Weiterentwicklung der GRAMOVIS-Anwendungswerkzeuge zur Unterstützung der AC^2-Simulationscodes

GRS - 664

Gesellschaft für Anlagen-
und Reaktorsicherheit
(GRS) gGmbH

Weiterentwicklung der GRAMOVIS-Anwendungswerkzeuge zur Unterstützung der AC²-Simulationscodes

Abschlussbericht

Matthias Behler
Volker Jacht
Kaloyan Pavlov
Benedikt Schätz
Josef Scheuer
Thomas Voggenberger

März 2022

Anmerkung:

Das diesem Bericht zugrunde liegende Forschungsvorhaben wurde mit Mitteln des Bundesministeriums für Wirtschaft und Energie (BMWi) unter dem Förderkennzeichen RS1572 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 BMWi übereinstimmen.

GRS - 664
ISBN 978-3-949088-55-1

Deskriptoren

ADM, AGM, AIM, ATHLET, ATLAS, ATLASneo, ATM, GRAMOVIS, Reaktorsicherheit, Simulation, Visualisierung

Kurzfassung

Die computergestützte Modellierung und Simulation des Verhaltens physikalischer Anlagen ist seit langem eine unverzichtbare Methode zum Verständnis der komplexen Vorgänge in deren Betrieb. Um der zunehmenden Komplexität bei der Durchführung und Analyse von Simulationen gerecht zu werden, müssen die bereits vorhandenen Werkzeuge zur grafischen Modellierung und Visualisierung (GRAMOVIS) kontinuierlich erweitert und an die neuen Anforderungen angepasst werden. Ziel dieses Projektes war die Weiterentwicklung der bereits existierenden Simulationsumgebung ATLAS-GRAMOVIS /VOG 18/ für die Nutzung der AC²-Rechencodes /AC2 19/. Diese Entwicklungen umfassen den AC² Design Modeller (ADM), vormals ATHLET Input Modeller (AIM; /VOG 18a/), zur Eingabegenerierung sowie ATLASneo / ATLAS zur interaktiven Steuerung und Ergebnisvisualisierung von Simulationen.

GRAMOVIS bietet Methoden zur grafischen Modellierung und Ergebnisauswertung für die AC²-Simulationsmodelle. Dieses Paket fasst Werkzeuge und Methoden zur grafischen Modellierung und Ergebnisbewertung zusammen, die den Anwender bei der Durchführung von Sicherheitsanalysen unterstützen können. Dazu eignen sich insbesondere interaktive Eingabewerkzeuge und Benutzeroberflächen zur Durchführung von Simulationsrechnungen sowie die visuelle Darstellung von Simulationsergebnissen in Form von Zeitreihenplots und Visualisierungen durch dynamische Bilder.

Für die interaktive Eingabenerstellung von ATHLET werden in ADM Module für thermohydraulische und regelungstechnische Systeme bereitgestellt, der ATHLET Thermohydraulic Modeller (ATM) und der ATHLET GCSM Modeller (AGM). Mit dem erreichten Entwicklungsstand ist der ATM in der Lage, einen großen Teil der ATHLET-Eingaben für Fluidsysteme zu unterstützen. Für den AGM, der bereits in der Praxis eingesetzt wird, wurden im vorliegenden Projekt hauptsächlich Wartungsarbeiten durchgeführt.

ATLASneo, das Reengineering des Analysesimulators ATLAS, ermöglicht eine interaktive Ausführung von Simulationen mit den in AC² enthaltenen Rechencodes sowie eine einfache Auswertung der Ergebnisse. Durch den Einsatz der GUI-Bibliothek Qt /QTC 22/ wurde eine plattformunabhängige Implementierung der Software möglich. Eine umfassende Modularisierung der enthaltenen Funktionalität stellt sicher, dass die Programmteile langfristig wartbar und erweiterbar bleiben. Die Entwicklungen der Kernfunktionalität von ATLASneo (Application Frame, MonitorWidget, HDF5- und PlotPanel) konnten im Projekt als AC² 3.3-kompatible Meilensteinversion abgeschlossen werden.

Abstract

Computer-aided modelling and simulation of physical plant behavior has long been an indispensable method for understanding the complex processes that occur in their operation. In order to cope with the increasing complexity in performing and analysing simulations, the already existing tools for graphical modelling and visualisation (GRAMOVIS) have to be continuously extended and adapted to the new requirements. The aim of this project was the further development of the already existing simulation environment ATLAS-GRAMOVIS /VOG 18/ for the use of the AC² computational codes /AC2 19/. These developments include the AC² Design Modeller (ADM), formerly ATHLET Input Modeller (AIM; /VOG 18a/), for input generation as well as ATLASneo / ATLAS for interactive control and result visualisation of simulations.

GRAMOVIS provides methods for graphical modelling and result evaluation for the AC² simulation models. This package groups tools and methods for graphical modelling and result evaluation, which can assist the user in performing safety analyses. Interactive input tools and user interfaces for performing simulation calculations as well as the visual representation of simulation results in the form of time series plots and visualisations through dynamic images are particularly suitable for this purpose.

For the interactive input generation of ATHLET, modules for thermohydraulic and control systems are provided in ADM, the ATHLET Thermohydraulic Modeller (ATM) and the ATHLET GCSM Modeller (AGM). With the level of development achieved, ATM is able to support a large part of the ATHLET input for fluid systems. For the AGM, which is already used in practice, mainly maintenance work was carried out in the present project.

ATLASneo, the reengineering of the ATLAS analysis simulator, provides an interactive execution of simulations with the computational codes contained in AC², as well as a straightforward evaluation of the results. With the use of the GUI library Qt, a platform-independent implementation of the software became possible. A comprehensive modularisation of the contained functionality ensures that the program parts remain maintainable and expandable in the long term. The developments of the core functionality of ATLASneo (application frame, MonitorWidget, HDF5- and PlotPanel) could be completed in the project as an AC² 3.3 compatible milestone version.

Inhaltsverzeichnis

1 Einleitung

Im Feld von Forschung und Sicherheitsbewertung von Reaktoranlagen stellt die Simulation physikalischen Anlageverhaltens seit langem eine unverzichtbare Methode dar, um die komplexen Vorgänge in deren Betrieb zu analysieren. Die Möglichkeiten der dabei zum Einsatz kommenden Rechencodes haben sich seit jeher ständig erhöht und erreichen in den durchführbaren Simulationen mit jedem Entwicklungsschritt neue Höhepunkte im räumlichen und zeitlichen Detaillierungsgrad. 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 versucht dieses Vorhaben für Simulationsprogramme der Reaktorsicherheit eine zeitgemäße Applikationsplattform bereitzustellen, um die Benutzer bei der Anwendung von Rechencodes zu unterstützen, fehleranfällige Arbeitsschritte möglichst zu automatisieren sowie den Erkenntnisgewinn aus den durchgeführten Simulationen zu maximieren. So tragen die hier geleisteten Entwicklungen dazu bei, Fehler zu vermeiden und das Verständnis der Ergebnisse von Simulationen zu verbessern. Dieser Bericht enthält eine detaillierte Beschreibung der wesentlichen Arbeiten und Ergebnisse des Vorhabens zur Erweiterung der GRAMOVIS-Anwenderwerkzeuge.

2 Zielsetzung und Arbeitsprogramm

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 an ATHLET gekoppelt, um diese um Funktionalitäten zu erweitern. Die GRS-Rechencodes, welche verwendet werden, um Vorgänge in Kernkraftwerken und deren Verhalten bei Ereignissen 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 Parametrisierung der angesprochenen Modelle und deren Verhalten im Zusammenspiel untereinander. 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 zum Problem 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. Viele Zeitreihen müssen in Relation zueinander gesetzt oder in einen geometrischen Kontext betrachtet werden, um richtig interpretiert werden zu 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 „Wissen“ über die simulierte Situation mitbringen, um die Daten aussagekräftig darstellen zu 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.

Um der Komplexität bei der Durchführung und Analyse von AC²-basierten Simulationen zu begegnen, müssen deshalb die bereits vorhandenen Werkzeuge zur graphischen Modellierung und Visualisierung, GRAMOVIS, erweitert und an die neuen Gegebenheiten angepasst werden. Des Weiteren ergeben sich durch den praktischen Einsatz der Codes und ihrer Unterstützungswerkzeuge oft Veränderungen in den Benutzeranforderungen. Erst durch die tägliche Verwendung zeigen sich die wahren Workflows und Anwendungsfälle (Use Cases), welche in der Projektplanung oder auch während der Entwicklung nur schwer abzuschätzen sind. Oft wird auch erst durch direkte Benutzerrückmeldungen die Notwendigkeit einer Weiterentwicklung von Anwendungskonzepten und Visualisierungsmethoden sichtbar, welche immer wieder nötig werden, um die Anwender bestmöglich in ihrer Arbeit unterstützen zu können. Auch diese Anpassungen sollten bei den im Projekt geplanten Entwicklungsschritten miteinfließen.

Nicht zuletzt muss in den Projektzielen auch die allgemeine Weiterentwicklung in Konzepten der Softwareentwicklung berücksichtigt werden, welche stete Wartung und Anpassung von Schnittstellen und eine Steigerung der Softwarequalität erfordern, um mit der allgemeinen Entwicklung Schritt halten zu können. Nur so kann die Anwendung der Werkzeuge auch unter den sich kontinuierlich weiterentwickelnden Bedingungen gewährleistet werden, welche zum einen durch die sich verändernden Anforderungen von Rechencodes und Anwendern, zum anderen auch durch die Betriebssysteme und Rechnerausstattung vorgegeben werden.

Das Vorhaben setzt inhaltlich auf dem durch das BMWi geförderten Vorhaben RS1537 „ATLAS-GRAMOVIS Weiterentwicklung von Methoden“ /VOG 18/ und der langjährigen Entwicklung des Analysesimulators ATLAS auf. Um die hierin entwickelten Methoden auf aktuelle Anforderungen der AC²-Rechencodes und der Anwender anzupassen, wurden für die Entwicklung die im folgenden beschriebenen Schwerpunkte gesetzt.

2.1.1 Allgemeine Weiterentwicklung der Softwarekonzepte

Das Projekt basiert auf bereits bestehenden, meist unabhängig voneinander entwickelten Softwareteilen, welche weiterentwickelt werden müssen, um deren Zusammenarbeit zu gewährleisten und zu verbessern. Gerade durch die oft sehr unterschiedlichen Entwicklungsansätze dieser Softwareteile müssen deren Schnittstellen erst überarbeitet werden, um eine einheitliche Anwendung zu ermöglichen. Durch die Erarbeitung gemeinsamer Schnittstellen ermöglicht dieses Projekt die Schaffung einer flexiblen

Softwarebasis zur einfachen Integration alter Bestandscodes, welche durch generische Steuerungsprogramme oder GUI-Elemente zur Bedienung und Konfiguration einfacher in den typischen Arbeitsablauf von Anwendern eingebunden werden können.

Im Rahmen dieser Weiterentwicklung muss auch die Wartbarkeit der Bestandscodes sichergestellt werden. Hierzu können in einzelnen Teilen verschiedene Maßnahmen zur Steigerung der Softwarequalität, wie Umstrukturierung und Refactoring, notwendig werden, 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.

2.1.2 Ausbau der Unterstützung bestehender Rechencodes

Der Fokus der vorangegangenen GRAMOVIS-Entwicklungen lag auf der Anwenderunterstützung bei der Durchführung und Ergebnisvisualisierung von ATHLET-Simulationen. Daneben gibt es eine ganze Reihe von Rechencodes, wie ATHLET-CD (ATHLET Core Degradation), COCOSYS oder QUABOX/CUBBOX, welche physikalische Phänomene behandeln, die in ATHLET nicht berücksichtigt werden. Diese Codes wurden vor der Schaffung des AC²-Pakets meist von ATHLET unabhängig entwickelt und konnten aufgrund der unterschiedlichen Programmstrukturen und Datenformaten in den Anwenderwerkzeugen zur Modellierung und Visualisierung nicht oder nur eingeschränkt unterstützt werden.

Mittlerweile hat die Kopplung, d. h. die gemeinsame Anwendung einzelner Rechencodes, maßgeblich durch die Spezifikation des Softwarepakets AC², stark an Bedeutung gewonnen und definiert Interoperabilität und Datenkompatibilität als zentrale Anforderung. Die inzwischen 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.

2.1.3 Weiterentwicklung der Anwendungswerkzeuge

Die Anwendung der Rechencodes birgt viele Bereiche, welche den Benutzern zum Teil sehr viel Detailwissen über die Eigenschaften der Codes abverlangen, um erfolgreiche Simulationen durchführen zu 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 weiter ausgebaut und neu auf die Anwendungsschwerpunkte abgestimmt werden. Besonders durch die Verschiedenartigkeit der Rechencodes, welche in Simulationsläufen auch gekoppelt eingesetzt werden, zeigt sich, dass die Anwendungswerkzeuge generell flexibler werden müssen, um mit der Entwicklung der Codes und den Anforderungen der Anwender Schritt halten zu können. Dadurch ergaben sich speziell in den Bereichen der Anwendungsunterstützung folgende Projektziele:

- **Modellierung und Input-Erstellung**
  - Unterstützung weiterer Rechencodes
    - ATHLET-CD, COCOSYS (Hauptmodul)
  - Integration manuell erstellter bzw. korrigierter ATHLET-Datensätze
  - Erweiterung der Funktionen und Überarbeitung des Input Modellers
    - Methode zur Erweiterbarkeit durch die Nutzer (Scripting)
    - Verwaltung gekoppelter Datensätze
- **Simulations-Durchführung**
  - Unterstützung aller relevanten Rechencodes
  - Starten und Überwachen von Cluster-Simulationsläufen
- **Ergebnis-Analyse und Visualisierung**
  - Gleichzeitige Nutzung verschiedener Datenquellen
  - Erweiterte Visualisierungsmöglichkeiten dynamisierter Bilder
  - Integration spezieller Visualisierungstools einzelner Rechencodes
  - Erweiterte Visualisierungsmöglichkeiten topologischer Strukturen
  - Ausgebaute Anwender-Dokumentation

3 Stand von Wissenschaft und Technik

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

3.1 Externe Entwicklungen

In der Simulation technischer Prozesse werden die Anwender in vielen Fällen durch interaktive graphische Werkzeuge oder Oberflächen (GUI) bei den Schritten von der Modellbildung (Eingabe) bis zur Ergebnisauswertung (Visualisierung) unterstützt. Dies gilt auch für wichtige Reaktorsicherheitscodes im internationalen Bereich. Exemplarisch werden zwei aktuelle amerikanische und französische Entwicklungen genannt. Sie stellen ähnliche Funktionen bereit, wie die für AC² entwickelten GRAMOVIS Werkzeuge.

Für die wichtigen US-NRC Codes TRACE, CONTAIN, COBRA, MELCOR, PARCS und RELAP5 ist SNAP „Symbolic Nuclear Analysis Package“ /AC2 19/ 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 Prozessdiagramme 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.

3.2 Arbeiten in Vorgängervorhaben

In vorangegangenen Projekten orientierten sich die Entwicklungen von Anwenderwerkzeugen fast ausschließlich an den Anforderungen die durch den zentralen Rechencode ATHLET vorgegeben wurden. Die folgenden Abschnitte beschreiben kurz die Arbeitsbereiche in denen ATHLET-Anwender bislang durch Werkzeuge unterstützt wurden, welche aus den Entwicklungen früherer Vorhaben, wie zuletzt RS1537, hervorgingen. Diese umfassten den ATHLET Input Modeller (AIM) zur Eingabeerstellung sowie ATLASneo zur Steuerung und Ergebnisvisualisierung der ATHLET-Simulation. Bei ATLASneo handelt es sich um ein plattformunabhängiges Reengineering der langjährig entwickelten ATLAS-Software auf Basis aktueller Softwaretechniken. ATLASneo hatte bis dato den Stand einer ersten Beta-Version erreicht, die zwar schon für einige Aufgabenstellungen in der überwachten Simulation und Ergebnisanalyse eingesetzt werden konnte, welche aber weder in Stabilität noch Funktionsumfang an den Vorgänger ATLAS heranreichte.

3.2.1 Modellierung und Input-Erstellung

Die Anwendung von Rechencodes erfordert Eingabe- bzw. Inputdateien, in denen alle notwendigen Parameter und Angaben für die Berechnungen spezifiziert sind. Diese Eingabedateien beinhalten in der Praxis meist viele Definitionen, die in einer ganz bestimmten Struktur vorgegeben werden müssen und daher oft unübersichtlich und fehleranfällig sind. Im Fall von ATHLET wurde die Erstellung dieser Dateien bereits durch das Werkzeug AIM (AGM + ATM) /VOG 18/ unterstützt, dessen Entwicklung in früheren Projekten

begonnen wurde. AIM verfolgte bereits einen visuellen, modellbasierten Ansatz, welcher die Erstellung konkreter Bereiche der Eingabedaten für die ATHLET-Modelle erlaubte. Graphische Objekte konnten auf Arbeitsblättern erzeugt und verknüpft werden. Die Daten der Objekte wurden entweder in Eingabemasken oder mit Geometrieobjekten für die Fluidobjekte (TFO) vorgegeben (siehe Abb. 3.1). Die Inputdaten wurden formal geprüft bevor sie im ATHLET-Format ausgegeben (exportiert) werden konnten.

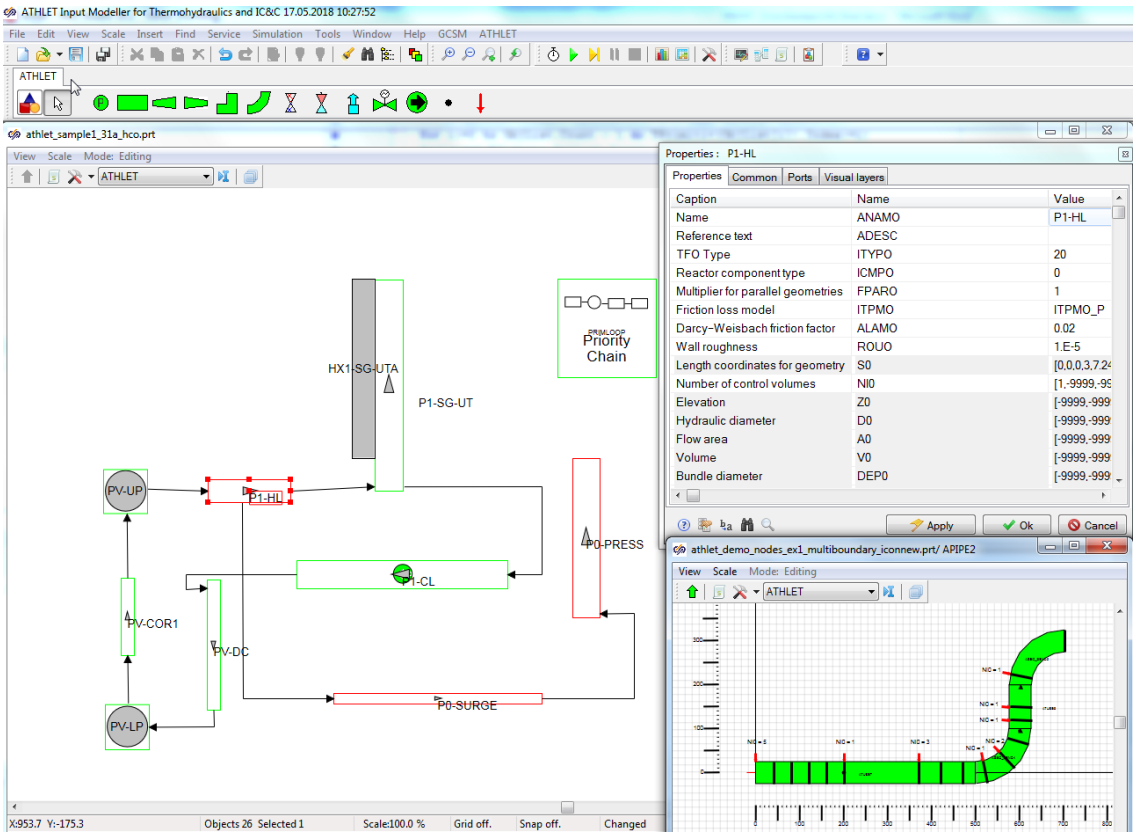

Abb. 3.1 Interaktive Eingabemodellierung für ATHLET mit AIM

3.2.2 Simulations-Durchführung

Wie sich durch den vielfachen praktischen Einsatz früherer Entwicklungen von ATLAS gezeigt hat, kann die Durchführung von Simulationen sehr gut durch eine graphische Benutzeroberfläche unterstützt werden. Diese ersetzt das manuelle Starten auf der Konsole und hilft dem Anwender, indem sich der zu verwendenden Rechencode und die nötigen Eingabedateien und -Parameter abfragen bzw. auswählen lassen. Diese Eingaben werden dann verwendet, um die oft komplexen Befehlsfolgen zusammenzusetzen und somit die Simulation zu starten. Nach erfolgreichem Start des Simulationsprozesses konnte dieser anschließend überwacht und in einem gewissem Rahmen gesteuert werden. Da allerdings nur ATHLET durch eine spezielle Erweiterung (Plugin) zur

Interpretation von Steuerbefehlen fähig war, konnte auch nur dieser im Online-Modus von ATLAS betrieben werden.

Das Steuermodul für ATLASneo, dessen Entwicklung im Vorgängerprojekt RS1537 begonnen wurde, realisierte diesen Schritt über einen allgemeineren Ansatz: Auf Basis der erweiterten Zugriffs- und Steuermöglichkeiten von ATHLET war es bereits möglich den Rechencode als „Shared Library“ (DLL) in den Prozess von ATLASneo einzuladen, diesen wie eine Funktion zu starten und auf dessen Simulationszustand direkt zuzugreifen. Dadurch wurde es möglich Simulationsläufe sehr detailliert und zeitschrittgenau zu überwachen sowie diese durch direkte Datenzugriffe zu steuern. Hierfür musste ATHLET zwar für den FDE-basierten Datenzugriff und Ablaufsteuerung (vgl. /FDE 21/) vorbereitet werden, die aktive Interpretation von Steuerbefehlen durch ATHLET ist aber für diese Art der Online-Simulation seither nicht mehr notwendig.

3.2.3 Ergebnis-Analyse und Visualisierung

Das im Vorgängerprojekt erarbeitete Konzept zum Reengineering von ATLAS und der implementierte Prototyp von ATLASneo (siehe Abb. 3.2) bot bereits eine generische Rahmenanwendung, die durch unabhängige Funktionsmodule (Gadgets) für unterschiedliche Aufgabenstellungen erweitert werden konnte und erlaubte so die Benutzer bei den verschiedensten Aufgaben innerhalb einer Applikation zu unterstützen. Neben dem in Abschnitt 3.2.2 beschriebenen Steuermodul zur Durchführung von Simulationen wurden auch einige Module zur Analyse und Visualisierung der Ergebnisse entworfen.

3.3 Gewonnene Erkenntnisse und neue Anforderungen

Aus den im Vorgängerprojekt durchgeführten Arbeiten und den von den ersten AIM- bzw. ATLAS-Anwendern eingeholten Rückmeldungen konnten viele Erkenntnisse über die Bedürfnisse der Benutzer und die Richtung zukünftiger Anforderungen gewonnen werden. Wie sich klar gezeigt hat, sind der modulare Aufbau der Anwendungen, eine plattformunabhängige und skriptbare Umgebung sowie der Einsatz von Standard-Dateiformaten und OpenSource-Komponenten unverzichtbare Voraussetzungen, um mit der Entwicklung und den Ansprüchen der Anwendern Schritt halten zu können. Die hierfür getroffenen Designentscheidungen wurden in diesem Projekt weiterverfolgt und ausgebaut.

Die im Vorgängerprojekt begonnene Entwicklung von ATLASneo und seinen Funktionsmodulen erfolgte mit dem Fokus, mittelfristig die in ATLAS vorhandene Funktionalität bereitzustellen und diesen somit langfristig ablösen zu können. Daher wurden vorrangig Funktionsmodule zur Trenddarstellung, zur Darstellung dynamisierter Bilder und zur Topologie-Visualisierung von Leittechnik-Netzwerken entworfen (siehe Abb. 3.2). Die vielversprechendsten Ansätze hieraus wurden dann im Rahmen dieses Projektes weiterentwickelt (siehe AP 4, Abschnitt 4.4), wobei die Priorisierung nach Praxistauglichkeit und Rückmeldungen der Anwender erfolgte. Dadurch konnten ATLASneo und seine Funktionsmodule schon während der aktuellen Projektlaufzeit als ergänzendes Werkzeug neben ATLAS praktische Anwendung finden.

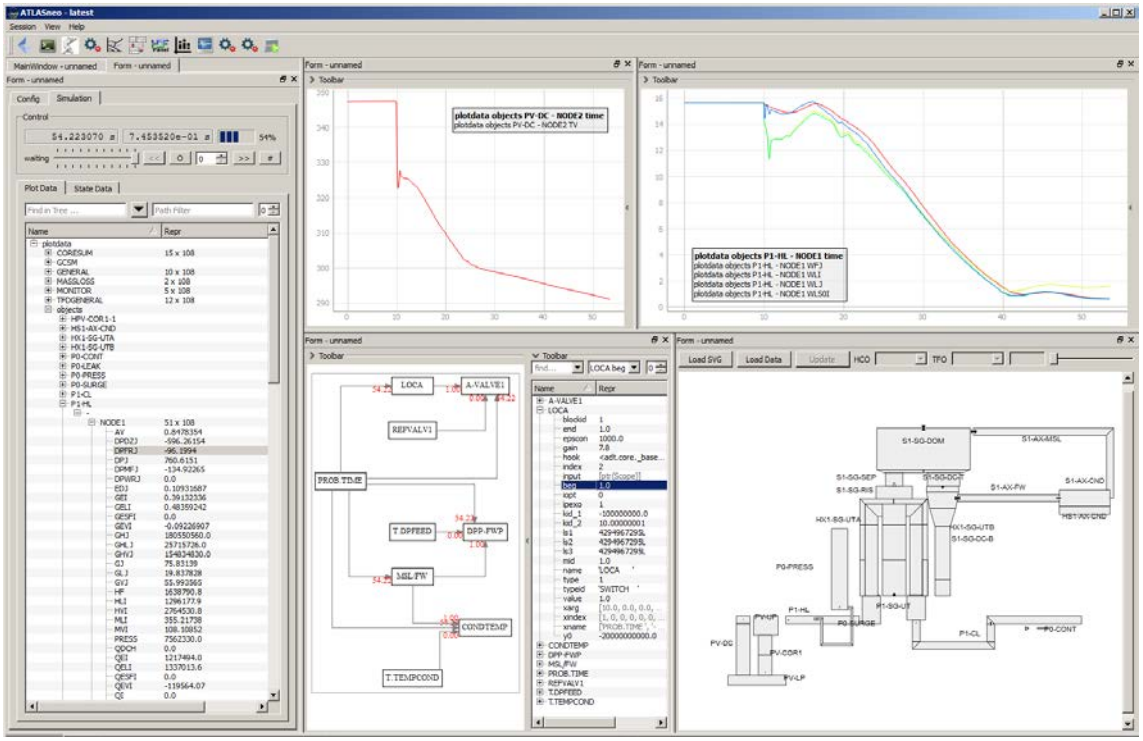

Abb. 3.2 Beispielanwendung einer frühen Version von ATLASneo

Die frühe Version von ATLASneo erlaubte bereits die rudimentäre Anwendung der entworfenen Funktionsmodule zur Online-Steuerung und zur Darstellung von Zeitreihen, Leittechnik-Netzwerken (GCSM) sowie dynamischer Bilder.

4 Arbeiten und Ergebnisse

Das vorliegende Projekt verfolgte das Ziel, die in den GRAMOVIS-Werkzeugen vorhandenen Möglichkeiten zur graphischen Modellierung und Ergebnis-Visualisierung zu erweitern. Die Anwender sollten beim typischen Vorgehen (Workflow) zur Simulation und Analyse von Anlagenverhalten noch besser unterstützt werden. Die bislang auf ATHLET ausgerichtete Funktionalität sollte jetzt auf das AC²-Softwarepaket erweitert werden und durch flexible Softwaretechniken auch andere bzw. zukünftige Entwicklungen im Blick behalten. Die folgenden Abschnitte geben einen Überblick über das breite Spektrum an Entwicklungsarbeiten, welches sich durch diese Zielsetzung ergab. Hierin werden die erarbeiteten Konzepte, Details zur Realisierung sowie die erzielten Ergebnisse dargestellt.

Wie in Abbildung 4.1 dargestellt, lässt sich die Anwendung verschiedener Rechencodes in drei typischen Phasen unterteilen: Input-Erstellung, Simulations-Durchführung und Ergebnis-Analyse. Diese drei Arbeitsschritte werden in GRAMOVIS durch zwei Hauptanwendungen unterstützt: Die Input-Erstellung durch das visuelle Modellierungssystem ADM; die Simulations-Durchführung und Ergebnis-Analyse durch ATLASneo, bzw. dessen Vorgänger ATLAS.

