden. Hierzu wurden in einzelnen Teilen verschiedene Maßnahmen zur

Steigerung der Softwarequalität, wie Umstrukturierung und Refactoring, notwendig, um deren Anpassung an die neuen Anforderungen zu ermöglichen. Auch längerfristig kann oft nur so die Erweiterbarkeit und Lauffähigkeit der Programme unter aktuellen bzw. zukünftigen Betriebssystemversionen erhalten werden.

1.2.2 Ausbau der Unterstützung von Rechencodes

Der Fokus des vorangegangenen GRAMOVIS-Projektes lag auf der Anwenderunterstützung bei der Durchführung und Ergebnisvisualisierung von AC²-Simulationen. Hierin sind neben ATHLET die Rechencodes ATHLET-CD (ATHLET Core-Degradation) und COCOSYS enthalten, welche in ATHLET durchgeführten Berechnungen um hierin nicht berücksichtigte Phänomene erweitern. COCOSYS wird von ATHLET weitgehend unabhängig entwickelt und kann aufgrund der unterschiedlichen Programmstruktur und des unterschiedlichen Datenformates von den auf ATHLET ausgerichteten Anwenderwerkzeugen bislang nur teilweise unterstützt werden. Somit war ein Ziel dieses Vorhabens, der Unterstützung möglichst aller AC²-Rechencodes näher zu kommen.

Inzwischen hat die Kopplung und die gemeinsame Anwendung der Rechencodes, maßgeblich auch durch die Spezifikation des Software-Pakets AC², stark an Bedeutung gewonnen und definiert Interoperabilität und Datenkompatibilität als zentrale Anforderung. Die in den Rechencodes vorhandenen Schnittstellen wurden und werden für die gemeinsame Anwendung als AC²-Softwarepaket weiter ausgebaut und schaffen so die Grundlage für eine breitere Unterstützung der Anwender bei Input-Erstellung, Simulationsdurchführung und Ergebnisanalyse bzw. Visualisierung.

1.2.3 Weiterentwicklung der Anwendungswerkzeuge

Die Anwendung der Codes birgt viele Bereiche, welche den Benutzern zum Teil sehr viel Detailwissen über die Eigenschaften der Codes abverlangen, damit Simulationen erfolgreich durchgeführt werden können. Hier versuchen die Anwendungswerkzeuge durch Zusammenstellung von Workflows oder auch Teilautomatisierung typischer Abläufe und durch intuitiv gestaltete graphische Benutzeroberflächen Hilfe anzubieten.

Um möglichst viele der inzwischen stark erweiterten Möglichkeiten der Rechencodes durch die Benutzerwerkzeuge unterstützen zu können, müssen diese stetig ausgebaut und immer wieder neu auf die Anwendungsschwerpunkte abgestimmt werden. Besonders durch die Verschiedenartigkeit der Rechencodes zeigt sich immer wieder, dass die

Anwendungswerkzeuge generell flexibler werden müssen, um mit der Entwicklung der Codes, den Anforderungen der Anwender sowie den allgemeinen Entwicklungen in den Betriebssystemen und der Software schritthalten zu können. Hieraus ergaben sich für das Projekt, gruppiert nach den typischen Arbeitsphasen von Sicherheitsanalysen, folgende Zielsetzungen:

- Modellierung und Input-Erstellung
  - Einheitliche Be- und Verarbeitung von Eingabeformaten
  - Ausbau der Unterstützung von AC²-Rechencodes
  - Wartung und Verbesserung der Anwendbarkeit bisheriger Entwicklungen
- Simulations-Durchführung
  - Weiterentwicklung von Kontroll- und Monitoring-Möglichkeiten
  - Wartung und Verbesserung der Anwendbarkeit bisheriger Entwicklungen
- Ergebnis-Analyse und Visualisierung
  - Ausbau der Unterstützung von Datenformaten und Anwender-Workflows
  - Weiterentwicklung generischer Datenvisualisierung
  - Wartung und Verbesserung der Anwendbarkeit bisheriger Entwicklungen

2 Stand von Wissenschaft und Technik

In diesem Abschnitt wird der Stand der Technik zum Projektbeginn im Bereich der vorhandenen Methoden zur Erzeugung der Daten für die Simulationsmodelle beschrieben. Hierbei werden Werkzeuge betrachtet, welche vor allem in Reaktorsicherheitsanalysen zur Modellierung vergleichbarer Simulationsmodelle genutzt werden.

2.1 Internationale Entwicklungen

Für die wichtigen US-NRC Codes TRACE, CONTAIN, COBRA, MELCOR, PARCS und RELAP5 ist SNAP, *„Symbolic Nuclear Analysis Package“* /APT 12/ verfügbar, welches für die Erstellung der Eingabedaten, Ausführung und Auswertung der Analysen verwendet werden kann. Neben der graphischen Eingabe wird auch der Import bestehender Datensätze unterstützt. Zusätzlich zur Standardsimulation sind Untersuchungen zur Sensitivität der Ergebnisse bei Parameteränderungen möglich. Die Visualisierung umfasst ähnliche 2D-Darstellungen wie sie in ATLAS möglich sind. SNAP ist als generelles, modulares Werkzeug konzipiert, in Java implementiert und kann mit „Plugins“ für andere Codes erweitert werden. SNAP ist plattformunabhängig, aber nicht OpenSource, d. h. es wird eine kostenpflichtige Lizenz für die Nutzung benötigt.

Für den Integralcode ASTEC wird von IRSN der „XASTEC Data Set Editor“ /IRS 14/ zur interaktiven Generierung der Eingabedaten entwickelt. XASTEC bietet ähnliche Funktionen und Methoden wie in SNAP und GRAMOVIS. Alle wichtigen Module von ASTEC werden unterstützt. XASTEC ist in Java implementiert und damit plattformübergreifend verfügbar. Bestehende ASTEC-Datensätze sollen importiert werden können. Von XASTEC aus können ASTEC-Rechnungen, die Ergebnisvisualisierung und ein Werkzeug für Unsicherheitsanalysen gestartet werden.

Darüber hinaus gibt es weltweit viele Ansätze zur Entwicklung von Unterstützungswerkzeugen, die sich mit der intuitiven Bedienbarkeit von digitalen Simulatoren beschäftigen, z. B. /BOR 12/ oder /WAL 17/. Als gemeinsamer Trend lässt sich hier eindeutig die Nachbildung der Bedienelemente aus Anlagenwarten durch digitale Kontrollelemente auf Gruppen von Computermonitoren ausmachen. Durch möglichst realitätsnahes Aussehen der Kontrollelemente und deren Anordnung in Funktionsgruppen, wie sie auch in reellen Anlagen vorkommen, wird versucht, die Verwendung der Werkzeuge zur Simulationssteuerung möglichst intuitiv zu gestalten. Ergänzend sind spezielle Prozess-

diagramme verfügbar, die mit dynamischen Daten die Vorgänge in der Simulation verständlich darstellen können. Hierdurch wird es auch möglich, diese Unterstützungswerkzeuge in Notfallzentren oder für Schulungen einzusetzen und Abläufe und Anlagenverhalten zu trainieren.

Ein weiteres in der Wissenschaft und Industrie weit verbreitetes OpenSource Werkzeug, welches eine graphische und auch textbasierte Modellierung von multiphysikalischen Prozessen erlaubt, ist (Open)Modelica. Mithilfe des frei verfügbaren GUI-Clients OMEdit können beliebig komplexe Systeme, z. B. integrale thermohydraulische und leittechnische Systeme wie bei einer kerntechnischen Anlage, relativ einfach und mit hoher Genauigkeit nachgebildet werden. Darüber hinaus können eigens geschriebene oder von Drittanwendern zur Verfügung gestellte Bibliotheken in Modelica importiert, bzw. exportiert werden; das ist besonders wichtig, wenn man spezielle Komponenten- oder Systemmodelle, die nicht in der Standardbibliothek enthalten sind, verwenden möchte. Insgesamt betrachtet, eignet sich Modelica wegen seiner breiten Anwendungsmöglichkeiten sowie der internationalen Gemeinschaft von Anwendern und Entwicklern aus verschiedenen Themengebieten /MOD 25/, hervorragend für wissenschaftlich-technische Anwendungen auf simulativer Basis.

2.2 Qualifikationen und Referenzen der GRS

Die bisherigen GRS-Entwicklungen von Anwenderwerkzeugen orientierten sich an den Anforderungen, die durch die Rechencodes des AC²-Pakets /AC2 19/ vorgegeben werden. Die folgenden Abschnitte beschreiben kurz die Arbeitsbereiche, in denen AC²-Anwender aktuell durch Werkzeuge unterstützt werden, welche in vorangegangenen Vorhaben (RS1537, RS1572) entwickelt wurden. Diese Entwicklungen umfassten den AC² Design Modeller (ADM) zur Eingabeerstellung sowie ATLASneo zur Simulationssteuerung und Ergebnisvisualisierung /SCH 22/. Bei ATLASneo handelt es sich um eine plattformunabhängige, auf modernen Technologien (Python /PYT 25/ und Qt /QTC 25/) basierende Weiterentwicklung der langjährig entwickelten ATLAS-Software /VOG 18/, die zunächst zusätzliche Funktionalität bereitstellt und langfristig die vollständige Ablösung von ATLAS vorbereitet. ATLASneo stellt bereits einige Funktionsmodule zur Online-Simulation und Ergebnis-Analyse zur Verfügung und wird von GRS-internen als auch ersten externen Anwendern verwendet.

2.2.1 Modellierung und Input-Erstellung

Nachdem der AC² Design Modeller (ADM) mit den darin enthaltenen Modulen *„ATHLET GCSM Modeller“* (AGM) und *„ATHLET Thermohydraulic Modeller“* (ATM) einen testbaren Entwicklungsstand erreicht hatte, wurde dieser im Entwicklungszeitraum des Vorgängerprojekts /SCH 22/ bereits für erste praktische Anwendungen eingesetzt (siehe Abb. 2.1). Hierbei zeigten die bisherigen Erfahrungen aber, dass dringend weitere Verbesserungen und Bugfixes benötigt wurden. Um die Verifikation der Applikationsfunktionalität und die Richtigkeit der mit ADM erzeugten Datensätze schon während der Entwicklungszeit zu gewährleisten, wurde in enger Zusammenarbeit mit den Anwendern ein Test-Workflow erarbeitet. Dieser gliederte sich wie folgt:

1. Vollständiger Import bestehender Beispiel-Datensätze und die Erzeugung der dazugehörigen graphischen Blöcke in ADM
2. Generierung des Datensatzes mit Hilfe von ADM ohne Modifikation
3. Textueller Vergleich zwischen generiertem und importiertem Datensatz und das Testen der Funktionalität durch ATHLET
4. Modifikation der ATHLET-Modellparameter in ADM
5. Generierung des modifizierten Datensatzes und Testen durch ATHLET

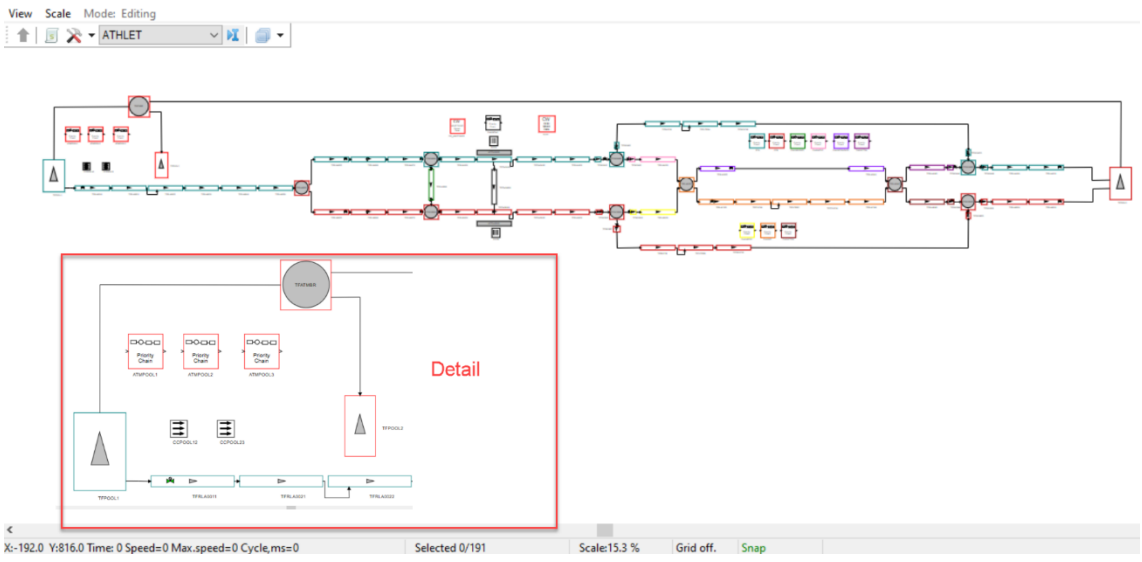

Abb. 2.1 Thermohydraulik-Modellierung einer Versuchsanlage in ADM

Durch dieses Vorgehen konnte der bestehende Source-Code vervollständigt bzw. ergänzt, der Import von alten ATHLET-Datensätzen generalisiert und somit einige Lücken

geschlossen werden. Die davor benötigte manuelle Nachbearbeitung der durch ADM erzeugten Datensätze konnte somit deutlich reduziert werden. Auch alle relevanten und von den Anwendern benötigten ATHLET-Modelle wurden für die interaktive Erstellung im ATM-Modul bereitgestellt. Die Export-Funktionalität wurde für Modelle bzw. graphische Blöcke ähnlicher Beschaffenheit weithingehend überholt, um eine vereinheitlichte und wartungsarme Methode zur Generierung von Datensätzen zu gewährleisten.

2.2.2 Simulations-Durchführung

Wie sich durch den vielfachen, praktischen Einsatz der Online-Simulation in ATLASneo gezeigt hatte, kann deren Durchführung sehr gut durch eine graphische Benutzeroberfläche unterstützt werden. Die Online-Simulation ersetzt das manuelle Starten der Rechnung auf der Konsole und hilft dem Anwender, indem sich der zu verwendende Rechencode sowie die nötigen Eingabedateien und -Parameter abfragen bzw. auswählen lassen. Die hierdurch gestartete Simulation kann anschließend überwacht und gesteuert werden. Das bisher entwickelte Steuermodul für ATLASneo realisiert die Möglichkeiten für Zugriff und Steuerung von Simulation auf Basis von Python-Controller-Klassen, die den Rechencode als „shared library“ (DLL/SO) einladen, starten und während der Simulation direkt auf deren Zustand zugreifen können (Online-Simulation). Im Gegensatz zur stand-alone Simulation, welche auch über die AC²-GUI gestartet werden kann, können hiermit Simulationsläufe sehr detailliert überwacht und gesteuert werden. Zudem wurden über die Controller-Klassen erste Möglichkeiten geschaffen, Simulationen in eigenen Prozessen zu starten und deren Ablauf über Netzwerk zu kontrollieren.

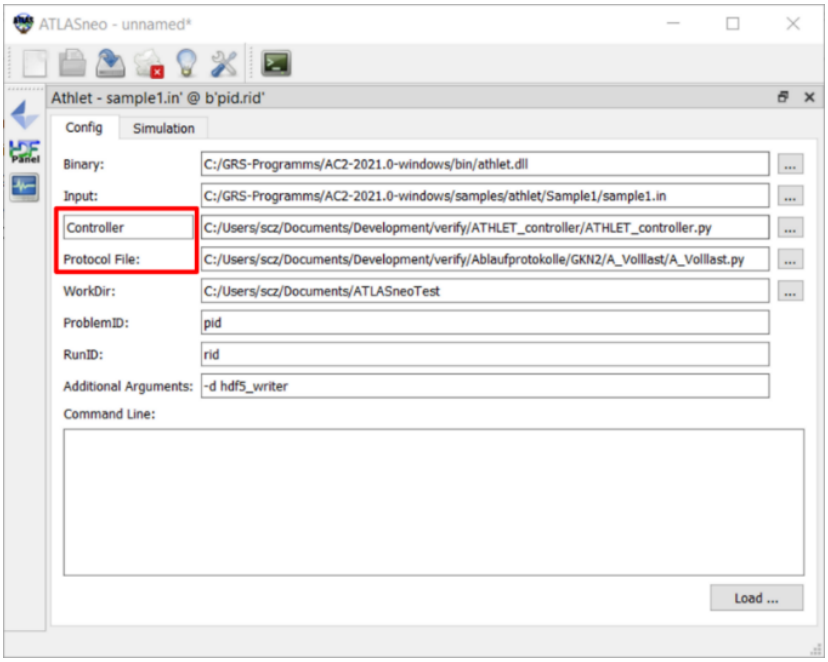

Abb. 2.2 Online-Simulation in ATLASneo. Neben der Dateiauswahl über Dialoge ist die Eingabe von Parametern (rot) zur Protokoll-gesteuerten Simulation möglich

Der bereits erreichte Entwicklungsstand von ATLASneo bietet eine Rahmenanwendung für sehr unterschiedliche Funktionsmodule und erlaubt so, die Benutzer bei verschiedensten Aufgaben innerhalb einer Applikation zu unterstützen. Neben dem Steuermodul zur Durchführung von Online-Simulationen (siehe Abb. 2.3) wurden bisher auch einige Module zur Analyse und Visualisierung der Ergebnisse entwickelt. Mit dem langfristigen Ziel, die in ATLAS vorhandene Funktionalität auch in ATLASneo bereitzustellen, wurden vorwiegend Module zur Trenddarstellung, zur Darstellung dynamisierter Bilder und zur Topologie-Visualisierung von Leittechnik-Netzwerken entworfen. Auch diese sollten im Rahmen des vorliegenden Projektes gepflegt und nach Möglichkeit gemäß neuer Benutzer- und auch Software-Anforderungen angepasst werden.

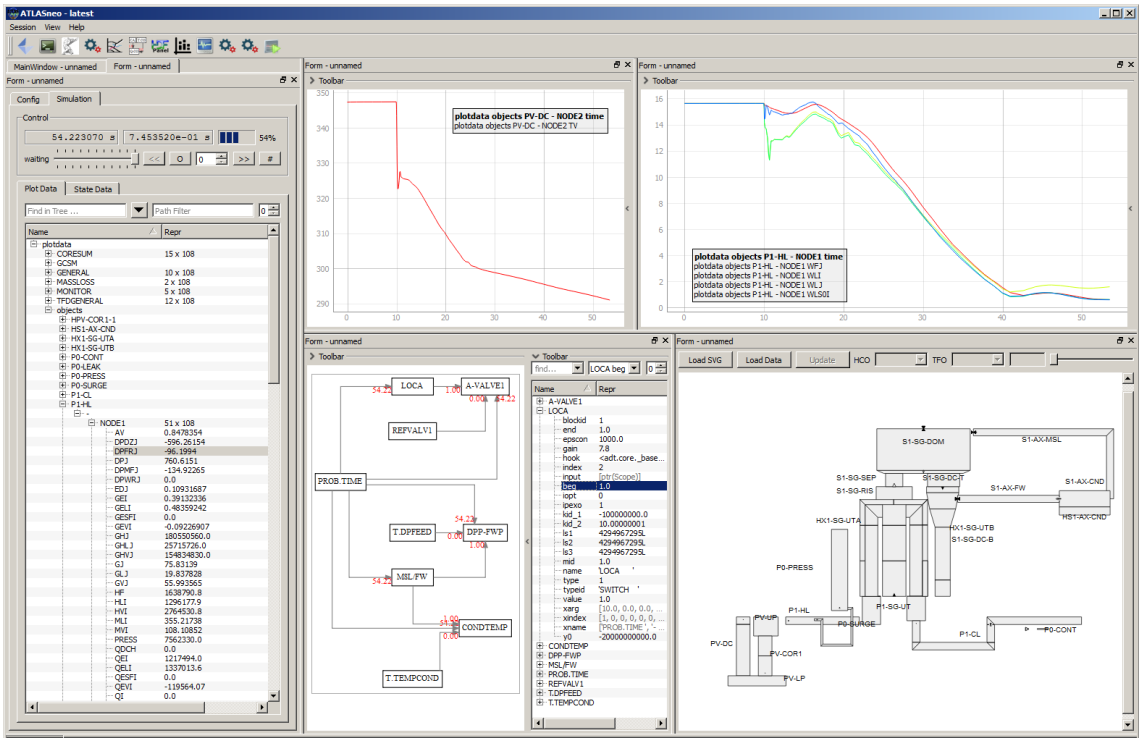

Abb. 2.3 Beispielanwendung einer frühen Version von ATLASneo und der entworfenen Funktionsmodule zur Online-Steuerung und zur Darstellung von Zeitreihen, Leittechnik-Netzwerken (GCSM) sowie der erste Prototyp dynamischer Bilder

3 Arbeiten und Ergebnisse

Das primäre Ziel der GRAMOVIS-Projekte ist es, den Anwender von AC² bei den Arbeiten zur Erstellung von Sicherheitsanalysen zu unterstützen und das typische Vorgehen (Workflow) durch Werkzeuge möglichst effizient zu gestalten. Durch die immer neuen Anforderungen in den Anwenderprojekten zur Analyse von Anlageverhalten war es auch im vorliegenden Projekt erforderlich, die hier eingesetzten Werkzeuge kontinuierlich weiterzuentwickeln und auf die Bedürfnisse der Anwender auszurichten.

Um in den durch GRAMOVIS bereitgestellten Werkzeugen die AC²-Rechencodes unterstützen zu können, müssen alle Software-Teile zusammenarbeiten und aufeinander abgestimmt sein. Zum Erreichen dieses Ziels wurden die drei typischen Phasen, **Input-Erstellung**, **Simulations-Durchführung** und **Ergebnis-Analyse** definiert (siehe Abb. 3.1), welche den Standard-Workflow in einer praktischen Anwendung beschreiben.

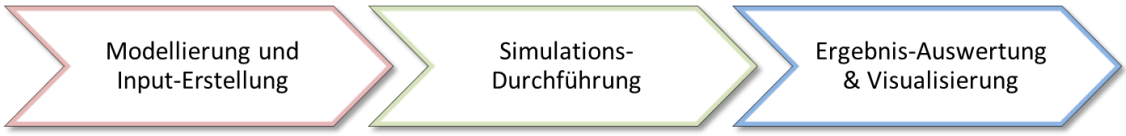

Abb. 3.1 Typische Phasen bei der Erstellung von Sicherheitsanalysen

In Abb. 3.2 wurden diesen Phasen die hierin zum Einsatz kommenden Werkzeuge zugeordnet. Die hier blau hervorgehobenen Werkzeuge wurden in den jeweils dargestellten Arbeitspunkten implementiert bzw. weiterentwickelt.

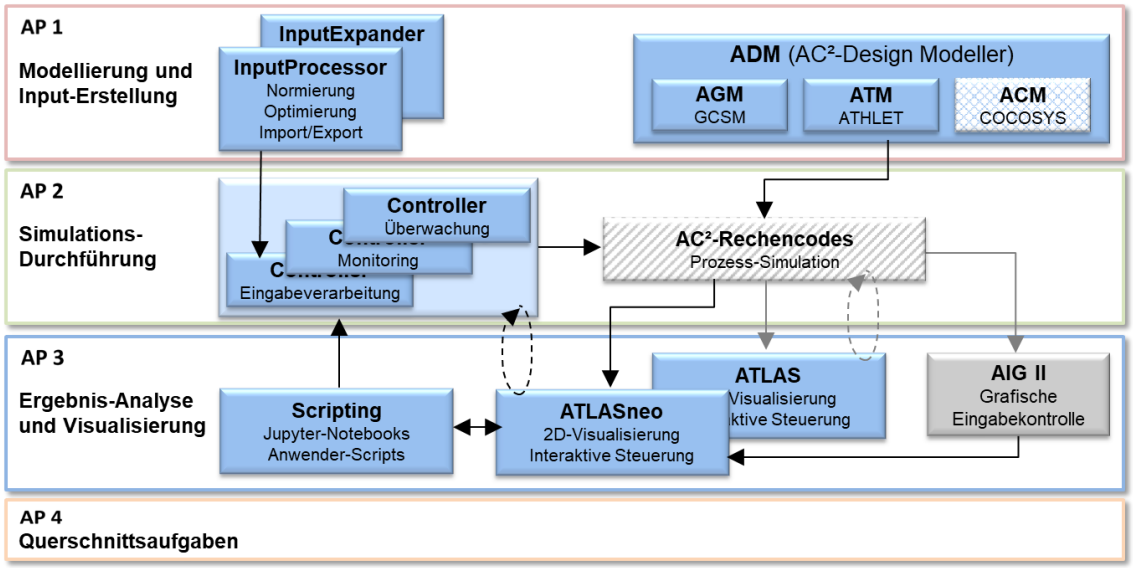

Abb. 3.2 Zusammenspiel von GRAMOVIS-Werkzeugen und ihr typisches Einsatzgebiet

Bereits im Vorgängerprojekt wurden die Konzepte erarbeitet, wie eine Zusammenarbeit der AC²-Codes über einheitliche Schnittstellen zum Starten, zur Steuerung und zum Datenzugriff realisiert werden kann. Neben den Entwicklungsarbeiten der einzelnen Werkzeuge war auch die konkrete Weiterentwicklung dieser Schnittstellen ein wichtiger Bestandteil des vorliegenden Projekts und wurde in den einzelnen Arbeitspunkten berücksichtigt:

- **Modellierung und Input-Erstellung**
  Um eine grundlegende Unterstützung bei der Input-Erstellung anbieten zu können, sollte eine Vereinheitlichung und Flexibilisierung in der Be- und Verarbeitung der Eingabedateien erreicht werden. Die dazu realisierten Arbeiten werden im *Arbeitspunkt 1* (Abschnitt 3.1) beschrieben.

- **Simulations-Durchführung**
  Sowohl das einheitliche Starten von Simulationsläufen als auch deren Überwachung und ggf. Steuerung soll durch ein leicht bedienbares User-Interface ermöglicht werden. *Arbeitspunkt 2* (Abschnitt 3.2) fasst die hierfür durchgeführten Arbeiten und Weiterentwicklungen zusammen.

- **Ergebnis-Analyse und Visualisierung**
  Die Anwendbarkeit der Werkzeuge zur Analyse und Visualisierung von Rechenergebnissen muss durch ein einheitliches Datenformat gewährleistet werden. Die Arbeiten zur weiteren Vereinheitlichung der Datenformate und die Weiterentwicklungen der Analysewerkzeuge finden sich in *Arbeitspunkt 3* (Abschnitt 3.3).

- **Querschnittsaufgaben**
  Die notwendigen Weiterentwicklungen der Software-Infrastruktur wurden im *Arbeitspunkt 4* zusammengefasst. Die hierzu geleisteten Arbeiten und deren Ergebnisse werden im Abschnitt 3.4 beschrieben.

Ergänzend zu den expliziten Weiterentwicklungen der Werkzeuge und ihrer Schnittstellen war ein weiteres Projektziel die Aufbereitung der bisherigen Entwicklungen, um eine Auskopplung der einsatztauglichen Werkzeuge für die effiziente Nutzung erstellen und diese auch externen Anwendern bereitstellen zu können (vgl. Abschnitt 3.3.1).

3.1 AP 1: Modellierung und Input-Erstellung

Um den Modell-basierten Ansatz zur Erstellung von Input-Datensätzen im AC² Design Modeller (ADM) zuverlässig und weitreichend anwendbar zu machen, sollte der früher erarbeitete Test-Workflow genutzt und ADM in enger Absprache mit Entwicklern und Anwendern verbessert werden.

Grundsätzlich kann die Anwendbarkeit von Modellierungswerkzeugen maßgeblich dadurch gesteigert werden, dass auch bereits bestehende Inputs weiterverarbeitet werden können. Somit ist in allen Werkzeugen die Import-Funktionalität ein unerlässliches Element, welches in der weiteren Entwicklung priorisiert werden muss. In diesem Vorhaben sollte deshalb eine Vereinheitlichung und Flexibilisierung in der Be- und Verarbeitung der Eingabedateien erreicht werden.

Hierzu war geplant, mittels Python-Packages, wie *PLY* /PLY 18/ oder *Lark* /LARK 24/ eine generelle Import-Funktionalität, modularisiert und generisch, zu realisieren. Hierbei sollte durch die Umformung eingelesener Formate in eine abstrakte Datenstruktur (AST: Abstract Syntax Tree) eine getrennte Behandlung von Ein- und Ausgabeformat ermöglicht und somit eine einfache und flexible Unterstützung verschiedener Formate realisiert werden. Da diese Art der Vorverarbeitung keine Eingriffe in die eigentlichen Rechencodes und deren Einleseroutinen erfordert, können auch alte Bestands- und Beispieldatensätze nutzbar gehalten werden und müssen nicht auf ein neues Format übertragen werden.