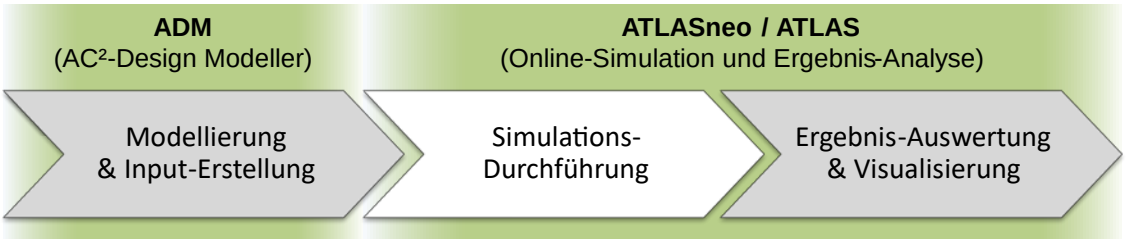

Abb. 4.1 Unterstützung durch GRAMOVIS-Werkzeuge

Die drei Phasen bei der Anwendung von Rechencodes werden durch die GRAMOVIS-Werkzeuge ADM und ATLAS/ATLASneo unterstützt.

4.1 AP 1: Anpassungen in bestehenden Rechencodes

Um die Simulationscodes des Softwarepakets AC² im GRAMOVIS Anwender-Workflow integrieren zu können, müssen diese die dafür notwendigen Voraussetzungen mitbringen. Je nach Grad der erforderlichen Einbindung ist es immer wieder notwendig, Anpassungen in den Schnittstellen und im Aufbau der Codes vorzunehmen, damit diese Kompatibilität erreicht und auch zukünftig sichergestellt werden kann. Im Folgenden wird das Zusammenspiel der Software-Komponenten graphisch dargestellt (siehe Abb. 4.2)

und im Anschluss die im Projekt geplanten Anpassungen kurz zusammengefasst. Diese werden dabei nach den drei Anwendungsphasen von GRAMOVIS gruppiert, damit deren Relevanz für die einzelnen Werkzeuge abgeschätzt werden kann. Die detaillierte Beschreibung der durchgeführten Arbeiten und die dabei erzielten Ergebnisse finden sich dann in den Abschnitten 4.1.1 und 4.1.2.

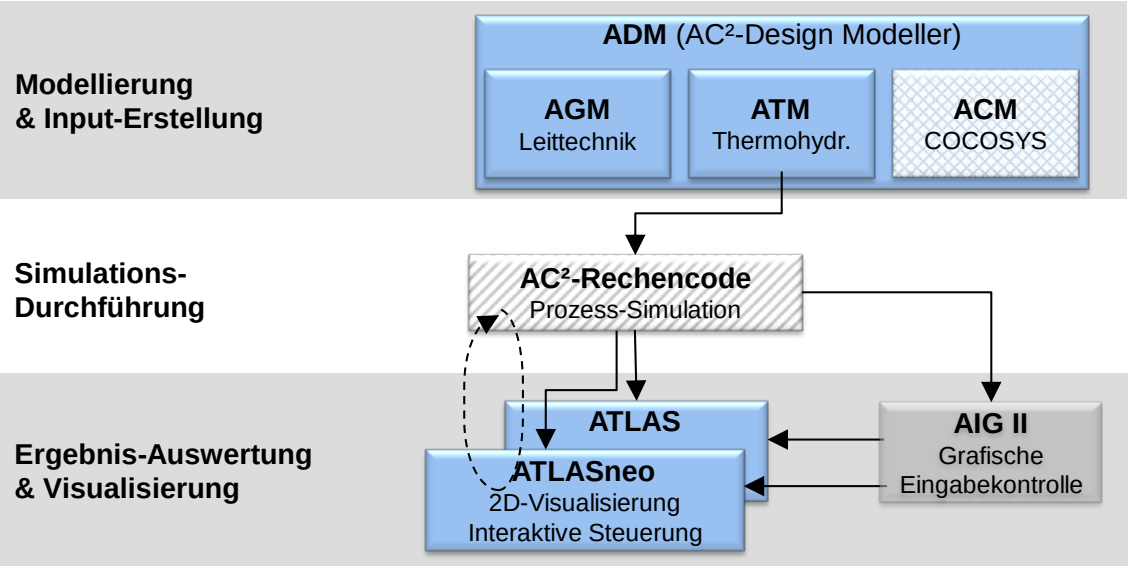

Abb. 4.2 Zusammenspiel von GRAMOVIS-Werkzeugen und Rechencodes

- **Input-Erstellung**

  Zur Unterstützung bei der Input-Erstellung sollte seitens der Rechencodes eine Vereinheitlichung und Flexibilisierung des Eingabeformates erreicht werden. Es war ein Konzept zu erarbeiten, wie eine Standardisierung des Dateiformats zur Input-Spezifikation realisiert werden kann (siehe Abschnitt 4.1.1).

- **Simulations-Durchführung**

  Die Durchführung von Simulationen kann durch Werkzeuge, wie einer graphischen Benutzeroberfläche, auf unterschiedliche Weise unterstützt werden. Simulationen können *offline* (eigenständig) oder *online* mit der Möglichkeit des interaktiven Zugriffs zur Steuerung und zur Überwachung des Simulationszustands gestartet werden. Durch die interaktive Arbeitsweise in ATLASneo konzentriert sich das darin enthaltene Funktionsmodul (MonitorWidget) vor allem auf die Durchführung von Online-Simulationen. Abschnitt 4.1.2 beschreibt die Funktionsweise und die im Projekt unternommenen Arbeiten, um die Rechencodes auf die notwendigen Datenzugriffe und die interaktive Steuerung der Simulationen vorzubereiten.

- **Ergebnis-Analyse**
  Die Möglichkeiten zur Darstellung von Simulationsergebnissen ist ein sehr wichtiger Aspekt bei der Verwendung von Rechencodes. Unter den GRAMOVIS-Werkzeugen fällt diese Aufgabe zwar hauptsächlich auf ATLAS/ATLASneo, allerdings ist für die generelle Zugänglichkeit und eine einheitliche Verarbeitung der Ergebnisse ein robustes und universell einsetzbares Datenformat unerlässlich. Mit der Festlegung auf HDF5 (/HDF 17/) wurde hier ein zukunftsfähiges Format gewählt, welches sowohl hohes Datenaufkommen handhaben kann als auch die Flexibilität bietet, den Anforderungen verschiedenster Rechencodes gerecht zu werden. Im Rahmen dieses Projekts sollten die Grundlagen geschaffen werden, um HDF5 als Ausgabeformat in den AC²-Rechencodes zu realisieren und durch eine Konvertierungsmöglichkeit auch die Nutzbarkeit der bisherigen Ausgabeformate zu erhalten (siehe Abschnitt 4.1.2).

4.1.1 Vereinheitlichung des Eingabeformates

Praktisch alle Rechencodes des AC²-Pakets welche mit ATHLET gekoppelt betrieben werden können, verwenden ein an ATHLET angelehntes Eingabeformat oder sogar dessen Routinen in abgewandelter Form. Aus dem allgemeinen Bestreben, die Eingaben für die Codes einheitlich eingeben und verarbeiten zu können, entstand das Projektziel, mögliche Ansätze für die Vereinheitlichung der Eingabeformate zu untersuchen und den zielführendsten soweit möglich, in einer entsprechenden Implementierung umzusetzen. Die folgenden Abschnitte beschreiben die untersuchten Ansätze, die erzielten Entwicklungsergebnisse sowie die dadurch gewonnenen Erkenntnisse für die zukünftige Weiterentwicklung.

4.1.1.1 Auslagerung des ATHLET-Parsers

Ein Ansatz, um das Eingabeformat der Rechencodes zu vereinheitlichen, war die Auslagerung der Einlese-Routinen von ATHLET in eine Bibliothek (*parser.dll*), welche dann von allen AC²-Codes (einschließlich ATHLET) zum Einlesen und Analysieren des Inputs genutzt werden sollte. Hierfür wurde geplant eine möglichst einfache Schnittstelle zu entwerfen, die dann von verschiedenen Codes genutzt werden konnte. Des Weiteren sollte ermittelt werden, welche Erweiterungen des Eingabeformats notwendig sind, um allen AC²-Codes gerecht zu werden und ob es möglich wäre diese Erweiterungen abwärtskompatibel zu definieren. Wie in Abbildung 4.3 dargestellt, sollte für die Implementierung des neuen Eingabeformats ein Parser-Generator (Lex, Yacc) zum Einsatz

kommen, wodurch das Format sehr flexibel und ohne Einschränkungen definiert werden kann. Die Parsing-Routinen des neuen Formats könnten dadurch eigenständig als C/C++-Bibliothek (*input.dll*), implementiert werden, welche in ihren Schnittstellen kompatibel wäre zur Bibliothek des ursprünglichen ATHLET-Eingabeformats.

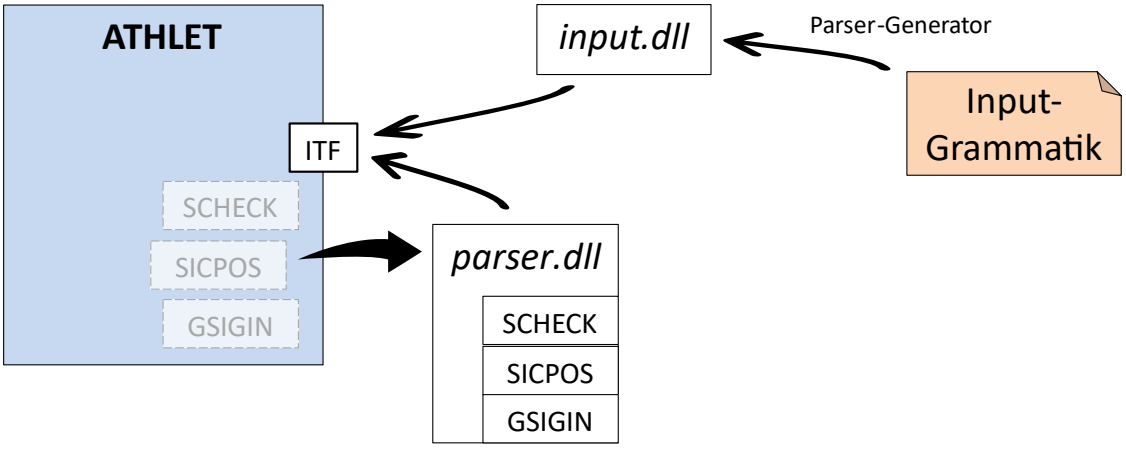

Abb. 4.3 Auslagerung der ATHLET-Parsing Routinen

Die Auslagerung der ATHLET-Parsing Routinen als dynamische Bibliothek sollte die gleichzeitige Entwicklung des neuen Eingabeformats und eine parallele Verwendbarkeit der Formate in allen AC²-Codes ermöglichen.

Für die Anbindung der so erstellten Bibliotheken sollte die in ATHLET bereits eingesetzte Funktion zum Nachladen dynamischer Bibliotheken aus der externen Kopplungs-Bibliothek FDE /FDE 21/ genutzt werden, welche auch leicht in die anderen AC²-Codes integriert werden könnte. Die Entwicklung der Parsing-Bibliothek könnte dadurch, ohne große Gefahr von Inkompatibilitäten durch die ATHLET-Weiterentwicklung erfolgen. Des Weiteren bietet dieser Ansatz den Vorteil einer gleichzeitigen Verwendbarkeit beider Eingabeformate, was den Umstieg für die Anwender erleichtern würde und darüber hinaus die Einlesbarkeit bestehender Eingabedatensätze sicherstellen könnte. Trotz aller Vorteile und der Flexibilität in der Entwicklung dieses Ansatzes stellte sich dieser leider als zu aufwendig heraus, da die aus ATHLET auszulagernden Einlese-Routinen sehr stark mit dem Rest des Codes verwoben sind. Da diese Routinen nach dem Herauslösen aus ATHLET über die generischen Schnittstellen wiederangebunden werden müssen, wären auch im ATHLET-Code sehr viele Anpassungen nötig und es würde sich ein unverhältnismäßig hoher Aufwand für die Implementierung ergeben. Um diesen zu vermeiden, wurde der Ansatz wie in Abschnitt 4.1.1.2 beschrieben weiterentwickelt.

4.1.1.2 Eingabeformat unter Einbettung des Python-Interpreters

Aufgrund der Erkenntnisse aus Abschnitt 4.1.1.1 wurde der Entwicklungsansatz verändert und das hierin geplante Vorgehen angepasst. Zusammen mit Anwendern und Codeentwicklern wurden die Anforderungen an ein neues Eingabeformat analysiert und die erforderlichen Spracheigenschaften festgelegt. Hierdurch zeigte sich klar die Notwendigkeit, die Bestandteile und Mechanismen objektorientierter Programmiersprachen (OOP) zu Grunde zu legen und so eine Eingabespezifikation für ein hierarchisch strukturierbares Eingabeformat zu erzielen. Des Weiteren sollte es Modularität und die Wiederverwendbarkeit von Definitionen unterstützen und die Erstellung übersichtlicher und einfach überprüfbarer Eingabedatensätzen fördern. Zwar bietet der in Abschnitt 4.1.1.1 geplante Einsatz eines Parser-Generators und die Implementierung einer entsprechenden C/C++-Bibliothek eine sehr große Flexibilität, um alle genannten Anforderungen an das Eingabeformat zu berücksichtigen. Allerdings ist der Aufwand zur Entwicklung einer entsprechenden, an OOP angelehnten Grammatik und der notwendigen Semantik nicht unerheblich. Des Weiteren zeigten sich in der Liste der Anforderungen sehr große Überschneidungen mit den Eigenschaften der Programmiersprache Python /PYT 22/, was die Designentscheidung nahelegte, den Python-Interpreter als dynamische Bibliothek in Eigenentwicklungen einzubetten und dadurch für das neue Eingabeformat Grammatik, Parser und Semantik-Prüfungen aus Python zu verwenden. Dadurch konnte der Entwicklungsaufwand für diese Teile umgangen werden.

Beim Einlesen von Eingabedateien müssen viele Überprüfungen vorgenommen werden, um alle enthaltenen Definitionen und deren Struktur und Zuordnungen zueinander richtig erkennen zu können. In Grammatik-basierten Parsern wird diese Aufgabe in mehrere Phasen unterteilt, wobei nach der lexikalischen (Erkennung von Keywords, Operatoren und Tokens) die syntaktische Analyse (Erkennung von Ausdrücken und Anweisungen durch Token-Reihenfolge) durchgeführt wird. Im Anschluss daran erfolgt dann meist die semantische Analyse, welche u.a. die logische Reihenfolge von Definitionen und Typ-Kompatibilität der in Ausdrücken verwendeten Symbole sicherstellt. Auf dieser bereits durch den Python-Interpreter bereitgestellten Funktionalität kann für die Verarbeitung von Modelldefinitionen aufgebaut werden, wodurch die dafür erforderlichen Programmteile dann nur für den strukturellen Aufbau von Modellen und ihrer Komponenten implementiert werden müssen.

Abbildung 4.4 gibt einen Überblick über diesen Realisierungsansatz und skizziert auch die Verwendung der FDE-basierten Plugin-Schnittstelle /FDE 21/ für die Anbindung an

den Rechencode, wie hier an ATHLET. Neben den grundlegenden Semantik-Prüfungen, die bereits durch den Python-Interpreter sichergestellt werden, müssen Modelldefinitionen für Simulationen erweiterten Überprüfungen standhalten (*checkModel()*), um in Rechenläufen der AC²-Codes verwendet werden zu können. Durch die Verwendung des Python-Interpreters erlaubt dieser Ansatz, diese Modell-Überprüfungen direkt in Form von Python-Klassen zu implementieren und sämtliche für den Aufbau von Modellspezifikationen erforderlichen Definitionen als Python-Package (Modell-RT) zusammenzufassen. Die hierin enthaltenen Routinen übernehmen auch den Daten-Transfer (*transfer()*) zurück in den Rechencode. Dazu müssen hier die passenden Schnittstellen des Codes angesprochen werden, um die ermittelten Modelldaten an die Daten-Modulen des Rechencodes weiterzuleiten und den Start der Simulation einzuleiten.

Wie in der Abbildung skizziert, erfolgt nach dem Einladen des Plugins (pythonbridge.dll) automatisch das Laden des rechencodespezifischen Modell-Runtime Packages, wodurch dann im Python-Interpreter alle Definitionen bekannt sind, um die Modellspezifikation (Input-Code) einlesen (*loadInputCode()*) zu können.

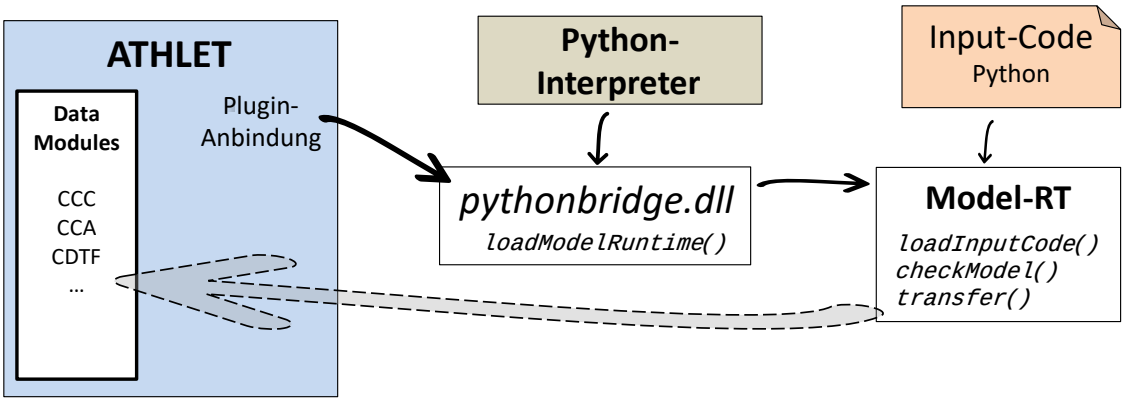

Abb. 4.4 Verwendung des Python-Interpreters durch ein ATHLET-Plugin

Durch die Einbettung des Python-Interpreters in ein ATHLET-Plugin (pythonbridge.dll) kann dieser zur Definition des neuen Eingabeformats (Input-Code) herangezogen werden und erlaubt die Implementierung aller notwendigen Modell-Überprüfungen in Python (Model-RT).

Gemäß der oben beschriebenen Konzeption wurde ein Prototyp des geplanten Eingabe-Plugins für ATHLET implementiert. Aufgabe des Plugins ist es, eine Verbindung zwischen dem in Fortran entwickelten ATHLET-Kernprogramm und den als Python-Code formulierten Eingabedateien herzustellen. Um den Python-Interpreter einfacher einladen und nutzen zu können, wurde die bisher realisierte Erweiterung nicht in Fortran, sondern in C++ implementiert. Der bisher erreichte Stand erlaubt mittlerweile das Einladen des

Python-Interpreters, das Laden eines Modell-Runtime packages und die Ausführung von Python-Statements.

Nach der grundlegenden Prototyp-Implementierung des ATHLET-Plugins wurde es hinsichtlich seiner Python Kompatibilität verbessert. Es erkennt jetzt zur Laufzeit die durch die Python Bibliothek zur Verfügung gestellten Funktionen und passt dahingehend seine Aufrufe an. Somit ist es möglich, mit einem einzigen Plugin Kompatibilität zu Python 2 und Python 3 zu gewährleisten. Das codespezifische Python-Package zur Definition der Eingabestrukturen und für den Datentransfer der Modelldaten zurück nach ATHLET konnte in der Projektlaufzeit nur exemplarisch implementiert werden. Für die tatsächliche Spezifikation von Modellen ist hier weiterer Entwicklungsbedarf erforderlich. Allerdings wurde durch das Plugin *„pythonbridge“* die Grundlage geschaffen, über die auch generell die Anbindung von Python-Code an die Rechencodes ermöglicht wird. Es erlaubt damit sowohl die Erstellung von in Python entwickelten Plugins als auch die Kopplung zu anderen Python-basierten Analysewerkzeugen. Die Möglichkeiten sollten in zukünftigen Projekten ausgelotet und ausgebaut werden.

4.1.1.3 Einheitliche Handhabung durch Grammatik-basierte Verarbeitung

Neben den beiden oben beschriebenen Ansätzen über eine nativ implementierte Erweiterung der Simulationscodes (Plugin) wurde auch ein dritter Ansatz verfolgt, welcher weniger die Vereinheitlichung der Formate und mehr auf die einheitliche Handhabung verschiedener Formate abzielt. Um eine Kompatibilität unabhängig der Divergenz bestehender, oft jedoch sehr ähnlicher Formate wie ATHLET und ATHLET-CD, erreichen zu können, wurden erste Versuche zur Grammatik-basierten Textverarbeitung mit PLY (Python Lex-Yacc; /PLY 21/) durchgeführt. PLY ist eine Implementierung der Parsing Tools *„Lex und Yacc“* in Python und ermöglicht die Kombination von sogenanntem „rapid prototyping“ und der Verarbeitung komplexer Grammatiken. Hierfür wurden bereits kleine Ausschnitte aus bestehenden Input-Dateien mit Hilfe von PLY, in einen sogenannten AST (Abstract Syntax Tree) zerlegt und für eine weitere Verarbeitung zur Verfügung gestellt. Dies ermöglicht unter anderem eine Überprüfung der Richtigkeit des Eingabeformats, automatische Korrektur von Fehlern und die Analyse der zu erzeugenden Datenobjekte.

Die Planung zur Realisierung einer Grammatik-basierten Eingabeverarbeitung wurde im weiteren Projektverlauf verfeinert und zusätzliche Tests mit aktuell verfügbaren Pythonpackages durchgeführt. Hier wurden neben PLY auch andere vergleichbare packages

bzgl. ihrer Anwendbarkeit, Reifegrad, Stabilität und Lizenzbedingungen untersucht. Die vielen Möglichkeiten, die sich dadurch für die Implementierung und Wartung von Lösungen zur Vereinheitlichung des Eingabeformats zeigten, sowie eine Abwägung von Aufwand und Nutzen legten die Schlussfolgerung nahe, den bisherigen Ansatz über ein Plugin für die Simulationscodes erstmal abzuschließen und aktuell nicht weiter auszubauen. Das Ziel, einen einheitlichen Umgang mit den unterschiedlichen Eingabeformaten, wie deren Generierung und Verarbeitung zu erreichen, erscheint durch eine Grammatik-basierte Verarbeitung deutlich einfacher realisierbar. Dieser Ansatz konnte zwar im Rahmen dieses Projekts nicht weiterverfolgt werden, soll aber Gegenstand des nachfolgenden Projekts sein.

4.1.2 Schnittstellenanpassungen in Rechencodes

Neben ATHLET, dem bisher zentralen Simulationscode, sollten auch die Erweiterungscodes des AC²-Softwarepakets durch Online-Simulationen und interaktiver Ergebnisanalyse mit Hilfe der GRAMOVIS-Werkzeuge unterstützt werden. Für die Sicherstellung der dazu notwendigen Schnittstellen zur Steuerung und zum Datenaustausch sowie der einheitlichen Ausgabe der Ergebnisdaten in HDF5 waren hier vielerlei Anpassungen in den einzelnen Rechencodes erforderlich. Abbildung 4.5 gibt am Beispiel einer ATLASneo Online-Simulation einen Überblick über die allgemeine Funktionsweise und das Zusammenspiel der einzelnen Komponenten. Zur Simulation eines gegebenen Modells wird hierbei ATHLET als zentraler Rechencode gestartet. Sowohl die Eingabedatei als auch die beim Start angegebenen Parameter erlauben ATHLET, die für die Simulation erforderlichen Erweiterungscodes zu bestimmen (hier: Core Degradation und QUABOX/CUBBOX (QC)) und als Plugins nachzuladen.

Durch die Nutzung von ATHLET als Hauptcode und die Einbindung der anderen Rechencodes als Plugin konnten die Prozeduren für Starts und Restarts von Simulationen vereinheitlicht werden. Allerdings ist für die interaktive Anwendung neben dem zentralen Simulationsstart auch eine, in allen Codeteilen einheitliche Fehlerbehandlung entscheidend. Besonders Fortran-Codes, die für eigenständige Rechenläufe entwickelt wurden, verwenden sehr oft *STOP*-Statements, um bei aufgetretenen Laufzeitfehlern das Programm zu beenden. Da diese Behandlung allerdings nicht mehr zum aufrufenden Programmteil zurückkehrt, sondern das Betriebssystem anweist, gleich den gesamten Prozess zu beenden, ist diese Art der Fehlerbehandlung für eine interaktive Betriebsart durch eine Benutzeroberfläche absolut ungeeignet. Daher mussten in den Rechencodes

alle *STOP*-Statements entfernt und durch eine ausnahmenbasierte Behandlung (Exceptions) ersetzt werden.

Neben der Plugin-Schnittstelle und dem Exceptionhandling wurde auch die Zugriffsmöglichkeiten auf den Simulationszustand unter Verwendung der externen Kopplungs-Bibliothek FDE /FDE 21/ realisiert. Durch Einbindung dieser Bibliothek wird es Rechencodes möglich, die Variablen und Datenstrukturen zur Speicherung ihres Simulationszustands für externen Zugriff freizugeben. Dabei werden die Variablen aller relevanten Simulationsgrößen unter ihrem symbolischen Namen in sog. HashMaps (State-Scopes) eingetragen und bilden so eine Art Inhaltsverzeichnis aller zugreifbaren Daten.

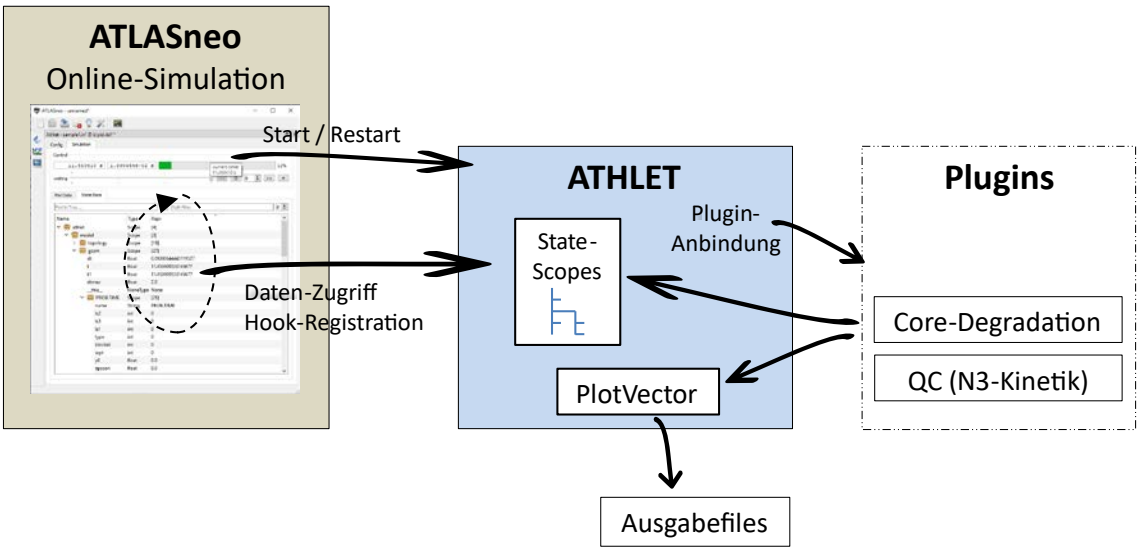

Abb. 4.5 Online-Simulation in ATLASneo

Die Online-Simulation in ATLASneo erfolgt über ein Funktionsmodul, welches das Kontrollprogramm (Controller) für die Einbindung und den Aufruf des ATHLET-Codes sowie die Steuerung und den interaktiven Zugriff auf die Daten des Simulationslaufs ermöglicht.

Darüber hinaus können an signifikanten Stellen im Code Synchronisationspunkte (Hooks) definiert werden, um externe Programmteile über Ereignisse, wie das Erreichen einer bestimmten Stelle im Rechenlauf zu benachrichtigen. Ein externes Kontrollprogramm (Controller) kann an diesen Synchronisationspunkten Kontrollfunktionen (Callback-Funktionen) registrieren, durch die der Rechenlauf vorübergehend angehalten wird, um z. B. Datenzugriffe auf exportierte Simulationsgrößen vorzunehmen. Die Hooks, die bereits vor dem Simulationsstart unter ihrem, möglichst deskriptiven Namen sichtbar sind, erlauben es dem externen Kontrollprogramm auf bestimmte Ereignisse, wie z. B. das erfolgreiche Einlesen der Eingabedatei am Hook „`INPUT_DONE`“ oder den Beginn eines neuen Zeitschritts in der Simulation („`TIMESTEP_BEGIN`“) zu reagieren.

Somit wird es möglich, Simulationen von außerhalb des Rechencodes kontrolliert ablaufen zu lassen, ihren Fortschritt zu überwachen und ggf. durch gezielte Eingriffe zu steuern.

Wie im Diagramm ersichtlich, nutzen auch die als Plugins implementierten Erweiterungscodes diese Mechanismen, um z. B. die Berechnung ihrer physikalischen Modelle abhängig vom Simulationszustand des Hauptcodes ATHLET und an dessen Ablauf gekoppelt durchzuführen. Des Weiteren können die Plugins die im Hauptcode implementierten Routinen und Datenstrukturen zur Ablage von Rechenergebnissen mitbenutzen. Das wichtigste Beispiel ist hier sicherlich der Ergebnisvektor (PlotVector), in dem alle für die Datenausgabe relevanten Simulationsgrößen gesammelt werden, bevor diese bei der Ausgabe eines Zeitschritts in die Ausgabedatei geschrieben werden können. Durch diese Anbindung profitieren die Plugins von den für ATHLET implementierten Ausgabeformaten, wie HDF5, und lassen hierüber ihre Ergebnisse in die zentrale Ausgabedatei mitausgeben.

Wie in Abbildung 4.5 dargestellt, arbeitet auch das in ATLASneo enthaltene Funktionsmodul für Online-Simulationen nach diesem Prinzip und implementiert alle Kontrollfunktionen, die für eine interaktive Steuerung von Simulationen benötigt werden. Sowohl der Hauptcode ATHLET als auch die als Plugins implementierten Erweiterungen mussten für diese Betriebsart vorbereitet werden. So war es, neben den Anpassungen für den Datenzugriff auch nötig, durch geeignete Absicherungen im Code zu gewährleisten, dass alle als Plugin eingesetzten Erweiterungscodes auch mit abrupten Änderungen des Programmablaufs z. B. Rücksprüngen in der Simulationszeit, extern ausgelösten Abbrüchen und Neustarts von Simulationen umgehen können. Darüber hinausgehende Arbeiten und spezielle Anpassungen in den einzelnen Rechencodes werden in den folgenden Unterabschnitten beschrieben.

4.1.2.1 Anpassungen in ATHLET/-CD

Die oben beschriebene interaktive Betriebsart, während der Online-Simulation stellt hohe Ansprüche an die Stabilität der Rechencodes. Besonders durch die abrupten Sprünge in der Durchführung von Rechenläufen oder unstetige Änderungen des Simulationszustands, die bei Online-Simulationen und -Analysen durch Benutzereingriffe immer wieder vorkommen, zeigten sich oft Probleme in den Rechencodes, die zu Abbrüchen der Simulation oder Abstürzen des gesamten ATLASneo-Prozesses führten. Die Ursachen hierfür lagen häufig in der Fortran-Codebasis deren Programmierung

schlichtweg nicht für diese Betriebsart ausgelegt war und z. B. durch fehlende explizite Initialisierung interner Variablen nicht mit wiederholter Ausführung zurechtkam. Mittlerweile können Fortran-Compiler die Entwicklung zwar sehr gut unterstützen und helfen, diese Fehler durch automatische Überprüfungen und Type-Checks zu vermeiden. Allerdings erfordert die Verwendung dieser zusätzlichen Compiler-Optionen meist die Einhaltung bestimmter Quellcode-Standards, die für ältere Codes für gewöhnlich nicht gewährleistet sind und die Modernisierungen notwendig machen.

Dadurch, dass sich bei der Anwendung von ATHLET-CD in Online-Simulationen immer wieder die oben genannten Stabilitätsprobleme zeigten, mussten für die zielführende Weiterentwicklung die im Folgenden beschriebenen Arbeiten zur Code-Modernisierung unternommen werden, wobei die Umstellung der gesamten ATHLET-CD Codebasis vom Fortran 77 „fixed-Format“ auf F90-Dateien im „free-Format“ ein sehr wichtiger Schritt zur Erleichterung der Weiterentwicklung war. In einem zweiten Schritt wurden alle Variablennamen im Code einer expliziten Typendeklaration zugeordnet, um die vielfältigen Möglichkeiten der Typüberprüfung des Compilers nutzen zu können. Darüber hinaus wurden alle, in neuen Compilern nicht mehr unterstützten Schleifenausdrücke aktualisiert und arithmetische *if*-Anweisungen in reguläre *„if else“*-Anweisungen abgeändert. Bei jedem Schritt wurde verifiziert, dass die Änderungen keine Unterschiede in Ablaufverhalten und Rechenergebnissen der Codes auslösten. Die notwendigen Schritte wurden mit Hilfe eines Python-Skripts implementiert und werden im Folgenden detailliert erläutert.

Umstellung des Codeformats

Die erste Aufgabe bestand darin, die stark einschränkende Syntax von Fortran 77, das so genannte Fixed-Format (Endung *„.f“*), zu aktualisieren. Dieses Format beschränkt den nutzbaren Bereich von Code-Dateien auf eine maximale Anzahl an Zeichen pro Zeile (meist 72) und erzwingt dadurch viele Zeilenumbrüche, worunter die Code-Lesbarkeit, besonders bei der Formulierung komplexer Bedingungen, sehr leidet. Auch haben hier die ersten 6 Zeichen jeder Zeile eine feste Bedeutung und können nicht für Programm-Statements verwendet werden. Um die zukünftige Entwicklung von diesen Restriktionen zu befreien, musste die Syntax aller ATHLET-CD-Dateien auf den moderneren Fortran-90-Standard aktualisiert werden, welcher das „Free-Format“ (Endung *„.f90“*) empfiehlt. Dieses neuere Format hebt die oben beschriebenen Beschränkungen auf und erlaubt den Code gemäß modernem Standard zu formatieren, welche die Lesbarkeit durch Einrückungen und viele weitere Maßnahmen steigern.

Vorgehensweise und Maßnahmen zur Verifikation

Die Hauptanforderung an die Vorgehensweise für die Aktualisierung des Codes war, dass diese rein syntaktisch sein sollte und dass es durch die Änderungen zu keinen Unterschieden in Programmablauf oder Rechenergebnissen kommen durfte. Da in der Praxis eine hundertprozentige Testabdeckung von Code kaum gewährleistet werden kann und diese auch sehr aufwendige Testläufe erfordert, wurde für die hier vorgenommenen Anpassungen ein anderes Vorgehen gewählt: Der direkte Vergleich kompilierter Routinen durch disassemblierten Code. Der Assemblercode enthält die low-level Anweisungen, die der Prozessor beim Programmablauf ausführt. Daher war die Maßgabe, dass sich bei der reinen Umstellung des Dateiformats die effektiven Anweisungen im Assemblercode unverändert blieben, wodurch Fehler sehr schnell aufgezeigt werden konnten.

Um dieses Vorgehen auf Praxistauglichkeit zu testen und um ein Verständnis für die verschiedenen spezifischen Änderungen zu erlangen, wurden die Konvertierung zu Beginn für jede Datei von Hand vorgenommen. Da es sich jedoch um knapp 600 Dateien handelte, von denen einige Dateien tausende Codezeilen enthalten, sollte dieser Konvertierungsprozess durch ein Python-Skript größtenteils automatisiert werden. Neben der schnelleren Bearbeitung konnten durch den automatisierten Ablauf menschliche Fehler in einzelnen Dateien ausgeschlossen werden. Auch die Überprüfung durch den oben genannten Vergleich von Assemblercode konnte in den automatischen Ablauf integriert werden, wodurch der Testaufwand in der Verifikation drastisch reduziert werden konnte.

Das für die Teilautomatisierung entwickelte Python-Skript basiert auf der Funktionalität verschiedener Linux Kommandozeilen-Werkzeuge, die unter Windows nicht direkt verfügbar sind und setzte daher die Installation des Ubuntu-Subsystem für Windows (WSL) voraus. Das Skript verwendet im Wesentlichen die folgenden Werkzeuge und automatisiert deren Aufrufe:

- **findent:** korrigiert die Einrückung von Fortran Quelldateien und konvertiert fixed-Format in free-Format.
- **cmake/make:** übersetzt die Quellcodedateien unter Verwendung des eingesetzten Fortran Compilers. Unter WSL wurde **gfortran** (GNU Fortran) eingesetzt.

- **objdump:** ermittelt Informationen aus kompilierten Codemodulen (object files) und erzeugt den Assemblercode der enthaltenen Programmteile (Routinen).
- **diff:** vergleicht zwei Textdateien, wie Quelldateien, und listet die Unterschiede blockweise auf.

Der folgende Pseudo-Code zeigt die grundlegenden Arbeitsschritte und deren Reihenfolge, in der das Skript die Konvertierung durchführt:

```
1) make *.f > *.obj          #< kompilieren aller f-Quelldateien zu object files
2) objdump *.obj > *.1.asm   #< disassemblieren aller object files und speichern als *.asm
3) findent *.f > *.f90       #< konvertieren der Quelldateien nach Free-Format
4) make *.f90 > *.obj        # kompilieren aller f90-Quelldateien zu object files
5) objdump *.obj > *.2.asm   #< disassemblieren aller object files und speichern als *.asm
6) diff *.1.asm *.2.asm      #< paarweises Vergleichen der asm-Dateien
```

Abb. 4.6 Arbeitsschritte zur Konvertierung von Fortran-Quellcode-Dateien

Das Ergebnis dieser Schritte ist die Ausgabe der Unterschiede, die sich im Assemblercode der einzelnen Module ergeben. Idealerweise sollten diese natürlich identisch sein, in der Praxis zeigten sich allerdings auch bei korrekter Umstellung einige Unterschiede, wie die Verschiebung von Startadressen oder Offsets, welche in der Sensitivität dieser Methode oder in der internen Arbeitsweise des Fortran Compilers begründet liegen. Daher mussten die Quelldateien, für die sich Unterschiede in den Assemblercodes ergaben, genau überprüft werden, um nachvollziehen zu können, ob diese durch eine fehlerhafte Konvertierung oder durch ein übersensibles Verhalten des Compilers ausgelöst wurden.

In der Phase der Prüfung auf Assemblercode-Unterschiede erzeugten einige Fortran-Dateien nach der Konvertierung in das Free-Format leicht unterschiedliche Assemblercodes. Es war auffällig, dass diese Unterschiede bei vielen Dateien der gleichen Art waren und sich immer auf einige wenige Zeilen beschränkten. Bei genauer Betrachtung der Änderungen in den Fortran-Dateien wurde deutlich, dass diese auf eine unterschiedliche Anzahl von Leerzeichen zwischen den Argumenten von Funktionsaufrufen zurückzuführen waren. Wurde zum Beispiel eine Funktion vor der Konvertierung durch *„function(a, b)“* aufgerufen, hatte der Konverter diesen Aufruf zu *„function(a,b)“* geändert, und es ergab sich ein Unterschied in den Offsets der Assemblercodes. Diese Unterschiede ändern die Funktionalität des Programms nicht und können ignoriert

werden. Um jedoch zu bestätigen, dass es keinen anderen Grund für den beobachteten Unterschied im Assemblercode gab, wurde diese Überprüfung für jede verdächtige Datei durchgeführt und die Ursache für die Unterschiede beseitigt.

Nicht-implizite Typdeklarationen

Erweiterte Fehlerüberprüfungen der Fortran-Compiler und die Optimierung bei der Übersetzung des Quellcodes erfordern meist explizite Typdeklarationen für Variablen und Prozeduren. Gerade in der Entwicklung komplexer Rechencodes ist es wichtig diese Funktionalität der Compiler nutzen zu können, damit Fehler, wie falsche Zuweisungen oder Argumentübergaben bei Prozeduraufrufen vermieden werden. Auch kann dadurch die Qualität und Wartbarkeit des Quellcodes deutlich gesteigert werden, was die Weiterentwicklung der Rechencodes sehr erleichtert. Hieraus ergab sich die Aufgabenstellung, bisher im Code von ATHLET-CD fehlende Typdeklarationen nachzutragen.

Da die Codebasis von ATHLET-CD sehr groß ist, war es auch hier nicht praktikabel, jede einzelne Datei manuell zu durchsuchen und tausende von Codezeilen auf nicht typisierte Variablen zu überprüfen. Daher wurde ein weiteres teilautomatisiertes Skript entwickelt, um nicht deklarierte Variablennamen in Prozeduren zu erkennen, den Typ der Variablen zu bestimmen und eine Tabelle mit den Typdeklarationen zu erstellen. Eine Prozedur (*`function`* oder *`subroutine`*), die nicht deklarierte Variablen beinhaltet, nutzt die sog. Implizite Deklaration, die den Typ über den Anfangsbuchstaben des Variablennamens zuordnet. Unter der Annahme, alles andere sei vom Typ Integer, wird für die Festlegung von Fließkommavariablen doppelter Genauigkeit meist eine Angabe wie *„IMPLICIT DOUBLE (A-H, O-Z)“* verwendet. Prozeduren, welche nicht deklarierte Variablen nutzten, konnten also über diese Art von Statements identifiziert werden. Hierfür musste nur ein regulärer Ausdruck gefunden werden, der syntaktisch richtige *„IMPLICIT“*-Deklaration erkennen kann.

Um das Aufspüren von Variablennamen, ihre korrekte Deklaration und schließlich das Hinzufügen der Typdeklarationen in eine Prozedur zu ermöglichen, wurde eine Python-Klasse „`TypeTemplate`“ erstellt. Diese Klasse verwaltet eine Liste von Variablennamen und kümmert sich intern darum, den Typ der bereitgestellten Variablennamen zu bestimmen. Außerdem liefert diese Klasse eine Ausgabe von Fortran-Code, der in der Quelldatei direkt in den Deklarationsteil der entsprechenden Prozedur eingefügt werden kann, und deklariert somit alle Variablen.

Der folgende Pseudo-Code zeigt die grundlegenden Arbeitsschritte und deren Reihenfolge, in der das Skript die Bearbeitung der Quelldateien und die Deklaration durchführt. Auch der hierzu entworfene Ablauf verwendet die schon bei der Format-Konvertierung angewendeten Maßnahmen zur Verifikation durch Disassemblierung und basiert auf der Annahme, dass explizit angegebene Typdeklarationen beim Vergleich des Assemblercodes zur impliziten Deklaration keine Änderungen zur Folge haben dürfen.

```
1)  make *.f90 > *.obj          #< kompilieren aller f90-Quelldateien zu object files
2)  objdump *.obj > *.1.asm     #< disassemblieren aller object files als *.asm
3)  for f in implicit(*.f90):   #< für alle Quelldateien mit impliziter Deklaration...
4)    „IMPLICIT NONE“ > f       #< implizite Deklaration in Quelldatei deaktivieren
5)    make f |                  #< kompilieren der aktuellen Quelldatei,
        find „undeclared“ |        auffangen von Fehlermeldungen undeklarierter Variablen
        TypeTemplate.add()         und merken der Variablen in TypeTemplate
6)    TypeTemplate > f          #< Deklaration für Variablen zu Quelldatei hinzufügen
7)    make f > f.obj            #< kompilieren der aktuellen Quelldateien
8)    objdump f.obj > f.2.asm   #< disassemblieren des aktuellen object files als .asm
9)    diff f.1.asm f.2.asm |    #< Vergleichen der asm-Dateien
10)     wait                       und bei Unterschied auf Entwickler-Bestätigung warten
```

Abb. 4.7 Arbeitsschritte zum Nachtragen von Typdeklarationen in Fortran-Code

Wie im Pseudo-Code dargestellt, kann die Typdeklaration nicht vollständig automatisiert ablaufen und erfordert manuelle Schritte des Entwicklers. Nachdem die Typdeklaration in 6) einer Quelldatei hinzugefügt und diese kompiliert 7) und disassembliert 8) wurde, können im Vergleich 9) Unterschiede auftauchen, welche durch das sensible Verhalten des Fortran-Compilers ausgelöst werden, die aber keine Änderungen im Codeverhalten bedeuten. So zeigt bereits die veränderte Zeilenanzahl einer Quelldatei Auswirkungen im Assemblercode, woraufhin in der Code-Bearbeitung durch das Python-Skript einige Vorkehrungen getroffen wurden, um diese falschen Unterschiede, wie hier durch konstant halten der Zeilenanzahl mit Hilfe von Kommentarzeilen zu vermeiden. Um dem Entwickler in 10) die Überprüfung zu erlauben, wartet das Skript auf einen Tastendruck, bevor es mit der Bearbeitung der nächsten Quelldatei fortfährt.

Neben der Ergänzung von Typdeklarationen wurde das Python-Skript auch um die Ersetzung von mittlerweile nicht mehr unterstützten Kontrollstrukturen, wie DO-Schleifen mit End-Sprungmarken und „arithmetischen IF“-Anweisungen, erweitert. Auch diese Modernisierungsmaßnahmen konnten durch die automatische Code-Bearbeitung mit Hilfe

von regulären Ausdrücken implementiert werden und verwenden das oben beschriebene Verfahren zur Verifikation durch Disassemblierung, um ungewollte Änderungen im Rechencode zu verhindern.

Alle für die Modernisierung entwickelten Hilfsskripte konnten die Code-Umstellung des Hauptentwicklungszweiges sehr beschleunigen und durch die Verifikationsmaßnahmen deutlich sicherer machen. Außerdem konnten diese auch bei der Integration alter, noch nicht modernisierter Entwicklungs-Branches anderer Codes schon gute Dienste leisten.

4.1.2.2 Anpassungen in COCOSYS

COCOSYS, das Containment Code System, simuliert Vorgänge im Containment und angrenzenden Gebäuden und wird typischerweise mit ATHLET und ATHLET-CD gekoppelt betrieben. Da COCOSYS als zentraler Koordinator agiert und wiederum andere Rechencodes, wie auch ATHLET, steuert, muss es in ATLASneo als zu startendes Hauptprogramm vorgesehen werden. Um diese Art von Simulationen auch überwachen und steuern zu können, sind die dafür notwendigen Mechanismen zur Kommunikation, zum Datenaustausch und zur einheitlichen Ablage von Ergebnisdaten erforderlich.

Durch COCOSYS stellt AC² weitere Simulationscodes (RALOC, AFP, CCI) bereit, die über MPI gekoppelt sind und über den COCOSYS-Main-Driver (COCOSYS-Treiber) synchronisiert werden. Im Rahmen des Projekts wurde an einem einfachen Verfahren gearbeitet und getestet, wie und in welchem Ausmaß ATLASneo genutzt werden kann, um den COCOSYS-Treiber zu starten, interaktiv zu steuern und die Ergebnisdaten anzuzeigen. Da dieser aber zwingend in einem MPI-Kontext (*mpiexec*) ausgeführt werden muss, und hierüber auch nur mit Prozessen kommunizieren kann, die Teil dieses Kontextes sind, muss bereits ATLASneo in dieser Betriebsart gestartet werden.

Der COCOSYS-Treiber lässt bislang noch keine interaktive Steuerung zu, welche eine direkte Online-Simulation über ATLASneo erlauben würde. Im getesteten Verfahren wurde COCOSYS daher als Subprozess aus der ATLASneo-Konsole gestartet, welcher dann auf die Initialisierung der ATHLET-Simulation wartet. Diese erfolgt, sobald die ATHLET-DLL vom Funktionsmodul in ATLASneo geladen und gestartet wird. Danach ist aus ATLASneo ein interaktiver Zugriff auf ATHLET-Simulationsgrößen möglich, nicht jedoch auf die der COCOSYS-Rechencodes. Somit ist ein unterstütztes Starten von COCOSYS-Simulationen zwar möglich, allerdings können diese noch nicht schrittweise angesteuert und nur teilweise überwacht werden. Dennoch ist die Auswertung der

Ergebnisdaten durch Nutzung der nach HDF5 konvertierten Ausgabedateien uneingeschränkt möglich. Die dafür notwendige HDF5-kompatible Ablagestruktur, die auch die in COCOSYS auftretenden Ergebnisvektoren veränderlicher Länge abbilden kann, wurde im Rahmen der Entwicklung des Datenkonverters erarbeitet und ist unter Abschnitt 4.4.2. beschrieben.

4.1.2.3 Anpassungen in QUABOX/CUBBOX

Der von ATHLET unabhängig entwickelte 3D-Neutronenkinetik-Code berechnet die 3D-Leistungsverteilung im Reaktorkern und wurde von den GRAMOVIS-Werkzeugen bis dato nicht unterstützt. Obwohl QUABOX/CUBBOX (QC) nicht mehr weiterentwickelt wird, findet dieser in einzelnen Projekten oder Referenzsimulationen noch immer Anwendung, wodurch es sich anbot diesen Code für die Einbindung in GRAMOVIS vorzusehen und somit das Konzept zur Integration externer Rechencodes auf Praxistauglichkeit zu überprüfen. So konnten durch die exemplarische Integration von QC alle wichtigen GRAMOVIS-Komponenten (Online-Simulation, HDF5-basierte Datenausgabe, Daten-Analyse in ATLASneo) auf die Anforderungen von Neutronenkinetik-Codes vorbereitet werden. Die dabei erarbeiteten Schnittstellen für die Kopplung von QC mit ATHLET können bei Bedarf weiter verallgemeinert werden, damit diese auch zukünftig entwickelten Neutronenkinetik-Codes, wie DYN3D, zur Verfügung stehen.

Die im Projekt unternommenen Anpassungen ermöglichen jetzt QUABOX/CUBBOX als ATHLET-Plugin zu erstellen und mit allen kompatiblen ATHLET-Versionen in gekoppelten Simulationen anzuwenden (vgl. Abb. 4.5). Durch die Nutzung der FDE-basierten Schnittstellen zur Datenüberwachung und Ablaufsteuerung (vgl. /FDE 21/) kann QC zusammen mit ATHLET auch als Online-Simulation in ATLASneo gestartet, überwacht und gesteuert werden. Als Plugin erweitert QC den zugreifbaren Zustand des Simulationsprozesses um alle Variablen, welche für den Online-Zugriff relevant sind, und nutzt über die Routinen von ATHLET auch dessen Ergebnisvektor, um seine Simulationsergebnisse in einheitlicher Art abzulegen und im Ausgabeformat von ATHLET mit ausgeben zu lassen.

Die Funktionstüchtigkeit des ATHLET-Plugins für QC wurde anhand eines „Open-Core“ DWR-Modells getestet. Diese Kopplung von ATHLET und QC konnte in ATLASneo durch das Funktionsmodul für Online-Simulationen gestartet werden und erlaubte den interaktiven Zugriff auf die internen QC-Simulationsgrößen. Bereits während der Simulation konnten die Plot-Möglichkeiten zur Online-Auswertung von Ergebnisvariablen

genutzt werden und auch der gesamte Rechenlauf reproduzierte die erwarteten Ergebnisse der Neutronenkinetik in der HDF5-Ausgabedatei von ATHLET.

Die angedachten Weiterentwicklungen zur erleichterten Bereitstellung der Eingabedaten sowie die Ergebnis-Visualisierung der Neutronenkinetik-Codes konnte im aktuellen Projekt nicht mehr angegangen werden. Diese Funktionalitäten sollten innerhalb eines eigenständigen Funktionsmoduls von ATLASneo realisiert werden, welches die Parameter verschiedener Neutronenkinetik-Codes (z. B. QC, FENNECS, PARCS, DYN3D) sowohl in 2D als auch in 3D visualisieren kann. Dennoch wurden durch die geleisteten Entwicklungen des ATLASneo-Anwendungsrahmens sowie die zur Visualisierung Dynamischer Bildgeometrie (vgl. Abschnitt 2.2.4) die Grundsteine gelegt, diese Funktionalität im Rahmen weiterer Projekte zu realisieren.

4.2 AP 2: Modellierung und Input-Generierung

Als eine Konsequenz aus der geplanten Unterstützung der im AC²-Softwarepaket enthaltenen Rechencodes (ATHLET, ATHLET-CD, COCOSYS) ergab sich die Notwendigkeit das aus AGM und ATM bestehende Modellierungssystem AIM umzubenennen. Vorher stand dieses Akronym für „ATHLET Input Modeller“, was die Anwendung dann durch die erweiterte Aufgabenstellung nicht mehr ausreichend charakterisierte. Um den hinzukommenden Rechencodes Rechnung zu tragen, wird das Modellierungssystem seither unter dem Namen ADM („AC² Design Modeller“) geführt. Dieser Sachverhalt ist in Abbildung 4.6 graphisch dargestellt.

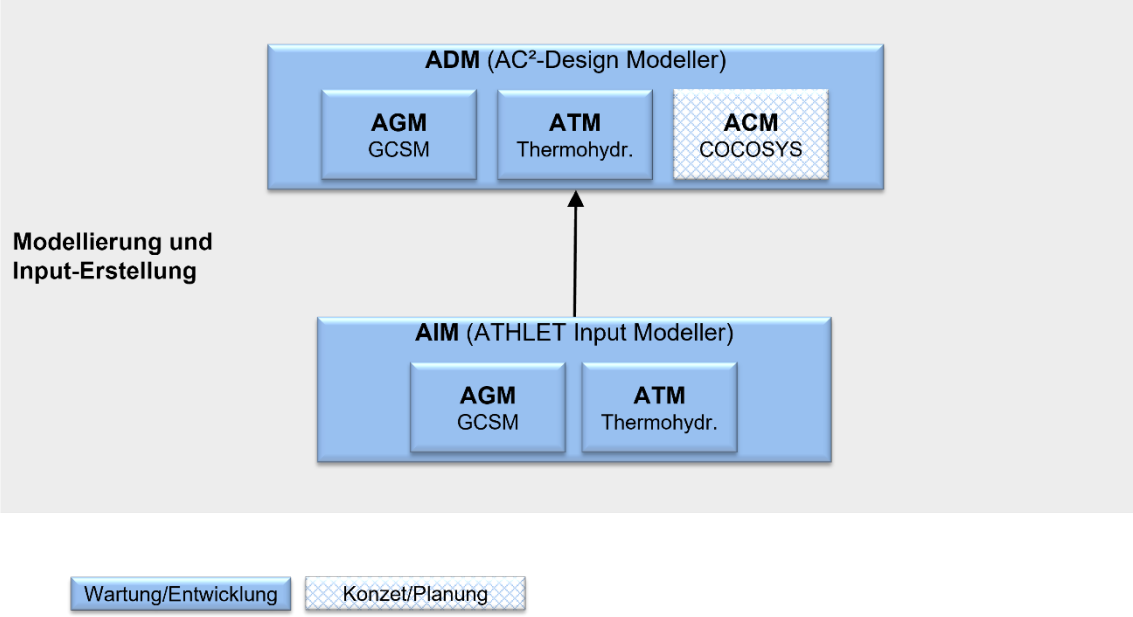

Abb. 4.8 Darstellung der Namenskonvention des AC² Design Modeller (ADM)

Die geplante Unterstützung der im AC²-Softwarepaket enthaltenen Rechencodes (ATHLET, ATHLET-CD, COCOSYS) erforderte die Umbenennung des aus AGM und ATM bestehende Modellierungssystem ATHLET Input Modeller (AIM) zu ADM.

Die Bereitschaft, ADM für die Input-Erstellung zu benutzen, kann vor allem dadurch gesteigert werden, dass auch bereits bestehende Daten, wie Eingabedatensätze für ATHLET, weiterverarbeitet werden können. Da die Daten für die Modelle in den Rechencodes jedoch große Unterschiede aufweisen können, war für einige Modelle die Entwicklung spezifischer Importfunktionen nötig, wodurch sich ein sehr hoher, über das erwartete Maß hinausgehender, Entwicklungsaufwand ergab.

4.2.1 Integration neuer Basissoftware

Zur Realisierung von ADM wird als Basissoftware „SimInTech“ /SIT 21/ verwendet, welches ein generelles System zur Modellierung und Simulation, ähnlich zum Beispiel zu „Simulink“ /MAT 22/. Der Source Code von SimInTech liegt in einem Repository vollständig vor. Ein Supportvertrag sichert die Bereitstellung aktueller Erweiterungen und Verbesserungen.