Die nachfolgend aufgeführten Teilarbeitspunkte verfolgten zum einen das Ziel, die Anwendbarkeit des aktuellen Entwicklungsstands von ADM zu verbessern. Zum anderen sollten die neu zu entwickelnden Funktionalitäten so gestaltet werden, dass sie auch im Falle einer möglichen Neuausrichtung der Entwicklung – unabhängig vom Basissystem *SimInTech* /SIT 21/ – für die Modellierung und Eingabeerstellung nutzbar bleiben.

3.1.1 Entwicklung Input-Parametrisierungs-System

Um Anwender bei der direkten Arbeit an Eingabedateien zu unterstützen, sollte ein Eingabe-Präprozessor zur Parametrisierung von zu generierenden Objekten entwickelt werden. Da in der Praxis auch mit Rechencodes und Eingabedatensätzen außerhalb des GRS-Umfelds gearbeitet wird, verfolgt das System einen allgemeinen Ansatz zur Modifikation textbasierter Input-Dateien. Ziel war die Bereitstellung einer flexiblen und plattformunabhängigen Anwendung, die eine vereinfachte und flexible Erstellung von Eingabedaten ermöglichte. Hierzu werden im Input-File die zu parametrisierenden Abschnitte mit sogenannten *„Tags“* markiert, welche Variablen oder auch komplexe Ausdrücke enthalten können. Durch eine leicht verständliche Syntax sollten sich sowohl gängige Programmstrukturen (z. B. Schleifen oder Bedingungen) als auch vordefinierte Funktionen zur Erzeugung komplexerer Eingabestrukturen verwenden lassen. Das System sollte dem Anwender erlauben, bestehende Datensätze effizient zu parametrisieren und für unterschiedliche Anwendungsszenarien wiederzuverwenden. Die folgende Abbildung (Abb. 3.3) veranschaulicht die Funktionsweise:

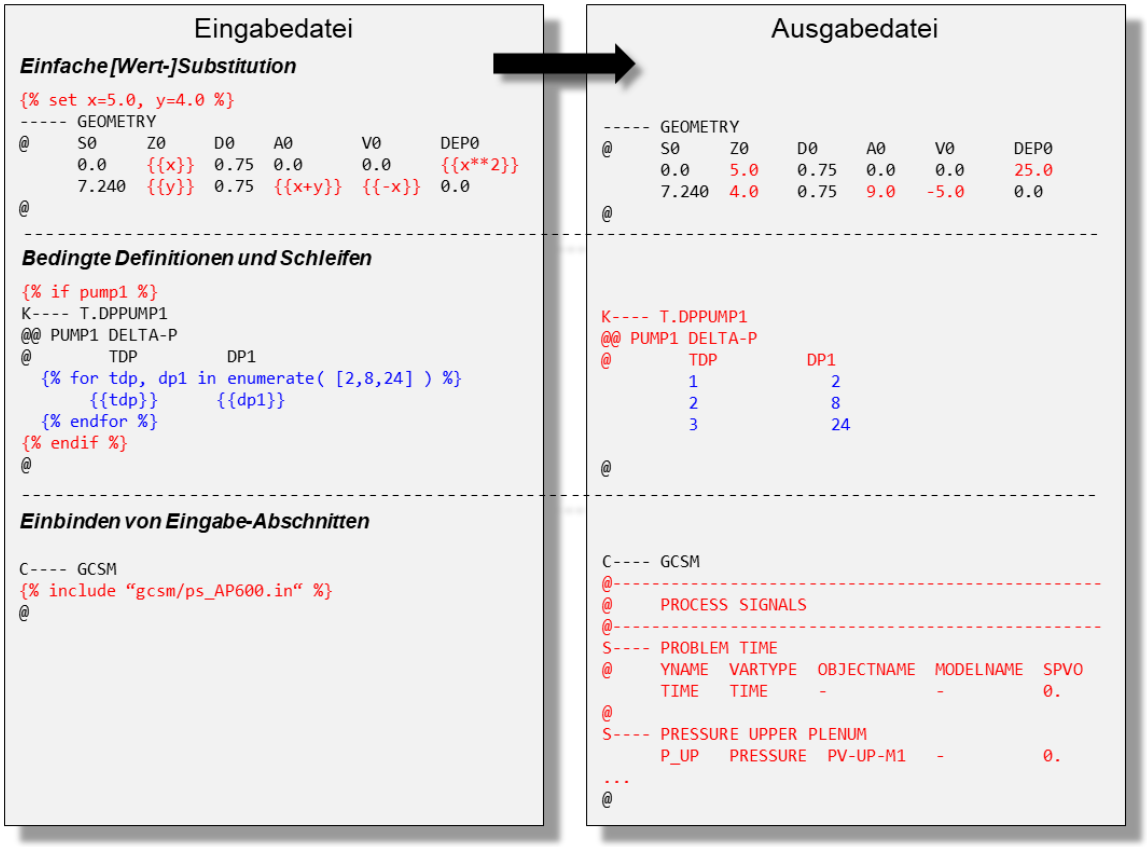

Abb. 3.3 Grundlegende Funktionsweise des Input Parametrisierungs-Systems

Die im Arbeitspaket entwickelte Funktionalität ermöglicht somit eine automatisierte Anpassung von Eingabedateien für die AC²-Codes ATHLET, ATHLET-CD und COCOSYS und kann sowohl in beliebigen Texteditoren als auch in ADM zur Erstellung von Makro-Bauteilen eingesetzt werden.

Konzeption und Technologieauswahl

Im Projektverlauf wurde das ursprüngliche Konzept präzisiert, die Implementierungsstrategie festgelegt und auf Basis von Python und der Template-Engine *Jinja* eine Kommandozeilen-Anwendung *InputExpander* entwickelt. Gestützt durch Recherchen zu potenziell einsetzbaren Bibliotheken sowie erste Tests, wurde Python /PYT 25/ als für die hier notwendigen Anforderungen am besten geeignete Programmiersprache gewählt. Die für die Umformung der Eingabe-Dateien notwendige Verarbeitung von Textblöcken und dem Ersetzen funktionaler Textmarken (Tags) kann bereits durch die in Python eingebauten Funktionalität realisiert werden. Daneben ist hierdurch auch die Auswertung mathematischer Ausdrücke bereits abgedeckt. Hierzu passen sowohl Funktionsumfang und Syntax der Template-Engine *Jinja* sehr gut, da hierin bereits eine sehr große Bandbreite an grundlegenden Textmarken enthalten ist.

Basis-Anwendung und Kommandozeilen-Interface

Zur einfachen und flexiblen Nutzung des Input-Parametrisierungs-Systems wurde, wie geplant, eine Kommandozeilen-basierte Python-Anwendung erstellt. Diese wird über das implementierte Command-Line Interface unter Angabe der zu verarbeitenden Eingabedateien gestartet und kann über Argumente und Optionen gesteuert werden (siehe Abb. 3.4). Deren Bedeutung und eine Kurzbeschreibung ihrer Funktion ist über eine Hilfe-Seite verfügbar.

```
inputexpander > python inputexpander  --help
Usage: expander [OPTIONS] [INFILES]...

Options:
  -O, --outfile PATH             File or  dir to which output gets written:  -
                                  = stdout (default)
  --assume-dir                    Try creating  --outfile as directory and use
                                  it for writing output file/s.
  -m, --outmode [override|append ]
                                  Mode for writing output file/s: override
                                  (default), append = append to  outfile/s
  -P, --path PATH                 Path for looking up template files,
                                  repeatable.
  -s, --set TEXT                 Define realization set, repeatable. e.g.
                                  --set "x=1; y=2"  --set "x=2; y=3"
  -E, --on-error [stop|next|ignore ]
                                  sets how to go on after a parsing error:
                                  `stop` (default), process `next` file (exit
                                  with errorcode) or `ignore` like next but
                                  exit without  errorcode.
  --help                         Show this message and exit.
```

Abb. 3.4 Das Commandline-Interface (CLI) und die Optionen des InputExpanders

Die wichtigsten Optionen sind im Folgenden kurz zusammengefasst:

- Die **`--outfile`** und **`--outmod`** Optionen steuern die Ausgabe der erzeugten Eingabe-Datensätze. Diese werden standardmäßig in eigene Dateien ausgegeben, können z. B. aber auch in einem bestimmten Zielverzeichnis oder in einer einzigen Datei gesammelt werden. Auch die Umlenkung nach Standard-Out wurde implementiert, was anderen Werkzeugen die direkte Weiterverarbeitung des Ausgabestroms erlaubt.

- Die **`--set`** Option erlaubt die Angabe der zum Verarbeiten der Templates und der Erzeugung der Eingabe-Datensätzen notwendigen Parameter. Mehrere Parameter können dabei als String zusammengefasst angegeben werden. Durch die Mehrfachangabe der **`--set`** Option können mehrere Realisierungen (d. h. Eingabe-Datensätze) auf einmal erzeugt werden.

- Die **`--on-error`** Option erlaubt das Verhalten des InputExpanders im Fehlerfall festzulegen. Hier stehen bisher die Optionen `stop`, `next`, `ignore` zur Verfügung und erweisen sich besonders bei der Verarbeitung vieler Eingabedateien in einem Aufruf als sehr nützlich.

Zur Sicherstellung der Qualität und der erreichten Stabilität des InputExpanders wurden in der CI automatisch durchgeführte Testprozeduren erstellt, welche im weiteren Verlauf der Entwicklung kontinuierlich gepflegt und ausgebaut werden. Zusammen mit internen Anwendern wurden die im Weiteren zu implementierenden funktionalen Textmarken

(Tags) und eine mögliche Anbindung an Protokoll-Definitionen für geskriptete Handmaßnahmen diskutiert und für eine zukünftige Weiterentwicklung als Issue festgehalten.

3.1.2 Entwicklung Grammatik-basierter Eingabe-Analyse

Im Rahmen dieses Arbeitspakets wurde ein Konzept zur Grammatik-basierten Analyse und Verarbeitung von Eingabedatensätzen der AC²-Rechencodes (siehe Abb. 3.5) entwickelt und umgesetzt. Ziel war die Schaffung einer generischen, automatisierten Methode zum Einlesen, Strukturieren und Verarbeiten textbasierter Eingabedateien, um die darin enthaltenen Informationen vollständig und konsistent maschinell erfassen zu können. Der bisher in ADM implementierte Import-Mechanismus sollte dazu generalisiert und so gestaltet werden, dass er auch von anderen Werkzeugen genutzt werden kann. Ausgehend von den typischen Merkmalen der ASCII-basierten Eingabeformate der AC²-Codes wurden die grundlegenden Sprachkonstrukte – wie Include-Mechanismen, mögliche Token-Arten oder der Umgang mit Kommentaren – identifiziert und formal beschrieben. Der strukturelle Aufbau von Eingabedatensätzen sollte in Form von formalen, durch Parser-Generatoren nutzbaren Grammatiken (z. B. für ATHLET, ATHLET-CD und COCOSYS) spezifiziert werden. Diese sollten dann von Python-Bibliotheken wie *PLY* oder *Lark* genutzt werden, um die Eingaben in syntaktische Elemente zu zerlegen und in Form einer hierarchischen Datenstruktur (AST: Abstract Syntax Tree) abzulegen. Diese Struktur bildet die Grundlage für weiterführende Verarbeitungsmethoden (siehe 3.1.3), wie etwa Konsistenzprüfungen, Definitionsabfragen oder die gezielte Modifikation, wie die Normierung oder auch Optimierung von Eingabedatensätzen. Abb. 3.5 stellt dieses Konzept schematisch dar:

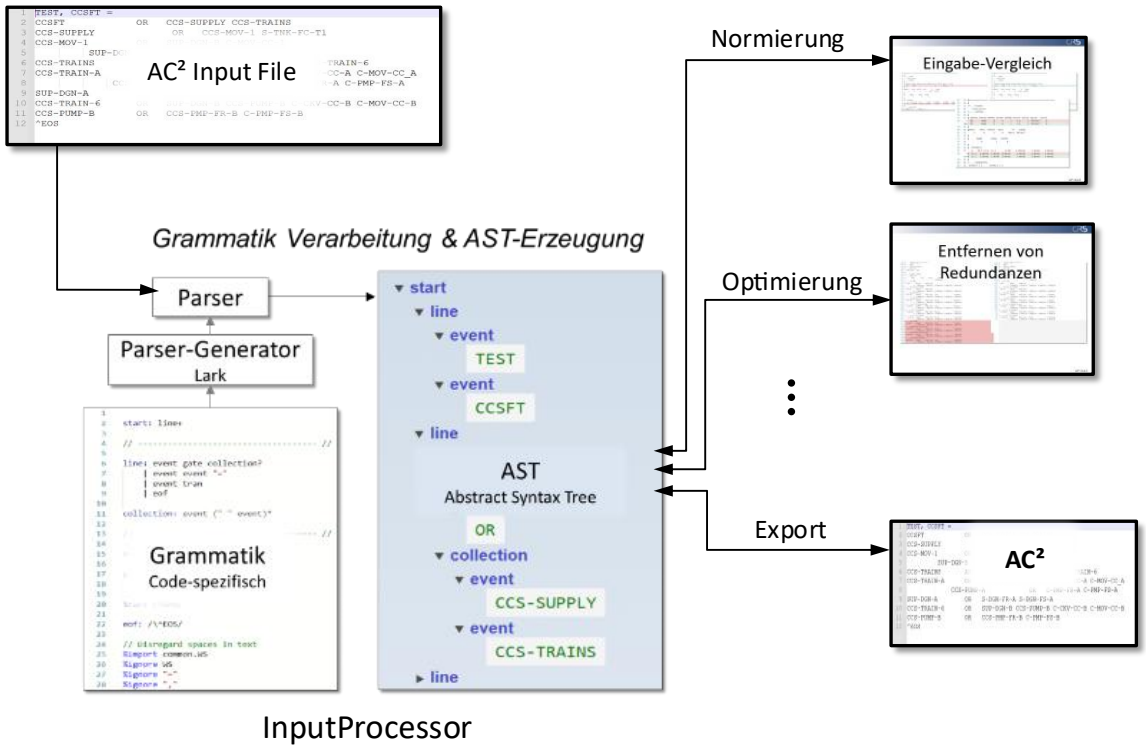

Abb. 3.5 Konzept der Grammatik-basierten Eingabe-Analyse

Konzeption und Technologieauswahl

Zu Beginn des Arbeitspakets wurde das Grundkonzept des Eingabe-Analysesystems ausgearbeitet und präzisiert. Ziel war die Definition einer flexiblen Architektur, die sowohl für die AC²-Rechencodes als auch für externe textbasierte Simulationssysteme genutzt werden kann. Als Programmiersprache zur Implementierung der Systemkomponenten wurde Python /PYT 25/ gewählt, das aufgrund seiner plattformunabhängigen Laufzeitumgebung, der umfangreichen Standardbibliothek und der hohen Flexibilität bei der Textverarbeitung eine ideale Basis bietet. In einer eingehenden Recherche zu potenziell geeigneten Python-Paketen für die formale Sprachanalyse und Parser-Generierung wurden verschiedene Bibliotheken untersucht. Im Fokus standen dabei *PLY* und *Lark*, die beide Mechanismen zum Definieren lexikalischer und grammatischer Regeln und zur automatischen Generierung von Parsern bieten. Aufgrund seiner modernen Architektur, besseren Fehlertoleranz und dem deutlich größeren Funktionsumfang für die Verarbeitung von Parse-Trees bzw. ASTs wurde schließlich *Lark* als Kernbibliothek ausgewählt.

Aufbau und Strukturierung der ATHLET-Grammatik

Im Anschluss wurde mit der Erstellung einer formalen Grammatik für den ATHLET-Input begonnen. Zunächst wurde die Eingabesprache in ihre groben Bestandteile zerlegt, um ein besseres Verständnis der Struktur, der verwendeten Kontrollwörter und der zulässigen Parameterbereiche zu gewinnen. Dabei kamen Parsing-Tests mit *Lark* zum Einsatz, um das Verhalten einfacher Prototyp-Grammatiken zu validieren und die grundlegenden Schritte für die Implementierung von Parsing-Mechanismen zu erarbeiten.

Da die Eingabe-Struktur von ATHLET sehr umfangreich ist, wurde zur Unterstützung der Entwicklungsarbeit das MindMapping-Werkzeug *Freemind* eingesetzt. Die Mindmap diente zur hierarchischen Darstellung der Struktur des Eingabeformats, insbesondere der zahlreichen ATHLET-Kontrollwörter und ihrer Beziehungen zueinander. Dadurch konnten die Abhängigkeiten zwischen Blöcken und Unterblöcken übersichtlich dokumentiert werden. Auf Grundlage dieser Mindmap entstanden die ersten Grammatik-Definitionen für die Schlüsselwörter `GCSM` und `OBJECT`, die als Prototyp für den Aufbau der restlichen Struktur dienten.

Entwicklung der lexikalischen und grammatischen Regeln

Im nächsten Schritt erfolgte die formale Definition der Eingabesprache mithilfe regulärer Ausdrücke und grammatischer Regeln in erweiterter Backus-Naur-Form (EBNF). Hierbei wurden zunächst **lexikalische Regeln** (reguläre Ausdrücke) erstellt, die dem Lexer von Lark erlauben, den Text eines Eingabedatensatzes in eine Abfolge logisch zusammengehöriger Einheiten (Tokens) zu zerlegen – beispielsweise Schlüsselwörter, Zahlenwerte, Bezeichner oder Operatoren. Anschließend wurden die **grammatikalischen Regeln** formuliert, welche beschreiben, wie diese Tokens zu größeren syntaktischen Strukturen zusammengefügt werden können, um die hierarchische Natur der Eingabe (Blöcke, Subblöcke, Parameterlisten) in einem sogenannten Parse-Tree, der Roh-Form eines AST, abzubilden.

Zur Weiterverarbeitung der erkannten Strukturen wurden erste **Transformer**-Methoden implementiert, die den erzeugten Parse-Tree bereinigen und in einen AST überführen. Dieser abstrahierte Baum enthält die gesamte im Eingabedatensatz enthaltenen Information logisch strukturiert, wodurch er als Basis für spätere Verarbeitungsschritte, wie Plausibilitätsprüfungen oder Konvertierungen in andere Datenformate, dienen kann.

Umstellung auf LALR(1) und Erweiterung der Grammatik

Im weiteren Projektverlauf wurde der zunächst verwendete *Earley*-Algorithmus durch den deutlich effizienteren Algorithmus LALR(1) ersetzt, was eine erhebliche Performance-Steigerung und die Speicherung von generierten Parsern ermöglichte. Grundsätzlich unterscheiden sich die beiden Algorithmen durch die Anzahl an Lookahead-Tokens, welche zum Erkennen und Unterscheiden von Strukturen herangezogen werden. Während *Earley* hier keine Grenzen setzt, kann LALR(1) nur ein einziges Token vorausschauen. Aus diesem Grund musste die bisher notierte Grammatik überarbeitet werden, sodass sich auch für einen Lookahead von 1 keine Widersprüche oder Mehrdeutigkeiten ergeben. In diesem Zug wurde die Grammatik auch inhaltlich vervollständigt, wodurch nun sämtliche ATHLET-Kontrollwörter eingelesen werden konnten.

```
%import .ac2input (CW, KW, SW, PW)
%import .ac2input (eol_1n, eol_0n, identifier, description, _expr_group)

#==================
# CW <generic>
#==================
?start                      : (CW _cw_generic)+      -> cw_section
_cw_generic                 : _annotatable_header                        \
                              section_data?                              \
                              (kw_section* | sw_section* | pw_section*)
    #
    kw_section              : KW  _annotatable_header                    \
                              section_data?                              \
                              (sw_section* | pw_section*)
        #
        sw_section          : SW  _annotatable_header                    \
                              section_data?                              \
                              pw_section*
            #
            pw_section      : PW  _annotatable_header                    \
                              section_data?

    _annotatable_header     : _section_id  description  eol_1n
                            | _section_id               eol_1n
        _section_id         : identifier
    section_data            : _expr_group               eol_0n

%import .base.term (BLANK, INDENT, LINE_CONT, INNER_COMMENT)
%ignore BLANK
%ignore INDENT
%ignore LINE_CONT
%ignore INNER_COMMENT
```

Abb. 3.6 Grammatik-Definition generischer Control-Words in AC²-Eingaben

Abb. 3.6 zeigt einen Teil der erstellten Grammatik-Definitionen zum Einlesen von AC²-Eingabedatensätzen. Für Sonderfälle, die vom standardisierten Aufbau der AC²-Eingaben abweichen, wurden spezielle **Teilgrammatiken** erstellt. Dazu gehören unter anderem `3D-MODULE`, `CDR1DIN`, `CONDRU`, `ECOREMOD`, `HEADER` oder `PARAMETER`. Diese Blöcke weisen besondere Strukturen auf, beispielsweise abweichende Regeln für Bezeichner, erweiterte Ausdrucksverarbeitung oder freie Textabschnitte.

Darüber hinaus wurde die **mathematische Ausdrucksverarbeitung** erweitert, um Vergleiche, logische Literale und Operatoren (`.true.`, `.false.`, `.or.`, `.and.`, `.not.`), Funktionsaufrufe und indizierte Zugriffe (Subscripts) zu unterstützen. Damit können komplexe numerische Definitionen in den Eingaben korrekt erkannt und verarbeitet werden.

Implementierung rekursiver Includes und der Basis-Anwendung

Ein weiterer wesentlicher Fortschritt war die Integration eines rekursiven Include-Mechanismus, der es ermöglicht, verschachtelte Eingabedateien einzulesen. Hiermit erkennt der Parser `include`-Anweisungen, welche auf einzubindende Eingabedateien verweisen. In diesem Fall wechselt er automatisch in die einzubindende Datei und kehrt nach deren Verarbeitung in den ursprünglichen Datenstrom zurück. Hierfür musste auch eine Logik zur Erkennung und Behandlung zirkulärer Includes implementiert werden, um Endlosschleifen zu vermeiden.

Zur praktischen Nutzung des Parsing-Systems wurde eine Kommandozeilen-Anwendung *InputProcessor* entwickelt, die über ein umfangreiches CLI (Command-Line Interface) gesteuert werden kann, siehe Abb. 3.7. Hierüber können die Anwender verschiedene Parameter wie Grammatikdateien, Lexer-Modi, Startsymbole oder Ausgabemodi (`plain`, `tree`, `pretty`) angeben. Bei speziellen Anforderungen kann neben den zu analysierenden Eingabedateien auch der Lexer und Parser-Typ (`earley` oder `lalr`) passend zur verwendenden Grammatik gewählt werden. Die Ergebnisse werden in Form von Parse-Bäumen bzw. ASTs gespeichert und entweder in Ausgabedateien geschrieben oder auch direkt auf der Konsole ausgegeben.

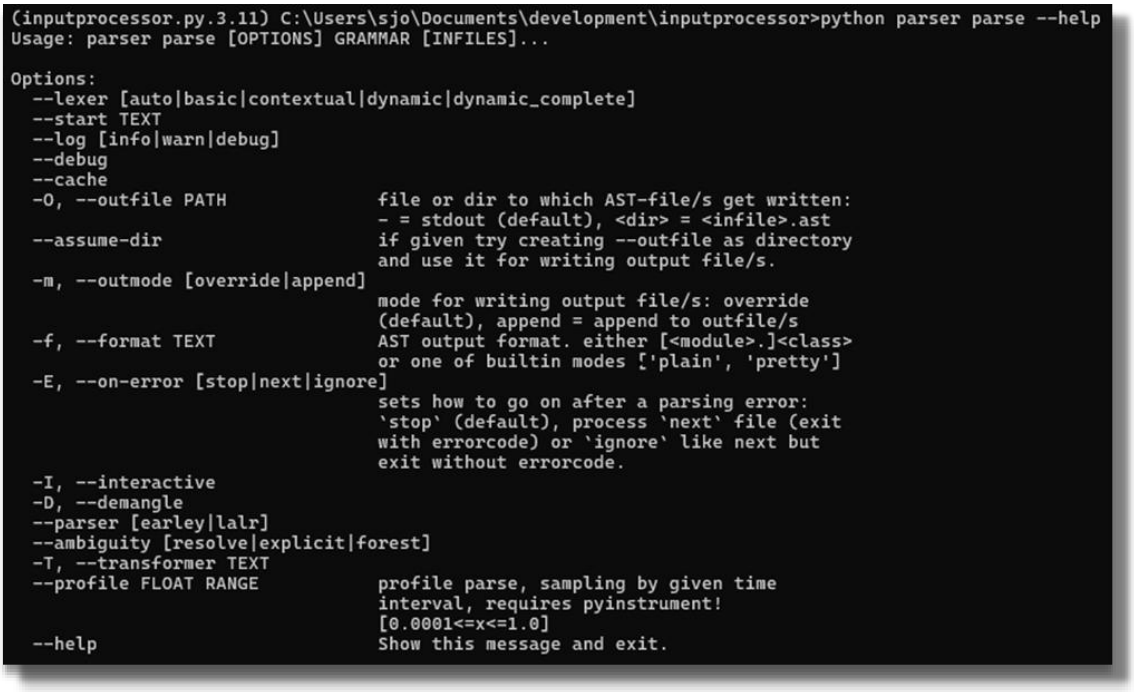

Abb. 3.7 Das Commandline-Interface (CLI) und die Optionen des InputProcessors

Das Command-Line-Interface erlaubt eine komfortable Integration in automatisierte Arbeitsabläufe und unterstützt bereits die Verarbeitung mehrerer Eingabedateien in einem Durchlauf, einschließlich detaillierter Logging-Optionen und Fehlerbehandlung. Alle verfügbaren Optionen, deren Bedeutung und eine kurze Beschreibung ihrer Funktion sind über die `--help` Option als Hilfe-Seite verfügbar.

Test und Qualitätssicherung

Zur Sicherstellung der Funktionsfähigkeit und Stabilität wurden umfangreiche Test- und Validierungsabläufe in die Continuous Integration (CI) eingebunden. Diese Tests überprüfen die Parser-Ergebnisse für verschiedene Eingabeszenarien und melden Abweichungen automatisch. Die CI-Konfiguration wurde so erweitert, dass neue Grammatik-Module und Parser-Komponenten jederzeit integriert und überprüft werden können. Dadurch ist eine langfristige Wartbarkeit und Erweiterbarkeit des Systems gewährleistet.

3.1.3 Implementierung fortgeschrittener Eingabeverarbeitung

Aufbauend auf der in 3.1.2 beschriebenen Eingabeanalyse sollte im Rahmen dieses Arbeitspunkts die Funktionalität des Input-Prozessors gezielt erweitert werden, um

komplexe Verarbeitungsschritte auf Basis der erzeugten abstrakten Datenstruktur (AST) zu ermöglichen.

Während die syntaktische Analyse bereits ein strukturiertes Abbild der Eingabedaten liefert, lag der Schwerpunkt dieses Arbeitspakets auf der semantischen und funktionalen Weiterverarbeitung dieser Daten, um sie für verschiedene Anwendungszwecke nutzbar zu machen. Ziel war die Entwicklung einer modularen Architektur, die eine flexible Nachbearbeitung der Eingabedaten erlaubt – beispielsweise zur

- **Normierung** von Eingabedatensätzen, um Unterschiede zwischen Datensätzen systematisch vergleichen zu können,
- **Optimierung**, durch das Erkennen und Entfernen ungenutzter oder redundanter Definitionen, sowie
- **Übersetzung in Standardformate** (z. B. YAML, JSON, XML), um den Datenaustausch mit externen Werkzeugen und Schnittstellen zu erleichtern.

Darüber hinaus sollte der Input-Prozessor in die Lage versetzt werden, komplexe Konsistenzprüfungen durchzuführen und die Eingabestrukturen der AC²-Rechencodes in abstrahierter Form weiterzuverarbeiten.

Aufbau und Erweiterung der AST-Verarbeitung

Nach der erfolgreichen Umstellung des Parsers auf den LALR(1)-Algorithmus konzentrierten sich die Entwicklungsarbeiten auf den systematischen Aufbau von Verarbeitungsschichten über dem AST. Dazu wurden mehrere Klassenhierarchien für **Transformation**, **Interpretation** und **Ausgabe** implementiert, die eine flexible Weiterverarbeitung der eingelesenen Daten ermöglichen.