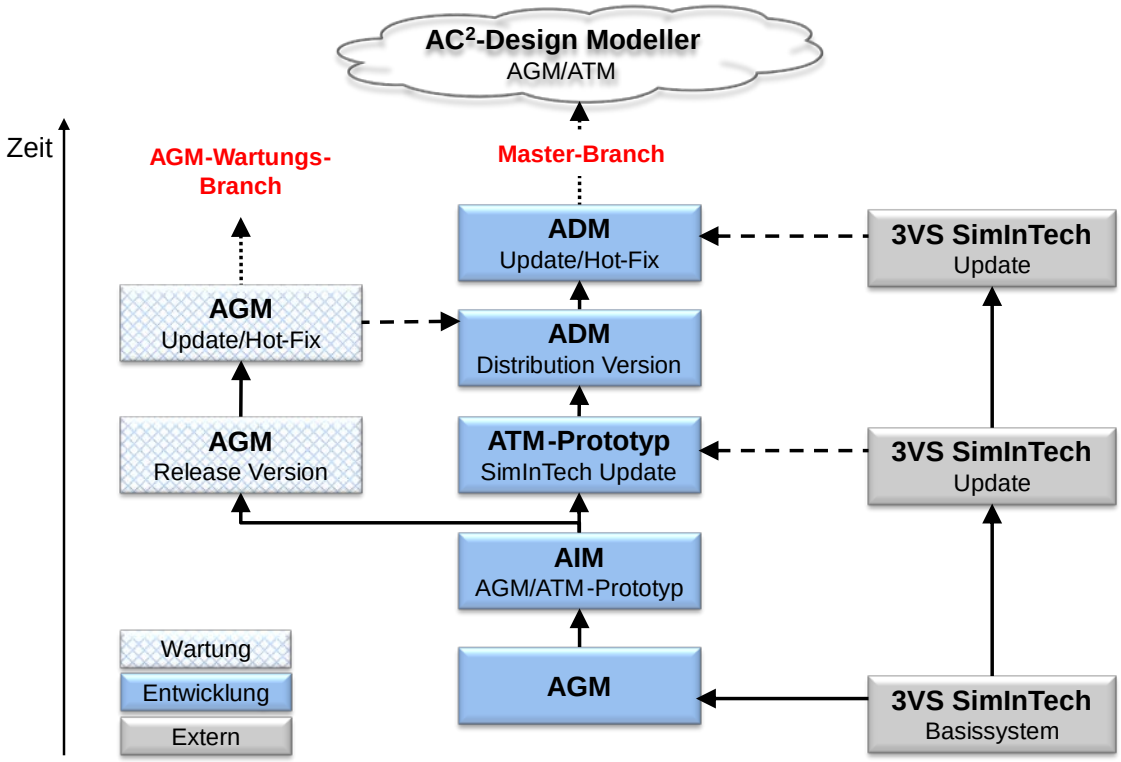

Abb. 4.9 Darstellung der ADM-Repository-Hierarchie

Die externe Entwicklungslinie vom Basissystem SimInTech des Herstellers 3VServices ist durch graue Kästchen dargestellt. Die GRS-interne Entwicklungslinie von ADM ist durch blaue Kästchen gekennzeichnet. Die gestrichelten Linien symbolisieren, die durch „Cherry-Picks“ erstellten „Merge“-Versionen.

Während der Laufzeit des Vorhabens wurde vom Hersteller 3VServices zweimal ein größeres Update von SimInTech zur Verfügung gestellt, die u.a. größere Änderungen an der Bedienoberfläche enthielten. Ein erheblicher Aufwand entstand bei der Übernahme dieser Änderungen im Wesentlichen durch die Integration („Merge“) der für den Master-Branch. Nachdem die Voraussetzungen für einen automatischen Merge durch die Versionsverwaltung (Git) nicht gegeben waren, mussten die nötigen Erweiterungen des Quellcodes durch „Cherry-Picks“ übernommen werden. Nach der Integration der neuen Version der Basissoftware SimInTech wurde deshalb die Funktionalität der Applikationskomponenten von ATM verifiziert, um eine ausreichende Testabdeckung zu gewährleisten. Nachdem der im Wartungs-Branch gewartete AGM bereits von den Anwendern genutzt wird, wurde eine generelle Überholung und Integration des Basissystems abgewogen und wegen den historisch bedingten Abhängigkeiten zum früheren Basissystem (Fortran-DLLs etc.) für dieses Vorhaben als zu umfangreich eingestuft. Für ATM erfolgten die Tests hauptsächlich entwicklerseitig, da dessen Komponenten noch kaum in praktischer Anwendung sind. Durch die Funktionstests zeigten sich auch unter

Verwendung des aktuellen Basissystems einige Probleme, welche in der zukünftigen Entwicklung für die Modellierung und Input-Generierung eingeplant werden müssen. Der Aufwand das Basissystem umzustellen, war dennoch gerechtfertigt, da im Master-Branch nur auf diesem Weg von der kontinuierlichen Weiterentwicklung von SimInTech profitiert und der weitere Support durch den Hersteller sichergestellt werden kann.

4.2.2 Überarbeitung der Benutzeroberfläche

Neben den programmtechnischen Änderungen mussten zusätzlich die Benutzeroberfläche und einige Algorithmen in ATM überarbeitet werden, da einige der Funktionen nur eingeschränkt verwendbar oder fehlerhaft waren, wie z. B.

- Spezielle Editoren für die einzelnen Properties
- Vererbte Daten (Properties) von einem Masterobjekt
- Auswahl der verbundenen TFOs in einem Cross Connection Object (CCO)
- Generierung von eindeutigen Objektnamen
- Implementierung von nicht editierbaren Properties (read only)

4.2.2.1 Spezielle Editoren

Das Basissystem SimInTech enthält keine spezifische Möglichkeit, den Properties von Objekten einen spezialisierten Editor hinzuzufügen. Diese Funktion ist für die Eingabeerstellung in ADM aber wünschenswert, da der Standard-Texteditor häufig zu generell ist, um die gewünschte Funktionalität, z. B. das Editieren einer Matrix oder das Auswählen eines Master-Objekts, bereitzustellen. Deshalb wurden konzeptionelle Methoden implementiert, die es ermöglichen, für bestimmte Objekt-Properties spezielle Editoren zu verknüpfen.

Einzelne Objekt-Properties können zum Beispiel über ihren Namen mit einem Editor Aufruf verknüpft werden. Dazu wird bei der Initialisierung der Objekte die Funktion „`RegisterExpressionEditorFor`“ benutzt.

Für Properties des Datentyps „`Matrix`“ kann ein genereller Matrix Editor (siehe Abb. 4.8) verknüpft werden, wenn der Name der Property eine spezielle Syntax benutzt, z. B. `GEOMETRY(matrix_of_double,SG0,DC0,CIRC0,SCROS0).`

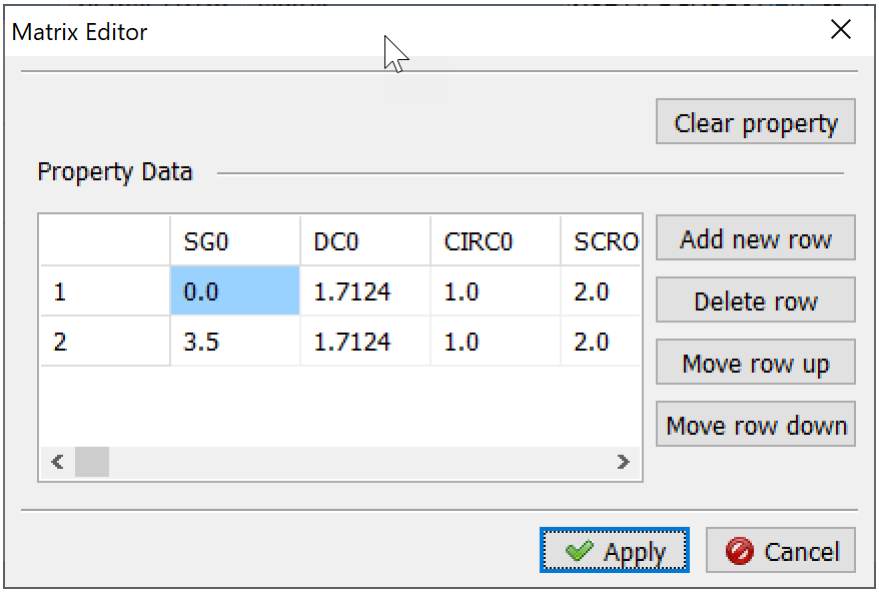

Abb. 4.10 Anpassung der GEOMETRY des Cross Connection Object im Matrix-Editor

4.2.2.2 Vererbte Daten

In einigen ATHLET-Objekten besteht die Möglichkeit, Daten von „Master“-Objekten zu erben, wie thermohydraulische Zustände (z. B. Anfangstemperatur). Die Umsetzung in ATM musste vollständig überarbeitet werden, da teilweise einige Daten von Objekten nicht geerbt werden können oder ungewollt immer geerbt werden, was in der bisherigen Implementierung nicht berücksichtigt war. Die Objekt-Properties werden nun bei der Initialisierung der Objekte mit „`IsInheritable`“ oder „`IsForcedInheritable`“ entsprechend gekennzeichnet. Eine weitere Kennung „`IsInherited`“ wird zur Laufzeit entsprechend den Benutzervorgaben gesetzt. Dies ist in einem speziellen Dialog (siehe Abb. 4.9) zum Setzen der „Master“-Property des Objekts durch die jeweiligen Checkboxes möglich. Wenn Daten vom Typ „`IsForcedInheritable`“ vererbt werden, sind die jeweiligen Eingabefelder deaktiviert.

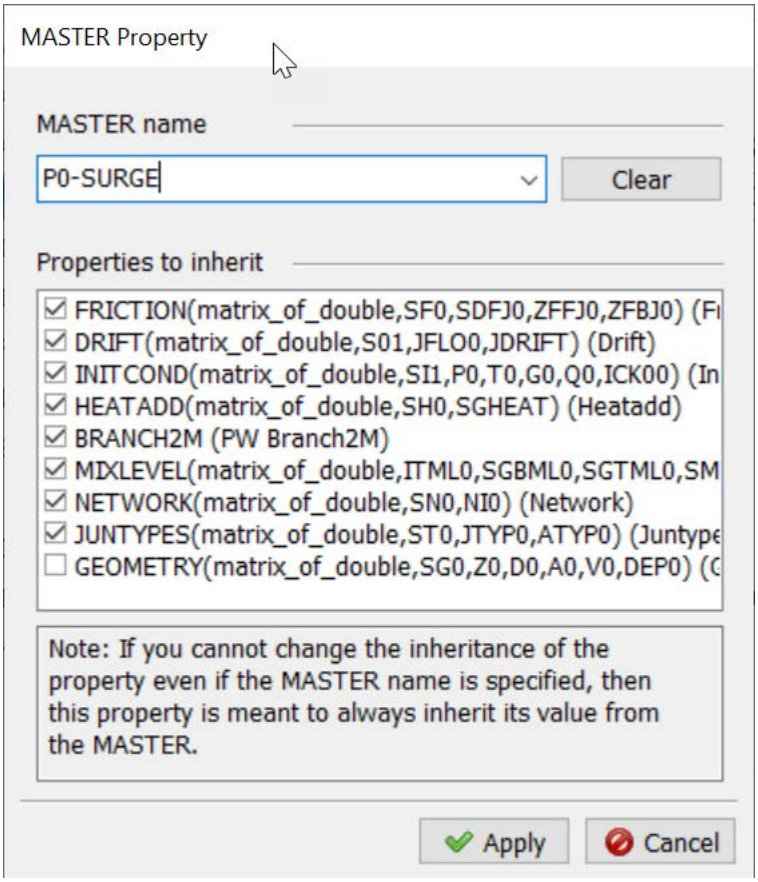

Abb. 4.11 Dialog für geerbte Daten eines TFOs

Dieser Dialog ist über einen externen Datentyp „`TMasterData`“ realisiert. Entsprechend dem jeweiligen Objekttyp werden im Dialog nur diejenigen Größen angezeigt, die tatsächlich geerbt werden können.

4.2.2.3 Auswahl von Objekten

An vielen Stellen in der ATHLET-Eingabe sind Referenzen auf andere Objekte nötig, wie beispielsweise die mittels eines CCO verbundenen TFOs (Source- und Target-Channel), Ventile und Pumpen in einer Leitung oder TFOs in einer Prioritätskette. Die Referenz erfolgt über den jeweiligen Namen der Objekte. Zur Erleichterung der Eingabe und Vermeidung von Fehlern wurde ein genereller Dialog (siehe Abb. 4.10) entwickelt, der die Objektauswahl ermöglicht.

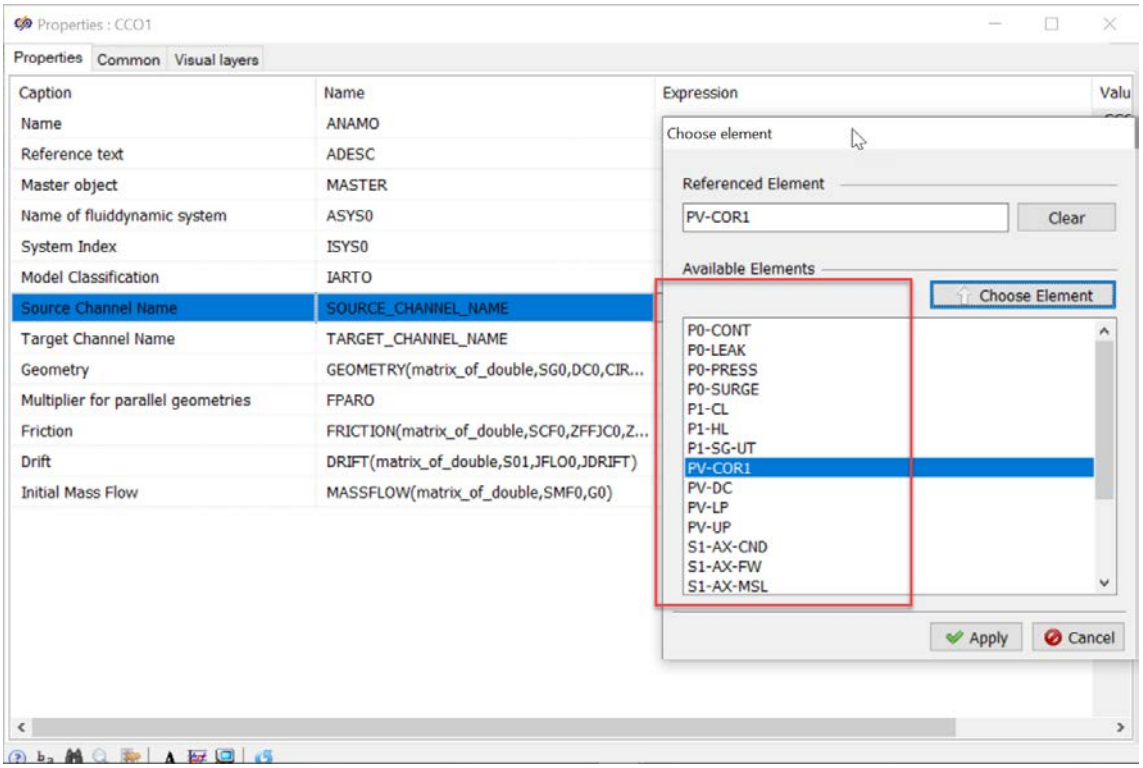

Abb. 4.12 Dialog für Objektauswahl

Die Liste der wählbaren Objektnamen wird dynamisch und entsprechend der verfügbaren und zulässigen Objekte erstellt. Dafür können je nach benötigtem Objekttyp (TFO, HCO, PUMP, …) Funktionen registriert und implementiert werden, die alle auf dem Workspace vorhandenen Objekte auswerten. Ihre Namen werden dann als alphabetisch sortierte Liste im Dialog angezeigt. Die Selektion des Nutzers wird mit „Apply“ zurückgegeben und gespeichert.

4.2.3 Methode zur Modifikation von ADM

Ziel des Arbeitspakets bestand darin, eine Methode bereitzustellen, die eine Modifikation der Modellierung in ADM ermöglicht, ohne den Quellcode zu verändern. Dies ist nur in gewissen Grenzen möglich, aber wünschenswert, wenn die Änderungen an der Code-Eingabe nur geringfügig sind, beispielsweise durch Hinzufügen neuer Variablen in einem existierenden Modell. Damit können Modellentwickler, z. B. beim Erstellen einer neuen Codeversion, einfache Anpassungen des Eingabegenerators selbstständig durchführen. Allerdings muss durch administrative Maßnahmen sichergestellt werden, dass die Änderungen nach einer Testphase ins Git-Repository von ADM übertragen werden.

Im Folgenden wird dieses Vorgehen am fiktiven Beispiel des Hinzufügens einer Abbrandzeit („TBURN“) für ein Brennstabobjekt erläutert. Im ersten Schritt muss das graphische Objekt die neue Eingabevariable berücksichtigen. Dazu kann der Blockeditor in ADM verwendet werden, der über das Objektmenü (siehe Abb. 4.11) zugänglich ist, sobald ADM in den „Developer Mode“ geschaltet wird.

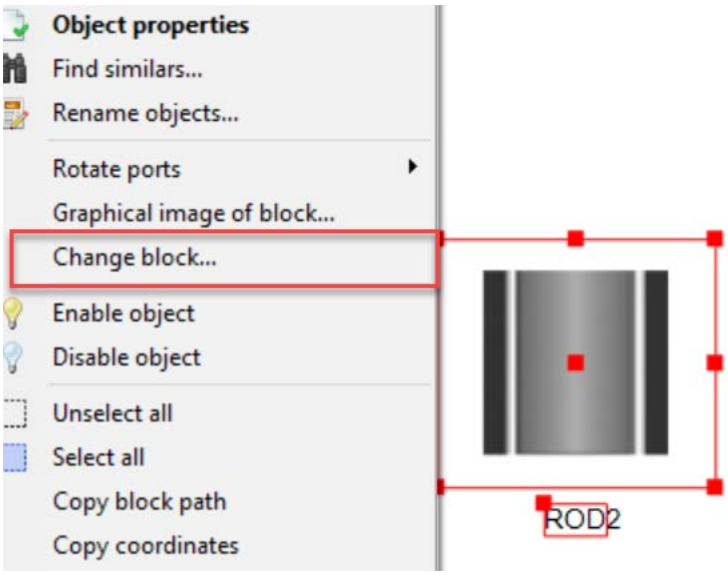

Abb. 4.13 Objektmenü eines graphischen Blocks

Der Blockeditor ist eine erweiterte Form des Standardeditors für die Vorgabe der Property-Werte. In diesem können Properties für das Objekt hinzugefügt, geändert oder gelöscht werden, indem man die mit dem Pfeil gekennzeichneten Buttons verwendet und den Namen der Property, ihren Datentyp und andere Details spezifiziert (siehe Abb. 4.12).

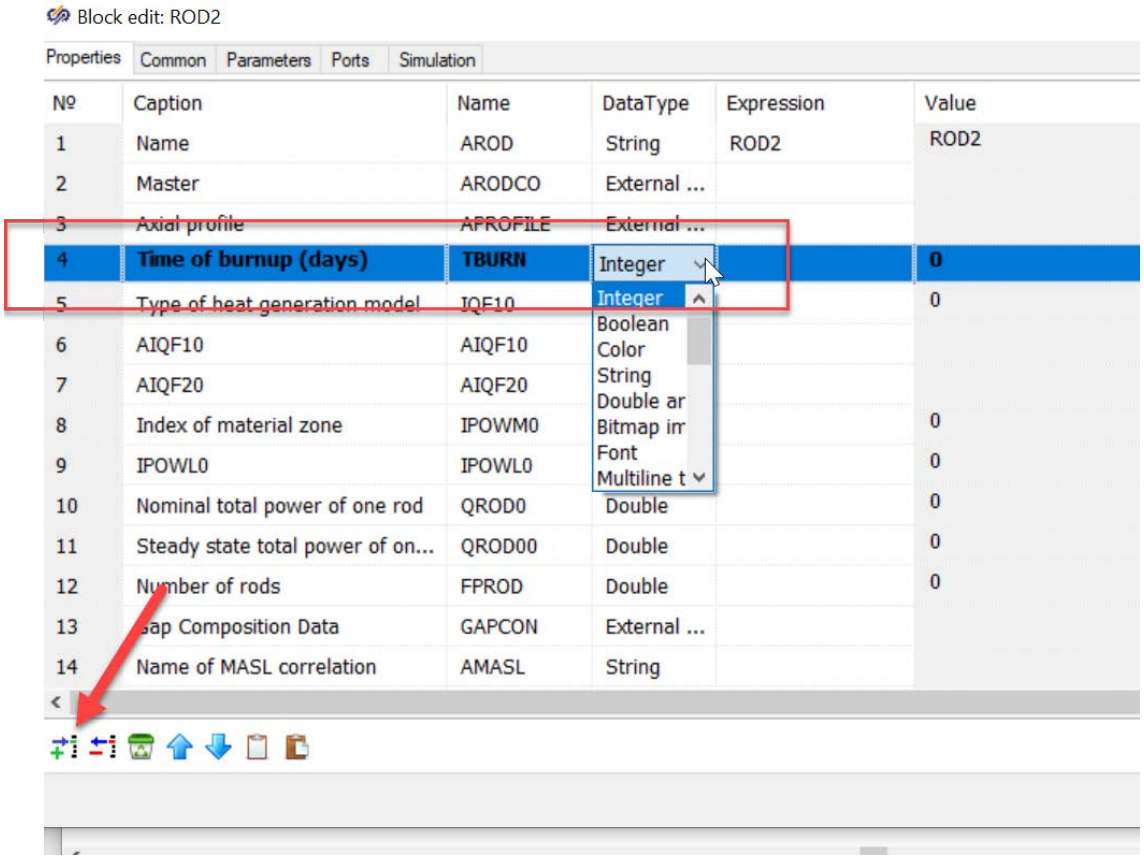

Abb. 4.14 Blockeditor für graphische Objekte

Ist die gewünschte Property auf diesem Weg erstellt, muss der Block im Anschluss in der Blockbibliothek gespeichert werden. Damit wird beim Generieren eines neuen Objekts dieses Typs über die Toolbar automatisch ein Block erzeugt, der die neue Property enthält. Die neue Property muss im Anschluss auch beim Erzeugen der ASCII-Ausgabe für das Objekt berücksichtigt werden. Wegen der strikten Formatierung, die gegenwärtig in der ATHLET-Eingabe verlangt wird, ist die Ausgabe des Variablenwerts an einer spezifischen Position einer Eingabezeile für das Objekt nötig, entsprechend der Vorgabe in der Eingabebeschreibung. Generell sind diese Vorgaben im ADM-Quellcode für den Datenexport der Objekte umgesetzt. Da der Quellcode nicht durch den Nutzer änderbar ist, wurde eine Option bereitgestellt, in der die Reihenfolge der Ausgabe der Werte für ein Objekt definiert werden kann. Dazu wird eine Liste von Namen (`StringMatrix`) verwendet, welche die Namen der Properties für die Ausgabe in der gewünschten Abfolge enthalten muss (siehe Abb. 4.13). Wenn diese Matrix Werte enthält, wird, abweichend von der im Quellcode definierten Reihenfolge, die Ausgabe entsprechend der Reihenfolge der Variablen in der Matrix verwendet. Im Beispiel wird die Abbrandzeit TBURN nach den Daten für das Leistungsprofil (APOFILE) ausgegeben.

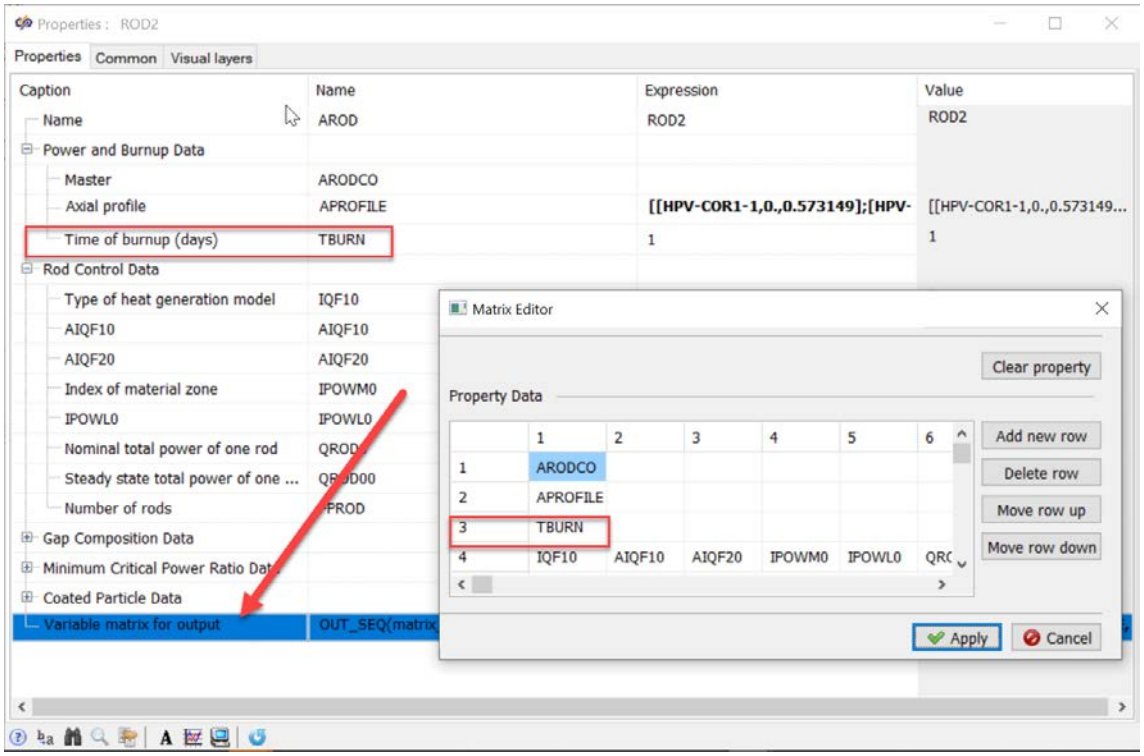

Abb. 4.15 Definition der Ausgabereihenfolge in einem Objekt

Es wird darauf hingewiesen, dass die implementierte Lösung nur einfache Erweiterungen im Input ermöglicht. In einigen Fällen ist beispielsweise die Ausgabe zusätzlicher Eingabedaten abhängig vom Zahlenwert einer anderen Eingabegröße nötig, was mit der Methode nicht realisierbar ist. Ebenso ist es nicht möglich, mit diesem Verfahren die nötigen Änderungen beim Datenimport zu realisieren.

Die Möglichkeiten zur Modifikation von ADM durch den Nutzer könnten durch den Einsatz von „Scripting“ noch erheblich erweitert werden. SimInTech enthält eine proprietäre Skript-Lösung mit einem eigenen Interpreter. Diese Skriptsprache umfasst sowohl übliche Befehle zur Programmsteuerung (Schleifen, Bedingungen, etc.) als auch vordefinierte Funktionen zum Zugriff und zur Modifikation von Objekten. Insgesamt ist der Umfang der Möglichkeiten aber zu sehr eingeschränkt und Erweiterungen sind nur aufwendig durchführbar. Ebenso ist wegen der fehlenden Dokumentation eine Verwendung nicht einfach. Aus diesem Grund wurden einige prinzipielle Tests zur Nutzung von Python Scripts („python4delphi“ /P4D 21/) in ADM durchgeführt. Die Tests der Einzelkomponenten des Bindings waren vielversprechend. Leider stellte sich die Einbindung in die Applikation als schwieriger heraus als angenommen und sie konnte über diesen Weg nur unzufriedenstellend realisiert werden.

4.2.4 Erweiterungen für AC²-Rechencodes

4.2.4.1 Erweiterung ATHLET-Eingabe (ATM)

In diesem Arbeitspaket wurden die interaktiven Werkzeuge zur Erstellung der ATHLET-Eingabe weiterentwickelt. Zur Modellierung von Reaktorsystemen mit Thermo-Fluid-Objects (TFOs), deren Netzwerktopologie und anderen Modellen aus ATHLET wurde der „ATHLET Thermohydraulic Modeller“ (ATM) im Master-Branch (ADM) weiterentwickelt. Zur Nachbildung der Reaktorleittechnik und von Hilfssystemen steht der „ATHLET GCSM Modeller“ (AGM) im Wartungs-Branch zur Verfügung, der bereits in breiter Anwendung ist und lediglich kleinere Ergänzungen und Bugfixes benötigte.

ATM hatte in RS1537 /VOG 18/ einen Entwicklungsstand erreicht, der eine eingeschränkte Anwendung für die interaktive Modellerstellung erlaubte. Erste Anwendungserfahrungen haben ergeben, dass für die Anwendung in der Praxis Erweiterungen und Verbesserungen benötigt werden. Dies betrifft besonders die Fähigkeit, existierende Daten automatisiert zu übernehmen, da eine manuelle Übertragung aus zeitlichen Gründen und der potenziellen Fehlerquellen nicht akzeptabel ist. Im kontinuierlichen Erfahrungsaustausch mit den Anwendern konnten Fehler zeitnah erfasst und beseitigt werden. Der dafür notwendige Aufwand war erheblich höher als geplant, wurde aber in Kauf genommen, um die Stabilität von ATM zu steigern.

4.2.4.2 Erweiterung der Modellierungsoptionen

Die Modellierung für ATHLET umfasste bisher im Wesentlichen die Nachbildung von Rohrleitungen und wärmeleitenden Strukturen mit TFOs und Wärmeleitungsobjekten (HCOs). Entsprechend der vorliegenden Anwendungserfahrung war für diesen Bereich in vielen Stellen eine Überarbeitung notwendig. Einige ATHLET-Modelle, z. B. für Ventile, Pumpen, Brennstab, wurden in ATM bisher noch nicht oder unzureichend unterstützt und mussten daher hinzugefügt werden.

4.2.4.3 TFO- und HCO-Modellierung

Eine Reihe von Anpassungsarbeiten wurden spezifiziert und durchgeführt. Diese Arbeiten waren für die praktische Anwendung kurzfristig erforderlich.

Für die Fluidobjekte ist die Vorgabe einer Reihe von tabellarischen Daten notwendig:

- Diskretisierung (NETWORK)
- Geometrie (GEOMETRY)
- Reibung (FRICTION)
- Driftdaten (DRIFT)
- Anfangswerte (INITCOND)
- Interne Komponenten (JUNTYPES)

Bisher konnten diese Daten zwar vorgegeben werden, der Editor zur Vorgabe war aber sehr schwer bedienbar. Mit der Entwicklung des Matrixeditors (siehe Abschnitt 4.2.2.1) konnte die GUI für die Datenvorgabe wesentlich verbessert werden. In Abbildung 4.14 ist die jetzt implementierte Dateneingabe am Beispiel der Anfangswerte zu sehen.

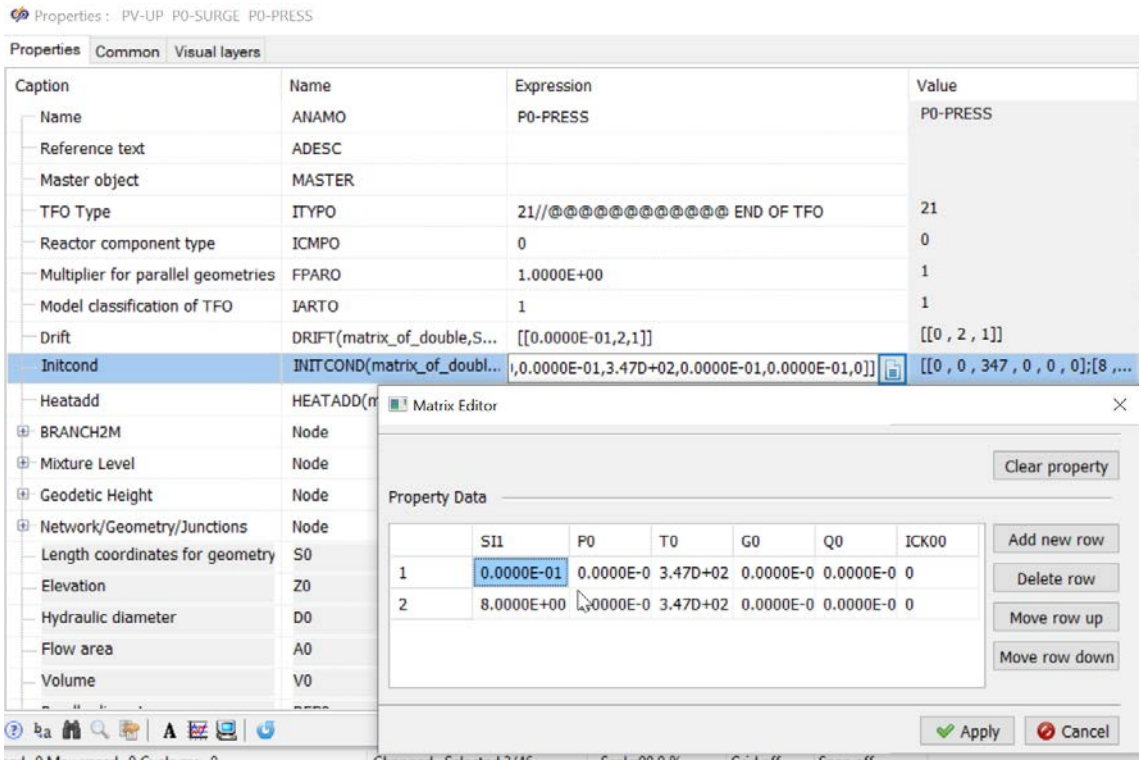

Abb. 4.16 Dateneingabe für tabellarische TFO-Daten

In ähnlicher Weise werden nun auch die tabellarischen Daten für HCOs vorgeben, beispielsweise für die ortsabhängigen Wärmeleitkoeffizienten.

Für einzelne TFOs benötigt ATHLET die Eingabe der geodätischen Höhen als Funktion der Längenkoordinate. Sind mehrere TFOs verbunden, müssen die Höhendaten konsistent sein. Bei notwendigen Änderungen entsteht dadurch ein erheblicher Berechnungsaufwand, um alle Daten korrekt anzupassen. Deshalb wurden in ATM Funktionen implementiert, die diese Datenanpassung unterstützen und automatisieren können. In den Properties der TFOs gibt es zusätzliche Eingabemöglichkeiten (siehe Abb. 4.15), die die Höhendaten modifizieren können.

Geodetic Height	Node	Node	Node
Initial geodetic height (added to Z0 values)	INIT_Z0	**1.0**	1
Calculate INIT_Z0 from previous TFO	CALC_INIT_Z0	☐	No
Has initial 3D-point	B3D_POINT	☐	No

Abb. 4.17 Daten zur Anpassung der geodätischen Höhen eines TFOs

Es ist damit möglich, alle Höhen mit einer Konstanten (INIT_Z0) zu beaufschlagen, die Daten vom Vorläufer TFO abzuleiten (CALC_INIT_Z0) oder sie mit einem absoluten Startwert (B3D_POINT) für das erste TFO in einer Kette zu versehen. Mit diesen Möglichkeiten kann bei komplexen Geometrien und Objektnetzwerken der Aufwand zur Datenmodifikation erheblich reduziert werden. Ebenso werden Berechnungsfehler vermieden.

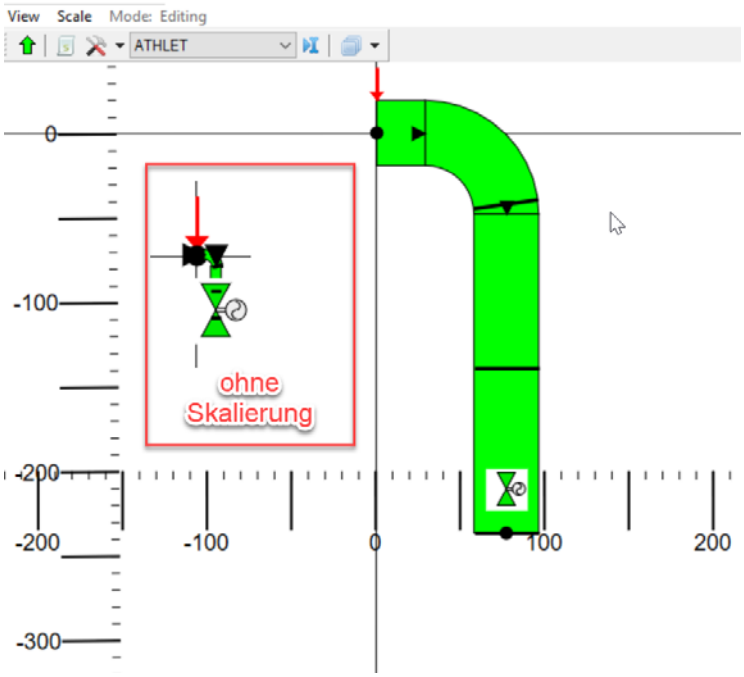

Abb. 4.18 Der Geometrie-Editor für TFOs

Im Geometrie-Editor (siehe Abb. 4.16) wird bei der Modellierung der TFOs die Geometrie der Leitungselemente (z. B. Rohr, Düse, Bogen) maßstabsgetreu angezeigt. Die Junctions, wie zum Beispiel Ventile und Pumpen, haben in ATHLET keine geometrische Ausdehnung und werden daher in fester Größe dargestellt.

Standardmäßig werden 100 Pixel für eine Länge/Durchmesser von 1 m verwendet. Für viele Anlagentypen ist dieser Maßstab gut geeignet. In Versuchsanlagen sind die Leitungsmaße jedoch oft sehr viel geringer, was in der Geometriedarstellung unbefriedigende Ergebnisse verursachte. Deshalb wurde eine Skalierungsoption (siehe Abb. 4.17) in ATM ergänzt.

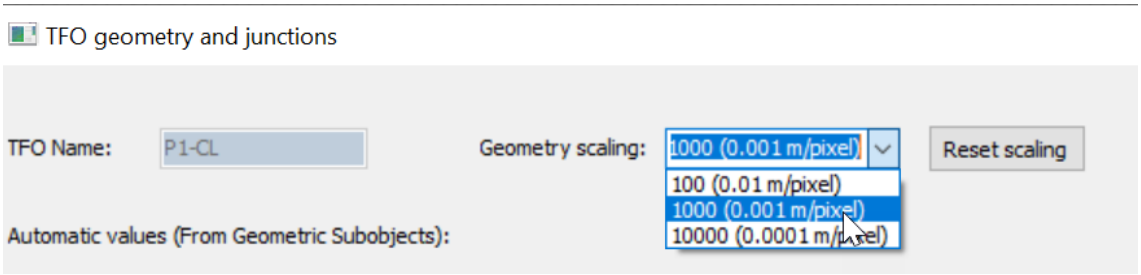

Abb. 4.19 Skalierung von Leitungselementen eines TFOs

In der bisherigen Version von ATM wurde versucht, am Beispiel des Cross Connection Objects (CCO), speziell angepasste Objekteditoren einzusetzen. Probleme ergaben sich aber durch den fehlenden Kontrollmechanismus für die Daten und den hohen Wartungsaufwand. Stattdessen können nun spezielle Editoren für die einzelnen Properties verwendet werden. Beim CCO wurde auf diese Weise die Vorgabe der TFOs, die durch das CCO verbunden sind, und die Auswahl eines Master-CCO realisiert. Ein Objektsauswahldialog stellt alle verfügbaren TFOs in einer Liste dar, aus der die gewünschten Objekte selektiert werden können (siehe Abb. 4.18).

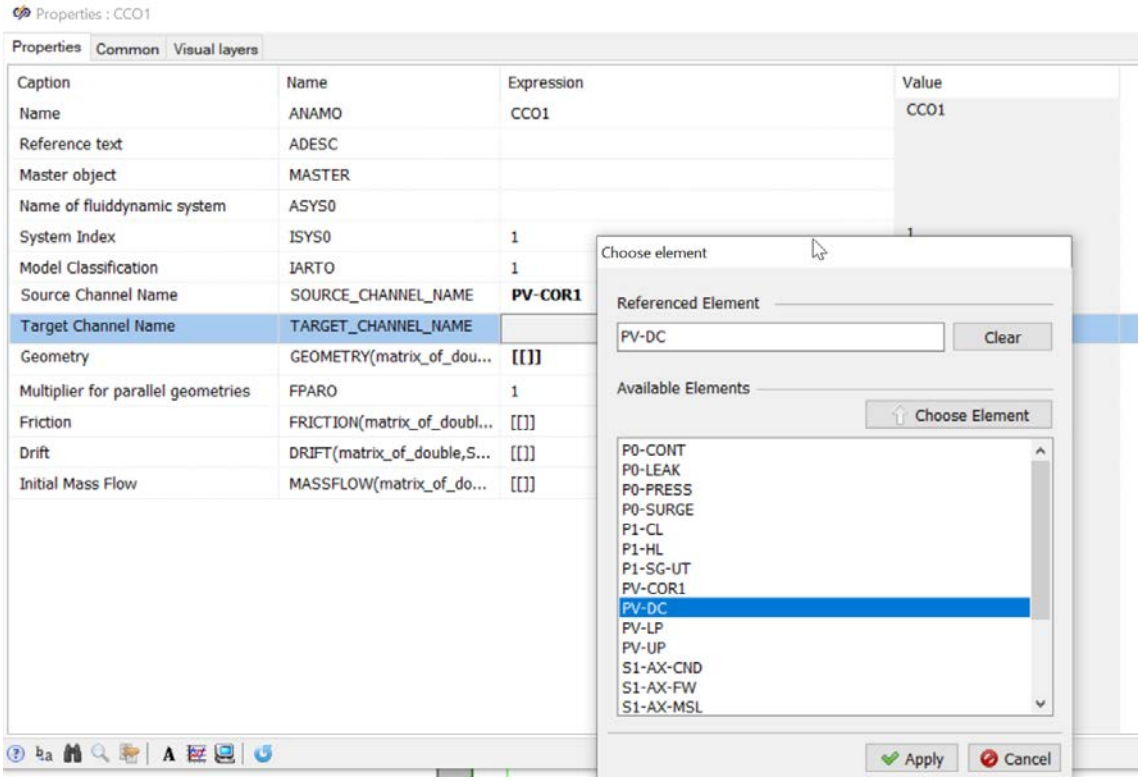

Abb. 4.20 Auswahl für Source/Target und Master TFOs

Besonders für die HCOs war das Fenster zur Vorgabe der Objektdaten wegen der sehr großen Anzahl von Properties sehr unübersichtlich. Daher wurde eine Strukturierung der Daten in Gruppen eingeführt, wie sie auch in der ATHLET-Eingabebeschreibung verwendet wird. Die Datengruppen werden als aufklappbare Baumelemente anzeigt (siehe Abb. 4.19). Dies erleichtert das Auffinden der benötigten Daten erheblich.

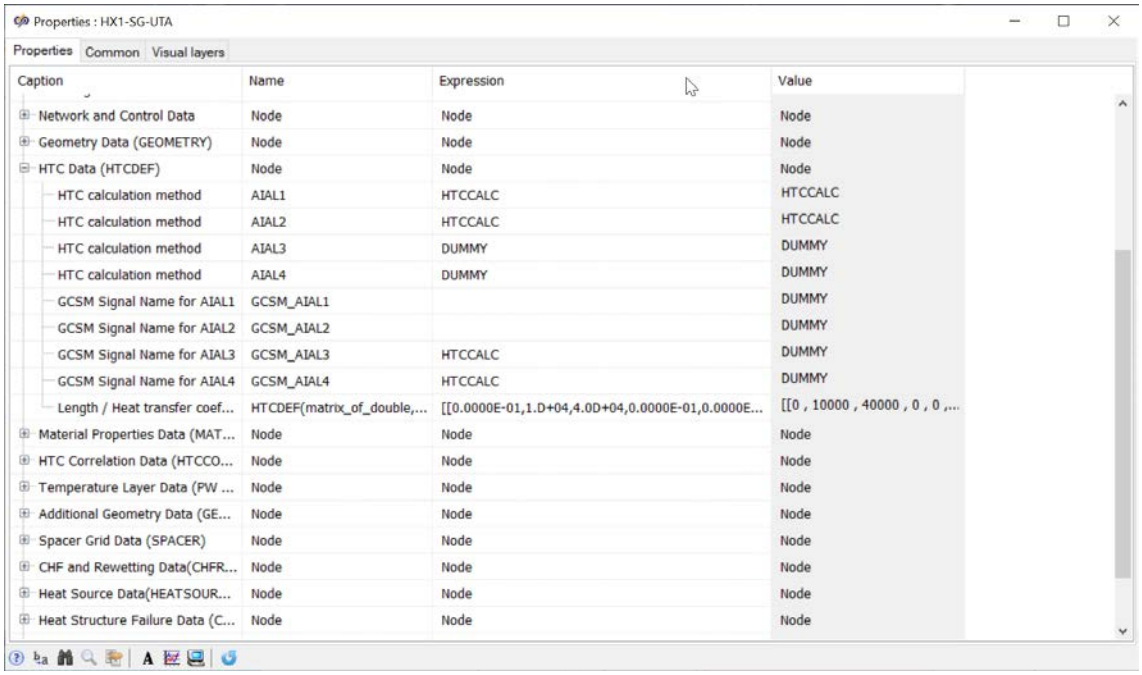

Abb. 4.21 Strukturierung von Heat-Conduction-Object-Daten in Gruppen

4.2.4.4 Brennstab Modellierung

Das Brennstabmodell (ATHLET Controlword ROD) besitzt umfangreiche Eingabedaten für die Beschreibung von

- Leistung und Abbrand
- Kontrolldaten zu Wärmeerzeugung (PW RODCON)
- Optionale Spaltdaten (PW GAPCON)
- Optionale Daten für Abstand zur Siedeübergangsleistung (PW MASL)
- Optionale Daten für Partikelbrennstoff (PW TRISO)

Zur Modellierung wurden in ATM ein generelles Kontrollwortobjekt und ein Brennstabobjekt (siehe Abb. 4.20) bereitgestellt.

Abb. 4.22 ATM-Objekte zur Brennstabmodellierung

Die Properties dieser Objekte (vgl. Abb. 4.21) dienen zu Vorgabe der benötigten Daten für die ATHLET-Eingabe. Die Namen der Properties entsprechen der Datenbezeichnung in der Eingabebeschreibung.

Properties : ROD11

Properties | Common | Visual layers

Caption	Name	Expression	Value
Name	AROD	ROD11	ROD11
Power and Burnup Data	Node	Node	Node
Master	ARODCO		
Axial profile	APROFILE	[[HPV-COR1-1,0.,0.573149,];[HPV-COR1-1,0.4875,...	[[HPV-COR1-1,0.,0.573149...
Rod Control Data	Node	Node	Node
Type of heat generation m...	IQF10	1	1
AIQF10	AIQF10	Q-CORE // qcore comment	Q-CORE // qcore comment
AIQF20	AIQF20	DEFAULT	DEFAULT
Index of material zone	IPOWM0	1	1
IPOWL0	IPOWL0	0	0
Nominal total power of one...	QROD0	82957.2	82957.2
Steady state total power of...	QROD00	82957.2	82957.2
Number of rods	FPROD	0 //FPROD inline com.	0
Gap Composition Data	Node	Node	Node
Minimum Critical Power Ratio ...	Node	Node	Node
Coated Particle Data	Node	Node	Node

Abb. 4.23 Properties eines ROD-Objekts

Für einige der Daten, beispielsweise das axiale Leistungsprofil, wurden spezielle Datenklassen entwickelt, die es erleichtern, auf die Werte zuzugreifen und sie zu editieren. Wie bei den TFOs und HCOs ist auch bei den ROD-Objekten die Auswahl eines „Master“ Objekts möglich, von dem die Daten für das Leistungsprofil vererbt werden können.

Das Erzeugen der von ATHLET lesbaren Eingabedaten wird von datenspezifischen Ausgabeklassen durchgeführt. Diese Klassen greifen auf Property-Daten der ROD-Objekte zu und generieren daraus, nach möglichst weitgehender Prüfung auf Vollständigkeit und Plausibilität, die für ATHLET korrekt formatierten Daten.

4.2.4.5 Pumpen Modellierung

Das Pumpenmodell (ATHLET Controlword PUMP) hat Parameter für verschiedene Optionen einer Pumpe:

- Druckdifferenzsteuerung
- Statische Impeller-Drehzahl Simulation
- Dynamische Impeller-Drehzahl Simulation

Die Daten können im Property Dialog der Pumpe (siehe Abb. 4.22) spezifiziert werden. Abhängig vom Pumpentyp (ILART2) können die formatierten Daten für ATHLET erzeugt werden. Der Import von Pumpendaten ist bisher nicht implementiert.

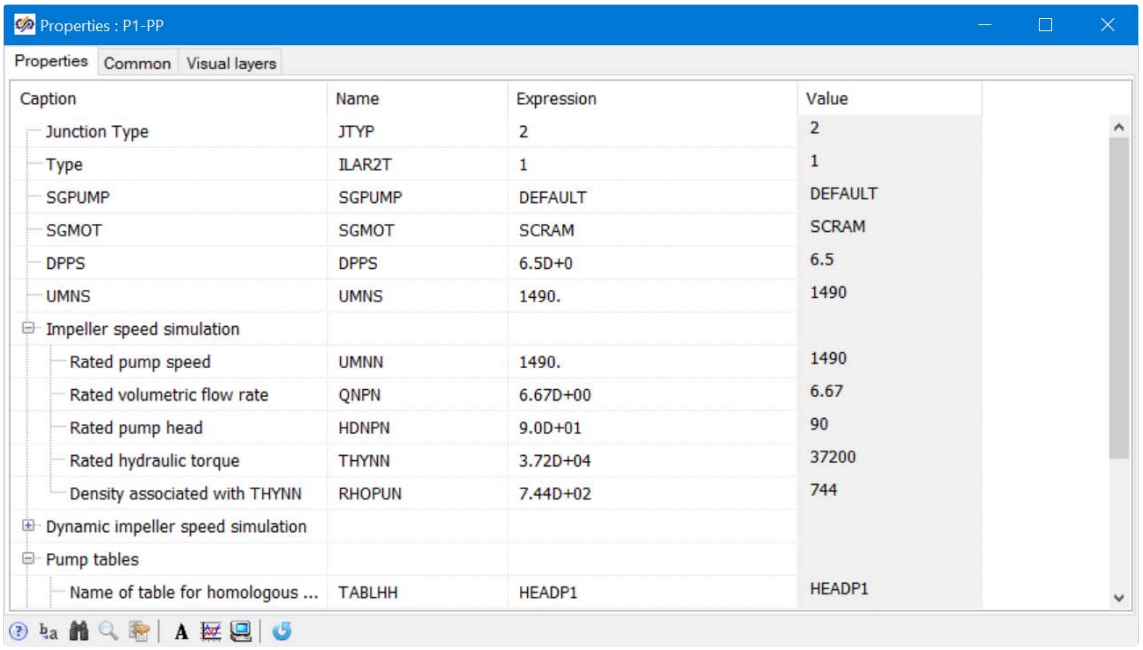

Abb. 4.24 Properties eines Pumpen-Objekts

Bei der Pumpe gibt es 2 unterschiedliche Möglichkeiten, ihre örtliche Position in einer Leitung (TFO) zu definieren. Wird der Geometrie-Editor nicht verwendet (NOTUSE_GEOMWS in Abb. 4.23), kann die Position als Ortskoordinate ST0 bei den TFO Junctions (JUNTYPES) als Zahlenwert vorgegeben werden.

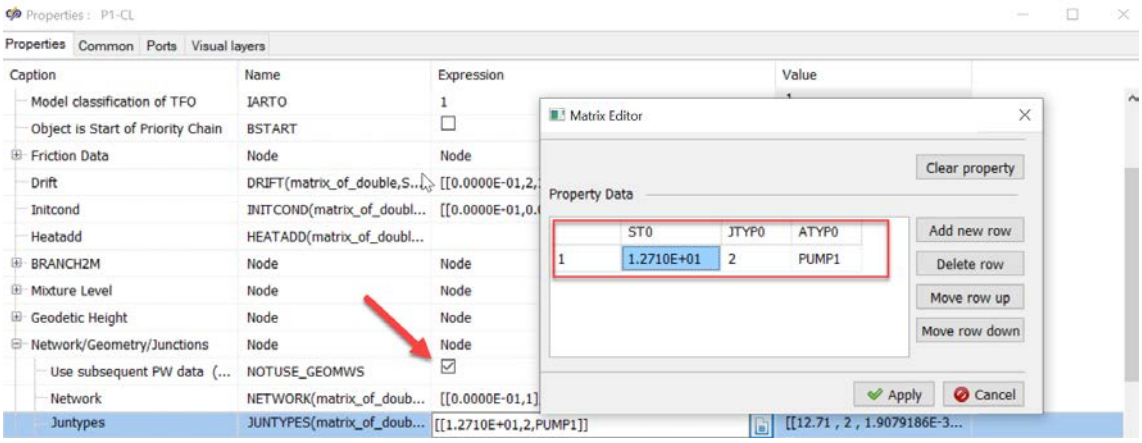

Abb. 4.25 Spezifikation des Pumpenorts im Editor

Wenn man die Leitungsgeometrie mit Geometrieelementen abbildet (siehe Abb. 4.24), muss man die Pumpenposition im Leitungsverlauf definieren. Aus der dort festgelegten Position wird dann die entsprechende Ortskoordinate der Pumpenposition im TFO automatisch berechnet.

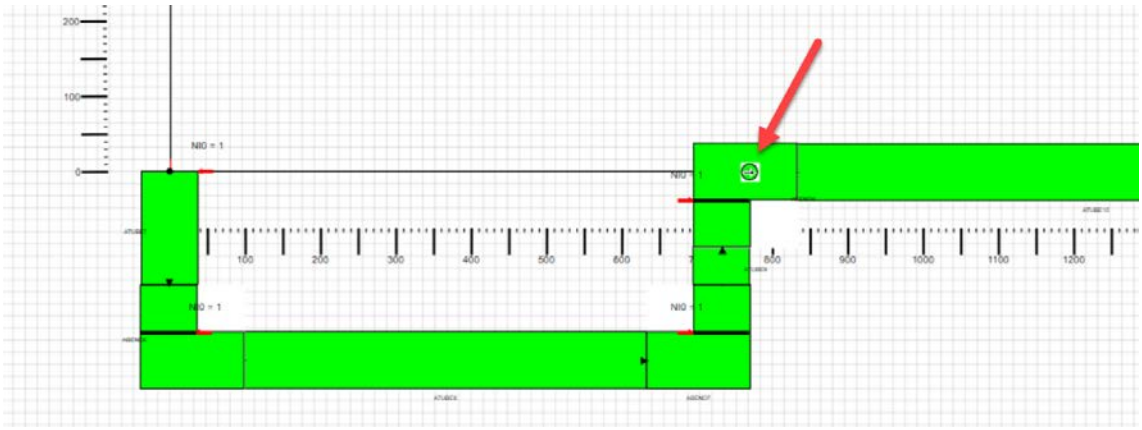

Abb. 4.26 Spezifikation des Pumpenorts in der Leitungsgeometrie

4.2.4.6 Ventil Modellierung

Das Ventilmodell (ATHLET Controlword VALVE) hat Parameter für verschiedene Optionen eines Ventils:

- Standardventil
- Varianten von Rückschlagklappen

Die Daten können im Property Dialog des Ventils (siehe Abb. 4.25) spezifiziert werden. Abhängig vom Ventiltyp (ITYPV) können die formatierten Daten für ATHLET erzeugt werden. Der Import von Ventildaten ist bisher nicht implementiert.

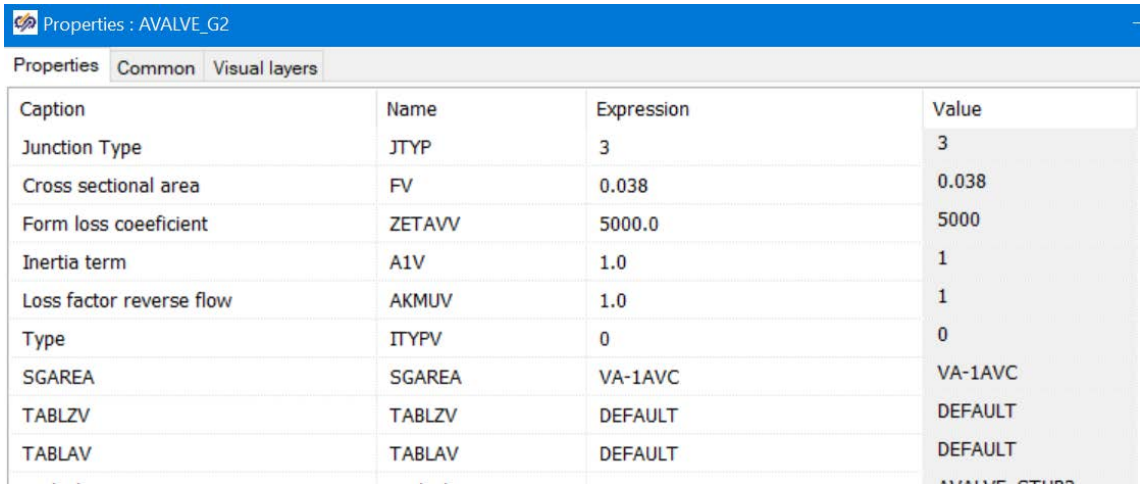

Abb. 4.27 Properties eines Ventil-Objekts

4.2.4.7 Tabellen Modellierung

Das Tabellenmodul (ATHLET Controlword TABLES) kann verschiedene Formen von Tabellen verwalten:

- Standardtabelle mit einer oder mehreren abhängigen Variablen
- Tabellen für Pumpen

Die nötigen Werte können im Property Dialog der Tabelle (siehe Abb. 4.26) spezifiziert werden.

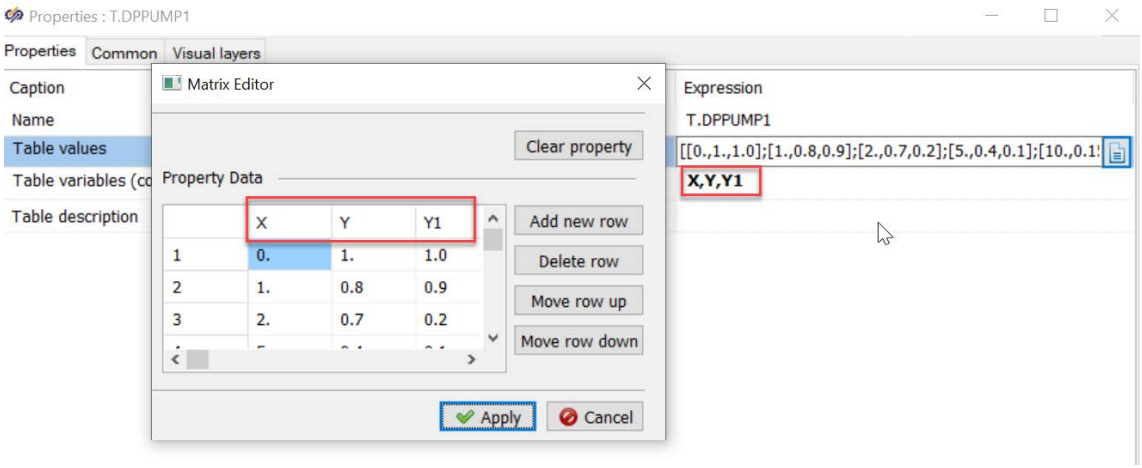

Abb. 4.28 Wertevorgabe einer Tabelle

Die Anzahl der Tabellenspalten kann beliebig gewählt werden. Dazu müssen die Namen der Spalten („Table variables“) angegeben werden, beispielsweise „X,Y,Y1“ für eine Tabelle mit 3 Spalten. Die Anzahl der Tabellenzeilen wird mit „Add row“ bzw. „Delete row“ eingestellt.