AST-Transformer

Transformer-Klassen führen eine **bottom-up**-Verarbeitung des AST durch und ermöglichen gezielte Umformungen, Vereinfachungen und Typisierungen der Baumstruktur. Hierzu wurden die folgenden Transformer-Typen entwickelt:

- **Demangler:** Entfernt die internen Typ-Präfixe der AST-Knoten, die zur Laufzeit für die Parser-Erkennung notwendig, für nachfolgende Verarbeitungsschritte jedoch hinderlich sind.

- **Prune:** Reduziert die Baumstruktur, indem irrelevante Tokens entfernt, Kommentar- und Include-Knoten zusammengeführt und bei Bedarf deaktivierte Eingabesektionen ausgeblendet werden.
- **Typing:** Wandelt Textliterale in native Python-Objekte um (`int`, `float`, `bool`, `str`). Für Fließkommazahlen wurde eine spezialisierte Klasse entwickelt, die zusätzlich Präzision und Notationsart speichert.
- **AC²:** Fasst AC²-spezifische Eingabestrukturen zusammen, wertet mathematische Ausdrücke aus und überführt tabellarische Definitionen in strukturierte Python-Objekte (z. B. 2D-Arrays).

Diese Mechanismen bilden die Grundlage für spätere Analysen, Optimierungen und Konvertierungen in verschiedene Ausgabeformate.

Interpretation und Semantische Auswertung

Zusätzlich zu den Transformern zur grundlegenden Umformung der AST-Struktur wurde eine zweite Art von Transformern eingeführt: Die sog. **AST-Interpreter**. Im Gegensatz zu den Standard AST-Transformern durchlaufen diese den AST **top-down**, um gespeicherte Informationen zu interpretieren oder mit zusätzlichen Metadaten anzureichern. Zu den wichtigsten der hier entwickelten AST-Interpretern zählen:

- **OperationEvaluator:** Bewertet mathematische Ausdrücke, löst Parameter auf und interpretiert definierte Konstanten und Funktionen (z. B. `pi`, `sin`, `sqrt` etc.).
- **DefinitionCollector:** Registriert alle Konstanten und mathematischen Funktionen, speichert Ergebnisse und erweitert Zuweisungsknoten im AST durch Textrepräsentationen und Ergebniswerte der Ausdrücke.

Diese Funktionalitäten ermöglichen eine erste semantische Analyse der Eingabedaten, einschließlich der Berechnung von Parametern und der Überprüfung logischer Verknüpfungen.

Ausgabe- und Formatierungsfunktionen

Für die kontrollierte Darstellung und den Export der verarbeiteten Daten wurden **Tree-Formatter-Klassen** implementiert. Diese erzeugen aus dem AST verschiedene Ausgabeformate, die je nach Zielsetzung unterschiedliche Zwecke erfüllen:

- **AST-Formatter**: Erstellt eine textuelle, hierarchische Darstellung des AST, die Struktur- und Tokeninformationen einschließlich Quellzeilenpositionen abbildet.
- **AC²-Formatter**: Erstellt eine Rekonstruktion der ursprünglichen Eingabe im Format der AC²-Rechencodes, soweit möglich unter Erhalt von Formatierung und Zeilenstruktur. Dadurch bleiben die erstellten Ausgabedateien mit den verarbeiteten Eingabedateien im Originalformat vergleichbar und können über Text-Vergleichswerkzeuge (Diff), ggf. automatisch, auf Unterschiede geprüft werden.

Diese Funktionalität bildet die Grundlage für die geplante **Normierung** und ggf. automatisierte **Differenzanalysen** zwischen Eingabedatensätzen. Abb. 3.8 stellt die Funktion der Grammatik-basierten Eingabe-Analyse und das Zusammenspiel der einzelnen Komponenten als Ablaufschema dar.

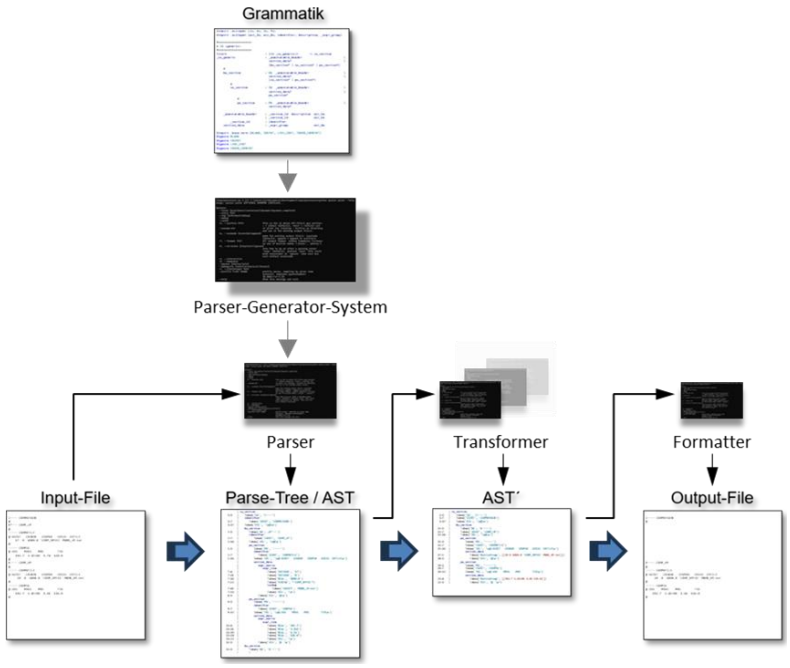

Abb. 3.8 Ablaufschema Grammatik-basierte Eingabe-Analyse. Der aus der Grammatik generierte Parser erzeugt einen Parse-Tree / AST mit den aus den Eingabedateien eingelesenen Informationen. Dieser wird gem. der gewählten Transformer bearbeitet (AST´) bevor er durch einen Formatter im Zielformat ausgegeben wird.

Ausbau des Kommandozeilen-Interfaces

Parallel zur inhaltlichen Erweiterung wurde das Kommandozeilen-Interface (CLI) des Input-Prozessors erheblich ausgebaut, um die Funktionen auch Anwendern unmittelbar zugänglich zu machen. Die wichtigsten der dazu implementierten CLI-Optionen sind:

- **Flexible Ausgabeformate** über die Option `--format` (`plain`, `pretty`, `AST`, `AC²` oder benutzerdefinierte Formate),
- **Transformationssteuerung** mit `--transformer`, um beliebige Kombinationen und Reihenfolgen von AST-Transformationen zu konfigurieren,
- **Parser-Caching** (`--cache`), wodurch die LALR(1)-Grammatik nur bei Änderungen neu geladen wird, was die Verarbeitung deutlich beschleunigt,
- **Fehlerbehandlung** (`--on-error`), um bei Massenverarbeitung das Verhalten bei fehlerhaften Eingaben festzulegen (`stop`, `next`, `ignore`),
- **Debug- und Interaktivmodus** (`--debug`, `--interactive`), beide geben Einblick in den Parserzustand und Token-Stack und helfen so, Fehlerursachen für nicht erkannte Eingabestrukturen und Shift/Reduce-Konflikte zu erkennen,
- **Option `--demangle`**, um Typpräfixe in Ausgaben zu entfernen – nützlich zur Erzeugung bereinigter AST-Ansichten.

Damit wurde ein Werkzeug geschaffen, das nicht nur Entwicklern, sondern auch Anwendern eine kontrollierte Analyse und Verarbeitung umfangreicher Eingabedatensätze erlaubt.

Erweiterung zur semantischen Analyse

In einem weiteren Schritt wurde das Konzept der semantischen Verarbeitung vorbereitet, um zukünftig komplexe Konsistenzprüfungen innerhalb der Eingabedaten durchführen zu können. Dazu wurden erste Transformer- und Datenklassen entwickelt, die komplexe Eingabeobjekte wie TFOs oder Bauteile zu Python-Objekten zusammenfassen. Diese Objekte können sowohl mit Standardwerten als auch mit variantenreichen Definitionen umgehen und bilden die Grundlage für Plausibilitätsprüfungen, die über rein syntaktische Analysen hinausgehen. Dieser Ansatz bildet den Einstieg in die semantische Konsistenzanalyse der Eingabedaten, deren Ausbau im Rahmen zukünftiger Projekte fortgeführt werden sollte.

Ein erster Anwendungsfall wurde mit der Verarbeitung tabellarischer Strukturen realisiert. Hierbei werden tabellenartige Eingaben als **Matrix**- oder **MatrixGroup**-Objekte zusammengefasst und auf Konsistenz geprüft, siehe Abb. 3.9. Darüber hinaus können durch Angabe von Indexintervallen und Spalten-Ausdrücken auch Abschnitte von Tabellenzeilen automatisch erstellt werden, was z. B. in ATHLET-Datensätzen eine deutlich einfachere Definition von GCSM-Tabellen möglich macht.

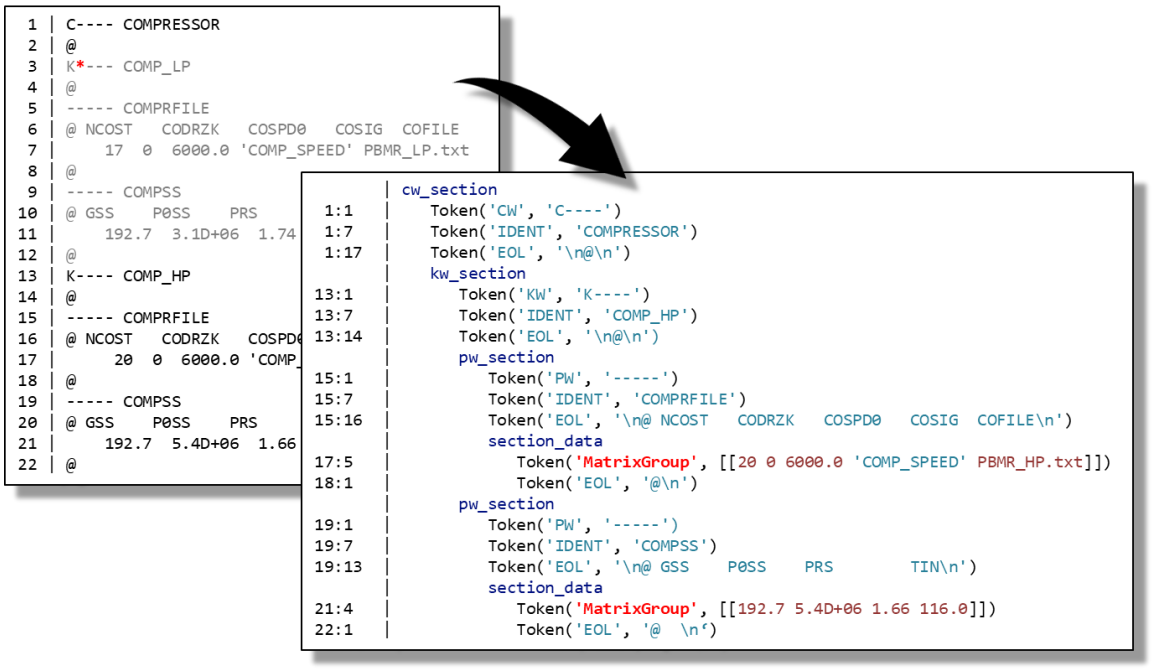

Abb. 3.9 Beispiel einer durch den InputProcessor verarbeite ATHLET-Eingabe (links). Die Ausgabe des daraus erstellte AST (rechts) zeigt die effektiven Daten der Eingabe und die zu MatrixGroup-Objekten zusammengefassten tabellarischen Daten

Qualitätssicherung und Validierung

Zur Überprüfung der Stabilität und Korrektheit der entwickelten Mechanismen wurde eine umfassende Test- und CI-Umgebung aufgebaut. Die Tests verwenden reale Eingabedateien aus den Validierungsumgebungen von ATHLET und ATHLET-CD und überprüfen die Reproduzierbarkeit der Ausgaben im AST- und AC²-Format. Nur wenn die generierten Dateien exakt mit den Referenzausgaben übereinstimmen, gilt der Test als bestanden.

In zusätzlichen Verifikationsläufen wurden die vom Input-Prozessor erzeugten AC²-Dateien direkt als Eingaben in ATHLET verwendet. Die Simulationsergebnisse stimmten, abgesehen von geringfügigen numerischen Abweichungen, welche durch höhere Rechengenauigkeit zu Stande kommen können, vollständig mit den Originaleingaben überein. Durch diese Maßnahmen werden die Funktion und Stabilität des Systems auch für zukünftige Weiterentwicklungen sichergestellt.

3.1.4 Vereinheitlichte Verarbeitung von Eingabeformaten

Die im Arbeitspunkt 3.1.2 entwickelte Eingabeanalyse ermöglicht bereits die syntaktische und strukturelle Erfassung des generischen Aufbaus von AC²-Eingabedateien. Um diese Funktionalität jedoch einheitlich für alle Rechencodes der AC²-Familie (ATHLET, ATHLET-CD und COCOSYS) nutzbar zu machen, sind zugehörige Eingabegrammatiken erforderlich, welche eventuell benötigte Sonderbehandlungen und zusätzliche Eingabeabschnitte in Form von Teilgrammatiken beschreiben.

Ziel dieses Arbeitspakets war es, neben ATHLET auch die Besonderheiten der anderen Eingabeformate als Grammatik abzubilden. Dadurch sollte der Input-Prozessor in die Lage versetzt werden, alle AC²-Formate auf derselben technischen Grundlage zu analysieren und zu verarbeiten. Diese Vereinheitlichung bildet eine zentrale Voraussetzung für spätere Schritte wie die automatisierte Konvertierung zwischen Formaten, die Qualitätssicherung von Eingabedaten oder die Entwicklung gemeinsamer Werkzeuge für die Input-Erstellung innerhalb der AC²-Toolkette.

Aufbau und Erweiterung der ATHLET- und ATHLET-CD-Grammatik

Im ersten Projektabschnitt stand die Vervollständigung und Validierung der Grammatik für ATHLET und ATHLET-CD im Vordergrund. Basierend auf dem bereits implementierten Input-Prozessor wurden umfangreiche Tests mit realen Eingabedatensätzen aus verschiedenen Projekten durchgeführt. Ziel war es, sämtliche noch fehlenden Kontrollwörter und Strukturelemente zu identifizieren und sukzessive in die Grammatik aufzunehmen. Durch diesen iterativen Prozess konnte die Grammatik schrittweise erweitert und verfeinert werden, bis schließlich alle relevanten Strukturen der ATHLET- und ATHLET-CD-Eingaben vollständig abgedeckt waren.

Ein besonderes Augenmerk lag dabei auf der Behandlung variabler und historisch gewachsener Eingabeabschnitte, die in älteren ATHLET-Versionen häufig uneinheitlich

definiert sind. Die entwickelte generische Grammatik für **Control-Words** (vgl. Abb. 3.6) erwies sich dabei als sehr flexibel und trug maßgeblich zur Abwärtskompatibilität des Systems bei. Letztlich steht damit nun ein Werkzeug zur Verfügung, das nicht nur aktuelle ATHLET- und ATHLET-CD-Versionen unterstützt, sondern auch ältere Eingabeformate dieser Codes zuverlässig verarbeiten kann, was einen entscheidenden Fortschritt für die Vereinheitlichung der gesamten AC²-Eingabeverarbeitung darstellt.

Erste Arbeiten zur Unterstützung von COCOSYS

Im weiteren Projektverlauf wurden die Arbeiten auf das Eingabeformat des dritten AC²-Codes, COCOSYS, ausgedehnt. Erste Tests zeigten, dass die bestehende generische Grammatik bereits einen großen Teil der COCOSYS-Strukturen korrekt verarbeiten und in der hierarchischen AST-Darstellung abbilden kann.

Allerdings traten hierbei auch spezifische Eingabemuster und Sonderstrukturen zutage, die sich deutlich von den Formaten der ATHLET-Familie unterscheiden. Diese erfordern gezielte Erweiterungen der Grammatik und zusätzliche Verarbeitungsschritte im Input-Prozessor. Die entsprechenden Anforderungen wurden identifiziert und für die weitere Entwicklung dokumentiert. Damit wurde zwar im Rahmen dieses Projekts das Ziel einer COCOSYS-Unterstützung noch nicht erreicht, jedoch eine solide Grundlage geschaffen, um die vollständige Unterstützung des COCOSYS-Eingabeformats in einer nachfolgenden Weiterentwicklung des InputProcessors gezielt umzusetzen.

3.1.5 Verbesserung der Anwendbarkeit bisheriger Entwicklungen

Mit dem AC² Design Modeller (ADM) steht ein Werkzeug zur Verfügung, das die interaktive Erstellung und Bearbeitung von Eingabedaten für die AC²-Rechencodes, insbesondere ATHLET und ATHLET-CD, ermöglicht. Zu Projektbeginn war die Entwicklung der beiden wesentlichen ADM-Komponenten – des ATHLET Thermohydraulic Modellers (ATM) und des ATHLET GCSM Modellers (AGM) – weit fortgeschritten, sodass bereits eine grundlegende Modellierung der thermohydraulischen und geometrischen Strukturen möglich war.

Da der ATM jedoch bislang nur in begrenztem Umfang praktisch eingesetzt wurde, bestand die Notwendigkeit, die **Anwendbarkeit und die Zuverlässigkeit der erzeugten**

Datensätze systematisch zu überprüfen. Der Arbeitspunkt verfolgte daher folgende Ziele:

- **Verifikation der erzeugten AC²-Eingabedatensätze** durch Vergleich mit etablierten ATHLET-Beispielen in enger Zusammenarbeit mit Anwendern und ATHLET-Entwicklern.
- **Identifikation und Beseitigung bestehender Fehler** in der Implementierung, insbesondere im Bereich der GCSM- und thermohydraulischen Modellierung.
- **Verbesserung der Benutzerfreundlichkeit und Dokumentation**, um einen durchgängigen und verständlichen Workflow zu gewährleisten.
- **Bewertung der Zukunftsfähigkeit der technischen Plattform „SimInTech“** und Erarbeitung eines Konzepts für eine mögliche Umstellung auf offene Softwarelösungen (OpenSource).

Ziel war es somit, ADM in einen **stabilen und praxistauglichen Zustand** zu überführen, der sowohl für interne GRS-Anwendungen als auch für externe Nutzer im Rahmen der AC²-Release nutzbar ist.

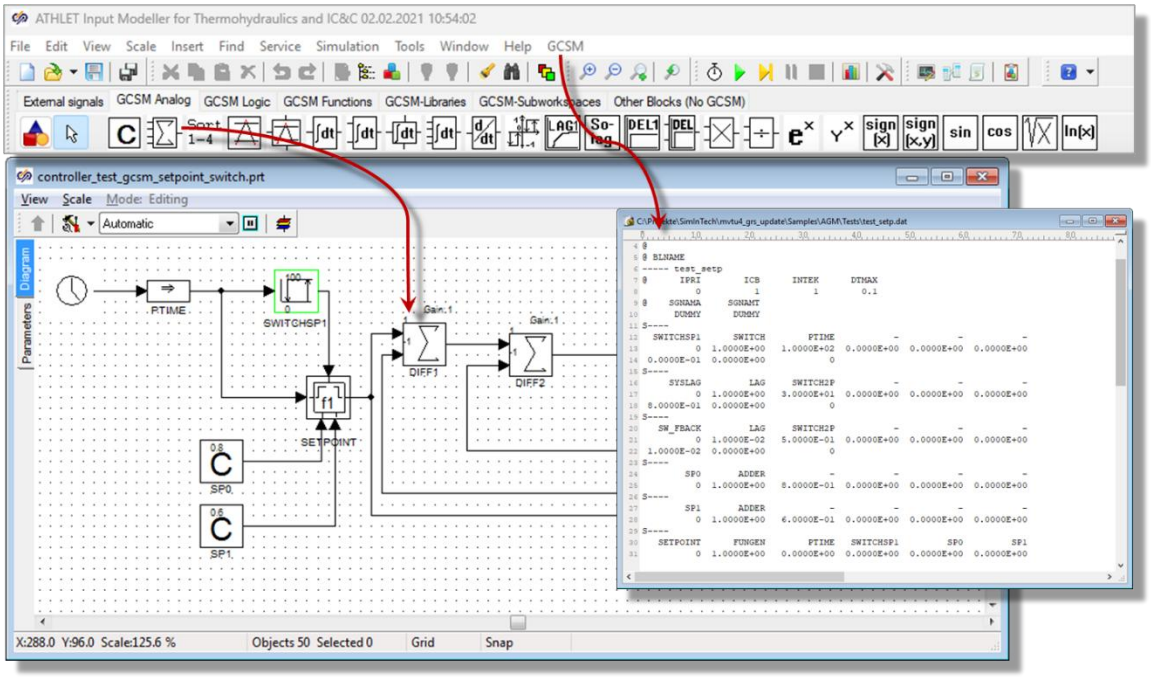

Abb. 3.10 ADM im Einsatz bei blockorientierter Leittechnik-Modellierung im AGM-Modul

Verifikation und Verbesserung der ADM-Funktionalität

In enger Zusammenarbeit mit den ADM-Anwendern und den ATHLET-Entwicklern wurde eine systematische Überprüfung der ADM-Funktionalität durchgeführt. Dazu wurden repräsentative ATHLET-Beispieldatensätze als ADM-Projekte abgebildet, erzeugt und mit den Originaldatensätzen verglichen. Diese Tests ermöglichten es, fehlerhafte Implementierungen zu identifizieren und gezielt zu korrigieren.

Ein zentrales Ergebnis war die Erkenntnis, dass die Korrektheit und Stabilität des GCSM-Moduls (AGM) oberste Priorität besitzen, gefolgt von der Verbesserung der Nutzbarkeit des Thermohydraulic Modellers (ATM), siehe Abb. 3.11. Weniger relevant wurde dagegen die direkte Eingabe zusätzlicher AC²-Parameter über ADM bewertet, da diese Eingaben meist außerhalb des grafischen Modells erfolgen. Im Zuge der Arbeiten wurden mehrere funktionale Fehler behoben und gezielte Verbesserungen implementiert:

- Korrekturen bei der grafischen Eingabe von Prioritätsketten,
- Anpassungen des Zahlenformats in den generierten ATHLET-Datensätzen,
- Verbesserungen bei der Erzeugung von GCSM-Grafiken für ATLAS,
- Korrekturen bei der Ausgabe von Ventildaten und *time-dependent volumes*.

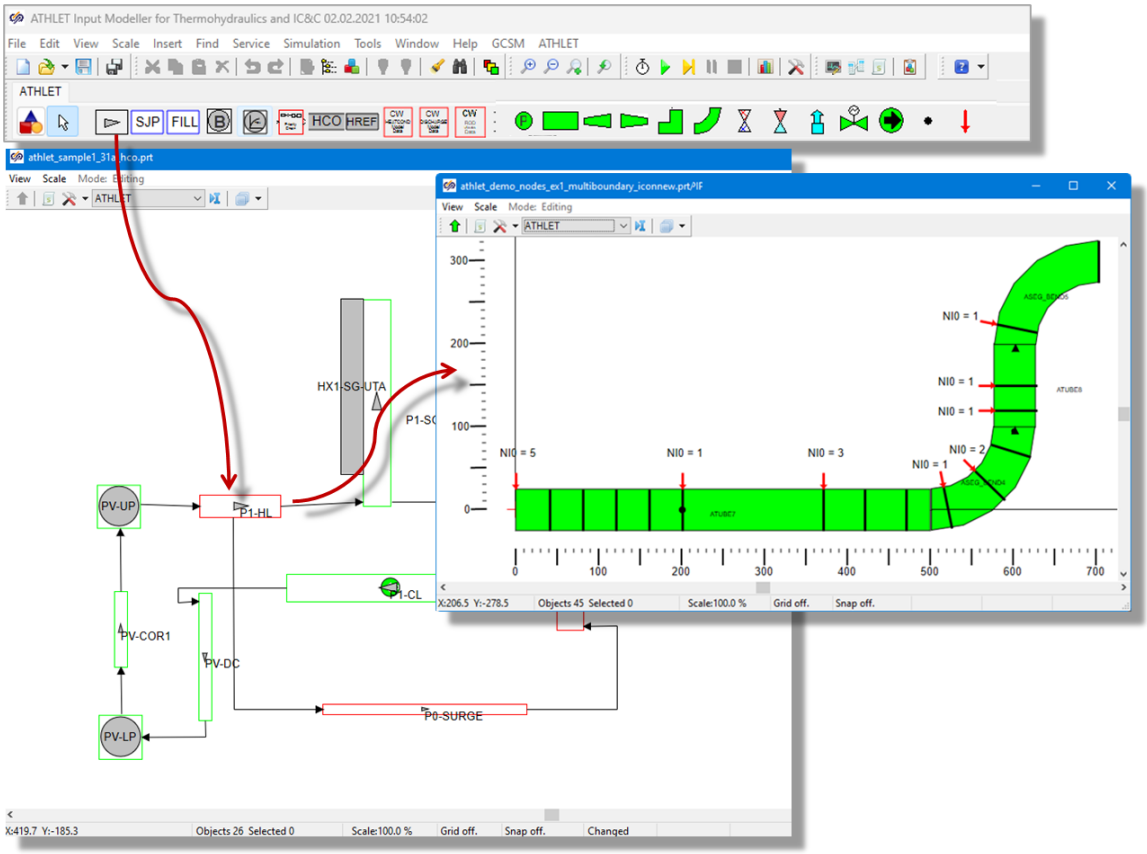

Abb. 3.11 ADM im Einsatz bei Modellierung der Thermohydraulik im ATM-Modul

Zur Sicherstellung der Nachhaltigkeit dieser Arbeiten wurden automatisierte Tests entwickelt, die bei jedem ADM-Build automatisch ausgeführt werden. Dadurch können zukünftig Fehler, die bereits behoben wurden, bei einer möglichen Regression sofort wieder erkannt werden. Damit ist auch die Grundlage geschaffen, repräsentative Datensätze als ADM-Projekte zu speichern und automatisiert zu überprüfen.

Verbesserungen im Entwicklungsprozess und in der Nutzerunterstützung

Parallel zu den funktionalen Anpassungen wurde die Paketierung und Dokumentation von ADM deutlich verbessert. Mit der AC²-Release v2023.0 wurde ADM erstmals auch an externe Nutzer ausgeliefert. Dafür wurde das Build-System erweitert, sodass

- die PDF-Dokumentationen automatisch aus dem Projekt-Wiki erzeugt werden,
- eine Liste verwendeter Drittprogramme und zugehörigen Lizenzen generiert wird
- und beide Dokumente direkt aus ADM heraus geöffnet werden können.

Darüber hinaus wurden zahlreiche Anwenderanregungen aus der praktischen Nutzung aufgenommen und in die Software integriert. Diese Rückmeldungen trugen wesentlich zur Verbesserung der Benutzerführung und zur Stabilisierung der Software bei.

Bewertung der technischen Basis und Zukunftsausrichtung

Ein wesentlicher Bestandteil des Arbeitspunkts war die Bewertung der technischen Basis von ADM, die aktuell auf der proprietären Plattform „SimInTech“ aufbaut. Dabei zeigte sich, dass insbesondere eine Portierung des GCSM-Moduls (AGM) auf eine andere Plattform mit erheblichem Aufwand verbunden wäre.

Für den Thermohydraulic Modeller (ATM) hingegen war das Ergebnis nicht so eindeutig, da die bisherigen Eingabemöglichkeiten in SimInTech für die Spezifikation zusätzlicher AC²-Parameter kaum Mehrwert bietet. Diese ließen sich ggf. durch eine einfachere, auf einer OpenSource-Plattform basierenden Benutzeroberfläche besser realisieren. Allerdings ist die Möglichkeit der Erstellung und Bearbeitung der Eingabeteile für die Leittechnik und die Thermohydraulik innerhalb einer einzelnen Anwendung für den Anwender als bedeutender einzustufen.

Eine weitere zentrale Voraussetzung für einen möglichen Plattformwechsel ist die Verfügbarkeit eines AC²-Input-Parsers, der eine zuverlässige Einbindung und Aktualisierung bestehender Datensätze erlaubt. Tests zur Nutzung der internen ATHLET-Routinen zum Einlesen von Eingabedaten in ADM wurden durchgeführt. Der Aufwand für eine vollständige Integration konnte dadurch zwar grob abgeschätzt werden, eine Umsetzung lag jedoch außerhalb des Umfangs des aktuellen Projekts, da für diesen Ansatz Anpassungen in ATHLET notwendig sind. Zielführender scheint daher die mittelfristige Einbindung des im Projekt entwickelten Eingabe-Parsers für AC²-Datensätze (siehe Abschnitt 3.1.2).

Darüber hinaus ist beim Betracht der Nutzung einer alternativen Plattform, wie eine moderne OpenSource-Umgebung, eine vollständige ADM-Portierung aufgrund des großen Codeumfangs (ca. zwei Mio. Codezeilen) aktuell kaum wirtschaftlich umsetzbar. Zudem müsste die neue Plattform die wesentlichen SimInTech-Funktionalitäten bereits mitbringen, was bislang keine der untersuchten OpenSource-Lösungen in ausreichendem Maße bietet. Insgesamt ist daher davon auszugehen, dass ADM mittelfristig auf SimInTech basiert bleibt, wobei mögliche Alternativen jedoch weiterhin beobachtet werden.

Zumindest konnte die Entwicklungsumgebung erfolgreich auf aktuelle Versionen des Delphi-Compilers und der benötigten Bibliotheken umgestellt werden, wodurch die technische Stabilität und Kompatibilität der Entwicklungsumgebung und somit die Wartungsfähigkeit von ADM in den kommenden Jahren gesichert sein sollte.

3.1.6 Allgemeine Wartungsarbeiten

Ein zentrales Ziel dieses Arbeitspunktes war die Sicherung der langfristigen Wartbarkeit und Stabilität der ADM-Software. Dazu sollten bestehende Entwicklungsprozesse modernisiert, die Softwarebereitstellung automatisiert und Abhängigkeiten von veralteten oder proprietären Werkzeugen schrittweise reduziert werden. Insbesondere standen folgende Aufgaben im Fokus.

- Zusammenführung des ADM-Quellcodes mit der neuen Version der Basissoftware SimInTech zur Übernahme von Verbesserungen und zur Vereinfachung künftiger Updates.
- Einrichtung einer automatisierten Build- und Testumgebung (Continuous Integration / Continuous Deployment, CI/CD) auf dem GRS-internen GitLab-Server.
- Modernisierung der Dokumentations- und Installationswerkzeuge, um sie in automatisierte Abläufe und Docker-Umgebungen integrieren zu können.
- Laufende Pflege und Anpassung an Änderungen der verwendeten Entwicklungsumgebungen.

Diese Arbeiten bilden die technische Grundlage für eine nachhaltige Weiterentwicklung und Verteilung des AC² Design Modellers (ADM) innerhalb der AC²-Toolkette.

Integration und Automatisierung der Build- und Paketierungs-Prozesse

Ein wesentlicher Fortschritt wurde durch die Einführung von Continuous Integration (CI) und der Nutzung von Docker-Images im GitLab-System /GIL 25/ erreicht. Die gesamte Erstellung des ADM-Installationspakets erfolgt nun vollständig automatisiert. Hierzu wurde ein dediziertes Docker-Image entwickelt, das alle benötigten Entwicklungswerkzeuge enthält. Die CI führt automatisierte Kompiliervorgänge durch, erstellt die Installationspakete und überprüft sie auf Vollständigkeit. Darüber hinaus werden die durch ADM erzeugten AC²-Eingabedatensätze automatisch getestet, wodurch eine fortlaufende Qualitätssicherung gewährleistet ist.

Die kontinuierliche Pflege der CI/CD-Pipeline stellte während des gesamten Projektverlaufs eine wiederkehrende Aufgabe dar. Änderungen an der GitLab-Plattform sowie neue Anforderungen, welche sich durch Beta-Tests und Pre-Release-Versionen ergaben, machten wiederholte Anpassungen erforderlich. Insgesamt führten diese Arbeiten aber zu einer stabilen, reproduzierbaren und transparenten Paketierung und somit zu einer einfachen Bereitstellung von ADM, was sowohl interne Tests erleichtert als auch die Integration in die offiziellen AC²-Releases ermöglicht.

Modernisierung der Installations- und Dokumentationstools

Um die Kompatibilität mit modernen Entwicklungs- und Testumgebungen sicherzustellen, wurde das bisher verwendete Installationswerkzeug Microsoft Visual Studio Installer Project durch das flexiblere Tool InnoSetup ersetzt. Die neue Lösung ermöglicht eine nahtlose Einbindung in die automatisierten Build-Prozesse und eine einfache Paketierung für verschiedene Release-Stände.

Parallel dazu wurde die Umstellung der Dokumentationserstellung auf *pandoc* /PAD 25/ vorbereitet. Das zuvor eingesetzte Tool *FastHelp* /FAH 21/ ist nicht automatisiert über die Kommandozeile ausführbar und wird inzwischen auch nicht mehr durch den Hersteller gewartet. Durch den Wechsel auf *pandoc* kann die Dokumentation künftig über Git versioniert, direkt im CI-Prozess generiert und in die Installationspakete eingebunden werden.

Diese Maßnahmen stellen sicher, dass ADM-Distributionen künftig vollständig automatisiert erstellt und mit aktueller Dokumentation ausgeliefert werden können.

3.2 AP 2: Simulations-Durchführung

Trotz der Verschiedenartigkeit der eingesetzten Simulationscodes und ihrer Anforderungen für die Ausführung soll eine möglichst einheitliche Anwender-Unterstützung erreicht werden, die das Starten und Überwachen von Simulationen erlaubt (Online-Simulation). Die typischen Optionen und Bedienfunktionen sollen dabei intuitiv zur Verfügung stehen. An die Arbeiten des Vorgängerprojekts soll hierzu angeknüpft und deren Ergebnisse weiter ausgebaut werden. Hierzu müssen insbesondere die im Folgenden aufgeführten Bereiche weiterentwickelt werden.

3.2.1 Auslagerung des Simulationsprozesses

Ziel dieses Arbeitspunktes war die Weiterentwicklung der Methoden zum Starten und Steuern von Simulationen, die in separaten Prozessen ausgeführt werden. Dieses Konzept hatte sich bereits im Vorgängerprojekt als robust und vielseitig erwiesen und sollte weiter verfeinert werden, um eine einheitliche und flexible Unterstützung unterschiedlicher Betriebsarten (z. B. stand-alone, online, überwacht) und Systemumgebungen (Workstation, Server oder Cluster) zu ermöglichen.

Ein weiterer Schwerpunkt lag in der Verbesserung der Steuerbarkeit von Simulationen über Netzwerkstrukturen hinweg. Dazu sollte der im Vorgängerprojekt begonnene Ansatz, Simulationscodes mittels Python-Kernels als Remote-Prozesse zu starten und zu steuern, weiterentwickelt werden. Ziel war eine standardisierte, leicht wartbare und möglichst plattformunabhängige Lösung unter Verwendung gängiger Python-Bibliotheken.

Darüber hinaus war geplant, die Integration der parallelen Ausführung der AC²-Rechencodes unter Verwendung von MPI-angebundenen Erweiterungen zu ermöglichen. Da diese Betriebsart nur in einem sogenannten MPI-Kontext funktioniert, der nur beim Start von Prozessen erstellt werden kann, ist für die Nutzung von MPI eine Durchführung der Simulationen in eigenen Prozessen zwingend erforderlich.

Die verschiedenen Ansätze zur Prozessauslagerung weisen jeweils spezifische Vor- und Nachteile auf und erfordern unterschiedliche Systemvoraussetzungen sowie Abhängigkeiten. Insgesamt sollte eine robuste, skalierbare und universell einsetzbare Infrastruktur erarbeitet werden, die Simulationen sowohl lokal als auch auf entfernten Systemen starten und auch während des Ablaufs kontrollieren kann.

Entwicklung des Microservice-Konzepts

Im Rahmen der Weiterentwicklung wurde der bisherige Ansatz um ein modernes, servicebasiertes Konzept erweitert. Alternativ zum direkten Prozessaufrufs über Python-Kernels wurde ein Microservice-Ansatz erarbeitet. Dieser Ansatz zeichnet sich durch hohe Modularität und Skalierbarkeit aus und ist durch eine webbasierte API steuerbar (siehe Abb. 3.12). Als einfacher Dienst lässt sich dieser sowohl lokal als auch auf anderen im Netzwerk verfügbaren Rechnern oder dem Linux-Cluster starten. Letztlich wurde dieser Ansatz ausgewählt, um die Auslagerung von Simulationsprozessen zu realisieren.

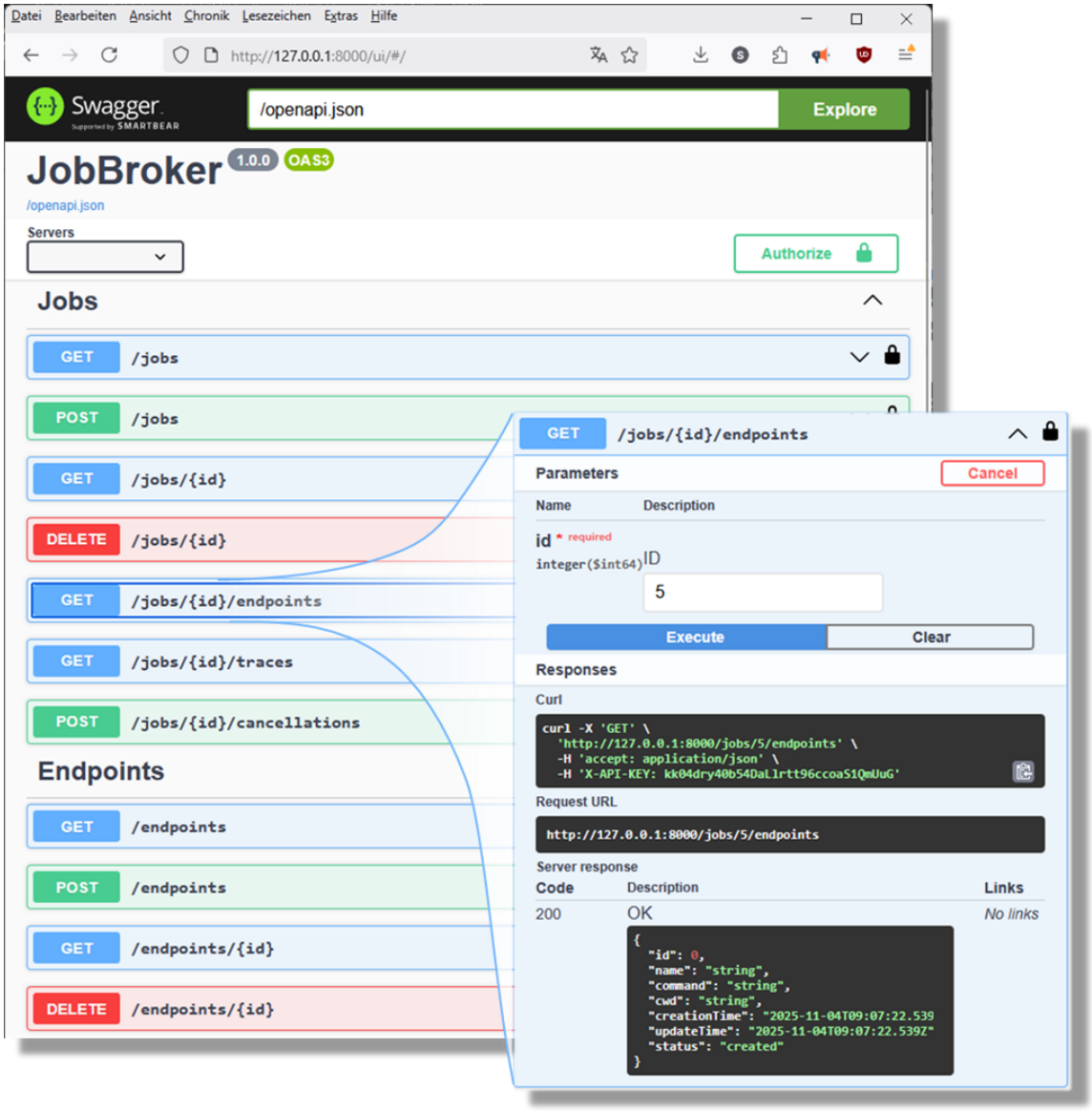

Abb. 3.12 API-Zugang zum JobBroker im Browser über das Webinterface. Hier gezeigt ist die Abfrage der Endpoints des Jobs 5 und die Alternative als `curl`-Aufruf

Nach der Festlegung aller Anforderungen und dem Entwurf einer passenden REST-API mit den notwendigen Befehlen (Start, Status-Abfrage und Stoppen) wurde mit der Implementierung einer entsprechenden Serverapplikation *"JobBroker"* begonnen. Aufgrund der hohen Anforderungen an eine zeitschrittgenaue Steuerung, wie sie in der Online-Simulation von ATLASneo verwendet wird, musste ein Design entwickelt werden, das unterschiedlichste Aufrufe von Simulationscodes über eine einheitliche REST-API ermöglicht und dennoch eine Code-spezifische Steuerung zulässt. Zudem mussten aufgrund des häufigen Austauschs von Binärdaten zwischen der Frontend-Anwendung (z. B. ATLASneo) und Simulationscodes direkte Netzwerkverbindungen möglich bleiben.

Die Architektur des JobBrokers wurde so konzipiert, dass sie eine hohe Modularität und Skalierbarkeit bietet. Dadurch kann der Dienst sowohl lokal als auch auf entfernten Rechnern oder in Clusterumgebungen betrieben werden. Neben der reinen Prozesssteuerung wurde eine standardisierte Schnittstelle definiert, über die externe Anwendungen, etwa ATLASneo oder andere GRAMOVIS-Komponenten, Simulationsjobs starten und deren Status abfragen können. Abb. 3.13 stellt diese Situation schematisch dar.

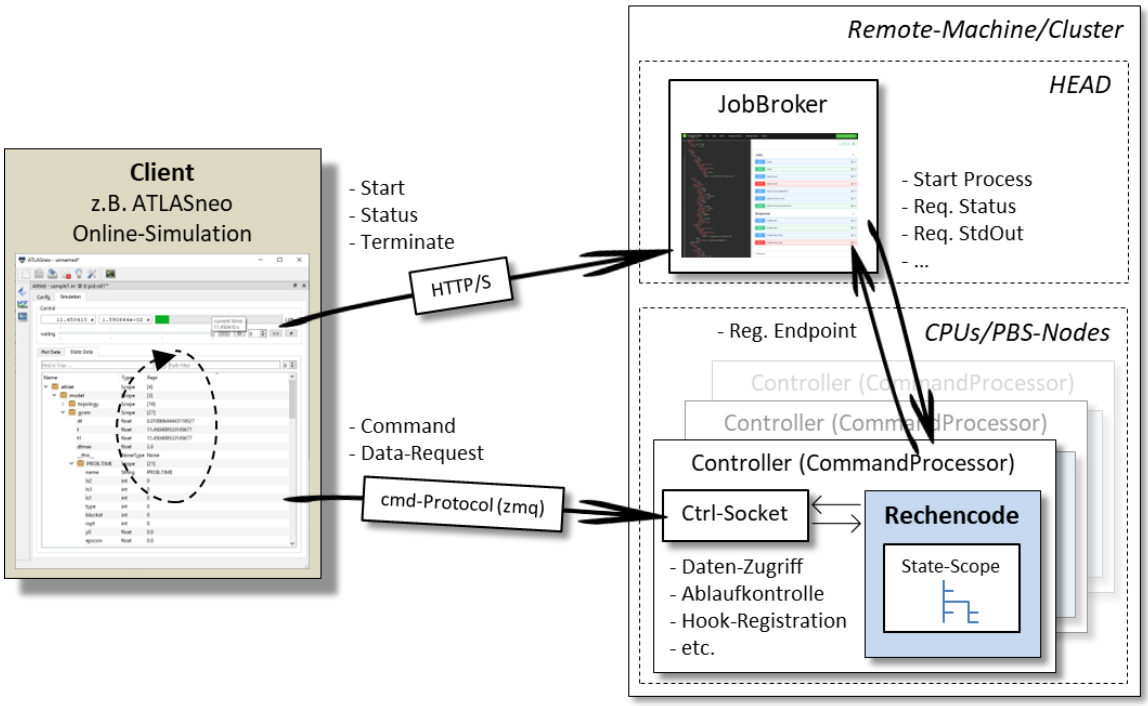

Abb. 3.13 Auslagerung von Simulationsprozessen unter Verwendung des JobBrokers

Da dieser Dienst im (lokalen) Netzwerk zur Verfügung steht, waren beim Design auch verschiedene Fragen zur Authentifizierung und Zugriffsberechtigungen zu berücksichtigen, um den sicheren und zuverlässigen Betrieb im Netzwerk zu gewährleisten.

Funktionale Erweiterungen und Implementierung

Im erreichten Entwicklungsstand besitzt der JobBroker bereits einen beachtlichen Funktionsumfang:

- Starten von Jobs, wie Simulationsprozesse, über beliebige Shell-Kommandos.
- Verwaltung von laufenden und bereits beendeten Jobs
  - Statusabfrage

  - Zugriff auf Konsolenausgaben (`stdout` / `stderr`)
  - Bereitstellung von Metainformationen wie Verbindungsendpunkte für den direkten Aufbau von Steuerverbindungen zu laufenden Jobs.
- Das gezielte / vorzeitige Beenden laufender Jobs.
- Persistente Speicherung von Job-Informationen in einer Datenbank, für die nahtlose Verwaltung von Jobs auch nach einem Neustart des JobBrokers.

Durch die webbasierte API können alle Abfragen und Aktionen sowohl durch das Standard-WebInterface des Serverdienstes als auch manuell durch Senden von HTTP-Requests, z. B. über Kommandozeilenwerkzeuge wie `curl` (vgl. Abb. 3.12) erfolgen.

Zur einfachen Verwendung der API aus Python-Umgebungen wurde zusätzlich ein **Python-Client-Package** entwickelt, das die Kommunikation mit dem JobBroker abstrahiert und welches in Python-Applikationen einfach eingebunden werden kann. Damit stehen Funktionen wie Verbindungsaufbau, Senden von Kommandos und Auswertung von Antworten über einheitliche Funktionsaufrufe zur Verfügung.

Um diese vereinfachte Nutzung auch außerhalb Pythons zu ermöglichen, wurde ergänzend ein **Command Line Interface (CLI)** entwickelt, das die eine Bedienung direkt über die Konsole und somit auch durch *„system-calls“* praktisch Programmiersprechen-unabhängig erlaubt. Diese Schnittstelle ist nicht nur für administrative Aufgaben und automatisierte Tests bedeutsam, sondern kann auch von als Job startbaren Anwendungen, wie den besagten Simulationscodes genutzt werden, um z. B. Endpunkte für den direkten Verbindungsaufbau beim JobBroker zu registrieren (vgl. Abb. 3.13).

Parallel dazu wurde die Nutzbarkeit des JobBrokers aus Frontend-Entwicklungen heraus überprüft. In Form einer Dash-basierten Webanwendung wurde ein Prototyp für ein nutzerfreundliches Webinterface geschaffen, das die Auswahl von Eingabedateien, Startparametern und Startmethoden ermöglicht. Dieses Frontend kommuniziert direkt mit dem Serverdienst über das Python-Client-Package und zielt darauf ab, den Anwendern eine intuitive und betriebssystemunabhängige Möglichkeit zu bieten, Simulationsprozesse zu starten und deren Status zu überwachen.

Erweiterung der Systemunterstützung und Einsetzbarkeit

In der weiteren Projektphase lag der Fokus auf der Erhöhung der Flexibilität und Portabilität des JobBrokers. Durch die Einführung unterschiedlicher **Executor**-Typen wurde die grundlegende Methodik zum Starten der als Job zu verwaltender Prozessaufrufe modularisiert: Neben dem bereits implementierten **Shell-Executor** zur lokalen Ausführung wurde ein **PBS-Executor** (Portable Batch System) entwickelt, der die Integration in Cluster-Umgebungen mit Warteschlangensystemen ermöglicht.

Damit wurde der Grundstein gelegt, um den JobBroker künftig auch in Hochleistungsrechenumgebungen einsetzen und hierüber Simulationsjobs flexibel auszuführen zu können. Neue Executor-Typen (z. B. Docker-basierte Container) können aufgrund der modularen Struktur einfach ergänzt werden. Dies ist ein wichtiger Punkt für die Zukunftsfähigkeit des Systems, insbesondere im Hinblick auf die wachsende Bedeutung verteilter und containerisierter Rechensysteme.

3.2.2 Weiterentwicklung Simulations-Controller

Die Steuerung und Überwachung von Simulationen erfolgen in den auf Python basierenden GRAMOVIS-Werkzeugen, wie z. B. ATLASneo oder CLI-Wrappern über sogenannte **Controller-Klassen**. Diese Klassen bilden die zentrale Schnittstelle zwischen der grafischen Benutzeroberfläche bzw. den Steuerlogiken und den zugrunde liegenden Simulationscodes. Wie in Abb. 3.14 dargestellt, können sie durch ihre objektorientierte Struktur und Mixin-Architektur über Vererbung zusammengestellt und flexibel an unterschiedliche Anwendungsfälle angepasst bzw. erweitert werden.

Im Projekt sollte diese Architektur um zusätzliche Funktionen ergänzt werden, die insbesondere die interaktive Steuerung und Laufzeitüberwachung von Simulationen verbessern. Ein wesentliches Entwicklungsziel bestand darin, sogenannte *„conditional breakpoints“* einzuführen. Diese sollen es Anwendern ermöglichen, benutzerdefinierte Bedingungen zu formulieren, bei deren Eintritt eine Simulation automatisch angehalten wird. Während dieses Haltepunkts soll der Anwender die Möglichkeit erhalten, über eine interaktive Konsole in den aktuellen Zustand der simulierten Anlage einzusehen, Variablen abzufragen, Werte zu verändern und anschließend die Simulation fortzusetzen. Damit sollte eine Funktionalität geschaffen werden, die einerseits die Fehleranalyse und Validierung von Modellen erleichtert und andererseits das Debugging komplexer Simulationsabläufe – insbesondere solcher mit eingebundenen Python-Steuerfunktionen

(z. B. zur Regelung oder Ansteuerung von GCSM-Signalen) – erheblich verbessert. Die Umsetzung war in Form von zusätzlichen Controller-Klassen geplant, die sowohl in interaktiven Anwendungen wie z. B. ATLASneo als auch in Python-Skripten verwendet werden können.

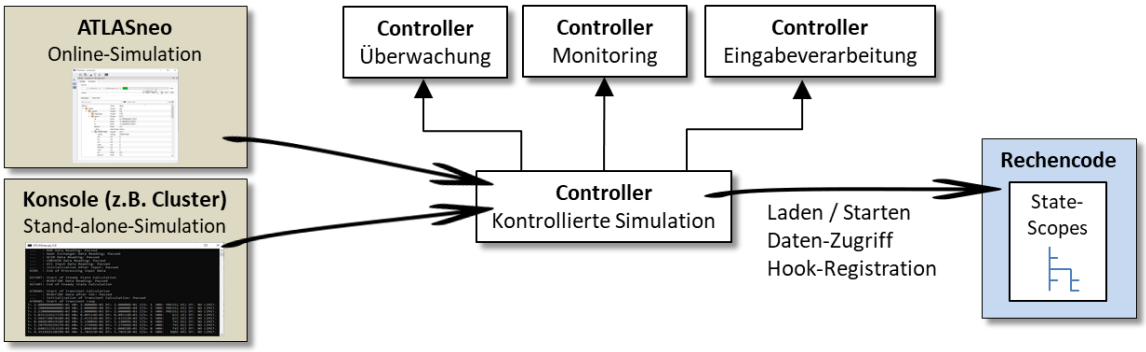

Abb. 3.14 Zusammenstellung eines Simluations-Controllers. Mixin-Klassen bieten verschiedene Funktionen und werden nach Bedarf durch Ableitung kombiniert

Konzeption und prototypische Implementierung

Zur Realisierung wurden zwei alternative Ansätze zur Implementierung einer textbasierten interaktiven Steuerkonsole untersucht und prototypisch umgesetzt:

- **Command-Shell-Ansatz auf Basis des Python-Packages `cmd`**
  Dieser Ansatz ermöglicht es, eine Simulation schrittweise ablaufen zu lassen und interaktiv über eine Kommandozeile zu steuern. Der Anwender kann während des Simulationslaufs Befehle eingeben, um Werte zu überwachen, Zwischenergebnisse abzufragen oder die Simulation fortzusetzen. Erste Ablauftests zeigten, dass dieser Ansatz eine einfache und robuste Grundlage bietet, um interaktive Steuerbefehle in laufende Simulationen einzubinden.

- **Escape-Shell auf Basis des Python-Debuggers `pdb`**
  Der zweite Ansatz erweitert das Prinzip der interaktiven Steuerung um die Möglichkeit, Bedingungen zum automatischen Anhalten festzulegen. Die Simulation wird hierbei beim Erreichen definierter Zustände (z. B. bestimmter Signalwerte, Druckgrenzen oder Steuerparameter) angehalten, und der Anwender erhält Zugriff auf eine interaktive Debugging-Shell. Diese erlaubt es, den aktuellen Simulationszustand zu analysieren, Variablen zu inspizieren und nach Bedarf manuell in den Simulationsablauf einzugreifen. Dadurch kann das Verhalten von Anwender-Funktionen, die in Python-Skripte eingebettet sind, gezielt überprüft werden.

Bewertung und Ausblick

Beide Ansätze konnten im Rahmen des Projekts erfolgreich getestet und funktional bewertet werden. Die Ergebnisse zeigen, dass insbesondere der `pdb`-basierte Ansatz großes Potenzial für die geplante Funktion der *conditional breakpoints* bietet, da er bereits Mechanismen zur Ablaufkontrolle, zum Variablenzugriff und zur Zustandsinspektion bereitstellt. Für den praktischen Einsatz in komplexen und langlaufenden Simulationen sind jedoch weitere Entwicklungsarbeiten erforderlich:

- Integration der Funktionalität in die bestehenden Controller-Klassen-Hierarchie,
- Definition einer benutzerfreundlichen Syntax zur Formulierung der Haltebedingungen,
- Anwendergeeignete Bündelung und Bereitstellung der Zugriffsmöglichkeiten in der interaktiven Konsole,
- Anpassungen für gemeinsamen Einsatz mit Controllern unterschiedlicher Arten der Simulations-Durchführung (schrittweise, Protokoll-gesteuert, ausgelagert).

Aufgrund der erforderlichen Anpassungen an die verschiedenen Simulationsumgebungen und den vielfältigen Anforderungen für den praktischen Einsatz konnte die vollständige Implementierung dieser Erweiterungen im aktuellen Projektzeitraum nicht abgeschlossen werden. Die begonnene Entwicklung soll aber im Rahmen eines nachfolgenden Projekts fortgeführt werden, um die prototypisch implementierten Funktionen zu einer stabilen und allgemein nutzbaren Komponente des Controller-Systems ausbauen und den Anwendern bereitstellen zu können.

3.2.3 Weiterentwicklung Monitoring und Simulationssteuerung

Das Entwicklungsziel dieses Arbeitspunktes war die Erweiterung der Simulationssteuerung und des Monitorings in der Online-Umgebung von ATLASneo, um Protokolle auch in interaktiven Simulationen nutzbar zu machen. Bisher war die Verwendung von Protokollen für die automatisierte Durchführung vordefinierter Eingriffe, beispielsweise zur Nachbildung von Bedienhandlungen, ausschließlich im Batchbetrieb möglich. Für den interaktiven Einsatz innerhalb von ATLASneo mussten daher die verschiedenen Controller-Klassen so erweitert werden, dass sie in Kombination lauffähig sind und eine synchrone Kommunikation zwischen dem ATLASneo-Frontend und den Simulations-

prozessen gewährleisten. Ziel war es, eine robuste und anwenderfreundliche Steuerung zu ermöglichen, die sowohl geskriptete Protokolle als auch manuelle Eingriffe unterstützt.

Erweiterung der Controller und Fehlerbehandlung

Ein Schwerpunkt der Arbeiten lag auf der technischen Abstimmung und Erweiterung der Controller-Klassen, um deren Zusammenspiel in der interaktiven Umgebung sicherzustellen. Insbesondere der Controller für die interaktive Simulation in der Qt-basierten ATLASneo-Oberfläche wurde um Mechanismen erweitert, die eine stabile Verarbeitung asynchroner Ereignisse ermöglichen. Hierzu wurde das **Signal-Handling** überarbeitet, um asynchrone Ausgaben (`print`-Statements) und Statusmeldungen zu behandeln. Außerdem wurde die Behandlung von **unerwarteten Exceptions** verbessert, damit Fehler, die während der Ausführung von Protokollen auftreten, kontrolliert abgefangen und an das User-Interface weitergegeben werden können. Diese Maßnahmen waren wesentlich, um die Robustheit der interaktiven Simulation auch unter Verwendung von Protokollen zu gewährleisten.