Zur Vorgabe von Pumpenkennlinien, wie Förderhöhe oder Drehmoment in Abhängigkeit vom Pumpendurchsatz, sind spezielle Tabellen notwendig, in denen jeweils 8 Teile einer Kennlinie eingegeben werden können. Diese werden in ATM mit den Elementen „HEAD“ und „TORQUE“ (siehe Abb. 4.27) realisiert.

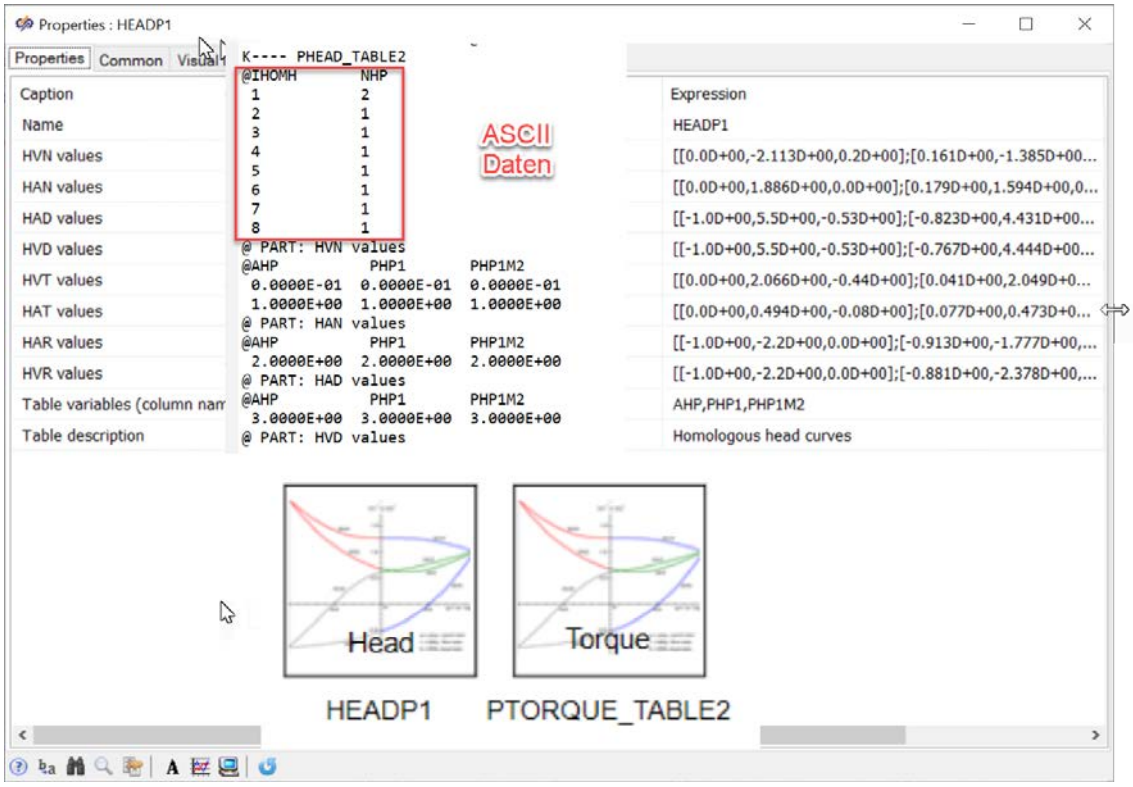

Abb. 4.29 Wertevorgabe für tabellarische Pumpenkennlinien

Die Bedeutung der einzelnen Tabellenteile (HVN, LZHVR,…) ist in der ATHLET-Eingabebeschreibung erläutert. Im Gegensatz zur zu den ASCII-Daten muss in ADM die Anzahl der Stützwerte (NHP) nicht angegeben werden, sondern wird bei der Datenerzeugung automatisch ermittelt.

4.2.4.7.1 Erweiterungen im Datenimport

Bereits in der in erste Version von ATM aus RS1537 war es möglich, die Daten von verschiedenen TFO-Typen aus ATHLET aus vorhandenen Eingabedaten in ASCII-Form automatisiert nach ATM zu übernehmen. Prinzipiell setzt dieser Datenimport auf einem Interpreter der Objektdaten auf, der die entsprechenden graphischen Objekte in ATM erzeugt oder modifiziert und die Datenwerte aus der Eingabe in die Properties der Objekte übernimmt. Eine manuelle Übertragung der Daten (Copy/Paste) ist zwar möglich, aber sehr zeitaufwendig. Vor allem lassen sich aber Fehler bei dieser Art der Datenübertragung nur schwer vermeiden. Die nachträgliche Identifikation dieser Übertragungsfehler kann im Einzelfall sehr schwierig sein. Im ungünstigen Fall bleiben solche Fehler sogar unerkannt.

Aus diesen Gründen hat sich abweichend von der ursprünglichen Planung im Vorhaben die Notwendigkeit ergeben, die Funktionen zum Datenimport in ATM erheblich zu erweitern und zu verbessern.

Das Format der ATHLET-Eingabedaten ist zwar strukturiert, setzt aber, wegen der historischen Entwicklung, auf keine Standards wie z. B. XML auf. Damit ist es nicht mit Standardbibliotheken lesbar. Für alle Objekte und Modelle von ATHLET liegen spezifische Eingabeformate vor, die sich zudem, je nach gewählter Modelloption, unterscheiden können und auch von der ATHLET-Version abhängen können. Eine genaue Beschreibung dieser Daten findet sich in der „ATHLET Input Data Description“ /LER 16/. Die genauen Details zu den Daten und Formaten sind im Eingabeparser von ATHLET programmiert. Dieser ist aber als elementarer Bestandteil von ATHLET in FORTRAN codiert, nicht modular aufgebaut und daher in ATM nicht verwendbar.

Für jeden Objekttyp aus ATHLET muss daher ein angepasster ATM-Interpreter für die Daten entwickelt werden. Wegen der teilweise sehr umfangreichen Eingabedaten und der verschiedenen Eingabeoptionen sind die Entwicklung und die nötigen Tests sehr aufwendig. Dennoch wurde für die überwiegende Anzahl der in ATM vorhandenen ATHLET-Objekte diese Funktionalität implementiert. Als zusätzlicher Nutzen ergibt sich, dass damit auch gleichzeitig der Reimport von manuell geänderten ASCII-Daten möglich ist. Dies wird durch eine entsprechende Modifikation der Properties von bereits vorhandenen graphischen Objekten erreicht. Ein weiterer Grund ist, dass die korrekte Abbildung von Objekten in ATM nur mit realen ATHLET Beispieldatensätzen geprüft werden kann. Gleichzeitig dient die Umsetzung der Beispieldatensätze in ATM als Referenz, wie man eine ablauffähige Anlagemodellierung in ATM erstellen kann

Gegenwärtig ist für die folgenden ATHLET-Objekte ein spezieller Datenimport vorhanden:

- Thermofluidobjekte (TFO, Pipe, Branch, CCO)
- Netzwerkobjekte/Prioritätsketten (PC)
- Wämeleitobjekte (HCO)
- Brennstabobjekte (ROD)
- Tabellenobjekte (TABLE)
- Neutronendynamik (NEUKINP)
- Variable und Formeln (PARAMETERS)

Wegen der komplexen Modelldaten war dafür ein sehr hoher, über das erwartete Maß hinausgehender, Entwicklungsaufwand nötig. Eine vollständige Automatisierung des Datenimports für alle Modelle ist im Gegensatz zur ursprünglichen Planung wünschenswert, da dann der Bestand vorhandener Datensätze einfacher übernommen werden kann. Deshalb wurde für Modelle mit ähnlicher Beschaffenheit eine generische Methode bereitgestellt, die es ermöglicht diese in Form von sogenannten Stub-Objekten (Texteditor-basierte Änderungsmöglichkeit, siehe Abb. 4.28) zu importieren.

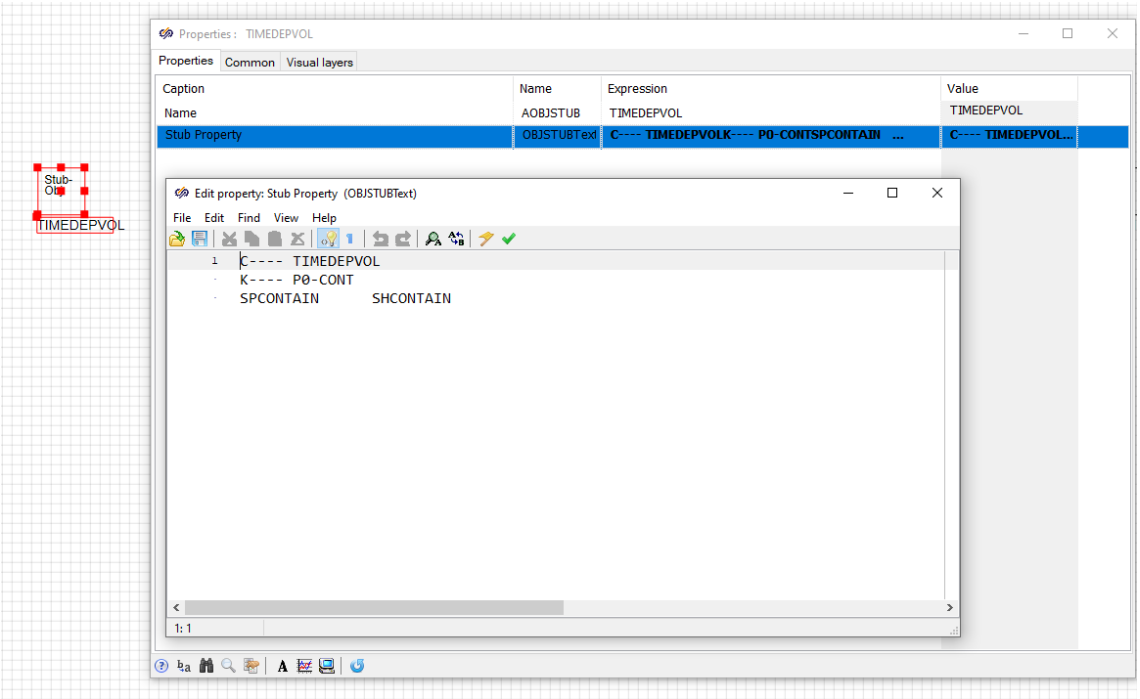

Abb. 4.30 Dateneingabe mittels Texteditors

Beispielhafte Darstellung der Dateneingabe mittels Texteditors für Modelle, welche über den generalisierten Importmechanismus erzeugt werden

Dieser generelle Ansatz ermöglicht auch den Import von Datensatzabschnitten für Modelle, die bislang nicht implementiert werden konnten oder bereits veraltet sind. Dabei werden für diese Modelle sogenannte "Stub-Objects" erzeugt, grafische Objekte deren Daten über einen Texteditor analysiert und nach Bedarf geändert werden können. Diese werden beim Export verwendet, um deren Information im generierten Datensatz auszugeben. Die Importfunktion konnte somit für alle ATHLET-Sample-Datensätze der AC^2-Simulationscodes durch das Schließen eines Use-Case Loops verifiziert werden. Hierzu wurden die Richtigkeit und Vollständigkeit der generierten Datensätze im Anschluss durch einen erfolgreichen ATHLET-Run bestätigt. Erste Tests zur Verarbeitung von ATHLET-CD Datensätzen wurden durchgeführt. Diese zeigten jedoch noch einige Mängel bei der Verarbeitung der hierin enthaltenen Modelle auf.

Die Positionierung der erzeugten graphischen Objekte erfolgt automatisch, so dass gleiche Objekttypen in einer Reihe auf dem Workspace (siehe Abb. 4.29) angeordnet werden.

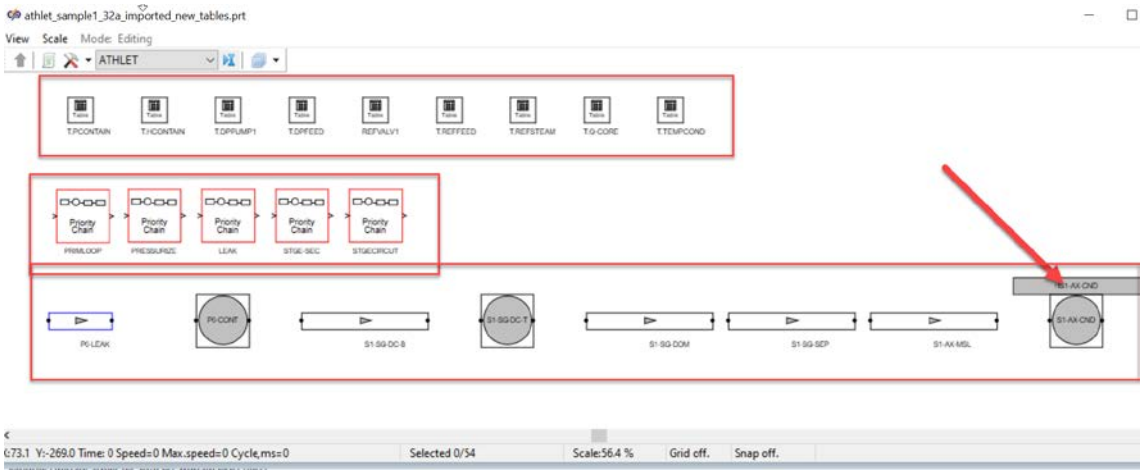

Abb. 4.31 Anordnung der importierten Objekte auf dem Workspace

Wie in der Abbildung ersichtlich (roter Pfeil), werden zur besseren Identifikation, davon abweichend, die erzeugten HCOs direkt bei dem zugeordneten TFO platziert. Zusätzlich sind diese dann auch ein „Subobject“ des TFOs und werden somit automatisch beim Verschieben mitbewegt.

4.2.4.8 Import von Kommentaren

Eine wesentliche weitere Anforderung war, dass Kommentare beim Import von Datensätzen nicht verloren gehen, da sie häufig wichtige Informationen zum Ursprung der Daten, Hinweise zur Berechnung, Autor usw. enthalten. Diese Informationen sollen beim Import der Daten möglichst weitgehend in ATM gespeichert werden.

Das Basissystem SimInTech enthält keine spezifische Möglichkeit, den Properties von Objekten Kommentare hinzuzufügen. Diese Funktion ist für die Eingabeerstellung in ATM aber notwendig, um die zusätzlichen Informationen zu den Daten zu verwalten. Aus diesen Gründen wurde eine Methode implementiert, die es ermöglicht, die Kommentare zusammen mit den Werten der Daten in den Properties zu speichern. Generell können nicht nur Zahlenwerte, sondern auch Formeln („Expressions“) für die Daten eingegeben werden. Dieses Feature wurde bereits zur Implementierung von Datenparametern verwendet. Die Interpretation der Formeln erfolgt durch einen eigenen Interpreter in SimInTech. Die Syntax für die Formeln erlaubt auch das Hinzufügen von Kommentaren, die auch mehrzeilig sein können. Kommentare werden durch ein vorangestelltes „//“ gekennzeichnet (siehe Abb. 4.30).

Abb. 4.32 Kommentare in den Property-Daten der Objekte

Ein Kommentar kann direkt im Eingabefeld für die Expression oder in einem Standardeditor bearbeitet werden. Dies ist dann nützlich, wenn sich ein Kommentar über mehrere Zeilen erstreckt. Kommentare zu Daten, deren Datentyp in den Properties der ATM-Objekte der als Bool (Yes/No) oder Enum (Aufzählungen) definiert ist, können nicht eingegeben werden. Die in ATM gespeicherten Kommentare werden bei der Erzeugung der ATHLET-Eingabedaten mit ausgegeben. Die Ausgabe in der ASCII-Datei erfolgt entweder direkt oberhalb der Zeile mit der Variablen oder in der gleichen Zeile am Ende nach den Zahlenwerten. Für die Ausgabe am Zeilenende werden die Kommentare der letzten Inputvariablen in der Zeile angefügt.

Der Datenimport für bestehende Eingabedateien wurde entsprechend des Speicherkonzepts erweitert, um vorhandene Kommentare, die in ATHLET mit einem „@“ gekennzeichnet sind, automatisch in die ATM Objekte zu importieren. Kommentare sind im ATHLET-Input nur zeilenorientiert vorhanden, entweder am Kopf oder am Ende einer Zeile. Somit ist eine Zuordnung zu einzelnen Variablen einer Zeile generell nicht möglich, wenn diese mehrere Variablen enthält. Die Kommentarzeilen werden daher beim Import entweder in der Expression der ersten oder letzten Variablen gespeichert.

Beim Import wird die Kommentarzeile, die direkt oberhalb der Werte steht, unterdrückt, da sie in den meisten Fällen nur die Namen der Eingabevariablen enthält. Diese entsprechen den Namen der Properties der Objekte und müssen daher nicht importiert werden. Ebenfalls unterdrückt werden Kommentarzeilen, die nur ein „@“ enthalten. Eine weitere Beschränkung sind Kommentare, die sich zwischen den Zeilen von ortsabhängigen Daten der Pseudokeywords befinden. Diese werden gegenwärtig ebenfalls beim Import nicht erfasst.

Das neue Verfahren hat umfangreiche Änderung in den Im- und Exportfunktionen für die verschiedenen Objekttypen erforderlich gemacht. Für das Verwalten der Kommentare zu den Daten wurde eine Reihe von Programmen neu entwickelt, die die nötigen Funktionen bereitstellen. Die Programmänderungen wurden mit umfangreichen Tests begleitet, die eine Vielzahl der möglichen Kommentare in den Daten berücksichtigt haben. Dabei hat sich gezeigt, dass Kommentare durch den in Delphi implementierten Parser-Ansatz nicht ausreichend automatisiert in ATM übernommen werden können.

4.2.4.9 Testanwendung

Ein Test von ATM ist durch die Erzeugung eines vollständigen ATHLET Inputs und die korrekte Ausführung eines Simulationslaufs mit den erzeugten Daten möglich. Dabei sind die folgenden Schritte im Fokus:

- Import von existierenden Daten
- Interaktive Modellierung in ATM
- Export der Daten und ATHLET-Simulationslauf

Mit diesem Vorgehen und der Nutzung aller Objekttypen im Datensatz lässt sich die Anwendung von ATM in der Praxis relativ gut beurteilen und Verbesserungsmöglichkeiten oder Fehler werden aufgezeigt.

In der Anwendung des ATM durch einen ATHLET Nutzer wurde ein thermohydraulisches Modell für eine geplante Versuchsanlage (siehe Abb. 4.31) entworfen. Aus den interaktiv erstellten Objekten und Daten können die äquivalenten ASCII-Daten in dem von ATHLET benötigten Format erzeugt werden. Eventuelle Fehler, wie fehlende Daten oder falsche Datentypen werden erkannt und in der Statuszeile des Programmfensters gemeldet. ATM unterstützt die aktuelle Release-Version 3.2A von ATHLET. Bei einer Anpassung der Eingabedaten in ATHLET, z. B. erweiterte Objektdaten für TFOs, muss ATM entsprechend angepasst werden.

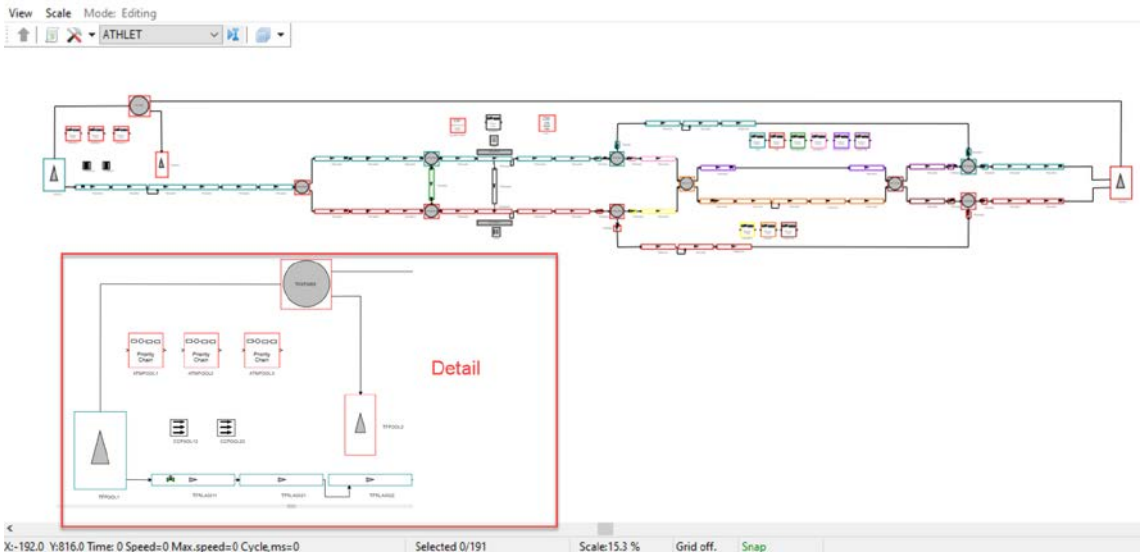

Abb. 4.33 Thermohydraulik-Modellierung einer Versuchsanlage in ATM

Es hat sich gezeigt, dass trotz vorhandener Dokumentation, häufig Fragen zur Realisierung von Details in der Modellierung mit ATM auftraten. Probleme oder Fehler wurden zeitnah mit dem Entwicklungsteam diskutiert und falls kurzfristig umsetzbar sofort beseitigt.

Die gewonnenen Erfahrungen sind direkt in Erweiterungen und Verbesserungen von ATM eingeflossen. Hier ist beispielsweise die Vorgabe von Ventil- und Pumpendaten zu nennen, die entsprechend der Beschreibung in den vorhergehenden Abschnitten realisiert wurden. Die Verwendung von ATM in einer realen Anwendung hat, durch die intensive Zusammenarbeit mit den Anwendern, einen erheblichen Gewinn für die Einsetzbarkeit zur Erstellung der ATHLET-Eingabedaten gebracht.

Eine andere wichtige Anwendung ist die ATM Modellierung eines mit ATHLET gelieferten Beispieldatensatzes zur vereinfachten Simulation eines DWR. Dieser Datensatz wurde überwiegend durch den Einsatz der Importwerkzeuge nach ATM übernommen. Für Teile, die bisher nicht importiert werden können, gibt es in ATM frei definierbare Datenblöcke (siehe Abb. 4.32), die mit einem Standardeditor per Cut und Paste eingefügt werden können.

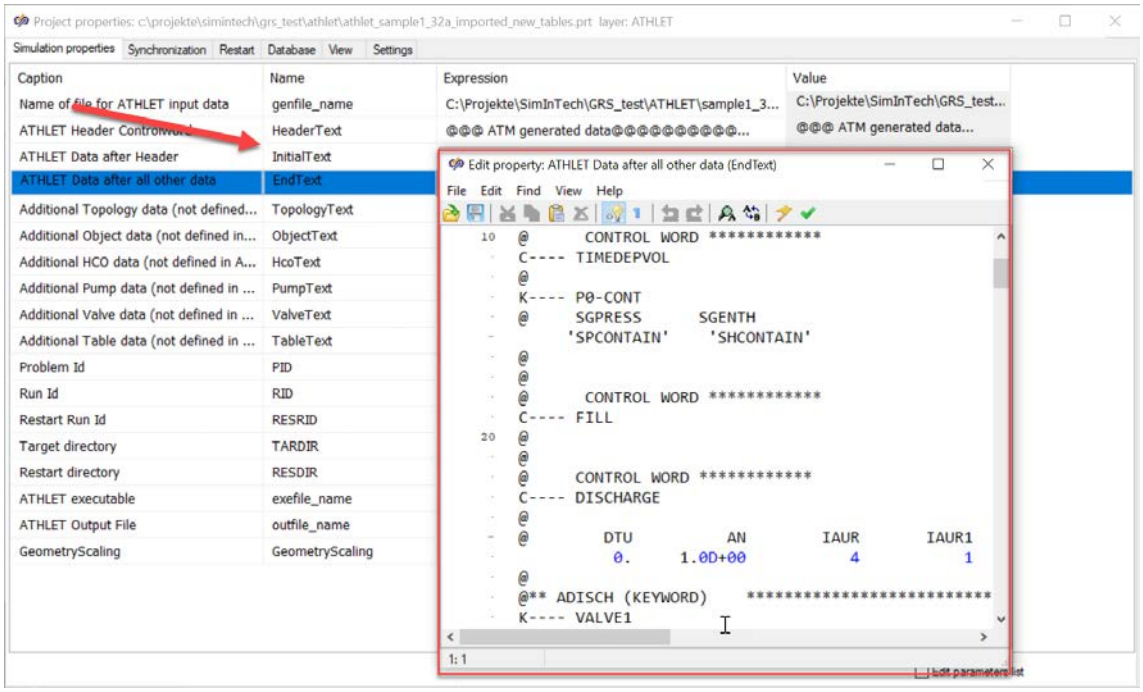

Abb. 4.34 Zusätzliche ASCII-Datenblöcke in ATM

In Kombination mit den Daten der graphischen Objekte kann ein vollständiger ASCII-Datensatz für ATHLET in ATM modelliert werden. Die Erzeugung der Eingabedatei und das Starten einer ATHLET-Simulation sind direkt aus ATM über das ATHLET Menü möglich (siehe Abb. 4.33). Der von ATM erzeugte Datensatz liefert dieselben Simulationsergebnisse wie der ursprünglich importierte Beispieldatensatz. Damit ist eine grundsätzliche Überprüfung des Datenimports möglich. Diese Prüfung muss durch weitere Beispiele mit anderen Modellen oder Eingabeoptionen sukzessive ergänzt werden.

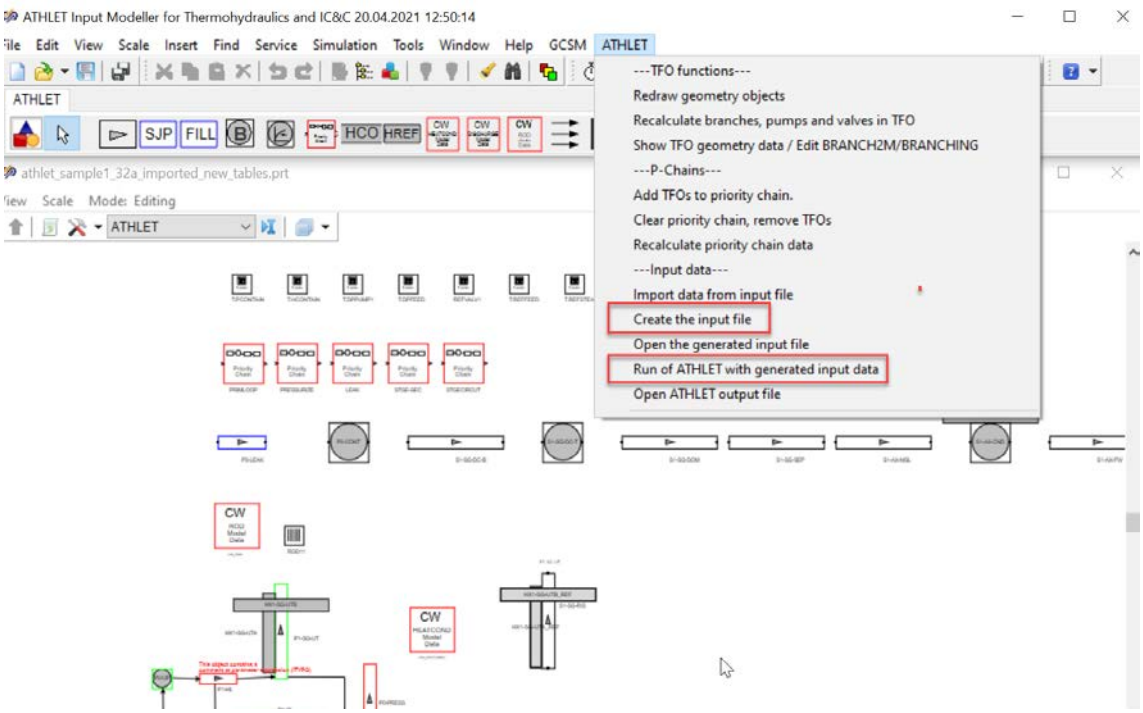

Abb. 4.35 Eingabedatei und ATHLET-Simulation über das ATHLET-Menü

4.2.4.10 Entwicklung ATHLET-CD ECORE-Eingabe

ATHLET-CD enthält eine Reihe von Modulen zur Simulation von Vorgängen bei schweren Unfällen. Das zentrale Modul ECORE beinhaltet Modelle für die Kernzerstörung, das heißt unter anderem für die Zerstörung von Brennstäben, Steuerstäben und Brennelementkästen. Die Eingabedaten umfassen dabei auch zentrale Steuergrößen, physikalische Parameter, Anfangswerte, Geometriedaten und die Kopplung der ATHLET-CD Objekte mit den Fluid- und Wärmeleitobjekten von ATHLET (TFO und HCO). Wie für die ATHLET-Objekte sollten die Geometrie und die Kopplung der Objekte graphisch dargestellt werden. Diese Funktionalität konnte bislang nur rudimentär mit Hilfe von Stub-Objekten (vgl. Abb. 4.28) realisiert werden. Die Bearbeitung dieser Objekte kann jedoch bereits durch Eingabe der Parameter in den interaktiven Dialogen vorgenommen werden. Alle weiteren Module aus ATHLET-CD wurden in diesem Arbeitspunkt nicht berücksichtigt.