Erweiterung der Protokollverarbeitung

Ein weiterer Entwicklungsaspekt betraf die Kompilation und Ausführung von Protokoll-Definitionen. Die Protokollverarbeitung wurde so erweitert, dass nun auch relative Importe unterstützt werden. Dadurch können komplexere, modular aufgebaute Protokollstrukturen erstellt und wiederverwendet werden, was insbesondere für größere Szenarien und standardisierte Versuchsabfolgen von Vorteil ist. Diese Funktionalität verbessert die Wartbarkeit und Erweiterbarkeit von Protokollen und schafft die Grundlage für die Einbindung umfangreicher Simulationsskripte in automatisierte und interaktive Abläufe.

Anpassung der Benutzeroberfläche

Um die neue Protokollfunktionalität für Anwender zugänglich zu machen, wurde das User-Interface der ATLASneo-Online-Simulation angepasst. Die Eingabemasken erlaubten dem Anwender im ATLASneo-Frontend bereits die grundlegenden Angaben, wie das Controller-Modul und die zu verwendende Controller-Klasse anzugeben. Darüber hinaus ist es jetzt auch möglich alle Controller-spezifischen Parameter direkt über die Oberfläche einzugeben. Abhängig vom Typ oder auch Erweiterungen durch den

Anwender können hier verschiedenste Argumente erforderlich werden. Durch eine neue Parameter-Eingabezeile können sowohl Positions- als auch Keyword-Argumente spezifiziert werden, die bei der Erstellung der Simulation direkt an die Controller-Klasse weitergeleitet werden.

Besonders für die im Protokoll spezifizierten Ausgaben, wie die überwachten Simulationsgrößen, die formulierten Trigger und die dadurch evtl. ausgelösten Actions wurde ein separater **Console**-Tab eingeführt der diese Ausgaben von anderen Konsolenmeldungen getrennt darstellt (Abb. 3.15). Damit ist die protokollgesteuerte Steuerung und Überwachung auch in interaktiven Sitzungen anwendbar, wodurch der bisher getrennte Workflow zwischen stand-alone und Online-Simulation vereinheitlicht wurde.

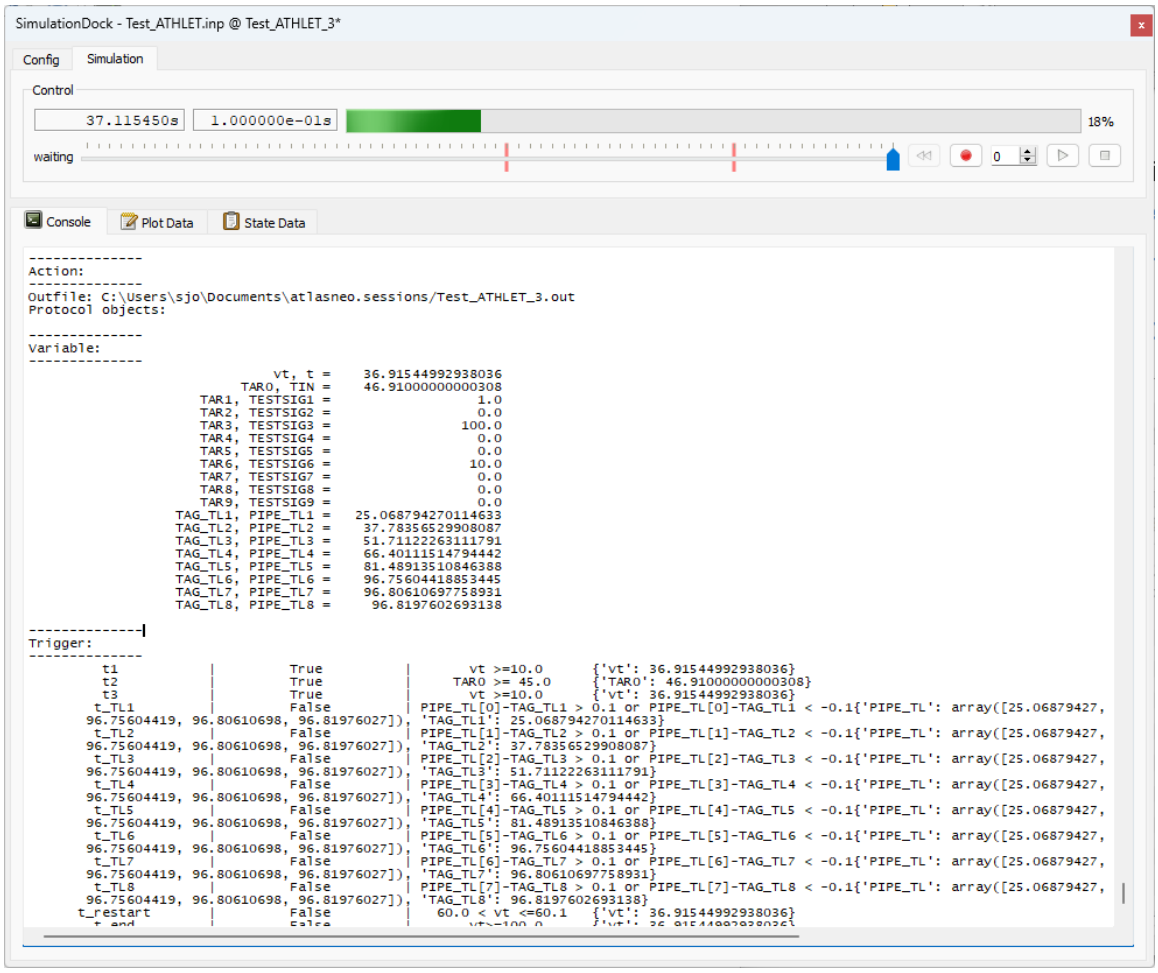

Abb. 3.15 Benutzerdefinierte Protokoll-Ausgaben werden im neuen Console-Tab separat gesammelt und erlauben die Nachverfolgung der geskripteten Handmaßnahmen

Ausblick: Textbasiertes Kontroll-Interface

Ein geplanter, jedoch im verbleibenden Projektzeitraum nicht mehr umgesetzter Arbeitspunkt war die Entwicklung eines rein textbasierten Kontroll-Interfaces zur Überwachung der direkten Interaktion mit der Simulation. Ein solches Interface würde eine rein Terminal-basierte Bedienung ermöglichen und könnte insbesondere in Kombination mit den im Projekt entwickelten Simulations-Controllern für *conditional breakpoints* und interaktive Kontrolle (siehe Abschnitt 3.2.2) einen deutlichen Mehrwert bieten. Hierdurch könnten Anwender, unabhängig von der graphischen Oberfläche Simulationen starten, diese überwachen und Parameter verändern. Dieser Ansatz sollte im Rahmen eines Nachfolgeprojekt weiterverfolgt werden.

3.2.4 Verbesserung der Anwendbarkeit bisheriger Entwicklungen

Ziel dieses Arbeitspunkts war die Optimierung der Anwendbarkeit und Stabilität der im Projekt bereits entwickelten Komponenten zur Simulationsdurchführung und -steuerung. Auf Grundlage von Rückmeldungen interner Anwender sollten gezielte Fehlerbehebungen sowie funktionale und ergonomische Verbesserungen umgesetzt werden, um den praktischen Einsatz der Softwarekomponenten im Arbeitsumfeld zu erleichtern. Konkret umfassten die Planungen

- die Erweiterung der Controller-Commandline-Interfaces (CLI), um neue Optionen der AC²-Rechencodes auch in stand-alone Simulationen nutzen zu können,
- den Ausbau der CLI-Hilfetexte, um Anwendern eine einfachere Orientierung und Hilfestellung zu implementierten Optionen zu bieten,
- sowie die Überarbeitung des ATLASneo-Steuermoduls für Online-Simulationen mit dem Ziel einer höheren Stabilität, Reaktionsfähigkeit und Benutzerfreundlichkeit. Dabei sollten insbesondere Bedienblockaden und Performanceprobleme bei großen Simulationsläufen reduziert werden.

Überarbeitung des Steuermoduls der ATLASneo Online-Simulation

Ein zentraler Schwerpunkt lag auf der grundlegenden Überarbeitung des Steuermoduls für die Online-Simulation in ATLASneo. Das bisherige Modul wurde in mehreren Schritten technisch modernisiert und funktional erweitert.

- Die **Qt-Benutzeroberfläche** wurde vollständig neu entwickelt und kann nun dynamisch zur Laufzeit geladen werden, anstatt vorab mit dem User Interface Compiler (UIC) kompiliert werden zu müssen. Dadurch erhöhte sich die Flexibilität bei der Entwicklung und die Wartbarkeit des Moduls deutlich. Die neue Benutzeroberfläche wurde parallel zur vorherigen bereitgestellt, um internen Anwendern die Möglichkeit zu geben, beide Varianten im Praxiseinsatz zu testen.

- die **Konsolenausgaben** der Controller, werden bei Anwendung innerhalb von ATLASneo über Signale an das Hauptfenster weitergeleitet und **synchronisiert** ausgegeben, was insbesondere bei nebenläufig gestarteten Simulationen wichtig ist, um Abstürze und unkontrolliertes Verhalten zu verhindern.

- Die interne **Zeitschrittsteuerung** wurde überarbeitet, um den Ablauf interaktiver Online-Simulationen zu beschleunigen und gleichmäßiger zu gestalten.

- Der bislang ausschließlich zur Auswahl von Restart-Punkten genutzte **Slider wurde auf die gesamte Zeitachse erweitert** und erlaubt nun die Auswahl beliebiger Zeitschritte. Zeitschritte mit Restart-Punkten werden im Slider zusätzlich durch eine rote Markierung hervorgehoben und können gezielt angewählt werden (vgl. Abb. 3.16). Die Auswahl des Zeitschritts kann jetzt sowohl über den Index als auch direkt über Eingabe des Zielzeitpunkts erfolgen. Zudem ist der Slider nun mit dem PlotData-Tree gekoppelt: Der ausgewählte Zeitschritt bestimmt den im Baum dargestellten Datenstand. Damit können Anwender auch bei laufender Simulation in frühere Zeitpunkte der Plotdaten zurückspringen, ohne die Simulation selbst zurücksetzen zu müssen.

- Die **Controller-Klassen** zur interaktiven Steuerung wurden angepasst, um die zukünftige Integration ausgelagerter Simulationsläufe vorzubereiten.

- Neue **Eingabeparameter für Simulation Restarts und Protokollverarbeitung** wurden ergänzt, wodurch diese jetzt direkt in der Oberfläche des Steuermoduls eingestellt werden können:
  - **Restart-Parameter** können nun in separaten Eingabefeldern angegeben und zugehörige Dateien über Dateidialoge ausgewählt werden, was die Einrichtung von Simulations-Restarts deutlich erleichtert.
  - Die **Simulation-ID**, welche für die Benennung von Arbeitsdateien verwendet wird, lässt sich nun halbautomatisch bestimmen. Das Format lässt sich jetzt mit Platzhalter festlegen und ermöglichen das automatische Hochzählen von Indizes, wodurch versehentliches Überschreiben früherer Ergebnisse verhindert wird.
  - Das **interaktive Testen von GCSM-Handmaßnahmen** wurde verbessert: Änderungen an Signalparametern (`iopt`, `xarg`, `gain`) können abschnittsweise vorgenommen und werden bei Rücksprüngen in der Simulation automatisch rückgängig gemacht. Diese Funktion erforderte auch Anpassungen an der Restart-Routine in ATHLET/CD.
  - Eingabefelder zur Definition und **Parameterübergabe für Protokolle** wurden integriert und freie Parameter können nun direkt an den zugehörigen Controller weitergeleitet werden.

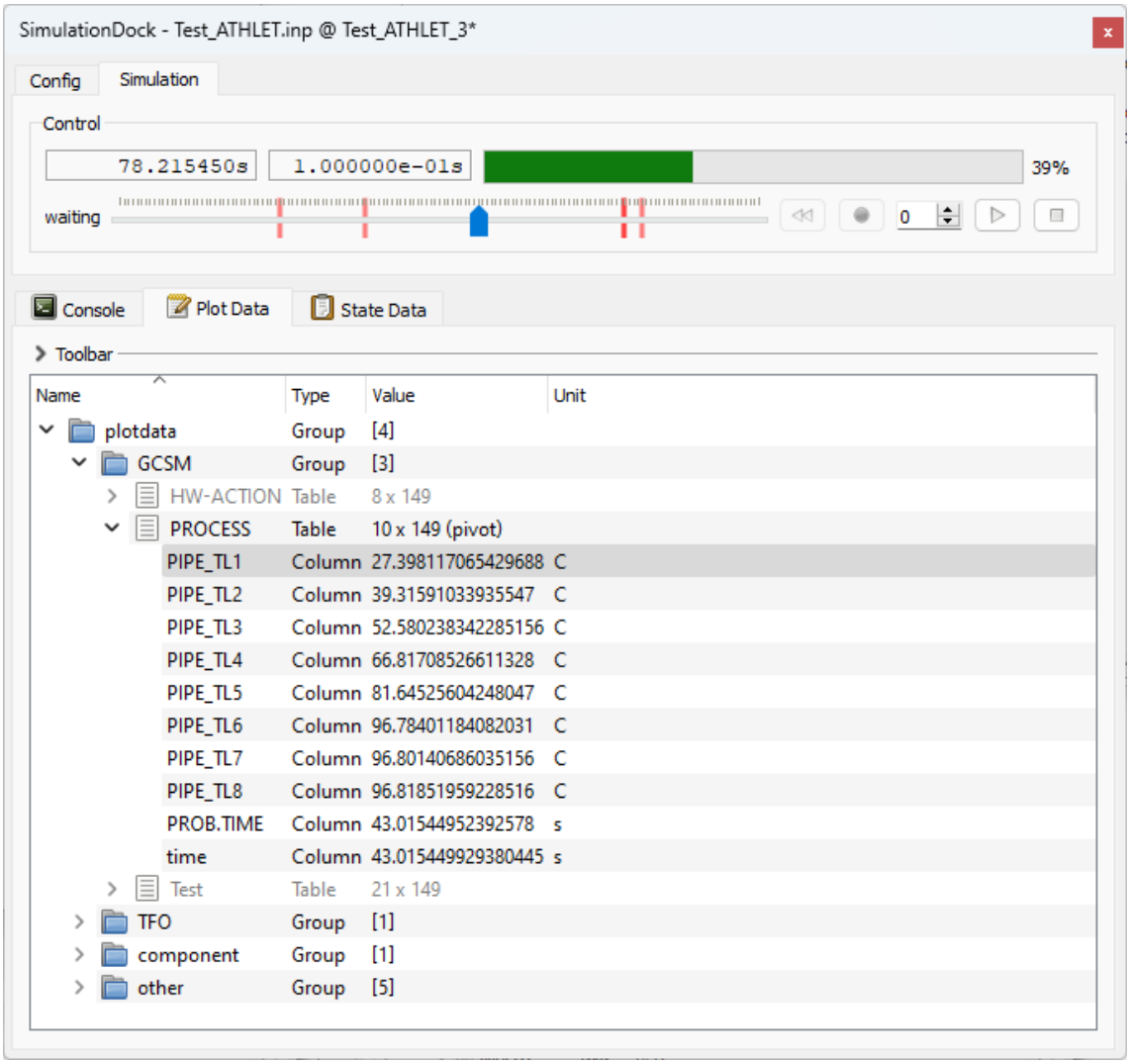

Abb. 3.16 Das überarbeitete Steuermodul der Online-Simulation erlaubt den Zugriff sowohl auf alle Zeitschritte als auch auf die im Slider markierten Restart-Punkte (rot)

Diese Arbeiten führten zu einer spürbar verbesserten Stabilität, Bedienbarkeit und Performance des Online-Simulationsbetriebs, welcher dadurch eine deutlich höhere Flexibilität und Fehlertoleranz im interaktiven Umgang mit laufenden Simulationen aufweist.

Integration und Erweiterung der Protokollverarbeitung

Mit der Version 1.1.0 des ATLASneo-Anwendungspakets (siehe Abschnitt 3.3.1) wurde die Verarbeitung von **Protokolldefinitionen für geskriptete Handmaßnahmen** in das Standard-Python-Paket für ATHLET integriert. Neben der Protokollüberwachung sind nun auch benutzerdefinierte, periodische Textausgaben möglich, mit denen der aktuelle Simulationszustand während der Laufzeit verfolgt werden kann. Diese periodischen

Ausgaben verbessern die Nachvollziehbarkeit des Verhaltens Protokoll-gesteuerter Simulationen erheblich und erleichtern die Anwendung von Protokollen insbesondere im interaktiven Betrieb der ATLASneo Online-Simulation.

Anwenderfeedback, Support und Fehlerbehebungen

Nach der Bereitstellung der Versionen **1.0.0**, **1.1.0** und **1.1.1** des ATLASneo-Anwendungspakets gingen, wie erwartet, zahlreiche Rückmeldungen und Fehlerberichte von internen Anwendern ein. Diese wurden im Rahmen kontinuierlicher Supportarbeiten bearbeitet und teilweise unmittelbar behoben. Beispiele für kurzfristig umgesetzte Verbesserungen sind

- die Korrektur der Darstellung von Retrace-Punkten in der Online-Simulation sowie
- die Absicherung von GUI-Komponenten bei mehreren geöffneten Online-Simulation Panels, da die parallele Ausführung von AC²-Simulationen bislang nicht möglich ist.

Andere gemeldete Punkte wurden als Issues für die weitere Entwicklung aufgenommen und werden in den Folgeversionen priorisiert umgesetzt.

3.2.5 Allgemeine Wartungsarbeiten

Die allgemeine Zielsetzung dieses Arbeitspunkts bestand darin, die im Projekt entwickelten Softwarekomponenten zur **Durchführung und Steuerung von Simulationen** dauerhaft **betriebsfähig**, **kompatibel und wartbar** zu halten. Da die zugrunde liegenden Laufzeitumgebungen und Packages, insbesondere Python und das Qt-Binding PySide (/PYS 17/ & /PYS 22/), kontinuierlich weiterentwickelt werden, war es notwendig, die eigenen Codestrukturen regelmäßig an diese Änderungen anzupassen. Damit sollten langfristig Stabilität, Kompatibilität und Nachhaltigkeit der entwickelten Software gewährleistet werden. Die Schwerpunkte der dabei notwendigen Wartungsarbeiten waren

- **Sicherstellung der Lauffähigkeit** auf aktuellen Plattformen und Betriebssystemen (Windows, Linux),
- **Anpassung** an neue *Python*-Versionen,
- **Umstellung** des Qt-Bindings *PySide* → *PySide2*,
- **Anpassung** an neue Versionen der Schnittstellenbibliothek *FDE* und Controller-Klassen,

- **Integration der Änderungen in die CI/CD** (Continuous Integration/Deployment) - Umgebung zur automatisierten Testdurchführung und Paketierung.

Aktualisierung der Schnittstellenbibliotheken und Controller

Ein Teil der Wartungsarbeiten bestand in Anpassungen zur Aktualisierung der FDE-Schnittstellenbibliothek /FDE 25/, die für die Steuerung von Simulationsprozessen und den Zugriff auf deren Laufzeitdaten verantwortlich ist. Mit der Integration der neuesten FDE-Version wurden auch die zugehörigen Controller-Klassen für die Online-Simulation angepasst, um die reibungslose Kommunikation zwischen Simulations-Backend und Benutzeroberfläche weiterhin sicherzustellen. Diese Aktualisierungen betrafen insbesondere die Datenübertragung zwischen den laufenden Simulationen und den interaktiven Steuerungsmodulen, wie z. B. ATLASneo und stellten sicher, dass alle Komponenten kompatibel mit den neuen Funktionsaufrufen der Bibliothek bleiben.

Anpassungen für Linux-Kompatibilität

Um die plattformübergreifende Lauffähigkeit der Online-Simulation zu gewährleisten, wurden gezielte Anpassungen für den Betrieb unter Linux vorgenommen. Eine der notwendigen Anpassungen diente dabei dem korrekten Setzen von Environment-Variablen, die insbesondere beim Laden dynamischer Bibliotheken (*shared objects*, `.so`) eine entscheidende Rolle spielen und mit im Environment definierten Funktionen kompatibel sein müssen. Hierdurch konnte sichergestellt werden, dass die Softwareumgebung für Linux-Systeme zuverlässig arbeitet und dieselbe Funktionalität wie unter Windows bereitstellt, was ein wichtiger Schritt zur Integration in heterogene Simulationsumgebungen ist.

Pflege und Standardisierung des Python-Environments

Für die Erstellung des freigegebenen Anwendungspakets war eine **Standardisierung und Wartung der Python-Umgebung** erforderlich. Hierbei wurden

- die im Projekt verwendeten Python-Pakete aufeinander abgestimmt,
- Kompatibilitätsprobleme durch gezielte Code-Anpassungen behoben,
- und die für den Release **v1.0.0 festgelegte Python-Version 3.7** als Referenzumgebung definiert, da sie zu diesem Zeitpunkt die größte Stabilität bei gleichzeitiger Abwärtskompatibilität gewährleistete.

Mit der Weiterentwicklung der Python-Laufzeitumgebung (Versionen 3.8 bis 3.11) ergaben sich unerwartete Änderungen im Fehlerverhalten der Controller-basierten Simulationen. Insbesondere zeigte sich, dass in Hook-Routinen ausgelöste **Exceptions** ab Python 3.8 nicht mehr korrekt an die höheren Routinen weitergereicht wurden, sondern vom Interpreter stillschweigend ignoriert wurden. Dieses Verhalten hatte direkte Auswirkungen auf die Fehlerbehandlung und Stabilität der Simulationssteuerung und machte eine eingehende Analyse erforderlich. Durch gezielte Anpassungen an den **Exception-Router**-Mechanismen konnte das Verhalten korrigiert und eine vollständige Kompatibilität zu Python 3.11 erreicht werden. Diese Arbeiten waren Voraussetzung für das **Update-Release v1.1.0**, das inzwischen auf der **Python-Version 3.11** basiert und somit auch zukünftige Wartbarkeit sicherstellt.

Erweiterung der Test- und CI-Infrastruktur

Parallel zu den Softwareanpassungen wurden auch die CI/CD-Prozesse weiterentwickelt, um eine verbesserte Testabdeckung der Online-Simulation zu erreichen. Hierzu wurden

- die Definitionen der CI-Jobs erweitert und restrukturiert,
- die im Testlauf verwendete ATHLET-Version aktualisiert,
- sowie Anpassungen in den Aufrufroutinen und Steuermechanismen vorgenommen, um eine konsistente Testausführung sicherzustellen.

Diese Maßnahmen ermöglichen es nun, regelmäßige und automatisierte Integrations- und Regressionstests durchzuführen, wodurch die Softwarequalität und die Verlässlichkeit der Releases langfristig verbessert werden.

3.3 AP 3: Ergebnis-Analyse und Visualisierung

Um eine schnelle und umfassende Anwendbarkeit von ATLASneo zu erreichen, wurde im Projektverlauf auch bei der Ergebnis-Analyse und Visualisierung eine enge Zusammenarbeit zwischen Entwicklern und Anwendern angestrebt. Dazu musste die Anwendbarkeit der Werkzeuge zur Analyse und Visualisierung von Berechnungsergebnissen durch ein einheitliches Datenformat sichergestellt werden. Als weiterer Punkt, um die Schnittmenge der produktiven Bereiche von Entwicklern und Anwendern zu erhöhen, war die Anbindung an Jupyter-Notebooks /JUP 25/ geplant. Auch die Ergebnisdar-

stellung durch dynamisierte Bilder und andere Formen der Visualisierung musste weiterentwickelt werden, um eine produktive Nutzbarkeit von ATLASneo schnell zu fördern. Zusammenfassend ergaben sich für diesen Arbeitspunkt die folgenden Themengebiete.

3.3.1 Erstellung Anwendungspaket

Ziel dieses Arbeitspunktes war es, den aktuellen Entwicklungsstand von ATLASneo so aufzubereiten, dass die stabilen und ausgereiften Funktionsteile als installierbares Anwendungspaket auch für externe Anwender bereitgestellt werden können. Hierzu sollten die bislang entwickelten Module (sogenannte Gadgets) vereinheitlicht, auf eine konsistente Python-Version migriert und in einer lauffähigen Umgebung zusammengeführt werden. Neben der technischen Integration waren auch die Erstellung der notwendigen Laufzeitumgebung, Lizenz- und Anwenderdokumentation sowie die Ausarbeitung geeigneter Installationsroutinen vorgesehen. Ziel war ein vollständiges, getestetes und dokumentiertes Softwarepaket, das eine einfache Installation und Nutzung von ATLASneo ermöglicht.

Konsolidierung und Aufbau der Python-Laufzeitumgebung

Ein wesentlicher Bestandteil der Arbeiten war die Erstellung und Optimierung der Python-Laufzeitumgebung (Environment), die als Grundlage sowohl für ATLASneo als auch für begleitende Hilfsprogramme dient. Dabei wurde besonderer Wert auf eine stabile und kompakte Zusammenstellung der erforderlichen Python-Pakete gelegt, um eine handhabbare Installationsgröße zu erreichen.

Im Zuge der Weiterentwicklung erfolgte ein mehrstufiges Upgrade der Python-Version: Zunächst wurde die Umgebung für das Initial Release 1.0.0 auf Python 3.7 festgelegt, um größtmögliche Kompatibilität sicherzustellen. Später wurde für das Update-Release 1.1.0 der Umstieg auf Python 3.11 vollzogen, um sicherheitsrelevante Schwachstellen zu beseitigen und die langfristige Wartbarkeit zu gewährleisten. Parallel wurde der Python-Basis-Installationsmanager von miniconda auf miniforge umgestellt, sodass nur noch OpenSource-Pakete, frei von Copyleft-Lizenzen, zur Erstellung des Environments verwendet werden.

Ergänzend wurden die Skripte zur Einrichtung des Python-Environments erweitert, sodass nun sowohl `conda`- als auch `pip`-basierte Paketinstallationen unterstützt werden. Dadurch steht eine flexible Methode zur Verfügung, um die benötigten Pakete im

Environment zusammenzustellen, zu aktualisieren und die jeweilige Paketkonfiguration einschließlich aller Versionsangaben in Form von requirements-Dateien abzulegen. Auf diese Weise lassen sich Environments konsistent reproduzieren, was sowohl für die Weiterentwicklung von ATLASneo als auch für die Erstellung zukünftiger Releases von zentraler Bedeutung ist.

Dokumentation von Lizenzinformationen

Zur Sicherstellung der rechtlichen und technischen Nachvollziehbarkeit wurde die Dokumentation, der in ATLASneo verwendeten **Third-Party- und OpenSource-Komponenten** erstellt und fortlaufend aktualisiert. Hierfür wurde ein Hilfsskript (*licenseCrawler*) entwickelt, das die in der Anwendung verwendeten Python-Pakete, deren Versionen und Lizenzinformationen automatisch erfasst und eine Lizenzübersicht mit integrierten Lizenztexten erzeugt. Da Lizenzangaben in Abhängigkeit von Quelle und Paket in den verschiedenen Paket-Datenbanken häufig unvollständig oder uneinheitlich sind, versucht das Tool durch heuristische Verfahren fehlende Informationen zu ergänzen und weist unvollständige Einträge zur manuellen Nachbearbeitung aus. Die so erstellte Lizenzdokumentation wird mit jeder neuen Version aktualisiert und dem Anwendungspaket in Form eines formatierten HTML-Dokuments beigelegt.

Anwenderdokumentation

Ein weiterer Schwerpunkt lag auf der Erstellung und kontinuierlichen Erweiterung der Anwenderdokumentation. In einem Workshop mit internen Nutzern wurden die wichtigsten Funktionsbereiche von ATLASneo identifiziert und in Folgenden deren Bedienung beschrieben. Die Dokumentation wurde in strukturierter Form innerhalb der Wiki-Seiten des GitLab-Projekts erstellt, um sowohl eine funktionale Einführung als auch praxisnahe Beispiele und Screenshots von internen Anwendern miteinfließen lassen zu können. Sie wird dem Anwendungspaket im PDF-Format beigelegt und ist komfortabel über einen Direktzugriff im „About“-Dialog von ATLASneo erreichbar. Im Zuge der späteren Releases wurden sämtliche Inhalte regelmäßig aktualisiert, um neue Funktionen und Änderungen der Benutzeroberfläche in der Dokumentation zu berücksichtigen.

Erstellung des Anwendungspakets und Automatisierung der Paketierung

Für die eigentliche Erstellung des Anwendungspakets wurden alle notwendigen Komponenten, Bibliotheken und Konfigurationsdateien in einem konsistenten Build-Prozess

zusammengeführt. Hierzu wurde eine *PyInstaller*-Konfiguration erarbeitet, die die Erzeugung eines sogenannten Binary Bundles ermöglicht, welches ATLASneo zusammen mit dem passenden Python-Interpreter und alle benötigten Abhängigkeiten enthält. Ein Skript steuert diesen Prozess automatisiert, sowohl lokal als auch innerhalb der GitLab-CI/CD-Umgebung, wodurch eine reproduzierbare Paketierung der freizugebenden Versionen gewährleistet wird. Für die Erzeugung der Installationsdatei wurde *InnoSetup* eingesetzt, welches den Ordner des Binary Bundle komprimiert und daraus einen benutzerfreundlichen Setup-Installer erstellt.

Im Projektverlauf wurden die entsprechenden CI/CD-Pipelines regelmäßig gewartet und an neue Anforderungen angepasst, insbesondere im Hinblick auf die bevorstehende Integration der Komponenten zur generischen Datenvisualisierung in den Hauptentwicklungszweig von ATLASneo. Dadurch wurde eine durchgängige und weitgehend automatisierte Prozesskette zur Erstellung von Releases geschaffen.

Bereitstellung und Weiterentwicklung der Releases

Im Rahmen dieser Arbeiten konnten sukzessive mehrere stabile Versionen von ATLASneo veröffentlicht werden:

- **Initial Release v1.0.0 (Dez. 2023):** Erste offizielle Bereitstellung eines lauffähigen und getesteten Installationspakets.
- **Update-Release v1.1.0 (Jun. 2024):** Aktualisierung von Python und FDE auf die Versionen 3.11 bzw. 2.8.3, Überarbeitung der Lizenz, Entfernung der Python 2.7-Kompatibilität und der dadurch obsoleten Kompatibilitätspakete (z. B. qt.py).
- **Patch-Release v1.1.1 (Apr. 2025):** Aktualisierung der integrierten Python-Pakete, Build-Skripte, Lizenzdokumentation und Anwenderunterlagen.
- **Update-Release v1.2.0 (Sep. 2025):** Aktualisierung der integrierten Python-Pakete, Build-Skripte, Lizenzdokumentation und Anwenderunterlagen. Vorbereitende Mitaufnahme anstehender Abhängigkeiten.

Die Build-Konfigurationen und CI/CD-Pipelines wurden fortlaufend angepasst und verbessert. Zum Projektende September 2025 konnte ein weiteres Update-Release fertiggestellt werden, welches letzte Fehlerbehebungen integriert und durch Umstellungen einiger Python-Packages die zukünftige Integration der generischen Datenvisualisierung vorbereitet.

3.3.2 Ausbau Unterstützung Ausgabeformate

Da ATLASneo auf dem HDF5-Format /HDF 25/ als zentraler Datengrundlage basiert – einem in wissenschaftlichen Anwendungen weit verbreiteten Format zur effizienten Speicherung und Verarbeitung großer Datenmengen – war es Ziel dieses Arbeitspunktes, den bereits vorhandenen Datenkonverter **x2hdf5** weiterzuentwickeln und um zusätzliche verarbeitbare Datenformate zu erweitern. Dadurch sollte die Integration und Visualisierung von Ergebnissen weiterer Rechenprogramme ermöglicht werden, insbesondere solcher, die bislang ausschließlich ASCII-basierte Ausgaben erzeugen. Damit sollte die bestehende Funktionalität, die bereits die Verarbeitung von Ergebnissen aus **ATHLET, ATHLET-CD** und **COCOSYS** unterstützt, auf weitere Datenquellen ausgedehnt werden.

Erreichte Ergebnisse

Im Rahmen der Arbeiten wurde das Commandline-Interface (CLI) des Konverters überarbeitet und um neue Optionen erweitert, siehe Abb. 3.17. Daneben wurden einige interne Fehlerkorrekturen und Verbesserungen vorgenommen:

- **Unterstützung zusätzlicher Datenformate:**
  Der Konverter wurde um die Fähigkeit ergänzt, NetCDF-Dateien zu verarbeiten und in das ATLASneo-kompatible HDF5-Format zu überführen. Dies stellt eine wichtige Erweiterung dar, da das NetCDF-Format bereits in ATLAS für die Verarbeitung von Simulationsdaten eingesetzt wurde. Darüber hinaus ist es in vielen wissenschaftlichen und technischen Anwendungen etabliert und ermöglicht somit die Kompatibilität mit Ergebnisdaten weiterer Rechencodes.

- **Fehlerbehebungen und Stabilitätsverbesserungen:**
  Im Zuge der Weiterentwicklung wurden verschiedene kleinere Fehler korrigiert, um die Zuverlässigkeit und Robustheit der Konvertierung zu erhöhen. Dies betrifft insbesondere die Verarbeitung größerer Datenmengen und den Umgang mit fehlerhaften Eingabedateien.

- **Anwenderfeedback und kontinuierliche Optimierung:**
  Nach der Bereitstellung des ATLASneo-Anwendungspakets wurde der Konverter auch externen Anwendern zugänglich gemacht. Dadurch ergaben sich Rückmeldungen, die zur Identifizierung weiterer kleinerer Fehler und Verbesserungsmöglichkeiten führten. Diese wurden umgesetzt, und gleichzeitig Anregungen für zukünftige Funktionserweiterungen gesammelt.

```
tools> python x2hdf5 --help
Usage: x2hdf5 [OPTIONS] DATAFILE...

  x2hdf5 Data Converter

  Arguments:
      DATAFILE  The data file to convert into HDF5-format (supports POSIX wildcards).
                Supported input formats: .pd, .ptf, .plt, .ncf
      OUTFILE   The path and name of the output file.
                If not existent the path gets created.
      OUTDIR    The output directory where to store output files.
                It is ignored if OUTFILE specifies an explicit path.
      ORDER     The order in which to process multiple DATAFILES.
                One out of: name, size, atime, ctime, mtime, asis
      RANGE     The range needs to be given in format <start>:[<stop>[:<step>]].
                Note that <start> is 0-based and <stop> is exclusive!
      PIVOT     The name and unit for the pivot column given in comma separated pair: <name>,<unit>

  Auto variables:
    {CWD}                current working directory
    {DATAFILE.name}      the basename of DATAFILE
    {DATAFILE.parent}    the parent directory of DATAFILE
    {DATAFILE.stem}      the basename of DATAFILE without suffix
    {DATAFILE.suffix}    the extension (including .) of DATAFILE

Options:
  -o, --output TEXT           The path and name of the output file. If not existent
                              the path gets created. [default: {OUTDIR}/{DATAFILE.stem}.h5]
  -d, --outdir TEXT           output directory.  [default: {CWD}]
  -p, --pivot TEXT            the name and unit of the pivot-column, '' => no pivot. [default: time,s]
  -I, --info-only             show time step info only, skip file conversion.
  -m, --maxcol INTEGER        limit number of table columns, 0 => no limit.  [default: 255]
  -s, --sort [name|size|atime|ctime|mtime|asis]
                              process files in given ascending order.  [default: asis]
  -r, --root TEXT             name of root group.  [default: plotdata]
  --range TEXT                the range of timesteps considered in data conversion.  [default: 0:]
  --verbosity INTEGER         verbosity level.  [default: 1]
  -E, --on-error [ignore|next|stop]
                              sets how to go on after an conversion error:
                              `stop`, process `next` file (exit with errorcode) or
                              `ignore` like next but exit without errorcode.  [default: next]
  --version                   Show the version and exit.
  --develop                   Enable developer mode.
  -h, --help                  Show this message and exit.
```

Abb. 3.17 Das überarbeitete Commandline-Interface (CLI) und die Optionen von x2hdf5

3.3.3 Anbindung an Jupyter-Notebooks

Ziel dieses Arbeitspunktes war es, die Nutzung von **Jupyter-Notebooks** als interaktive Analyseplattform enger mit **ATLASneo** zu verknüpfen. Da beide Systeme auf der Programmiersprache Python basieren, bieten sich hier erhebliche Synergien: Anwender können über Notebooks komplexe Analyse- und Visualisierungsaufgaben flexibel durchführen, während gleichzeitig eine enge Kopplung an bestehende ATLASneo-Funktionalitäten ermöglicht wird. Geplant war, typische Workflows, wie das Einlesen und Verarbeiten von HDF5-Daten, als **Notebook-Prototyp** bereitzustellen und die dafür benötigten Funktionen in Form wiederverwendbarer **Python-Packages** zusammenzufassen. Diese sollten den Anwendern als Grundlage für eigene, anpassbare Analyse-

szenarien dienen. Langfristig sollen auf diese Weise erprobte Workflows bei Bedarf zu vollwertigen Funktionsmodulen (Gadgets) innerhalb von ATLASneo weiterentwickelt werden.

Erreichte Ergebnisse

Im Rahmen der durchgeführten Arbeiten wurde zunächst eine **technische Analyse potenzieller Integrationspfade** zwischen ATLASneo und Jupyter-Notebooks vorgenommen. Dabei standen insbesondere die folgenden Themenfelder im Fokus:

Evaluierung geeigneter Bibliotheken

Zur Umsetzung der Anbindung wurden verschiedene OpenSource-Lösungen untersucht, die eine Interoperabilität zwischen Jupyter-Notebooks und bestehenden ATLASneo-Komponenten ermöglichen können.

- **Integration interaktiver grafischer Komponenten:**
  Die Bibliothek **anywidget** wurde als besonders geeignet identifiziert, um JavaScript-basierte Komponenten als Widgets direkt in Jupyter-Notebooks einzubinden. Damit besteht die Möglichkeit, bereits in ATLASneo entwickelte webbasierte Anwendungen, wie die dynamische Visualisierung *SVGViz*, auch innerhalb von Notebooks nutzbar zu machen. Dadurch ließen sich interaktive Darstellungen und Analysen auch außerhalb der Hauptanwendung umsetzen, was insbesondere für Forschungs- und Präsentationszwecke von Vorteil sein kann.

- **Steuerung externer Programme aus Notebooks:**
  Ergänzend wurde die Bibliothek **xeus-python** als potenzielle Lösung identifiziert, um die Steuerung externer Anwendungen, wie ATLASneo, direkt aus einem Notebook heraus zu realisieren. Diese Methode ermöglicht es, reproduzierbare Analyse- und Auswertungsabläufe zu definieren und eine Steuerung von ATLASneo zu entwickeln. Anwender könnten damit automatisiert Simulationen ausführen, Daten extrahieren und die Ergebnisse unmittelbar in Plots und Diagrammen für Berichte aufbereitet.

Ausblick: Perspektiven zur Integration

Durch die Analyse wurde die Grundlage geschaffen, ATLASneo zukünftig enger mit Jupyter-Notebooks zu verzahnen. Insbesondere für automatisierte und reproduzierbare **Auswertungs-Workflows** bietet diese Anbindung ein hohes Potenzial:

- Anwender könnten Simulationen aus Notebooks heraus starten und die erzeugten Daten direkt weiterverarbeiten.
- Visualisierungen und Berichtsgrafiken ließen sich automatisiert generieren und in Dokumentationsprozesse einbinden.
- Entwicklerseitig bietet sich die Möglichkeit, neue Analysefunktionen zunächst prototypisch in Notebooks zu testen, bevor sie als ATLASneo-Gadgets implementiert werden.

3.3.4 Weiterentwicklung generischer Datenvisualisierung

Ziel dieses Arbeitspunktes war die Weiterentwicklung der generischen Datenvisualisierung in ATLASneo. Dabei sollte untersucht werden, inwieweit moderne Webtechnologien und JavaScript-Bibliotheken /JSC 25/ zur Darstellung spezialisierter Diagrammtypen eingesetzt werden können, um eine **flexible, plattformunabhängige und wartungsarme Visualisierungsumgebung** zu schaffen.

Ein besonderes Augenmerk lag auf der Nutzung standardmäßig eingesetzter Technologien wie *SVG*, *HTML5*, *CSS* und *D3.js (*/SVG2/*,* /HTML5/*,* /CSS 25/ *und* /D3JS/*)*, um Darstellungen sowohl innerhalb von ATLASneo als auch unabhängig davon, etwa im Webbrowser, nutzbar zu machen. Langfristiges Ziel war es, eine einheitliche Basis für interaktive und dynamische Visualisierungen zu schaffen, die sich für verschiedene Rechenprogramme (ATHLET, ATHLET-CD, COCOSYS) eignet und zugleich die Entwicklung, Erweiterung und Wartung neuer Visualisierungsmodule vereinfacht.

Technologische Grundlagen und Entwicklungsumgebung

Im ersten Schritt wurde die technische Machbarkeit der Nutzung webbasierter Technologien geprüft. Durch die Nutzung von SVG als Basisformat für die Visualisierung ergab sich eine Reihe von Vorteilen:

- Nutzung der vollen **Funktionsvielfalt moderner Webtechnologien** (z. B. interaktive Steuerung, CSS-basierte Stildefinition, Browserdruckfunktionen),

- Aufbau der Bild-Visualisierung auf **Angular**, einen OpenSource-Framework für Webapplikationen,
- **Unabhängigkeit von der ATLASneo-Anwendung**, wodurch sich Darstellungen in vielen Anwendungen, wie jedem gängigen Webbrowser anzeigen lassen,
- Nutzung von **Browser-Entwicklungsumgebungen** (z. B. Chrome DevTools, Firefox Developer Tools) zur schrittweisen Analyse und Fehlerbehebung von Visualisierungskomponenten.

Diese Herangehensweise ermöglicht eine deutliche Verkürzung von Entwicklungszyklen sowie eine Reduzierung des langfristigen Wartungsaufwands. In der Browser-IDE entwickelte Visualisierungsmethoden können anschließend mit geringem Aufwand in ATLASneo integriert werden.

Festlegung und Einführung des SVG-Formats

Für die Visualisierung von ATHLET-Nodalisierungsbildern wurde ein SVG-basiertes Grafikformat definiert, das sich an der Funktionalität des bisherigen APG-Formats orientiert, die darin abgelegten Daten jedoch in ein offenes und zukunftsfähiges Format überführt. Dabei wurde auf eine möglichst codeunabhängige und generische Struktur geachtet, sodass das Format perspektivisch auch für andere Rechenprogramme nutzbar ist.

Der in einem Vorgängerprojekt entwickelte Konverter, der bestehende **APG-Dateien** in das neue SVG-Format übersetzt, wurde von der Programmiersprache Nim /NIM 18/ nach Python /PYT 25/ portiert und aktualisiert. Damit können sowohl aus dem ATHLET Input Graphics Tool (AIG) generierte APG-Dateien als auch manuell erstellte APG-Übersichtsbilder, z. B. von Kraftwerkskomponenten konvertiert werden.

Die aktuelle Version des Tools **ATHLET Input Graphics (AIG-II)** /KON 17/ kann neben dem bisherigen APG-Format nun auch direkt SVG-Dateien erzeugen. Da die SVG-Ausgabe jedoch noch nicht die erforderliche Stabilität erreicht hatte und die zur Verknüpfung geometrischer Formen mit den Quelldaten verwendeten Attribute zunächst nicht vollständig mit den Anforderungen von ATLASneo übereinstimmten, war es zeitweise notwendig, die erzeugten SVG-Dateien mithilfe eines **Konverter-Skripts** anzupassen, bevor sie in der Visualisierungsanwendung *SVGViz* geladen werden konnten. Inzwischen wurden die dafür notwendigen Umwandlungsschritte direkt in die **Laderoutine**

von *SVGViz* integriert, sodass nun auch SVG-Ausgaben aus AIG-II ohne Zwischenschritte direkt importiert werden können.

Zusätzlich wurde ein Konzept erarbeitet, um **erweiterte Visualisierungsinformationen** (z. B. abgeleitete Werte, Abbildungsarten, Formatierungs-Parameter, präferierte Farbskalen) in SVG-Grafiken als eigene Namensräume zu speichern. Dies ermöglicht zukünftig die dynamische Nutzung dieser Zusatzinformationen innerhalb von ATLASneo, beispielsweise zur Animation oder zur anwendungsspezifischen Darstellung.

Entwicklung interaktiver Visualisierungskomponenten

Im Zuge der Arbeiten wurden Prototypen interaktiver Komponenten entwickelt, um die dynamische Steuerung und Analyse der dargestellten Simulationsdaten zu ermöglichen:

Farbskala-Komponente

Eine vollständig interaktive Farbskala-Komponente erlaubt es dem Benutzer, die Zuordnung von Werten zu Farben in Echtzeit zu verändern. Dabei werden alle SVG-Elemente entsprechend der gewählten Skala neu eingefärbt. Über eine Anzeige der minimalen und maximalen Werte sowie einer Werteskala wird die Farbverteilung nachvollziehbar dargestellt. Zur Umsetzung wurde ein Farbdienst (Color Service) implementiert, der die Erstellung und Verwaltung mehrerer Farbskalen ermöglicht. Die Architektur folgt dem Observable-Design-Pattern (auf Basis von RxJs), wodurch Änderungen an Steuerkomponenten unmittelbar auf mehrere SVG-Elemente übertragen werden können. Diese Komponente wurde im Verlauf des Projekts erweitert:

- Integration von **lokalen und globalen**, vom Nutzer wählbaren **Farbskalen**,
- Einführung von **„Griff“-Elementen** zur interaktiven Steuerung der Gewichtung in Farbverläufen,
- Einbindung eines **ColorPickers** zur direkten Farbwahl über ein Pop-up-Menü.

Tooltip- und Plot-Komponente

Exemplarisch wurde eine benutzerdefinierte Tooltip-Komponente implementiert, die beim Anklicken eines SVG-Elements kontextspezifische Informationen anzeigt.

Diese Komponente

- erkennt angeklickte SVG-Elemente,
- ruft über die Web-Bridge spezifische Daten ab,
- und stellt sie dem Benutzer in einem kleinen Diagramm dar.

Zur grafischen Darstellung wurde die Bibliothek Plotly eingebunden, mit der die Tooltip-Komponente einfache X/Y-Zeitreihenplots direkt im Tooltip erzeugen kann. Perspektivisch muss weiter an der Darstellung kontextspezifischer Informationen gearbeitet werden, da die Komponenten eigene Zeitreihenplots erzeugt, welche nicht in Verbindung mit den Data-Models der umgebenden ATLASneo-Applikation stehen und dadurch schwer mit den Datenquellen synchron gehalten werden können.

Umsetzung in ATLASneo und SVGViz

Die erarbeiteten Konzepte und Komponenten wurden in die ATLASneo-Anwendung integriert. Dabei stand insbesondere das Gadget *Angulator* mit der zugehörigen Webanwendung *SVGViz* im Fokus, die die dynamische Darstellung der simulierten Daten übernimmt. Die wesentlichen Fortschritte waren:

- Erweiterung von ATLASneo um die automatische Initialisierung und **Start des Serverdienstes** und der Webanwendung *SVGViz* beim Programmstart,
- Anpassung der **Datenhaltung** im *Angulator* zur effizienteren Kommunikation zwischen Python-Backend und Webfrontend,
- Verminderung der **Datenübertragung** und Performance-Verbesserung in der **Darstellung** durch gezielte Updates,
- Flexibilisierung und Ausbau der ***SVGViz*-Laderoutine**, um in AIG-II erzeugte SVG-Ausgaben direkt einlesen zu können.

Die neuen Funktionen wurden in Testversionen von ATLASneo integriert und internen Nutzern für erste Anwendungen zur Verfügung gestellt (siehe Abb. 3.18). Diese konnten erstmals eigenständig in AIG2 erstellte SVG-Dateien importieren, um damit Transienten in Simulationen durch Farbverläufe dynamisch zu visualisieren.

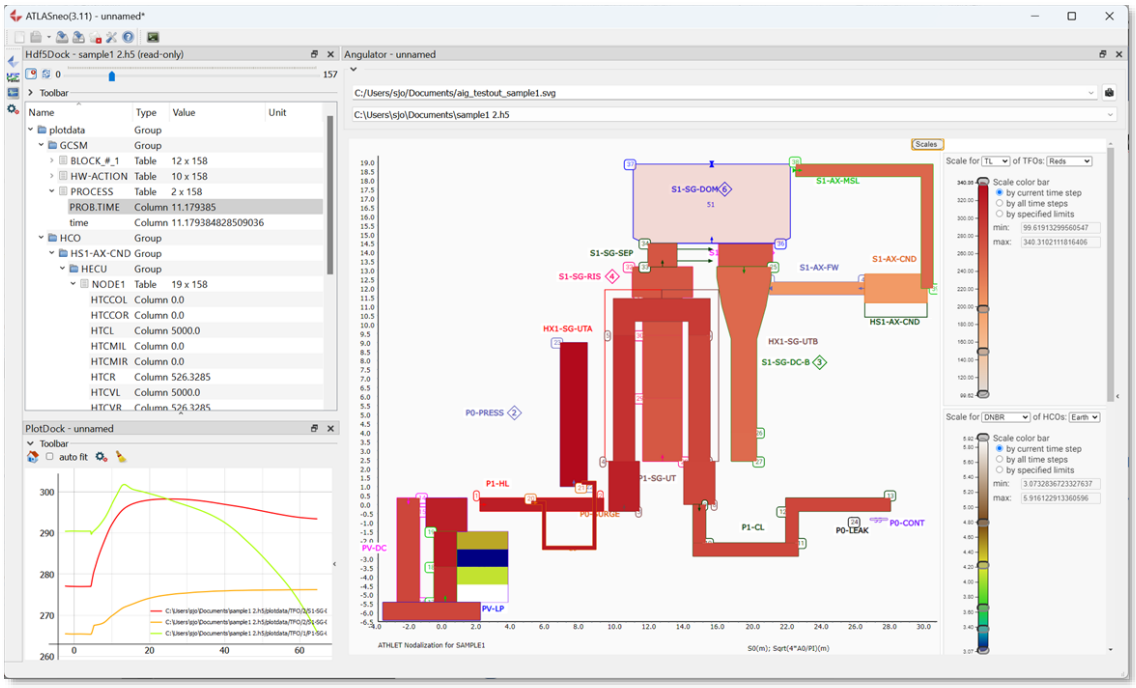

Abb. 3.18 Die auf Angular basierende Bilddynamisierung in Anwendung. Die interaktiven Bedienelementen, wie Farbskalen und ColorPicker erlauben die Feineinstellung der Wertabbildung auf die Füllfarben der eingeladenen SVG-Geometrie

Stabilisierung und Integration in den Hauptzweig

In der abschließenden Entwicklungsphase des Projektverlaufs wurde die Integration der Visualisierungskomponenten in den Hauptentwicklungszweig von ATLASneo vorbereitet, um die Wartung des *Angulator*-Gadgets und der Webapplikation *SVGViz* zu erleichtern und deren Funktionalität zukünftig auch externen Anwendern über reguläre Releases zugänglich machen zu können. Hierbei wurden

- **Layout und Ansichtssteuerung** in *SVGViz* umfassend überarbeitet,
- die **Kategorie**-spezifische **Variablenauswahl**, die **Kalibrierungsart** der Farbskala auf die Wertebereiche der Quelldaten und die benutzerdefinierte **Farbskalenverwaltung** verbessert,
- erste Vorbereitungen erarbeitet, um **Nutzereinstellungen** speichern und wiederherstellen zu können,
- und die **Screenshot**-Funktion erweitert, um exakt den aktuell sichtbaren Bildausschnitt zu exportieren.

Darüber hinaus wurde die Node-Auswahl per Mausklick überarbeitet. Die zugehörigen Zeitreihen können nun sowohl einzeln als auch gruppiert in einem verknüpften Plotter dargestellt werden. Die Übertragung der Metadaten wurde überarbeitet, sodass die Verbindung zur Datenquelle im Plotter bestehen und somit auch während laufender Online-Simulationen eine **konsistente Aktualisierung** der so zusammengestellten Diagramme gewährleistet bleibt.

3.3.5 Verbesserung der Anwendbarkeit bisheriger Entwicklungen

Ziel dieses Arbeitspunkts war die Erhöhung der Stabilität, Benutzerfreundlichkeit und Flexibilität von ATLASneo, insbesondere im Bereich der Ergebnisanalyse und Visualisierung. Aufbauend auf Rückmeldungen der internen Anwender wurden Maßnahmen zur Fehlerbehebung, Optimierung der Bedienung und Erweiterung der Funktionalität umgesetzt.

Allgemeine Verbesserungen und Fehlerbehebungen

Sowohl in der Rahmenanwendung als auch im Anwendungsframework und in mehreren Gadgets wurden zahlreiche Korrekturen und Erweiterungen vorgenommen, um die tägliche Arbeit mit ATLASneo zu erleichtern. Dazu zählen unter anderem:

- Eine **Splash-Screen**-Anzeige visualisiert den Ladefortschritt beim Start.
- **Tastenkürzel** (*CTRL + 1–9*) erlauben schnellen Gadget- und Konsolenzugriff.
- Das **„Speichern unter“** von Sitzungen ist jetzt über die Sessionbar möglich.
- Die verbesserte **Sitzungsverwaltung** stellt jetzt in der Session gespeicherte Fenstergrößen und Gadget-Anordnungen beim Laden zuverlässiger wieder her.
- Verschiedene Fehlerbehebungen im **Gadget-Management** erlauben jetzt das korrekte Schließen abgedockter Fenster.
- **Statusmeldungen** (StatusTips), wie Suchergebnisse oder Pfade ausgewählter TreePanel-Elemente werden jetzt zentral in der Haupt-Statusleiste angezeigt.
- **Dateiauswahlfelder** erlauben eine Filterung nach vordefinierten Dateimustern.
- **Filtereingabefelder** in TreePanels bieten nun eine hierarchische Autovervollständigung (vgl. Abb. 3.19, a)**.**

- Das **Delegate-System** wurde erweitert und unterstützt jetzt vollständig Qt-Enum-Typen, was ihre Darstellung und interaktive Konfigurationsanpassungen, wie im TreeView des ConfigPanels, ermöglicht.

Außerdem wurde die **Python-Konsole** um einen neuer **Popup-Completer** erweitert, der die bisherige Attributliste ersetzt. Der Completer filtert Vorschläge dynamisch anhand bereits eingegebener Zeichen und erleichtert so die Verwendung von Methoden und Objekten erheblich.

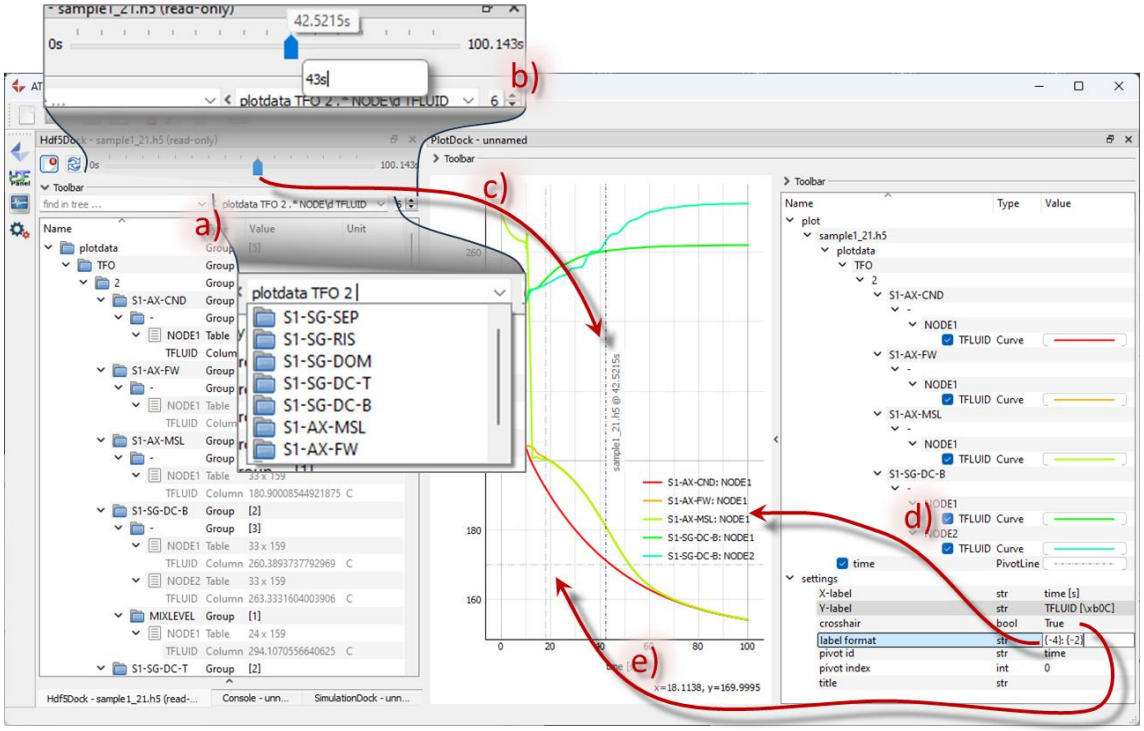

Abb. 3.19 Diverse Verbesserungen in Rahmenanwendung und Gadgets von ATLAS-neo

Erweiterungen in HDF5Dock und PlotPanel

Ein wesentlicher Fokus der vorgenommenen Verbesserungen lag auf der Analyse von Simulationsdaten und damit auf der Weiterentwicklung des **PlotPanels** und des **HDF5-Datenhandlings**.