4.2.4.11 Entwicklung COCOSYS THY-Eingabe

COCOSYS enthält eine Reihe von Modulen zur Simulation der Vorgänge im Containment. Das Thermohydraulik-Hauptmodul THY enthält als zentrales Modul u. a. Modelle zur Berechnung der thermodynamischen Zustände (Wasser, Dampf, Gase) im

Containment, zur Wasserstoffdeflagration, zu Druckabbausystemen und zur Abbildung von Sicherheitssystemen wie z. B. Pumpen und Kühlern, Türen und Klappen, Sprühanlagen oder passiven Rekombinatoren. Neben dem Modul THY existieren weitere Hauptmodule, die in diesem Arbeitspunkt jedoch noch nicht berücksichtigt werden können.

In einem COCOSYS Eingabedatensatz werden die Raumbereiche eines Containments oder von Betriebsräumen in Zonen (Nodes) diskretisiert. Zwischen diesen Zonen werden Verbindungen definiert, mit denen der Massen- und Energieaustausch durch Wasser- und Dampf-/Gasströme berücksichtigt wird. Aufgrund dieser Zusammenhänge kann ein Eingabedatensatz sehr gut durch ein Netzwerk aus Zonen und ihrer Verbindungen repräsentiert werden. Solche Netzwerke, wie beispielhaft in Abbildung 4.36 dargestellt, sollte in ADM interaktiv erzeugt und bearbeitet werden können. Die Daten der COCOSYS-Zonen und den Verbindungen werden als Objektparameter vorgegeben. Da die geometrischen Daten häufig in tabellarischer Form vorliegen, war für die eine Importmöglichkeit vorgesehen. Wichtige Parameter, wie Anfangswerte, sollten direkt als Zahlenwert in der graphischen Netzwerkdarstellung angezeigt werden. Ferner sollten die geodätischen Höhen und Volumina der Zonen im Schema ersichtlich sein, um fehlerhafte Daten oder unzulässige Verbindungen leichter erkennbar zu machen.

Wie auch für ATHLET-CD ist bislang nur ein rudimentärer Import von COCOSYS-Datenblöcken über Stub-Objekte sowie deren manuelle Nachbearbeitung möglich. Die Implementierung einer automatischen Prüfung der Daten auf Vollständigkeit, Plausibilität und Konsistenz war zwar im Rahmen dieses vorgesehen, konnte jedoch noch nicht umgesetzt werden.

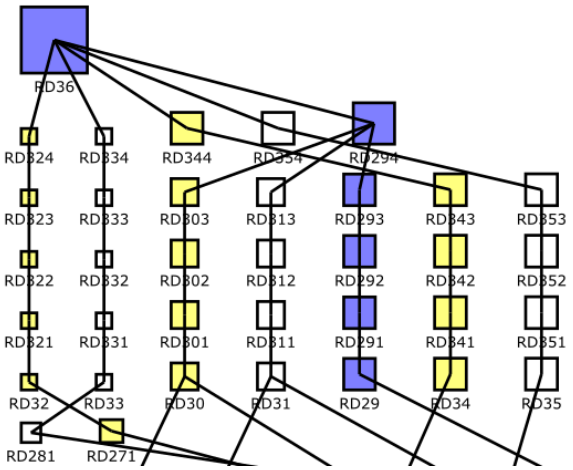

Abb. 4.36 Beispiel für eine Netzwerktopologie in COCOSYS

Wie oben beschrieben kann die Spezifikation eines COCOSYS-Eingabedatensatzes schematisch in ADM als abstraktes Netzwerk aus Zonen und Verbindungen modelliert werden. Die Lage der Zonen wird häufig in Schnitten des Gebäudes manuell dokumentiert, welche meist als Bitmap vorliegen (vgl. Abb. 4.35). Es wäre wünschenswert dieses Vorgehen auch bei der interaktiven Eingabeerstellung zu unterstützen. Hierzu wäre es nötig, Polygone als Objektgrafiken zu erzeugen und ihre Anordnung auf einer Bitmap zu ermöglichen. Mit dieser Darstellung würde dann sowohl die automatisierte Erstellung der Zonen-Topologie als auch eine Dokumentation der Raum-Zonenzuordnung in ADM ermöglicht. Die Zonen könnten Einbauten (Strukturen) und Sicherheitssysteme enthalten. Die notwendigen Daten würden in ADM vorgegeben werden können und die Zuordnung zu den Zonen, soweit möglich, als grafische Objekte dargestellt. Die Implementierung dieser Funktionalität muss nach weiterer Überprüfung und Verfeinerung des Konzepts im Rahmen eines zukünftigen Projekts bewerkstelligt werden.

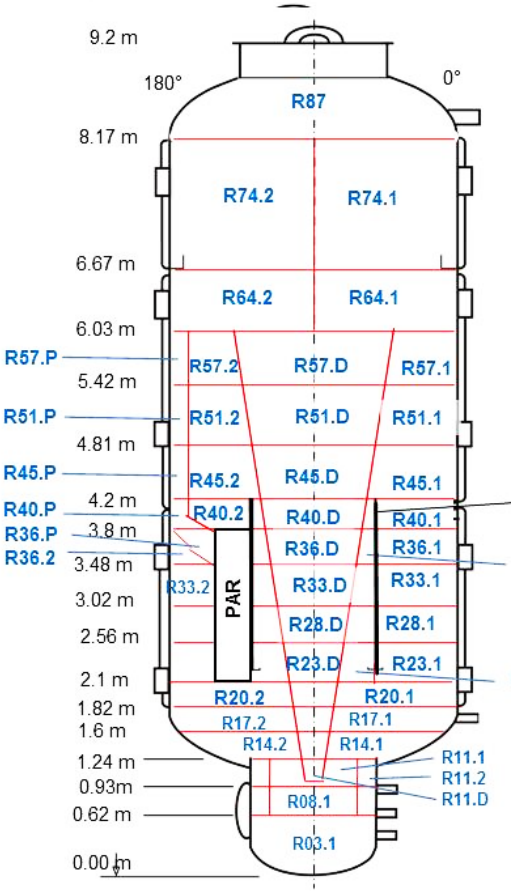

Abb. 4.37 Beispiel für die Diskretisierung der THAI Versuchsanlage in COCOSYS

4.2.5 Qualitätssicherung und Dokumentation

Über die in AP 5 beschriebenen allgemeinen Maßnahmen zur Qualitätssicherung hinaus, wurden die folgenden Arbeiten durchgeführt, welche speziell in der ADM-Entwicklung notwendig waren.

4.2.5.1 Refactoring des Quellcodes

Da die Entwicklung schon einen Zeitraum von ca. 10 Jahren umfasst, wurden im objektorientierten Delphi-Programmcode /EMB 15/ viele Änderungen durchgeführt, mit dem Ziel, den Code lesbarer, wartungsfreundlicher und robuster gegen Fehler zu machen. Insbesondere wurde das Design der Objektklassen vollständig überarbeitet. Damit konnten unter Nutzung von Vererbungsmechanismen Funktionen zentralisiert werden und viele Stellen von sehr ähnlichem, doppeltem Code eliminiert werden. Außerdem wurde eine neue Modularisierung des Quellcodes durch eine Aufteilung in Dateien und Verzeichnisse durchgeführt (siehe Abb. 4.36). Diese können, je nach Umfang, komplette Klassen, Teile von Klassen wie Funktionen, Properties, Datentypen, Editoren usw. enthalten Beispiele sind

- ATHLET-Objektklassen (TFO, HCO, PC, …)
- Datenexport- und -importklassen
- ATHLET-Datenklassen (Drift, Friction, Master, …)
- Zentrale Funktionen (Ausgabeformatierung, Fehlerbehandlung, …)

Insgesamt konnte mit dem durchgeführten Refactoring der Umfang des Codes reduziert und die Verständlichkeit verbessert werden.

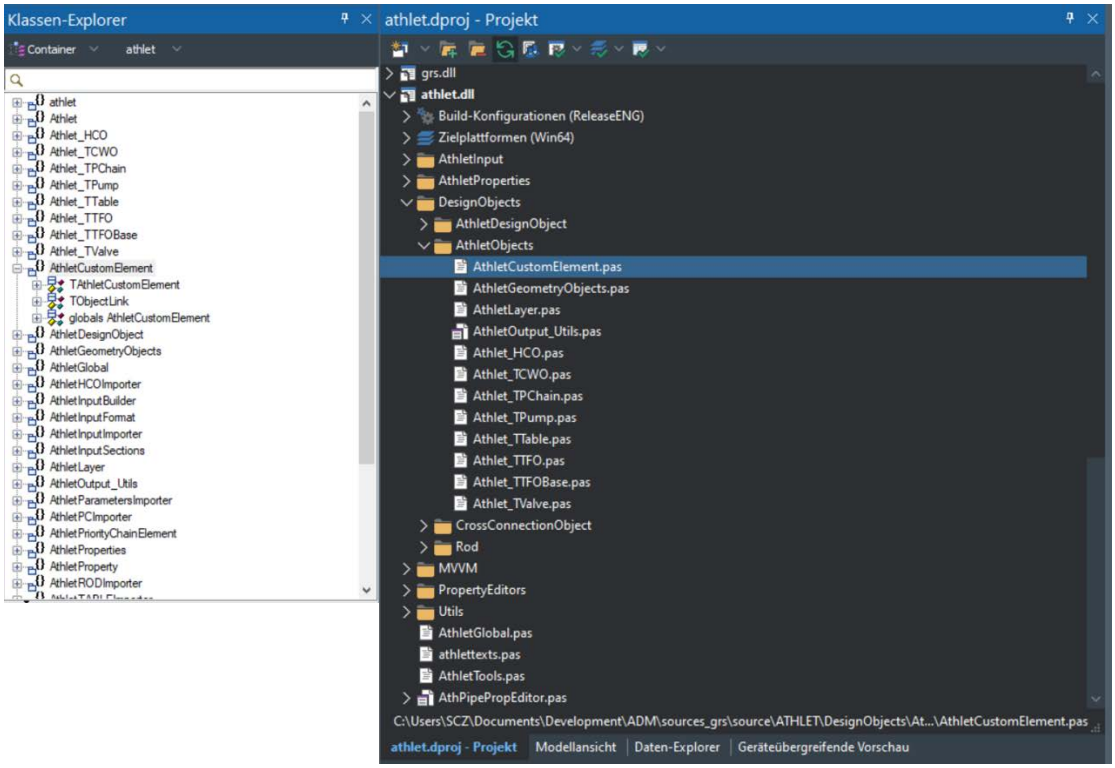

Abb. 4.38 Klassen und Verzeichnisstruktur in Embarcadero Studio (Delphi 10.3)

4.2.5.2 Modernisierung der ADM-Codeverwaltung durch GitLab

Die Verwaltung des Sourcecodes und weiterer Programmteile (Konfigurationsdateien, Dokumentation etc.) von ADM mit Git /GIT 21/ wurde in GitLab /GIL 21/ der GRS vollständig reorganisiert. Das Repository wurde aufgeteilt in eigenständige Repositories für Entwicklung, Komponenten, Anwendung und Dokumentation. Große Dateien werden im Large File System (LFS) von Git verwaltet, um den Zugriff zu optimieren. Außerdem können die Nutzer in GitLab „Issues“ erstellen, in denen Probleme oder Vorschläge zur Erweiterung von ATM erfasst werden. Damit ist es beispielweise möglich, Diskussionen und den Bearbeitungsstand der Probleme zu verfolgen oder Softwareänderungen (Commits) mit diesen zu verknüpfen. Die Bearbeitung eines Problems und die zu dessen Lösung durchgeführten Maßnahmen können direkt im jeweiligen Issue dokumentiert werden und sorgen für eine gute Nachvollziehbarkeit der durchgeführten Änderungen. Zusätzlich werden Informationen zu ADM und dessen Entwicklung im WIKI von GitLab bereitgestellt und kontinuierlich ergänzt. Insgesamt bietet der Einsatz von GitLab eine bessere Planbarkeit und Dokumentation der ADM-Entwicklung.

4.2.5.3 Automatisierung des ADM-Build durch GitLab-CI

Die automatisierte Kompilierung und Bereitstellung der ADM-Softwarekomponenten im Rahmen der auf GitLab zur Verfügung gestellten CI/CD, konnte nun auch Server-seitig erfolgreich durchgeführt werden. Dadurch kann nun die volle von GitLab bereitgestellte Funktionalität der CI/CD (Continuous Integration / Continuous Deployment) genutzt werden, um das automatisierte Erstellen und Testen des gesamten Anwendungspakets, wo möglich zu realisieren. Durch die vorangegangene Integration in GitLab wurden einige Änderungen notwendig. Dabei wurde der Installer von Visual Studio Installer auf den einheitlich verwendeten Installer (InnoSetup) umgestellt. Auch die Bereitstellung der Dokumentation über die Softwarekomponente FastHelp /FAH 21/ musste aufgrund der fehlenden Kompatibilität zum automatisierten Ablauf in der GitLab-CI umgestellt werden. Dabei wurden die Dokumentationsinhalte in das Markdown-Format übertragen, welches jetzt mit Pandoc /PAD 22/ automatisiert in gängige und zeitgemäße Dokumentationsformate (wie HTML oder PDF) übersetzt werden kann. Um die Erstellung der Image-seitigen Embarcardero-Entwicklungsumgebung weiter zu automatisieren, wurde die Neuauflage („pyscripter/MultiInstaller“; /MID 22/) des Silverpoint MultiInstaller getestet. Dieser ermöglicht die automatisierte Installation von Delphi-Plugins via Git-Repositories, Zip-Dateien als auch bestehenden Plugin-Ordnern. Leider konnte dieser Test nur unzufriedenstellend abgeschlossen werden. Dennoch konnte der zuvor benötigte, mühevolle Installationsprozess der Embarcadero Packages beziehungsweise Komponenten (Jedi Component Library (jcl), Jedi Visual Component Library (jvcl) etc.), durch das Einbinden gängiger GitHub-Repositories und das Ausführen eines Python-Skripts welches die benötigten Delphi-Bibliothekspfade in das XML-basierte (EnvOpts.proj [AppData\Roaming\Embarcadero\BDS\20.0 bei Delphi 10.3]) Embarcadero-Projektfile einträgt, erheblich reduziert werden.

4.2.5.4 Dokumentation und Deployment von ADM

Für eine leichtere Durchführung von Anwendungstests wurde ein eigenes Repository für die Anwender („ADM/Distribution“) mit verschiedenen Entwicklungsschritten von ADM eingerichtet. In dieses Repository können kurzfristige Änderungen eingestellt werden, wie sie häufig für eine zeitnahe Fehlerbeseitigung nötig sind. Die Nutzer können durch einen „Pull“ dann sehr einfach, ohne weiteren Installationsaufwand auf die Änderungen zugreifen. Sollten sich dadurch Probleme ergeben, ist eine einfaches „Reset“ auf ältere Versionen möglich.

Die Entwickler-Dokumentation des Programms erfolgt überwiegend direkt im Quellcode. Zusätzliche Erläuterungen von Entwicklungsdetails sind in speziellen Dokumenten in einem eigenen Repository („ADM/HowTo“) abgelegt. Diese Beschreibungen bieten eine Grundlage vorhandenes Know-how zu erhalten und an neue Entwickler weiterzugeben.

In der Anwender-Dokumentation wird die Anwendung von ADM im Detail beschrieben. Im Allgemeinen wird ein kurzer Überblick zum jeweiligen Objekt gegeben, teilweise mit Hinweisen zur Verwendung oder Bedienung. Falls Eingabeparameter für ein Objekt erklärt werden müssen, ist die ATHLET-Eingabebeschreibung eingebunden, die ebenfalls im HTML-Format verwendet wird. Dieses Vorgehen vereinfacht die Nutzung einer jeweils aktuellen Eingabebeschreibung von ATHLET.

4.3 AP 3: Simulations-Durchführung

ATLASneo bietet bereits die Möglichkeit, eine Simulation direkt zu starten und den Lauf sowohl online zu überwachen als auch interaktiv zu beeinflussen. Der ATHLET-Code wird dabei als Shared Library in den ATLASneo-Prozess eingebunden und als Funktion aufgerufen, wodurch sich ein detaillierter Datenzugriff und die vielfachen Steuermöglichkeiten ergeben (vgl. Abschnitt 4.1.2). Die Grenzen dieser Methode liegen aktuell darin, dass der ATHLET-Code durch seinen statischen Aufbau innerhalb eines Prozesses nicht ablaufinvariant (reentrancy) betrieben werden kann, was bedeutet, dass es in ATLASneo immer nur eine aktive ATHLET-Simulation geben kann. Des Weiteren kann eine Simulation somit nur auf demselben Rechner laufen, auf dem auch ATLASneo gestartet wurde, was auch das Starten und Überwachen von Cluster-Simulationen ausschließt. Um diese Einschränkungen zu beheben, musste eine Ansteuerung erarbeitet werden, welche die Simulation in einen eigenständigen Prozess auslagert und die detaillierten Möglichkeiten des Datenzugriffs und der Steuerung über einen Client-Server-Ansatz implementiert. Das in ATLASneo bestehende Funktionsmodul, welches zum Start und zur Steuerung von ATHLET-Simulationen entwickelt wurde, sollte erweitert werden, um eine einheitliche Benutzeroberfläche für verschiedene Simulationscodes bereit zu stellen. Zusammenfassend ergaben sich für diesen Arbeitspunkt die im Folgenden erläuterten Themengebiete.

4.3.1 Auslagerung des Simulationsprozesses

Der für das Starten der Simulation verwendete Programmteil (Controller) war bisher sehr auf ATHLET zugeschnitten und musste generalisiert werden, um diesen in einem eigenständigen Python-Prozess einbetten zu können. Wie in Abbildung 4.39 grafisch dargestellt, wurde in einem ersten Schritt der Einhänge-Mechanismus des Controllers überarbeitet, um sowohl allgemeine Namen von Simulations-Ereignissen (Hooks) als auch, wie bisher, die ATHLET-spezifischen Namen der Callback-Einhängepunkte verwenden zu können. Daneben mussten auch einige Umstellungen vorgenommen werden, welche die Verwendung des Controllers unter den Python-Versionen 2 und 3 /PYT 22/ erlauben.

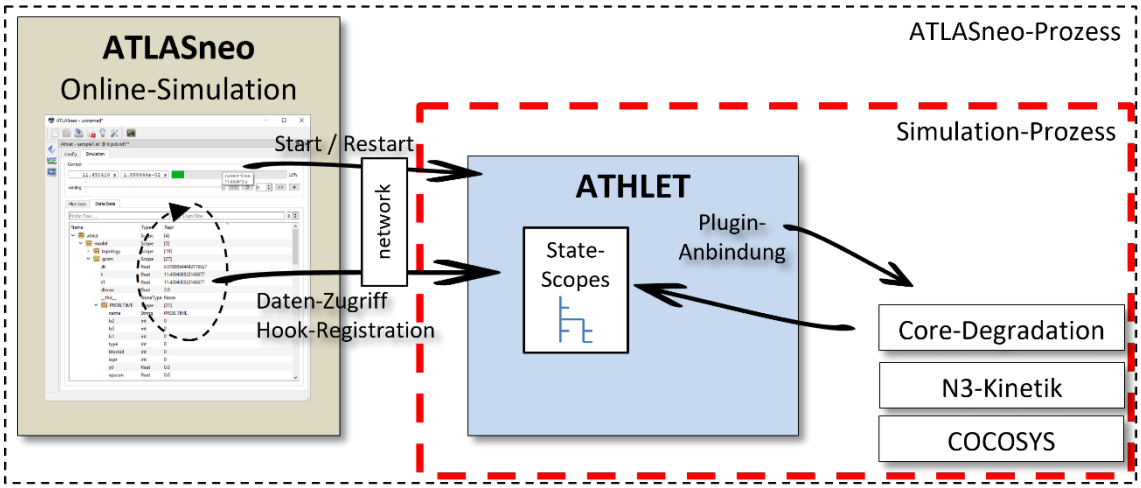

Abb. 4.39 Darstellung der Simulationssteuerung über ATLASneo

Der rot markierte Bereich zeigt beispielhaft den ATHLET-Lauf als ausgelagerten Prozess. Eine eigens implementierte Netzwerk-Schnittstelle steht bereits zur Verfügung

Auch die Arbeiten für die notwendigen Erweiterungen der Simulations-Steuerungsfunktionen wurden weitergeführt. Hiermit erlaubt der ATHLET-Controller bereits den grundlegenden Zugriff auf die Daten von TFO- und HCO-Strukturen der Simulation mittels Objekt-Namen und Node-Nummer. Der bisherige Direktzugriff auf die entsprechenden Datenarrays des Simulators, welcher bei eigenständigem Simulationsprozess ohnehin nicht mehr möglich ist, ist damit nicht mehr notwendig. Dieser kann jetzt durch den namensbasierten Zugriff ersetzt werden, welcher sich durch seine Abstraktion deutlich besser eignet, um ihn über die limitierte Schnittstelle ausgelagerter Prozesse durchzuführen.

Das Einladen von als dynamische Bibliothek vorliegenden Simulationscodes in eigenständige Prozesse und die schrittweise Durchführung von Simulationen konnte in Form

eines dedizierten Controllers implementiert werden. Dazu mussten an den bislang bestehenden, zum Starten und Steuern notwendigen Programmteilen weitere Anpassungen vorgenommen werden, um die Verwendung unter neueren Python-Versionen (3.8 und 3.9) zu ermöglichen. Eine rudimentäre Steuerung kann dabei bereits über eine eigens implementierte Netzwerk-Schnittstelle durch die Verwendung eines Kontroll-Ports erfolgen. Um den Programmteil des ATHLET-Controllers als eigenen Prozess ausführbar zu machen und hierzu standardisierte Schnittstellen anzubieten, wurde am Ansatz weitergearbeitet, den Controller als IPython-Kernel zu implementieren. Erste Testimplementierungen haben gezeigt, dass sich diese Methode sehr gut für die Ausführung ausgelagerter Prozesse eignet und durch die vielseitig unterstützten Schnittstellen leicht in andere Umgebungen und Anwendungen integriert werden kann. Somit sollte für die mittelfristige Weiterentwicklung vorrangig dieser Ansatz weiterverfolgt werden. Der Erhalt der bisherigen Steuermöglichkeiten konnte bislang noch nicht implementiert werden, weshalb dies ein primärer Fokus des Folgevorhabens sein wird. Auch soll in den weiteren Entwicklungsschritten dieser Ansatz für die generische Anwendbarkeit ausgebaut werden, um auch zum Start komplexer Simulationsläufe, wie z. B. COCOSYS, benutzt werden zu können. Die Verwendung von IPython-Kernels ist vielversprechend, seine Anwendbarkeit kann allerdings aufgrund der nicht mehr gegebenen Kompatibilität der benötigten packages zu Python 2 und Python 3 nicht mehr versionsübergreifend realisiert werden. Deshalb werden diese Entwicklungen erst im Folgevorhaben, nach Abschluss der Umstellung von ATLASneo auf Python 3 durchgeführt, um die zusätzlich zu erwartenden Portierungsarbeiten so gering wie möglich zu halten.

4.3.2 Weiterentwicklung des Funktionsmoduls Simulationssteuerung

Basierend auf den bereits bestehenden Controllerklassen wurde der Ausbau der Simulations-Steuerungsfunktionen über Trigger und Actions weiter vorangetrieben. Aufgrund der bereits weitläufigen Anwendung dieser Klassen musste hierbei auf den Erhalt der Python-2-Kompatibilität geachtet werden. Es wurden verschiedene Arbeiten und Bugfixes am Simulations-Controller für ATHLET/-CD vorgenommen. Unter anderem musste hier die thread-Synchronisation zwischen Steuer- und Controller-Code überarbeitet werden, um diese mit dem etwas veränderten Simulationsablauf beim Einladen von Restartpunkten kompatibel zu machen. Darüber hinaus konnte ein Problem behoben werden, welches unter Umständen den Start bzw. Neustart von Online-Simulationsläufen verhinderte. Hierzu wurde die Prozedur für Aufräumarbeiten nach dem Beenden einer Simulation angepasst. Diese stellt jetzt sicher, dass File-Channels explizit vom Python-

Controller geschlossen werden, welche vom nativen Simulationscode beim Start zwar geöffnet, bei unerwartetem Simulationsende, z. B. Abbrüchen, jedoch geöffnet bleiben. Dadurch können Online-Simulationen jetzt immer wieder neu gestartet werden, welche vorher aufgrund blockierter File-Channels oft fehlschlugen. Demnach kann nun sowohl die Restart-Funktionalität von ATHLET als auch Controller-Protokolldateien über die grafische Benutzeroberfläche verwendet werden, wie in Abbildung 4.40 zu sehen ist.

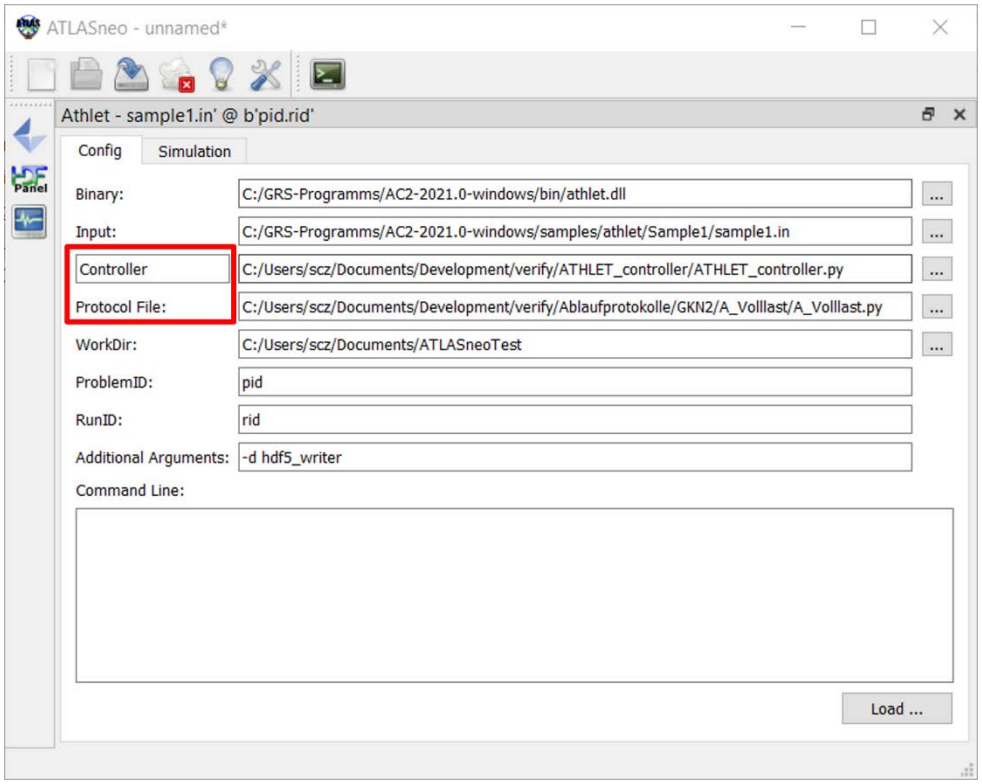

Abb. 4.40 Monitor-Widget zur Simulationssteuerung in ATLASneo

Rot markiert sind die beiden Felder, die es dem Anwender ermöglichen Controller und zugehörige Protokolle einfach über File-Dialoge auszuwählen.

Auch wurde die von Python zur Verfügung gestellte print-Funktion als Action implementiert. Diese erleichtert die Anwendbarkeit und Automatisierung von Simulations-Steuerungsmaßnahmen durch Protokolle. Für die hierin definierten Trigger und Actions wird es damit möglich print-Ausgaben zu verwenden. Dies erleichtert das Überprüfen der automatisierten Steuerungsmaßnahmen erheblich und erlaubt die einfache Verfolgung des Simulationslaufs über die Konsolenausgabe. Eine Generalisierung und Vereinfachung der Definition von Triggern und Actions sollte weiterhin angestrebt werden, um deren Widerspruchsfreiheit und eine einfache Anwendbarkeit zu gewährleisten.