- Das **HDF5Dock** wurde erweitert und erlaubt jetzt die Anzeige des Wertebereichs der Pivot-Spalte (z. B. Zeit) der aktiv ausgewählten Datentabelle. Dazu ermöglicht der überarbeitete **Zeitschritt-Slider** nun die Auswahl des Zeitschritts, die direkt über

Eingabe von Index- oder Pivot-Werten, wie z. B. der Zeit, erfolgen kann (siehe Abb. 3.19, b).

- Eine **Pivot-Line** visualisiert im Plot für jede Datenquelle (z. B. HDF5-Datei oder Simulation) den aktuellen Zeitschritt bzw. Datenindex als vertikale Linie. Sie ist dabei vollständig mit der jeweiligen Datenquelle synchronisiert und – wie im HDF5Dock – direkt mit dem Zeitschritt-Slider gekoppelt (Abb. 3.19, c).
- Ein neuer **Caching-Mechanismus** reduziert die HDF5-Datenzugriffe und sorgt für deutliche Performance-Verbesserung bei der Anzeige von HDF5-Daten.
- Das PlotPanel erkennt und konvertiert **verschiedene Datentypen** jetzt automatisch (z. B. HDF5-Spalten, 2D-Arrays, Wertreihen) und meldet Fehler beim Droppen inkompatibler Datentypen.
- **Kurven dynamisch aktualisierter Daten** werden jetzt im Fehlerfall, z. B. nicht verfügbare Daten, automatisch deaktiviert und vermeiden so Fehlermeldungen.
- Der neue **AutoFit-Modus** sorgt für eine automatische Anpassung der Achsen während Online-Simulationen.
- Über eine neue Option zur **Legendenbenennung** können Kurvennamen nun per Format-String generiert und dabei über verschiedene Platzhalter flexibel konfiguriert werden (Abb. 3.19, d).
- Ein **Crosshair-Cursor** mit Anzeige der Koordinaten an der aktuellen Maus-Position erleichtert die präzise Auswertung (Abb. 3.19, e).

Darüber hinaus wurde für spezielle Anwendungsfälle ein Modul zur Erstellung von **Distributions-Plots** implementiert, das Datenreihen über benachbarte Kontrollvolumen hinweg erzeugt und dynamisch mit der Zeitschrittsteuerung synchronisiert. Die benutzerfreundliche Ansteuerung dieser Funktion sollte im weiteren Entwicklungsverlauf über geeignete Bedienelemente des PlotPanels zur Verfügung gestellt werden.

Dokumentation und Anwender-Tutorial

Zur Unterstützung einer effektiven Nutzung wurde im GitLab-Wiki ein konsolenbasiertes Beispiel zur Erstellung von *Distribution-Plots* als **Tutorial** dokumentiert. Auch wenn diese Funktionalität perspektivisch über grafische Bedienelemente im PlotPanel nutzbar sein soll, dient das Tutorial bereits jetzt als praxisnahe Anleitung und erleichtert insbesondere

neuen Anwendern den Einstieg in fortgeschrittene Funktionen und typische Anwendungsfälle der ATLASneo-Konsole.

3.3.6 Allgemeine Wartungsarbeiten

Für die im Projekt bereits bestehenden Funktionalitäten von ATLASneo war zu Projektbeginn klar, dass deren langfristige Nutzbarkeit nur sichergestellt werden kann, wenn die Software kontinuierlich an neue Laufzeit-umgebungen, Bibliotheken und Betriebssysteme angepasst wird. Ziel dieses Arbeitspunktes war daher

- die regelmäßige Durchführung von Wartungsarbeiten,
- die Aktualisierung von Abhängigkeiten,
- die Sicherstellung der Funktionsfähigkeit bei neuen Python- und Qt-Versionen,
- sowie der Ausbau der automatisierten Tests, um sowohl bestehende als auch neue Funktionen dauerhaft funktionssicher zu halten.
- Dies betraf insbesondere die Bereiche Ergebnisauswertung, Visualisierung und die zugehörigen UI-Komponenten.

Die im Laufe des Projektes umgesetzten Maßnahmen können in die folgenden vier Themenfelder gruppiert werden.

Anpassungen an sich weiterentwickelnde Bibliotheken

Es wurden in mehreren Iterationen Anpassungen vorgenommen, um neue Versionen von Python, Qt, PySide/PySide2, PyQtGraph und weiteren Bibliotheken zu berücksichtigen. Dazu gehörten:

- Vereinheitlichung der **locale-Einstellungen** für eine konsistente Darstellung numerischer Werte,
- **Erweiterung des Delegate-Systems** für zusätzliche Datentypen (u. a. numpy-Arrays),
- Erhalt der **Rückwärtskompatibilität** für ältere Python-Versionen (nur bis zur geplanten Einstellung < 3.7).

Stabilisierung und verbesserte Nutzbarkeit

Parallel dazu wurden kleinere Wartungsarbeiten und funktionale Verbesserungen vorgenommen. Diese beinhalteten u. a.

- Verbesserungen in Start- und Testskripten,
- Weiterentwicklungen des HDF5Docks (Zeitschrittinformation, robustere Handhabung verschiedener Layouts von Datentabellen und Pivot-Spalten),
- Verbesserungen an der Sitzungsverwaltung und der Wiederherstellung von Layouts der Benutzeroberfläche.

Aufbau automatisierter UI-Tests mittels AutoIt

Um die Funktionsfähigkeit der grafischen Oberfläche automatisiert überprüfen zu können, wurde ein AutoIt-basiertes /AUI 22/ Testframework aufgebaut:

- AutoIt-Tests wurden so integriert, dass sie auf GitLab-CI auch ohne grafische Benutzeroberfläche ausgeführt werden können.
- Fehlerquellen in der CI-Umgebung (z. B. überlagernde Fenster bei Maus-Bedienung) wurden identifiziert und behoben.
- Grafische Anwendungen können automatisiert gestartet, über Maus und Tastatur bedient und geschlossen werden.
- Screenshots werden in der CI gesammelt und zusätzlich zu einem kurzen Video zusammengeführt, um Fehler schneller nachvollziehen zu können.

Ausbau der Testabdeckung in der CI

Der Umfang der automatisierten Tests wurde im Projektverlauf stetig erweitert:

- Für Anwendungsstart und grundlegende Bedienung von ATLASneo wurden Skripts und grundlege Tests implementiert.
- Fortgeschrittene Funktionalitätstests wurden zu den Themen Sessionmanagement, Konfigurationsdialogen, Gadgets (PlotDock / Console) und Protokoll-basierten Online-Simulationen erstellt.
- Erste Methoden zur Prüfung von sichtbaren UI-Elementen wurden implementiert.

- Erste Möglichkeiten zur Validierung der grafischen Inhalte (Plots) wurde implementiert, bedarf aber weiterer Entwicklung über das Projekt hinaus.

3.4 AP 4: Querschnittsaufgaben

Neben den geplanten Arbeiten und den konkreten Entwicklungsaufgaben im Projekt müssen weitere Maßnahmen unternommen werden, um hohe Qualität und Nachhaltigkeit der entwickelten Funktionalität zu erzielen. Diese müssen kontinuierlich durchgeführt werden, auch um die Entwicklungen der einzelnen Arbeitspunkten aufeinander abzustimmen. Diesem zusätzlichen Aufwand wird durch die folgenden Arbeitsgebiete Rechnung getragen.

3.4.1 Software-Infrastruktur

Um die langfristige Nutzbarkeit und Weiterentwicklung von ATLASneo sowie den zugehörigen Komponenten sicherzustellen und deren Kompatibilität zu den jeweiligen AC²-Codes gewährleisten zu können, sollte im Projekt begleitend eine robuste und nachhaltige Software-Infrastruktur aufgebaut werden. Ziel des Arbeitspunkts war daher,

- eine verlässliche CI-/CD-Pipeline aufzubauen,
- automatisierte Tests (inkl. GUI-Tests) kontinuierlich auszubauen,
- die Bereitstellung von Anwendungspaketen zu automatisieren und
- Entwickler- und Anwenderdokumentation fortlaufend aus dem jeweiligen Systemzustand generierbar zu machen (Code → API-Dokumente, Wiki → Anwenderhandbücher).

Damit sollte die Software nicht nur weiterentwickelt, sondern auch reproduzierbar, testbar und langfristig wartbar werden. Dies ist die Voraussetzung, um den in der Software-Entwicklung gängigen Leitsatz *„release early, release often“* praktisch umzusetzen. Erst diese regelmäßige, verlässliche Bereitstellung von aktualisierten Versionen ermöglicht den kontinuierlichen Austausch mit Anwendern und anderen Projekten. Die im Projekt erreichten Ergebnisse lassen sich in drei thematische Schwerpunkte gliedern.

Aufbau und Erweiterung der CI- / CD-Infrastruktur

Um die Kompatibilität und Pflege zu verbessern, wurden die für ATLASneo entwickelten CI/CD-Mechanismen in Anlehnung an das für AC²-Codes angewendete Vorgehen aufgebaut. Damit steht eine weitgehend automatisierte Build- und Release-Strecke bereit. Die wichtigsten Ergebnisse waren dabei:

- Die CI-Automatisierung kann die Projekte ADM und ATLASneo vollständig bauen, starten, testen und paketieren.
- Notwendige Python-Environments werden automatisiert erzeugt und halb-automatisch, d. h. durch explizite Angabe, aktualisiert.
- Release-Pakete (inkl. Lizenzübersichten) werden automatisch erzeugt und in die Paketierung übernommen.
- Standardisierte CI-Jobs wurden in wiederverwendbare Abschnitte ausgelagert, welche über den Include-Mechanismus eingebunden werden können. Dadurch werden Konfigurationen kleiner und wartbarer und können somit unter Vermeidung typischer Fehler deutlich schneller aufgebaut werden.

Automatisierte GUI-Tests unter Linux

Ein wesentlicher technischer Meilenstein war die vollständige Ausführung AutoIt-basierter GUI-Tests (nur Windows-Version verfügbar) innerhalb von Linux-Jobs der GitLab-CI. Damit können GUI-Funktionalitäten, welche normalerweise einen Desktop erfordern, nun reproduzierbar und automatisiert in Container-Umgebung ohne graphische Oberfläche wie den CI-Runnern getestet werden.

- Der Start der Windows-Version von ATLASneo unter Wine wurde erfolgreich konfiguriert.
- Virtuelle Displays (*Xvfb*) + *VNC*-Zugriff zum Live-Monitoring der Testläufe wurden aufgebaut.
- Screenshots aller Testschritte werden automatisch als CI-Artifacts bereitgestellt.

Automatisierte Dokumentationsgenerierung aus GitLab-Wikis

Für die Anwenderdokumentation wurde ein Werkzeug (wiki2pdf) entwickelt und sukzessive weitergeführt. Dieses Werkzeug wurde für die automatisierte Verwendung in der CI entworfen und ermöglicht, Aktualisierungen im Wiki ohne manuelle Schritte als Anwenderhandbuch zu formatieren. Es bietet dabei folgende Funktionalität:

- automatischer Export von GitLab-Wiki-Seiten nach PDF (inkl. Bilder, Diagramme, Codeblocks),
- konfigurierbare Auswahl der zu exportierenden Kapitel (Wiki-Sidebar wird interpretiert),
- einheitliches Layout via CSS, inkl. Seitenrändern, Seitenumbrüchen und verlinktem PDF-Inhaltsverzeichnis.

Dieses Werkzeug erlaubt es, das in der Paketierung von ATLASneo und ADM eingebundene Anwenderhandbuch mit der Dokumentation im Wiki synchron zu halten und Aktualisierungen manuellen Aufwand zu übernehmen.

3.4.2 Projektmanagement

Gegenstand dieses Arbeitspaketes waren Aufgaben des Projektmanagements und des Projektcontrollings. Durch das Projektmanagement und das Projektcontrolling wurde sichergestellt, dass alle Arbeiten in dem Vorhaben konform zu den GRS Projekt- und Qualitätsmanagement-Prozessen und -Regeln und im Einklang mit den Vorgaben des Auftraggebers koordiniert und sach- und termingerecht abgewickelt sowie EDV-technisch erfasst, vorgehalten und bedarfsgerecht aufbereitet wurden.

Das Projektmanagement umfasste u. a. alle administrativen Aufgaben, die sich im Zusammenhang mit dem Kontakt zum Auftraggeber und zur Projektbegleitung stellten. Aufgaben des Projektmanagements waren insbesondere die kontinuierliche Koordination, Durchführung und Überwachung der administrativ vertraglich zugesicherten Aufgaben, dazu gehörten unter anderem die halbjährliche Berichterstattung, die Beantragung außereuropäischer Dienstreisen, die Budgetkontrolle, das Stellen von Änderungsanträgen sowie die Veranlassung von Quartalsabschlägen.

Das Projektmanagement unterstützte den fachlichen Projektleiter bei der Mittel- und Personalplanung, der Zuweisung von Arbeiten mit Blick auf das vorhandene Budget sowie

der Genehmigung der Mittel nach Überprüfung auf vertragliche Übereinstimmung. Durch das Projektmanagement erfolgten schließlich die formale Qualitätssicherung aller Arbeitsergebnisse, die Überprüfung der Arbeitsergebnisse hinsichtlich der Einhaltung der GRS-Qualitätsstandards sowie die finale Freigabe aller nach extern gehenden oder zu veröffentlichenden Ergebnissen.

Im Projektcontrolling erfolgte die EDV-technische Eingabe, Pflege und kontinuierliche Aktualisierung der vertragsrelevanten Daten sowie die EDV-technische Erfassung und Vorhaltung der projektrelevanten Unterlagen (Angebote, Verträge und Änderungsdienste). Durch das Projektcontrolling wurde sichergestellt, dass die GRS dem Auftraggeber jederzeit alle projektrelevanten Informationen auf elektronischem Wege bedarfsgerecht zur Verfügung stellen konnte.

Im fachlichen Teil des Projektmanagements wurde bereits früh im Projekt eine klar strukturierte Arbeits- und Kommunikationsroutine festgelegt, die den kontinuierlichen Austausch im Team und Projektbegleitung sicherstellte. Wichtige organisatorische Entscheidungen, insbesondere im Kontext der Release-Vorbereitung, wurden systematisch vorbereitet, Aufgaben wurden verteilt und deren Fortschritt wöchentlich verfolgt.

Ein zentraler Mechanismus war dabei die Nutzung von GitLab-Issues als verbindliches Planungs- und Tracking-Werkzeug. Offene Punkte aus fachlichen Arbeitsabschnitten wurden dort erfasst, priorisiert und in den wöchentlichen Team-Meetings abgestimmt. Dieses Vorgehen hat sich besonders vor den Release-Meilensteinen sehr bewährt, da so sowohl fachliche als auch organisatorische Restarbeiten deutlich leichter synchronisiert werden konnten.

4 Zusammenfassung und Ausblick

Ziel des vorliegenden Projekts war es, die Werkzeuge der GRAMOVIS-Plattform technisch und methodisch so weiterzuentwickeln, dass die umfangreichen Workflows für die Anwendung von AC²-Rechencodes – von der Eingabeerstellung über die Ausführung und Überwachung von Simulationsprozessen bis hin zur Analyse und Visualisierung der Ergebnisdaten – möglichst benutzerfreundlich, reproduzierbar und konsistent durchführbar werden.

Im Projekt wurden hierfür grundlegende Weiterentwicklungen in allen drei wesentlichen Workflow-Schritten erzielt und mit Maßnahmen zur nachhaltigen Wartbarkeit der bestehenden Anwendungswerkzeuge unterlegt. Diese Arbeiten haben gezeigt, dass die Entwicklungen der GRAMOVIS-Projekte – insbesondere durch konsequente Modularisierung und durch die im Projekt eingeführten Mechanismen zur Qualitätssicherung, wie die CI-gestützte Automatisierung von Funktionstests, Dokumentation und Paketierung – jetzt technologisch und konzeptionell deutlich besser auf zukünftige Erweiterungen und Weiterentwicklungen vorbereitet sind.

Modellierung und Input-Erstellung

Die im Projekt erzielten Ergebnisse im Bereich der Eingabeerstellung zielen darauf ab, die Erstellung, Modifikation und Validierung von Eingabedaten für AC²-Rechencodes stärker zu automatisieren und methodisch zu vereinheitlichen.

Durch die Arbeiten am **AC²-Design Modeller (ADM)** konnten einige Fehler behoben werden. Zudem lieferten sie wichtige Erkenntnisse zur Zukunftsfähigkeit der bisherigen technischen Basis. Es wurde deutlich, dass die derzeitige Bindung des ADM an SimInTech wesentliche Weiterentwicklungen limitieren. Allerdings ist aufgrund der Größe und Komplexität der Anwendung eine kurzfristige Portierung auf ein anderes Basis-System nicht möglich. Künftige Arbeiten sollten daher die Nutzung neuer Systeme zur Eingabeerstellung prüfen – insbesondere vor dem Hintergrund etablierter OpenSource-Plattformen. Nichtsdestotrotz sind durch die im Projekt durchgeführten Aktualisierungen und Verbesserungen die Lauffähigkeit von ADM sowie dessen Wartbarkeit gewährleistet. Der für die Weiterentwicklung zentrale Mechanismus zum Import und Export von Datensätzen kann perspektivisch durch stabilere Methoden, wie dem im Projekt entwickelten ImputProcessor realisiert werden.

Zur Bereitstellung eines **Input-Parametrisierungs-Systems** wurde mit dem **InputExpander**, ein generisches Werkzeug zur parametergesteuerten Erzeugung und Variation von Eingabedatensätzen entwickelt. Dieser ermöglicht es Anwendern, textbasierte Inputs automatisiert, reproduzierbar und anwendungsspezifisch anzupassen. Der Input-Expander basiert auf einem generischen Ansatz, sodass er perspektivisch auch in anderen Aufgabenstellungen mit textbasierten Eingabesystemen eingesetzt werden kann.

Um eine **vereinheitlichte Verarbeitung von Eingabeformaten** zu ermöglichen, wurde der **InputProcessor** implementiert. Dieser stellt eine einheitliche, Grammatik-basierte Grundlage bereit, um unterschiedliche Eingabeformate (ATHLET, ATHLET-CD, COCOSYS, etc.) strukturiert analysieren und verarbeiten zu können. Damit wird eine Basis für die automatische Überprüfung, Formatumwandlung und weitergehende Optimierung von Eingabedatensätzen gelegt, die in Folgevorhaben auch für die Validierung und automatische Konsistenzprüfungen von Eingabedatensätzen nutzbar ist.

Simulations-Durchführung

Die Arbeiten in diesem Themenfeld zielten darauf ab, eine einheitliche, flexible Infrastruktur für die Ausführung und Überwachung von Simulationsprozessen verschiedener AC²-Rechencodes zu schaffen. Aufbauend auf den Ergebnissen des Vorgängerprojekts wurde die Auslagerung von Simulationen in separate Prozesse weiterentwickelt, einschließlich der **Steuerung über Python-Kernels**, parallele **Ausführung unter MPI** und Unterstützung unterschiedlicher Systemumgebungen (lokal, Netzwerk, Cluster).

Als alternativer Ansatz wurde der **JobBroker**, ein servicebasierter, modularer Serverdienst mit REST-API entwickelt, der die Planung, Ausführung, Überwachung und Verwaltung von Simulationsjobs ermöglicht und sowohl lokal als auch im Netzwerk oder auf dem Cluster betrieben werden kann. Der JobBroker bietet Funktionen zum Starten und Beenden von Jobs, Zugriff auf Statusinformationen und Konsolenausgaben sowie persistente Speicherung von Job-Metadaten. Die Architektur erlaubt die flexible Integration unterschiedlicher Executor-Typen, darunter lokale Shell-Aufrufe und PBS-Cluster-Integration, und bildet so die Grundlage für eine skalierbare, portable und zukunftsfähige Ausführung von Simulationen, sowohl im Batch-Betrieb als auch in interaktiven Szenarien. Zur einfachen Kommunikation mit dem Serverdienst wurden ein package für **Python-Clients** und ein **Command-Line-Interface** entwickelt.

Darüber hinaus wurde die Architektur der **Simulations-Controller**, welche in den GRAMOVIS-Werkzeugen zum Start und Steuern von Simulationen eingesetzt werden, weiterentwickelt. Das Ziel war die Steuerung, Überwachung und interaktive Kontrolle von Simulationen zu verbessern. Hierzu wurde die objektorientierte Mixin-Struktur der Controller-Klassen erweitert, um unter anderem „conditional breakpoints“ zu ermöglichen. Diese erlauben Simulationen bei bestimmten Bedingungen anzuhalten, um eine **interaktive Inspektion und Manipulation** von Variablen durchführen zu können. Dazu wurden zwei Ansätze – als Command-Shell oder als pdb-basierter Debugger – untersucht, welche beide großes Potenzial für die geplante interaktive Kontrolle bieten.

Parallel dazu wurde die **Verarbeitung geskripteter Protokolle** aus dem Batchbetrieb in die **Online-Simulation** integriert, die Controller für asynchrone Ereignisse, Fehlerbehandlung und Statusmeldungen erweitert und die Benutzeroberfläche angepasst, um die Eingabe von Controller-Parametern und die Anzeige von Protokollausgaben zu ermöglichen. Dadurch wurde der bisher getrennte Workflow zwischen stand-alone und interaktiver Simulation vereinheitlicht. Nicht mehr umgesetzte, aber geplante Arbeiten betreffen ein rein textbasiertes Kontroll-Interface, das eine vollständige terminalbasierte Bedienung erlauben und die Flexibilität und Unabhängigkeit von der GUI weiter erhöhen soll. Die Weiterentwicklung dieses Ansatzes sollte in einem Folgeprojekt angegangen werden.

Ergebnis-Analyse und Visualisierung

Im Arbeitspaket „Ergebnis-Analyse und Visualisierung“ wurden **ATLASneo,** dessen Funktionsmodule und der Datenkonverter **x2hdf5** gezielt für die produktive Nutzung weiterentwickelt. Zentrales Ziel war, die Analyse und Darstellung von Simulationsdaten durch ein konsistentes Datenformat zu vereinheitlichen, moderne Anwendungstechnologien einzubinden und die Nutzbarkeit durch Anwender zu erhöhen. Dazu wurde in enger Abstimmung mit Entwicklern und internen Nutzern gearbeitet, um sowohl die fachlichen Kernanforderungen als auch die technischen Grundlagen für reproduzierbare, flexible Auswertungen zu schaffen.

Ein Schwerpunkt lag auf der Bereitstellung von ATLASneo als installierbares **Anwendungspaket**. Dazu wurden Funktionsmodule und genutzte Pakete vereinheitlicht, auf moderne Python-Versionen migriert, die Laufzeitumgebung konsolidiert und automatisiert paketiert. Mit **mehreren Releases** (1.0.0 bis 1.2.0) wurde eine stabile Basis für

interne und externe Anwender geschaffen; inklusive automatisierter Lizenzdokumentation und anwendungsorientierter Benutzerdokumentation.

Zur Erweiterung der Datenbasis wurde der zentrale Konverter **x2hdf5** um zusätzliche Formate wie **NetCDF** ergänzt und überarbeitet, um mehr Rechencodes und deren Ausgabedaten in ATLASneo verarbeiten zu können. Parallel wurde eine technische Analyse zur **Anbindung von Jupyter-Notebooks** durchgeführt. Diese soll perspektivisch eine engere Verzahnung von prototypischer Entwicklung reproduzierbarer Auswertungsworkflows mit ATLASneo-Funktionalität ermöglichen.

In der generischen Datenvisualisierung wurde eine Umstellung des zugrundeliegenden Technologie-Stacks vollzogen: Die als Webapplikation realisierte SVG-basierte Visualisierung ***SVGViz*** wurde auf Basis von Standard-Bibliotheken, wie ***D3.js*** und des ***Angular***-Frameworks neu aufgebaut. Interaktive Elemente wie dynamische Farbskalen, Tooltips und Plot-Integration wurden entwickelt, ins System integriert und für reale Nutzungsfälle verfügbar gemacht. Das als ATLASneo-Gadget aufgebaute Funktionsmodul ***Angulator*** zum Laden und Darstellen der Webapplikation wurde überarbeitet und entsprechend angepasst. Der Code wurde zunehmend so strukturiert, dass Visualisierungen perspektivisch nicht nur in ATLASneo selbst, sondern auch in Webumgebungen genutzt werden können.

Abschließend wurden zahlreiche Verbesserungen der Anwendbarkeit vorgenommen – von Performance-Optimierungen im **HDF5Panel**, über verbesserte Bedienung und Konfiguration des **PlotPanel** bis hin zum Aufbau automatisierter **UI-Tests in der CI**. Damit wurde die Stabilität erhöht und die Grundlage für nachhaltige Weiterentwicklungen der Analyse- und Visualisierungsfunktionen gelegt.

Querschnittsaufgaben

Das Arbeitspaket Querschnittsaufgaben umfasste alle Arbeiten zum Aufbau einer nachhaltigen Software-Infrastruktur für ATLASneo und die zugehörigen Komponenten. Ziel war, die Entwicklung, Bereitstellung und Qualitätssicherung dauerhaft auf ein professionelles, reproduzierbares Fundament zu stellen. Dazu wurden CI/CD-Pipelines aufgebaut, standardisiert und so erweitert, dass Builds, Tests und Paketierungen weitgehend automatisiert ablaufen können. Python-Laufzeitumgebungen werden dabei automatisch erzeugt, Lizenzübersichten gemäß den verwendeten Paketen zusammengestellt und bei Bedarf auch Release-Installer automatisiert erstellt. Durch die Auslagerung von

Standard-Jobs in wiederverwendbare CI-Abschnitte konnten Konfigurationen deutlich vereinfacht und die Wartung beschleunigt werden.

Ein entscheidender Meilenstein war die im Projekt geschaffene Möglichkeit, **GUI-Tests in CI-Runnern** automatisiert auszuführen. Hierfür werden von ATLASneo die Windows-Version mittels Wine unter Linux gestartet, ein virtuelles Display eingerichtet und über AutoIt gesteuerte Testprozeduren durchgeführt. Regelmäßig erstellte Screenshot erlauben eine Überprüfung des Ablaufs. Dadurch können GUI-Funktionalitäten – die bisher zwingend eine graphische Oberfläche erforderten – nun reproduzierbar und automatisiert getestet werden.

Parallel wurde ein Werkzeug zur automatisierten **Erzeugung der Anwenderdokumentation** aus GitLab-Wikis entwickelt (**wiki2pdf**). Damit lassen sich Wiki-Inhalte automatisiert zu PDFs verarbeiten, inklusive Bilder, Code-Beispiele und strukturiertem Inhaltsverzeichnis. Dieses System erlaubt, die Dokumentation ohne manuellen Aufwand synchron zum Software-Stand bereitzustellen.

Begleitend dazu stellte das **Projektmanagement** sicher, dass die Arbeiten des Vorhabens regelkonform, budgetorientiert und termingerecht durchgeführt wurden. Es koordinierte alle Abstimmungen zum Auftraggeber, bereitete Releases organisatorisch vor und war für die formale Qualitätssicherung der Arbeitsergebnisse verantwortlich. Mit einer klaren Kommunikationsroutine und der konsequenten Nutzung von GitLab-Issues als verbindliches Planungsinstrument wurden fachliche und organisatorische Aufgaben laufend priorisiert und bis zu den Release-Meilensteinen systematisch nachverfolgt. Dadurch konnte eine robuste Projektsteuerung etabliert werden, die maßgeblich zum erfolgreichen Abschluss der Arbeiten beitrug.

4.1 Ausblick

Die Entwicklungen des Projekts haben gezeigt, dass alle drei Stufen des Workflows – Input, Simulation, Analyse – nun technologisch so aufgebaut sind, dass Weiterentwicklungen nicht mehr isoliert, sondern auf einer gemeinsamen Architektur vorgenommen werden können. Die im Projekt geschaffenen Bausteine bilden eine belastbare Grundlage für zukünftige Erweiterungen, die im Rahmen eines Folgevorhabens systematisch ausgestaltet werden können.

In einem möglichen Anschlussvorhaben sollten die Arbeiten in den folgenden vier Themenfeldern weitergeführt und vertieft werden.

Modellierung und Input-Erstellung

Ziel zukünftiger Arbeiten ist die weitergehende Automatisierung und Modularisierung der Eingabeerstellung. Der **InputExpander** soll um fortgeschrittene Konzepte ergänzt werden, welche die Generierung komplexer Anlagenmodelle durch modulare, wiederverwendbare Bauteile ermöglichen. Dazu sind erweiterte Modularisierungsmechanismen (z. B. Vererbung), zusätzliche Befehle und Kontrollstrukturen sowie die Einbindung externer Python-Routinen vorgesehen. Aufbauend darauf könnten beispielhafte parametrisierte Bauteilbibliotheken bereitgestellt werden. Parallel dazu soll der **InputProcessor** zu einem zentralen Werkzeug für die Grammatik-basierte Analyse von AC²-Eingaben ausgebaut werden. Hierbei sind die Unterstützung weiterer Codes (z. B. COCOSYS, FENNECS) sowie die Speicherung und Wiederverwendung abstrakter Syntaxbäume (AST) in standardisierten Austauschformaten (z. B. JSON, XML) geplant, um externe Applikationen anzubinden. Darüber hinaus ist die **Einbindung von InputExpander und InputProcessor in ADM** vorgesehen, um eine einheitliche Verarbeitung, Bearbeitung und Generierung von Eingabedatensätzen zu ermöglichen.

Simulationsdurchführung

Für die Simulationsausführung soll die Integration und Erweiterung des servicebasierten **JobBrokers** fortgesetzt werden. Dieser soll die verteilte Durchführung und zeitschrittgenaue Steuerung von Simulationen auf Clustern ermöglichen, einschließlich SSH-Tunneling und erweiterter Befehlsprotokolle für die Interaktion der Client-Anwendungen mit entfernten Prozessen. Hierzu sollen auch die **Controller**-Klassen erweitert werden, um dann eine einfache Anbindung an den JobBroker und eine interaktive Steuerung der verteilt laufenden Simulationen zu erlauben. Darauf aufbauend ist die Weiterentwicklung der **Online-Simulation in ATLASneo** vorgesehen, um auch remote ausgeführte Simulationen, etwa im Rahmen von BEPU-Analysen, einbinden zu können. Hierzu gehören unter anderem die Anpassung der Benutzeroberfläche und die Umstellung der HDF5-Ausgabe auf den SWMR-Modus („Single Write, Multiple Read“) zur Verarbeitung von Daten parallel ablaufender Simulationen.

Ergebnisanalyse und Visualisierung

Im Bereich Ergebnisanalyse und Visualisierung steht die Integration der dynamischen Datenvisualisierung in den stabilen ATLASneo-Release-Zweig im Vordergrund. Die Webanwendung **SVGViz** und das zugehörige **Angulator**-Gadget müssen funktional und

technisch vereinheitlicht werden, um interaktive Visualisierungen mit verbessertem Layout, stabiler Datenanbindung und erweiterter Abbildungslogik der Simulationsdaten, z. B. geometrische Repräsentationen und generierter Geometrie, bereitzustellen. Darüber hinaus soll der Datenkonverter **x2hdf5** um zusätzliche Funktionen zur performanten und flexiblen Datenkonvertierung erweitert werden. Neben der Einführung eines Append-Modes und der Auswahl von zu übertragenden Nodes/Teil-Bäumen ist seine direkte Anbindung an ATLASneo über eine grafische Oberfläche geplant. Langfristig ist auch die Integration des **JobBrokers in ATLASneo** vorgesehen, um rechenintensive Operationen, wie Datenkonvertierungen oder Aggregationen, auf externe Rechenunits auszulagern und diese somit parallel ablaufen lassen zu können.

Software-Infrastruktur und Querschnittsaufgaben

In einem Folgevorhaben wird auch der nachhaltige Ausbau der Software-Infrastruktur ein wichtiger Aspekt sein. Die geplanten Arbeiten umfassen die Erweiterung der Testautomatisierung, den **Ausbau der Testabdeckung**, die Weiterentwicklung der CI/CD-Pipelines für **kontinuierliche Bereitstellung** (Continuous Deployment) sowie die verbesserte **Entwicklerdokumentation** durch automatische Extraktion aus dem Quellcode (z. B. Sphinx, Doxygen). Parallel dazu sollte die **Anwenderdokumentation** systematisch erweitert werden. Ein weiterer Schwerpunkt liegt auf der plattformunabhängigen **Bereitstellung der GRAMOVIS-Werkzeuge**. Hierzu gehören die Erstellung von binary bundles für Linux, die Sicherstellung der Plattformunabhängigkeit im Code, die Automatisierung von Build-Prozessen und Installer-Erstellung sowie **eine OpenSource-Freigabe** auf geeigneten Plattformen (z. B. GitLab.com). Damit ließe sich sowohl die Transparenz als auch die Nachhaltigkeit der Entwicklungen erhöhen und die langfristige Wartung der GRAMOVIS-Werkzeuge unabhängig von Projektlaufzeiten sicherstellen.

Literaturverzeichnis

/AC2 19/ Weyermann, F. et al., *"Development of AC2 for the simulation of advanced reactor design of Generation 3/3+ and light water cooled SMRs"* Kerntechnik, vol. 84, no. 5, 2019, pp. 357-366, URL: https://doi.org/10.3139/124.190068.

/APT 12/ Applied Programming Technology, Inc., Symbolic Nuclear Analysis Package (SNAP), User's Manual, October 2012.

/AUI 22/ AutoIt Consulting Ltd, *AutoIt, a freeware BASIC-like scripting language for automating the Windows GUI and general scripting*, v3.3.16.1, URL: https://www.autoitscript.com/site/, Zugriff: 07.11.2025.

/BMWi 21/ BMWi-Forschungsförderung zur nuklearen Sicherheit, A2, 2021 URL: https://www.bmwi.de/Redaktion/DE/Publikationen/Energie/projektfoerderprogramm-nukleare-sicherheitsforschung.pdf.

/BOR 12/ Ronald L. Boring et al., *"Digital Full-Scope Mockup of a Conventional Nuclear Power Plant Control Room, Phase 1: Installation of a Utility Simulator at the Idaho National Laboratory"*, Idaho National Lab., 2012.

/CSS 25/ W3C, *CSS Snapshot 2025*, W3C Group Note, 18 September 2025, URL: https://www.w3.org/TR/css/#css, Zugriff: 07.11.2025.

/D3JS/ Mike Bostock and Observable, Inc. D3.js - a free, open-source JavaScript library for visualizing data, Stand: 07. November 2025 URL: https://d3js.org/, Zugriff: 07.11.2025.

/FDE 25/ Scheuer, J. et al., *libfde, Fortran Development Extensions*, v2.8.5, 2025, URL: https://gitlab.com/Zorkator/libfde, Zugriff: 07.11.2025.

/FAH 21/ EC Software GmbH, *FastHelp, a Windows Help File Generator*, Stand: 06. Dezember 2021, URL: https://www.fast-help.com/, Zugriff: 07.11.2025.

/GIL 25/ GitLab, *The DevOps Platform*, Stand: 07. November 2025,
URL: https://about.gitlab.com/, Zugriff: 07.11.2025.

/HDF 25/ The HDF Group, *“HDF5, a technology suite that makes possible the management of extremely large and complex data collections”*,
URL: https://www.hdfgroup.org/, Zugriff: 07.11.2025.

/HTML5/ Wikipedia, *HTML5, the hypertext markup language*, ver. 5,
Stand: 07. November 2025,
URL: https://de.wikipedia.org/wiki/HTML5, Zugriff: 07.11.2025.

/IRS 14/ Institute de Radioprotection et de Sûreté Nucléaire, *“XASTEC Data Set Editor, User’s Guide”*, Draft Version, 2014.

/JSC 25/ Wikipedia, *JavaScript (kurz JS)*, Stand: 07. November 2025,
URL: https://de.wikipedia.org/wiki/JavaScript, Zugriff: 07.11.2025.

/JUP 25/ Fernando Pérez et al., *Project Jupyter*,
URL: https://jupyter.org/, Zugriff: 07.11.2025.

/KON 17/ Technische Notiz zur AIG-Code Dokumentation, Version 1, 27.06.2017.

/LARK 24/ Erez Shinan, *Lark* - a parsing toolkit for Python, Rev. 1.2.2, 2024
URL: https://pypi.org/project/lark/, Zugriff: 07.11.2025.

/MOD 25/ Open Source Modelica Consortium, *A Modelica-based modeling and simulation environment*,
URL: https://www.openmodelica.org/, Zugriff: 07.11.2025.

/NIM 18/ Nim Lang Team, *Nim, a statically typed compiled systems programming language*, Stand: 03. März 2025,
URL: https://nim-lang.org/, Zugriff: 07.11.2025.

/PAD 25/ Pandoc, *A universal document converter*, Stand: 03. März 2025,
URL: https://pandoc.org/, Zugriff: 07.11.2025.

/PLY 18/ David Beazley, *PLY* - Lex and Yacc for Python, Version 3.11, 2018
URL: https://pypi.org/project/ply/, Zugriff: 07.11.2025.

/PYS 17/ PySide, *Python for Qt, provides LGPL-licensed Python bindings for Qt*, Stand: 10. Juli 2017, URL: https://wiki.qt.io/PySide, Zugriff: 07.11.2025.

/PYS 22/ PySide2, *Python bindings for the Qt cross-platform application and UI framework*, Version 5.15, 2022,
URL: https://pypi.org/project/PySide2/, Zugriff: 07.11.2025.

/PYT 25/ Python Software Foundation, *Python is a programming language that lets you work more quickly and integrate your systems more effectively,* Stand: 07. November 2025,
URL: https://www.python.org/, Zugriff: 07.11.2025.

/QTC 25/ The Qt Group, *Cross-platform software libraries and APIs*,
Stand: 07. November 2025,
URL: https://www.qt.io/#, Zugriff: 07.11.2025.

/SCH 22/ Scheuer, J. et al., *„Weiterentwicklung der GRAMOVIS-Anwendungswerkzeuge zur Unterstützung der AC²-Simulationscodes“*, März 2022, GRS-664.

/SVG2/ W3C, *Scalable Vector Graphics (SVG) 2*, Stand: 04. Oktober 2018,
URL: https://www.w3.org/TR/SVG2/, Zugriff: 07.11.2025.

/SIT 21/ 3V Services, *SimInTech*, Stand: 08. Dezember 2021,
URL: http://3v-services.com/#simintech, Zugriff: 08.12.2021.

/VOG 18/ Voggenberger, T. et al., *„Weiterentwicklung von Methoden zur interaktiven Modellierung und zur Visualisierung in ATLAS-GRAMOVIS“*, Dezember 2018, GRS-533.

/WAL 17/ Edward Waller et al., *“A simulator-based nuclear reactor emergency response training exercise”*, Journal of Emergency Management Vol. 15, No. 6, November/December 2017.

Abbildungsverzeichnis

Abb. 1.1 Übersicht zum Einsatz von GRAMOVIS-Werkzeugen bei Sicherheitsanalysen 2

Abb. 2.1 Thermohydraulik-Modellierung einer Versuchsanlage in ADM 9

Abb. 2.2 Online-Simulation in ATLASneo. Neben der Dateiauswahl über Dialoge ist die Eingabe von Parametern (rot) zur Protokoll-gesteuerten Simulation möglich 11

Abb. 2.3 Beispielanwendung einer frühen Version von ATLASneo und der entworfenen Funktionsmodule zur Online-Steuerung und zur Darstellung von Zeitreihen, Leittechnik-Netzwerken (GCSM) sowie der erste Prototyp dynamischer Bilder 12

Abb. 3.1 Typische Phasen bei der Erstellung von Sicherheitsanalysen 13

Abb. 3.2 Zusammenspiel von GRAMOVIS-Werkzeugen und ihr typisches Einsatzgebiet 13

Abb. 3.3 Grundlegende Funktionsweise des Input Parametrisierungs-Systems 16

Abb. 3.4 Das Commandline-Interface (CLI) und die Optionen des InputExpanders 18

Abb. 3.5 Konzept der Grammatik-basierten Eingabe-Analyse 20

Abb. 3.6 Grammatik-Definition generischer Control-Words in AC²-Eingaben 22

Abb. 3.7 Das Commandline-Interface (CLI) und die Optionen des InputProcessors 24

Abb. 3.8 Ablaufschema Grammatik-basierte Eingabe-Analyse. Der aus der Grammatik generierte Parser erzeugt einen Parse-Tree / AST mit den aus den Eingabedateien eingelesenen Informationen. Dieser wird gem. der gewählten Transformer bearbeitet (AST´) bevor er durch einen Formatter im Zielformat ausgegeben wird. 27

Abb. 3.9 Beispiel einer durch den InputProcessor verarbeite ATHLET-Eingabe (links). Die Ausgabe des daraus erstellte AST (rechts) zeigt die effektiven Daten der Eingabe und die zu MatrixGroup-Objekten zusammengefassten tabellarischen Daten 29

Abb. 3.10 ADM im Einsatz bei blockorientierter Leittechnik-Modellierung im AGM-Modul 32

Abb. 3.11 ADM im Einsatz bei Modellierung der Thermohydraulik im ATM-Modul 34

Abb. 3.12 API-Zugang zum JobBroker im Browser über das Webinterface. Hier gezeigt ist die Abfrage der Endpoints des Jobs 5 und die Alternative als `curl`-Aufruf 39

Abb. 3.13 Auslagerung von Simulationsprozessen unter Verwendung des JobBrokers 40

Abb. 3.14 Zusammenstellung eines Simluations-Controllers. Mixin-Klassen bieten verschiedene Funktionen und werden nach Bedarf durch Ableitung kombiniert 43

Abb. 3.15 Benutzerdefinierte Protokoll-Ausgaben werden im neuen Console-Tab separat gesammelt und erlauben die Nachverfolgung der geskripteten Handmaßnahmen 46

Abb. 3.16 Das überarbeitete Steuermodul der Online-Simulation erlaubt den Zugriff sowohl auf alle Zeitschritte als auch auf die im Slider markierten Restart-Punkte (rot) 50

Abb. 3.17 Das überarbeitete Commandline-Interface (CLI) und die Optionen von x2hdf5 58

Abb. 3.18 Die auf Angular basierende Bilddynamisierung in Anwendung. Die interaktiven Bedienelementen, wie Farbskalen und ColorPicker erlauben die Feineinstellung der Wertabbildung auf die Füllfarben der eingeladenen SVG-Geometrie 64

Abb. 3.19 Diverse Verbesserungen in Rahmenanwendung und Gadgets von ATLASneo 66

Abkürzungsverzeichnis

ADM	**A**C² **D**esign **M**odeller *(GUI-Anwendung)*
AGM	**A**THLET **G**CSM **M**odeller *(ADM-Modul)*
AIG	**A**THLET **I**nput **G**raphics *(GUI-Anwendung)*
APG	**A**TLAS **P**icture **G**enerator *(GUI-Anwendung)*
API	**A**pplication **P**rogramming **I**nterface *(Programmierschnittstelle)*
ASCII	**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange
AST	**A**bstract **S**yntax **T**ree
ATHLET	**A**nalyse der **Th**ermohydraulik von **Le**cks und **T**ransienten *(Code)*
ATHLET-CD	**A**THLET **C**ore-**D**egradation *(Code)*
ATLAS	**AT**H**L**ET **A**nalyse **S**imulator *(GUI-Anwendung)*
ATM	**A**THLET **T**hermohydraulic **M**odeller *(ADM-Modul)*
CI/CD	**C**ontinuous **I**ntegration / **C**ontinuous **D**eployment
CI-Runner	Ausführumgebung für CI-Jobs (GitLab Runner)
CLI	**C**ommand-**L**ine **I**nterface
CSS	**C**ascading **S**tyle**s**heets *(Standard für Layout-Spezifikation)*
COCOSYS	**Co**ntainment **Co**de **Sys**tem *(Code)*
DLL	**D**ynamic **L**ink **L**ibrary *(MS-Windows)*
EBNF	**E**xtended **B**ackus-**N**aur-**F**orm
FDE	**F**ortran **D**evelopment **E**xtensions *(Programmbibliothek)*
GCSM	**G**eneral **C**ontrol **S**ystem **M**odule *(ATHLET-Leittechniksystem)*
GRAMOVIS	**Gra**fische **Mo**dellierung und **Vis**ualisierung *(Projekt)*
GUI	**G**raphical **U**ser **I**nterface
HCO	**H**eat **C**onduction **O**bject *(ATHLET-Wärmeleitungsobjekt)*
HDF5	**H**ierarchical **D**ata **F**ormat, Version **5** *(Datenformat)*
HTML	**H**yper**t**ext **M**arkup **L**anguage
HTTP	**H**yper**t**ext **T**ransfer **P**rotocol
JSON	**J**ava**S**cript **O**bject **N**otation
LALR(*k*)	**L**ook **A**head **L**eft-to-Right, **R**ight-Reduction, *k*-Lookahead
MPI	**M**essage **P**assing **I**nterface
NetCDF	**Net**work **C**ommon **D**ata **F**ormat *(Datenformat)*
PBS	**P**ortable **B**atch **S**ystem
PDF	**P**ortable **D**ocument **F**ormat
PLY	**P**ython **L**ex-**Y**acc *(Parsergenerator-Framework)*
REST-API	**Re**presentational **S**tate **T**ransfer-**API** *(Schnittstelle)*

SO	**S**hared **O**bject *(Unix/Linux, vgl. DLL)*
SVG	**S**calable **V**ector **G**raphics *(Vektor-Grafikformat)*
TFO	**T**hermo-**F**luid-**O**bjekt *(ATHLET)*
VNC	**V**irtual **N**etwork **C**omputing
XML	E**x**tensible **M**arkup **L**anguage
Xvfb	**X** **v**irtual **f**rame**b**uffer *(virtueller Grafikserver unter Unix/Linux)*
YAML	**Y**et **A**nother **M**arkup **L**anguage

Glossar

- **AC²**

  GRS-Softwarepaket, das mehrere System- und Containment-Codes (ATHLET, ATHLET-CD, COCOSYS) zur Simulation von thermohydraulischen und physikalischen Prozessen in kerntechnischen Anlagen umfasst.

- **ADM (AC² Design Modeller)**

  Grafisches Modellierungssystem zur Erstellung und Bearbeitung von Eingabedaten für die AC²-Rechencodes. Besteht aus den Komponenten AGM und ATM.

- **AGM (ATHLET GCSM Modeller)**

  Modul des ADM-Systems zur Modellierung von leittechnischen Anlagensystemen (GCSM = General Control System Model).

- **Angular**

  OpenSource-Webframework zur Entwicklung moderner, komponentenbasierter Benutzeroberflächen. Angular wird vor allem für komplexe Webanwendungen eingesetzt und bietet Werkzeuge zur Datenbindung, Komponentenstrukturierung und reaktiven Darstellung von Inhalten. In GRAMOVIS wird Angular für die Realisierung interaktiver und dynamischer Visualisierungskomponenten genutzt.

- **API (Application Programming Interface)**

  Eine standardisierte Programmierschnittstelle, über die Softwarekomponenten miteinander kommunizieren können. APIs legen fest, wie Funktionen, Datenstrukturen und Dienste genutzt werden können, ohne deren interne Implementierung zu kennen. In GRAMOVIS wird u. a. über APIs der Zugriff auf externe Rechenkerne oder Serverdienste (z. B. den JobBroker) realisiert.

- **AST (Abstract Syntax Tree)**

  Abstrakter Syntaxbaum – eine hierarchische Datenstruktur, die den logischen Aufbau eines Quelltexts oder Eingabefiles beschreibt. ASTs dienen der syntaktischen Analyse und Weiterverarbeitung von Eingaben, etwa zur Validierung oder automatischen Umwandlung von Rechenmodelldaten.

- **ATM (ATHLET Thermohydraulic Modeller)**

  Modul des ADM-Systems zur interaktiven Modellierung thermohydraulischer Systeme und Komponenten.

- **ATHLET / ATHLET-CD**
  Systemcodes der GRS zur Simulation thermohydraulischer Vorgänge (ATHLET) bzw. zur Erweiterung um kernphysikalische und stoffchemische Effekte bei Kernschmelzen (ATHLET-CD).

- **ATLASneo**
  Visualisierungs- und Analysewerkzeug zur Darstellung, Auswertung und Nachbearbeitung von Simulationsdaten (z. B. aus AC²-Codes). Unterstützt interaktive Visualisierung, Online-Simulation und dynamische Ergebnisbewertung.

- **CI (Continuous Integration)**
  Softwareentwicklungsverfahren zur automatisierten Prüfung, Kompilierung und Bereitstellung von Software. In GRAMOVIS über GitLab realisiert, um Stabilität und Nachvollziehbarkeit der Entwicklungsstände sicherzustellen.

- **CLI (Command Line Interface)**
  Textbasierte Benutzerschnittstelle zur Bedienung von Programmen über Kommandozeilenbefehle. Eine CLI ermöglicht skriptgesteuerte Abläufe und wird in GRAMOVIS z. B. zur Steuerung des JobBroker-Dienstes genutzt.

- **COCOSYS**
  Containment-Code der GRS zur Simulation physikalischer Prozesse im Containment von Kernkraftwerken.

- **CSS (Cascading Style Sheets)**
  CSS ist eine Sprache, die u. a. das Aussehen und die Gestaltung von HTML-Dokumenten beschreibt. Sie trennt die inhaltliche Struktur einer Webseite von deren visueller Darstellung (Layout, Farben, Schriftarten, Abstände). Durch sogenannte Kaskadierung werden Formatierungsregeln hierarchisch angewendet, sodass globale und lokale Stilvorgaben flexibel kombiniert werden können.

- **Delegate-Konzept/-System**
  Softwarearchitekturprinzip, innerhalb von ATLASneo genutzt, um eine einheitliche Darstellung und Verwaltung von Einstellungsgrößen ermöglicht.

- **DLL (Dynamic Link Library)**
  Dynamische Programmbibliothek unter Windows, die gemeinsam genutzte Funktionen und Routinen enthält. DLLs werden zur Laufzeit geladen und erlauben die modulare Erweiterung von Anwendungen ohne Neukompilierung.

- **EBNF (Extended Backus–Naur Form)**
  Formale Notation zur Beschreibung der Syntax von Programmiersprachen oder Datenformaten. In GRAMOVIS wird EBNF verwendet, um die Struktur von Eingabedateien (z. B. für AC²-Codes) als sog. Grammatik zu definieren und daraus Parser zu erzeugen.

- **HDF5**
  Hierarchisches Datenformat zur Speicherung großer, strukturierter numerischer Datenmengen. Wird in GRAMOVIS und ATLASneo als Standardformat für Simulationsergebnisse verwendet.

- **HTML5 (HyperText Markup Language, Version 5)**
  HTML5 ist die aktuelle Version der Standard-Auszeichnungssprache (*engl: markup langauge*) im Internet. Sie definiert die Struktur und Bedeutung von Web-inhalten und enthält moderne Funktionen wie Multimedia (Audio, Video) und interaktive Grafiken, wie SVG. HTML5 bildet zusammen mit CSS und JavaScript die Grundlage moderner Webanwendungen.

- **JavaScript**
  Eine interpretierte, objektorientierte Programmiersprache, die vor allem zur Dynamisierung von Webseiten eingesetzt wird. Sie erlaubt Ereignissteuerung und die direkte Manipulation des dargestellten Inhalts. JavaScript wird auch zur Entwicklung komplexer Webframeworks, z. B. Angular genutzt.

- **Jinja**
  Python Template-Engine, die eine Trennung von Daten und Darstellung ermöglicht. Jinja wird häufig verwendet, um strukturierte Textdateien, z. B. HTML-Seiten oder Eingabedateien, zu generieren. In der GRAMOVIS-Umgebung dient Jinja der flexiblen und automatisierten Erzeugung von Eingabemodellen.

- **Jupyter**
  Offene Entwicklungsumgebung zur interaktiven Ausführung, Dokumentation und Visualisierung von Code, insbesondere für Python. Jupyter-Notebooks kombinieren Programmcode, Simulationsergebnisse, Texte und Grafiken in einer gemeinsamen Oberfläche und werden zunehmend auch in der Qualitätssicherung und Ergebnisanalyse von Simulationen verwendet.

- **LALR(1)**
  Abkürzung für **L**ook **A**head **L**eft-to-Right, **R**ight-Reduction, **1**-Lookahead – eine Klasse deterministischer Parser, die häufig in Compilerbau-Tools wie Yacc eingesetzt wird. LALR(1)-Parser erzeugen aus einer Grammatik effiziente Parser, die Texte, etwa Eingabedateien nach formalen Regeln analysieren können.

- **MPI (Message Passing Interface)**
  Standardisiertes Kommunikationsprotokoll für paralleles Rechnen. MPI ermöglicht den Datenaustausch zwischen Prozessen, die auf mehreren Rechenknoten laufen, und wird für die parallele Ausführung von Simulationen genutzt (z. B. im Rahmen von AC²-Rechencodes).

- **NetCDF (Network Common Data Format)**
  Selbstbeschreibendes, plattformunabhängiges Datenformat zur Speicherung, Dokumentation und Verteilung wissenschaftlicher Mess- und Simulationsdaten. Es ermöglicht die strukturierte Ablage großer multidimensionaler Datensätze (z. B. Zeitreihen, Gitterdaten) und wird in der Simulationstechnik häufig als Austauschformat eingesetzt, vgl. HDF5.

- **Online-Simulation**
  Betriebsmodus, bei dem Simulationen interaktiv gestartet, überwacht und während der Laufzeit beeinflusst werden können (z. B. für Handmaßnahmen oder Prozessüberwachung).

- **Python-Kernel**
  Ausführungsumgebung für Python-Code, die von interaktiven Anwendungen wie Jupyter Notebooks genutzt wird. Der Kernel führt Befehle aus, verwaltet Variablen und kommuniziert über ein standardisiertes Protokoll mit der Benutzeroberfläche. Dadurch lassen sich komplexere Abläufe und Berechnungen in separaten Prozessen durchführen. In GRAMOVIS wird der Ansatz zur Auslagerung von Simulationsprozessen eingesetzt.

- **REST-API (Representational State Transfer – API)**
  Architekturstil für webbasierte Programmschnittstellen, der auf einfachen HTTP-Operationen (GET, POST, PUT, DELETE) basiert. REST-APIs erlauben eine lose gekoppelte Kommunikation zwischen Anwendungen. Im GRAMOVIS-Umfeld wird z. B. der JobBroker über eine REST-API angesteuert.

- **SVG (Scalable Vector Graphics)**
  SVG ist ein XML-basiertes Dateiformat zur Darstellung zweidimensionaler Vektorgrafiken. Im Gegensatz zu Pixelgrafiken bleiben SVGs beliebig skalierbar, ohne an Qualität zu verlieren. Sie werden häufig in Webanwendungen verwendet, da sie direkt vom Browser interpretiert werden und per CSS oder JavaScript animiert und interaktiv gestaltet werden können.

- **VNC (Virtual Network Computing)**
  Protokoll zur Fernsteuerung grafischer Benutzeroberflächen über ein Netzwerk. VNC ermöglicht den Zugriff auf Desktop-Umgebungen anderer Rechner, was z. B. für die Überwachung oder das Testen von GUI-Anwendungen in der CI genutzt werden kann.

- **x2hdf5 (Converter)**
  Datenkonverter zur Vereinheitlichung und Transformation von Simulationsausgaben in das HDF5-Format. Ermöglicht die konsistente Nutzung und Visualisierung von Ergebnissen in ATLASneo.

Gesellschaft für Anlagen- und Reaktorsicherheit (GRS) gGmbH

Schwertnergasse 1
50667 Köln
Telefon +49 221 2068-0
Telefax +49 221 2068-888

Boltzmannstraße 14
85748 Garching b. München
Telefon +49 89 32004-0
Telefax +49 89 32004-300

Kurfürstendamm 200
10719 Berlin
Telefon +49 30 88589-0
Telefax +49 30 88589-111

Theodor-Heuss-Straße 4
38122 Braunschweig
Telefon +49 531 8012-0
Telefax +49 531 8012-200

www.grs.de

ISBN 978-3-911727-27-3