4.4 AP 4: Ergebnis-Analyse und Visualisierung

Die Entwicklungen zur Anwenderunterstützung im Bereich der Durchführung von Simulationsläufen und der Auswertung von Simulationsergebnissen werden in der Anwendung ATLASneo zusammengefasst. Durch die im Projekt geplanten Entwicklungsarbeiten sollte ATLASneo weiter ausgebaut werden, um sowohl die AC²-Rechencodes zu unterstützen als auch zukünftig eine Basisanwendung bieten zu können, in die weitere Codes und Visualisierungstools einfach integriert werden können. Um die Modularität und Wiederverwendbarkeit sicherzustellen und somit die weitere Entwicklung zu erleichtern, wurde die Anwendung in einer Schichten-Architektur aufgebaut.

Abbildung 4.41 gibt einen Überblick über die verschiedenen Schichten und die jeweiligen Bestandteile. Der Anwendungsrahmen (Application Frame) bildet die oberste Verwaltungsebene, die die enthaltenen Funktionsmodule (Gadgets) organisiert. In ATLASneo stehen diese dann als dockbare Fenster zur Verfügung, die aus Komponenten der darunter liegenden Schichten (Panels + Widgets) aufgebaut sind und bilden die graphische Benutzeroberfläche für die in der Methoden-Schicht (Methods/Tools) vorhandenen Funktionalität (Methoden). Dadurch, dass diese getrennt von den Bestandteilen der Darstellungsschichten (Presentation-Layers) gehalten werden, können die Methoden auch in anderen Umgebungen, z. B. Konsolenanwendungen, eingesetzt werden.

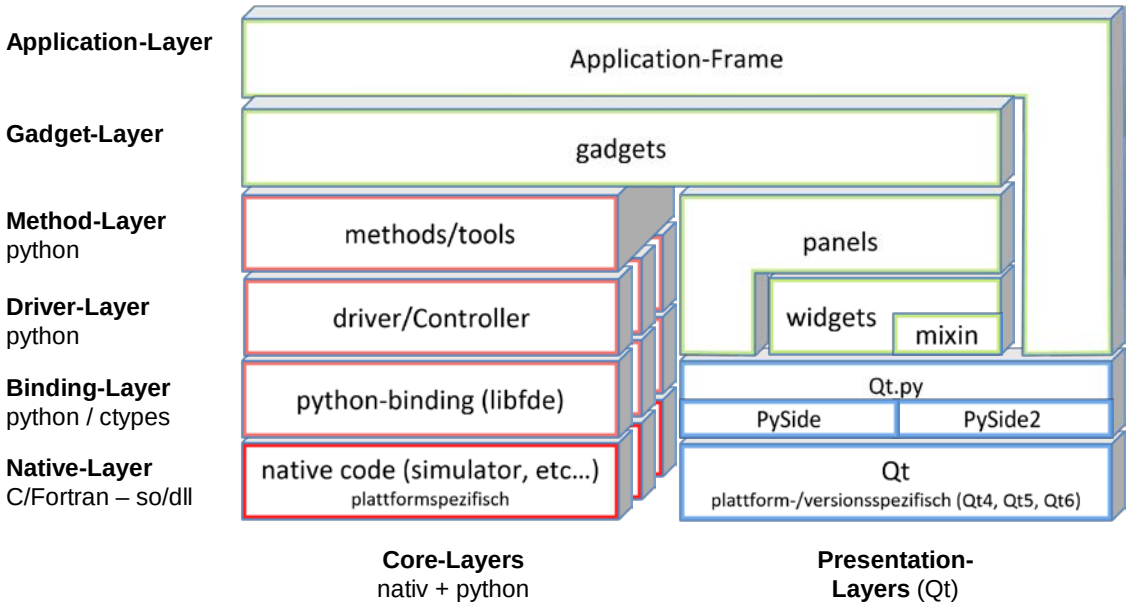

Abb. 4.41 Anwendungs-Architektur von ATLASneo

Zur Sicherstellung von Modularität und Wiederverwendbarkeit, ist das Framework in Schichten unterteilt, welche die Kernfunktionalität von der Benutzeroberfläche trennt.

Für die Entwicklungen im Projekt musste sowohl an der Architektur als auch an den einzelnen Komponenten weitergearbeitet werden. Die dazu durchgeführten Arbeiten werden in den folgenden Abschnitten beschrieben.

4.4.1 Weiterentwicklung des ATLASneo-Anwendungsrahmens

Der Anwendungsrahmen von ATLASneo musste für die zusätzlich unterstützten Rechencodes und die Portierung von Python 2 auf Python 3 weiterentwickelt werden. Durch hinzukommende Funktionsmodule und Darstellungsarten ergaben sich weitere Einstellungsmöglichkeiten und Parameter, die im Konfigurationsdialog verwaltet werden müssen. Hierzu wurde eine hierarchische Konfigurationsstruktur mit den Ebenen „Sitzung“, „Benutzer“ und „Applikation“ implementiert. Um eine über die gesamte Applikation einheitliche Behandlung von Parametern zu unterstützen, wurde ein auf das Qt-Delegate Konzept basierende typspezifische Behandlung von Konfigurationsgrößen entwickelt. Die jeweiligen Arbeiten und notwendigen Schritte sind im Folgenden näher erläutert.

Portierung Python 2 auf Python 3

Aus lizenztechnischen Gründen nutzte ATLASneo einige Softwarepakete, welche bis vor einiger Zeit nur für Python 2 vorhanden waren. Mittlerweile gibt es aber verfügbare Alternativen oder die jeweiligen Pakete wurden auf die aktuelle Sprach-Version 3 portiert. Daher konnte nun auch ATLASneo auf Python 3 aktualisiert werden, ein Entwicklungsschritt, der durch das Entwicklungsende von Python 2 (seit Januar 2020) noch weiter an Bedeutung gewonnen hatte. Das betraf vor allem das für die Qt-Benutzeroberfläche zuständige Package „PySide“/PYS 17/, welches jetzt nur noch indirekt über die Kompatibilitäts-Schicht Qt.py angesprochen wird. Diese übersetzt unter Python 3 alle nötigen GUI-Aufrufe auf das dort verfügbare Package „PySide2“ /PYS 22/ (vgl. Abb. 4.39). Auch das bisher nicht verfügbare automatische Laden und Übersetzen von UI-Designdateien konnte überarbeitet werden und erleichtert jetzt die Erstellung von Gadget-Benutzeroberflächen. Die Umstellung auf Python 3 betraf auch die Startskripte von ATLASneo. Da diese für den automatischen Download, die Installation und die Aktivierung der notwendigen Python-Umgebung zuständig sind, mussten die Skripte erweitert und auf eine neue Package-Zusammenstellung angepasst werden. Der erreichte Stand erlaubt nun eine automatische Installation und einen Start von ATLASneo unter beiden Python-Versionen. Für das reibungslose Zusammenspiel der Komponenten mussten jedoch sowohl im Anwendungsrahmen als auch in den Funktionsmodulen weitere Inkompatibilitäten

behoben werden. Diese betrafen vor allem Codeteile, die für die Erzeugung der grafischen Benutzeroberfläche zuständig sind und wurden oft durch Unterschiede der eingesetzten Qt-Binding PySide (für Python 2.7) und PySide2 (für Python 3.x) verursacht. Das inzwischen auch für Python 2 verfügbare PySide2-binding wurde getestet, zeigte sich im weiteren Entwicklungsverlauf allerdings als noch zu instabil und fehlerhaft, um für ATLASneo generell als Qt-Binding eingesetzt werden zu können. Daher wird bislang noch die Qt.py-abstrahierte Lösung und der duale Einsatz von PySide vs. PySide2 verwendet. Die Entwicklungen der Kernfunktionalität von ATLASneo (Anwendungsrahmen, Monitor-Widget, HDF5 und PlotPanel) konnte als AC² 3.3-kompatible Milestone-Version abgeschlossen werden. Diese ist die letzte Referenzversion, die unter beiden Python-Versionen (2/3) betrieben werden kann. Für zukünftige Erweiterungen und neuere Entwicklungen ist geplant, die explizite Unterstützung von Python 2 nicht mehr zu gewährleisten.

Hierarchische Konfigurationsstruktur

Die möglichen Konfigurationseinstellungen sowohl des Anwendungsrahmens als auch die der einzelnen Funktionsmodule (Gadgets) werden in einer gemeinsamen hierarchischen Struktur abgelegt, um diese mit einem „Konfigurations-Gadget“ gemeinsam, zentral und einheitlich verwalten zu können. Für die hierarchische Datenstruktur stehen drei Hierarchieebenen zur Verfügung, wie sie in Abbildung 4.42 zu sehen sind. Damit können Standardeinstellung aus der Anwendungsinstallation getrennt von anwenderspezifischen Einstellungen und diese wiederum getrennt von Einstellungen der Sitzung verwaltet und gespeichert werden. Hierdurch ist es möglich anwender- und sitzungsspezifische Einstellungen in späteren Sitzungen einfach und unabhängig einzuladen.

Das Einlesen der Konfigurationsdaten zu Beginn der Sitzung wird durch den Anwendungsrahmen veranlasst. Die interne Datenhaltung erfolgt dabei zentral im Anwendungsrahmen und der hierin enthaltenen Klasse „ConfigBroker“, sodass Änderungen in der Konfiguration sowohl dem Anwendungsrahmen als auch sämtlichen Funktionsmodulen zur Verfügung stehen. Die Kommunikation des Konfigurations-Gadgets mit den verschiedenen Anwendungsteilen wurde durch den "Signal-Slot-Mechanismus" (Qt-Signals) realisiert. Die Änderung eines Konfigurationsparameters durch den Anwender wird dabei an den Anwendungsrahmen signalisiert, welcher diese direkt an alle aktiven Gadgets weiterleitet. Die vom jeweiligen Parameter beeinflussten Gadgets können dann darauf reagieren, den neuen Wert abfragen und diesen geeignet verarbeiten.

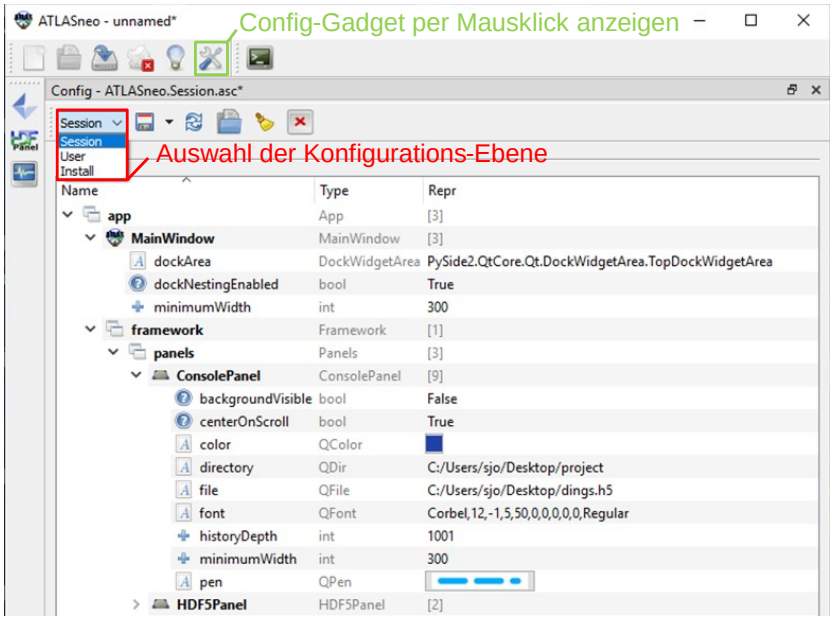

Abb. 4.42 Config-Gadget in ATLASneo

Über eine hierarchische Konfigurationsstruktur können voneinander unabhängige Einstellungen mit den Ebenen „Sitzung“, „Benutzer“ und „Installation“ realisiert werden.

Über diesen Mechanismus werden Änderungen in den betroffenen Anwendungsmodulen bei Bedarf sofort aktiv. In einigen Modulen wurden bereits verschiedene Konfigurationsparameter implementiert, die sich somit über diesen Mechanismus einstellen lassen. Die zur Verwaltung der Konfigurationsparameter eingesetzte, hierarchisch organisierte Datenstruktur wurde vom Konfigurations-Gadget getrennt und sehr generisch implementiert und kann somit auch für andere Funktionsmodule eingesetzt werden. Das Konfigurations-Gadget selbst wurde soweit möglich aus bereits bestehenden Bausteinen für Funktionsmodule („Panels“) erstellt. Sofern notwendig, wurden dazu bestehende Panels erweitert oder neu entwickelt. Ferner wurden dem Konfigurations-Gadget benötigte Funktionalitäten, wie z. B. zum Laden und Speichern der Konfigurationsparameter aus bzw. in Dateien, hinzugefügt. Als Datenformat wurde dazu das standardisierte Format JSON genutzt. So konnte auch ein robuster Ansatz zur Serialisierung von Qt-basierten Einstellungsgrößen realisiert werden, wodurch jetzt auch sämtliche Gadget-Einstellungen zu Schriftarten, Farben und Linienstilen in den Benutzereinstellungen abgespeichert und beim späteren Öffnen von Sessions wiederhergestellt werden können.

Qt-Delegates

Durch erste praktische Benutzung des Konfigurations-Gadgets zeigte sich die Notwendigkeit den Anwender auch bei der Einstellung komplexer Größen, die sich aus mehre-

ren einfachen Größen zusammensetzen, interaktiv zu unterstützen. Zwar können praktisch alle Einstellungen in Form von numerischen Werten oder Zeichenketten eingegeben werden, jedoch erwarten Anwender hier typischerweise Dialoge, die eine interaktive Auswahl, von z. B. Verzeichnispfaden, Farben oder Schriftarten erlauben. Erste Testimplementierungen des Qt-Delegate-Konzepts erlaubten bereits im Plot-Gadget die Farben und Stricharten der Graphen auszuwählen. Durch die weitere Ausarbeitung des Delegate-Konzepts und die Implementierung Typ-spezifischer Darstellung und Auswahldialoge können dem Anwender nun viele Einstellungen ermöglicht und über die ganze ATLASneo-Benutzeroberfläche hinweg einheitlich angeboten werden.

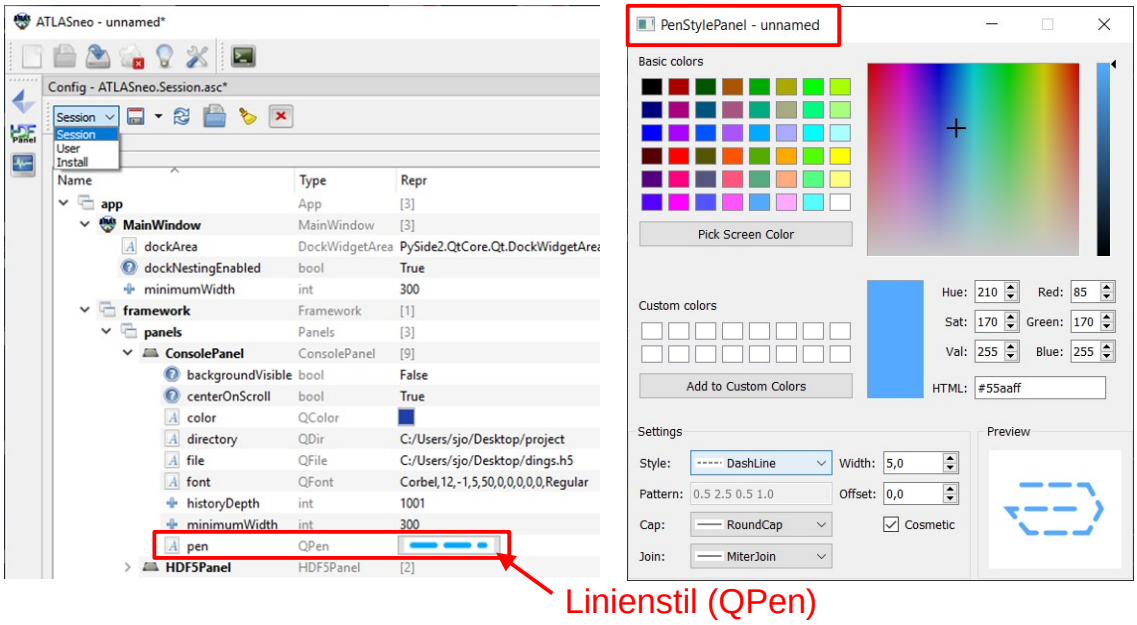

Abb. 4.43 Delegate-Konzept in ATLASneo

Ein Doppelklick auf einen dargestellten Linienstil öffnet den zugehörigen Konfigurationsdialog (PenStylePanel).

Abbildung 4.41 zeigt das Ergebnis der somit generalisierten Funktionsweise anhand des Config-Gadgets und der komplexen Einstellungsgröße QPen, welche alle Parameter von Linienstilen zusammenfasst. Da es für den Anwender weder hilfreich noch praktikabel ist, all diese Parameter getrennt voneinander angezeigt zu bekommen und diese somit auch einzeln einstellen zu müssen, werden sie nun vielmehr als Linienstil graphisch zusammengefasst dargestellt und können in einem geeigneten Dialog (PenStylePanel) interaktiv konfiguriert werden.

Im weiteren Entwicklungsverlauf sollte geprüft werden, inwieweit die typspezifische Behandlung von Einstellungen in früher implementierten Funktionsmodulen genutzt werden

kann, um deren Handhabung zu erleichtern und an die des Config-Gadgets anzugleichen. Ein Beispiel ist hier das Plot-Gadget, welches durch seine Vielzahl an Parametern noch reichlich Einsatzmöglichkeiten für Qt-Delegates bietet. Unter anderem sind das die Einstellungen zu den Abszissenachsen und dem formatierbaren Legendentext, welche dadurch zusammen mit den anderen Funktionen einheitlich konfigurierbar wären. Auch könnte dazu die Unterstützung für weitere Parametertypen implementiert und z. B. um eine Auswahl für die bisher nicht unterstützen Enum-Typen erweitert werden.

4.4.2 Verwendung mehrerer Datenquellen

Da als zentrales Datenformat für die Speicherung von Simulationsdaten auf das für große Datenmengen ausgelegte Format HDF5 gesetzt wird, wurde das Konzept für das Einlesen und Konvertieren tabellarischer Daten (Zeitreihen) erweitert, um in ATLASneo auch die Daten von Simulationen ohne HDF5-Ausgabeformat oder alten Bestandsdatensätzen nutzen zu können. Neben dem bisher schon unterstützten Datenformat von ATHLET können im Konverter nun auch Ausgabedateien von COCOSYS und MELCOR verarbeitet werden. Durch die Arbeiten im Berichtszeitraum konnte die Unterstützung für das MELCOR-Format vollständig und ohne Abhängigkeit von der früher notwendigen Index-Datei implementiert werden. Die Konvertierung von COCOSYS Dateien in das HDF5-Format wurde weiter ausgearbeitet, um die grundlegenden Unterschiede zum ATHLET-Datenformat, wie die Längenänderung des Ausgabevektors (veränderliche Anzahl der Ausgabegrößen zur Laufzeit), behandeln zu können. Diese werden jetzt in HDF5 als getrennte Werte-Tabellen abgebildet, welche dann in ATLASneo identisch zu ATHLET-Ausgabedateien verarbeitet werden können. Wie sich in der praktischen Anwendung allerdings herausstellte, gibt es im COCOSYS-Ausgabeformat einige Spezialfälle, die bisher noch nicht zufriedenstellend verarbeitet werden können.

Das für das Einladen von HDF5-Dateien zuständige HDF5-Panel (siehe Abb. 4.42) wurde erweitert und ermöglicht jetzt für eine aktuell geöffnete Datei einen „reload“ durchzuführen und diese somit neu einzulesen. Diese Funktionalität ist vor allem bei einer Vorabauswertung von aktuell noch laufenden Simulationen, welche „stand-alone“ gestartet wurden, wie zurzeit noch COCOSYS-Rechenläufe, wichtig. Außerdem wurden die Routinen zum Öffnen von HDF5-Dateien überarbeitet. Diese können jetzt alternativ über den Drag-and-Drop-Mechanismus angesteuert werden und akzeptieren damit auch Dateien, welche über externe Programme, z. B. den Datei-Explorer, in das Anwendungsfenster von ATLASneo gezogen werden. Darüber hinaus wurde die Unterstützung für

die Einheitenanzeige sowie eine Vorschau für die in der HDF5-Datei enthaltenen Zeitreihen integriert. Letztere zeigt jetzt direkt in der Baumansicht den grundlegenden Zeitverlauf sowie Minima, Maxima und Mean als Tooltip an. Hierdurch können gesuchte Größen noch schneller lokalisiert und bereits grob bewertet werden.

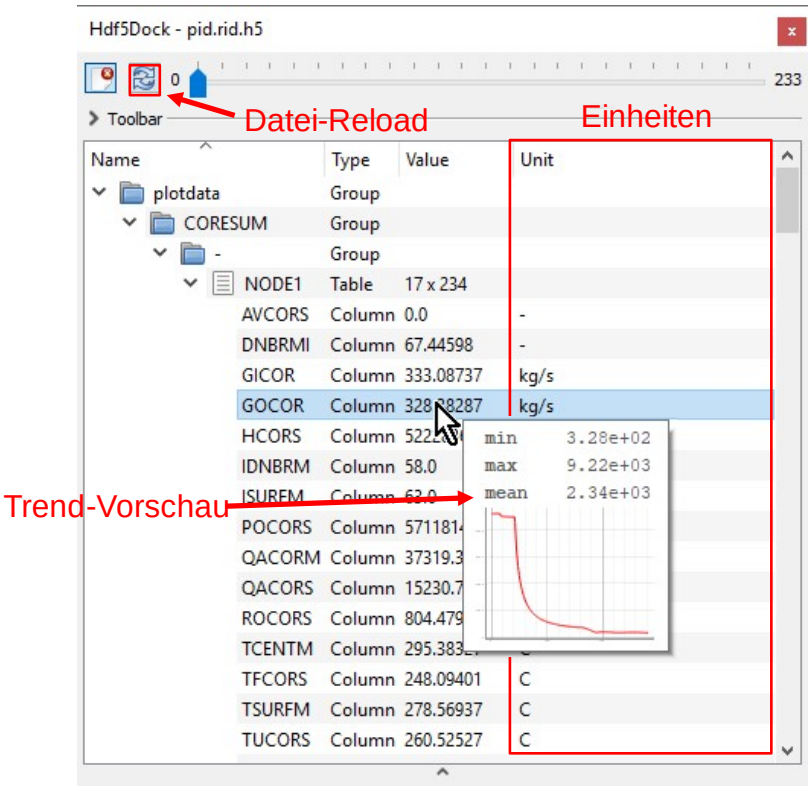

Abb. 4.44 Das HDF5-Panel in ATLASneo

HDF5-Dateien können jetzt per Drag-and-Drop geöffnet sowie über den „reload"-Button neu eingeladen werden. Vorhandene Einheiten werden als Spalte angezeigt. Der Tooltip über Werten einer Zeitreihe zeigt den Trend als Vorschau.

4.4.3 Erweiterung der Ergebnisdarstellung durch dynamisierte Bilder

Für die Visualisierung von Zusammenhängen, die über die reine zeitliche Entwicklung einzelner Ergebnisgrößen hinausgehen bietet ATLASneo das Modul zur Darstellung dynamisierter Bilder, welches Werte auf geometrische Eigenschaften eines vorbereiteten Bildes überträgt. Somit lassen sich komplexe Zusammenhänge aussagekräftig als Farbverläufe oder Veränderung in Position und Größe darstellen. Das Funktionsmodul wurde im Vorgängerprojekt auf Basis von standardisierten Webtechnologien (SVG /SCA 11/, HTML5 /HTM 08/, JavaScript /JSC 22/) komplett neu entworfen.

Um die Ergebnisvisualisierung durch dynamische Geometrie in ATLASneo weiter zu generalisieren, wurde die JavaScript library D3 getestet. Dabei wurden verschiedene Beispiele des Frameworks auf Anwendbarkeit in der Anwendungsumgebung von ATLASneo geprüft. Die hierfür notwendigen Grundfunktionalitäten umfassen zum Beispiel die Projektion aktueller Simulationsdaten auf zugeordnete SVG Geometrien, um deren Form und Farbe zu aktualisieren, als auch das Laden von extern erstellten SVG Dateien, z. B. durch AIG /KON 17/. Des Weiteren wurden erste Schritte zur dynamischen Generierung regelmäßiger Geometrien und von Kontrollelementen gemacht.

Für die Visualisierung von Ergebnisdaten aus ATHLET/-CD-Simulationen wurden die Möglichkeiten des Funktionsmoduls zum Anzeigen dynamischer Bilder noch weiter ausgebaut. Unter Verwendung der bereits getesteten JavaScript Bibliothek D3 wurde auch die Generierung spezieller Diagrammtypen weiter ausgearbeitet.

Um die Verwendung modernerer Sprachfeatures zu ermöglichen, wurde der Typescript-Compiler in den Entwicklungsprozess von Visualisierungen integriert. Typescript übersetzt JavaScript-Code in den von ATLASneo aktuell unterstützten Sprach-Standard und gewährleistet eine gewisse Abwärtskompatibilität, was den Wartungsaufwand während der Weiterentwicklung deutlich reduziert.

Des Weiteren wurde an der grundlegenden Implementierung gearbeitet, die den Python-basierten Rahmen des RenderWidgets mit dem JavaScript-basierten Teil einer geladenen Visualisierung verbindet. Durch diese sprachübergreifende Verbindung wird es möglich, mit den in der Szene dargestellten Geometrieobjekten zu kommunizieren und Daten auszutauschen. Hierüber werden dann die aktualisierten Daten, z. B. die Simulationsdaten eines neuen Zeitschritts zu deren Darstellung übertragen.

Zusätzlich wurde das React-Framework getestet, um dieses auf Funktionalität und Kompatibilität zu überprüfen. Es hat sich gezeigt, dass React bei automatischer Aktualisierung von in Web-Frames dargestellten Elementen viele Vorteile bietet und seine zukünftige Anwendung sollte für die weitere Entwicklung von webbasierten Visualisierungen in Erwägung gezogen werden.

Für die Portierung nach Python 3 und das dadurch notwendige Upgrade auf Qt5, muss das RenderWidget (QWebView/Qt4; /QWE 17/), das zum Anzeigen und dynamischen Anpassen von Web-Inhalten (JavaScript) bereitgestellt ist, grundlegend überarbeitet werden. Um die bisherige Funktionalität unter Python 2 abzubilden, wurden Tests mit

der Python 3-Version des Qt WebKit-Frameworks /WEB 22/ durchgeführt, welches jetzt die Nutzung eines sog. WebChannel-Mechanismus erfordert. Als Resultat steht in ATLASneo unter Python 3 nun ein neues Gadget mit der Renderqualität von modernen Internetbrowsern zur Verfügung.

4.4.4 Integration spezieller Visualisierungslösungen

Für die Unterstützung der ATHLET-Erweiterung Core Degradation (ATHLET-CD) in der Ergebnis-Visualisierung von ATLASneo sind spezielle Darstellungsarten erforderlich. Die in den Werkzeugen „CoreView“ und „SView“ bestehenden Visualisierungen sollen deshalb in ATLASneo integriert und erweitert werden. Die bislang erarbeiteten 2D-basierten Darstellungen (siehe Abb. 4.43) können zwar sehr gut in Form von dynamischen Bildern realisiert werden, setzen aber auch einige der in Abschnitt 4.4.3 angegangenen Weiterentwicklungen voraus. Auch für die Visualisierung von Simulationsergebnissen anderer Rechencodes, wie COCOSYS und QUABOX/CUBBOX, hat sich gezeigt, dass sich die Möglichkeiten der 2D-basierten Darstellung sehr gut eignen. Hier war zu erarbeiten, welche Arten von Bildgeometrie für eine sinnvolle Visualisierung der Ergebnisse verwendbar sind und wie diese erstellt oder ggf. generiert werden können (vgl. Abschnitt 4.4.3).

Die Abbildung zeigt das Prinzip anhand der Darstellung von ATHLET-CD-Ergebnissen in CoreView sowie auf Basis dynamisch generierter SVG-Geometrie. Im Gegensatz zur programmatisch fest implementierten Darstellung in CoreView, ist die SVG-basierte Visualisierung deutlich flexibler und kann, vergleichbar zu APG-Bildern in ATLAS, teils statisch als Datei vorgegeben, durch dynamisch generierte Elemente erweitert werden. Durch die Verwendung der Web-Technologien zur Darstellung und Dynamisierung können die Bilder sowohl im WebBrowser als auch in ATLASneo dargestellt werden und sollen zukünftig die Grundlage aller dynamischer Ergebnis-Visualisierung bilden.

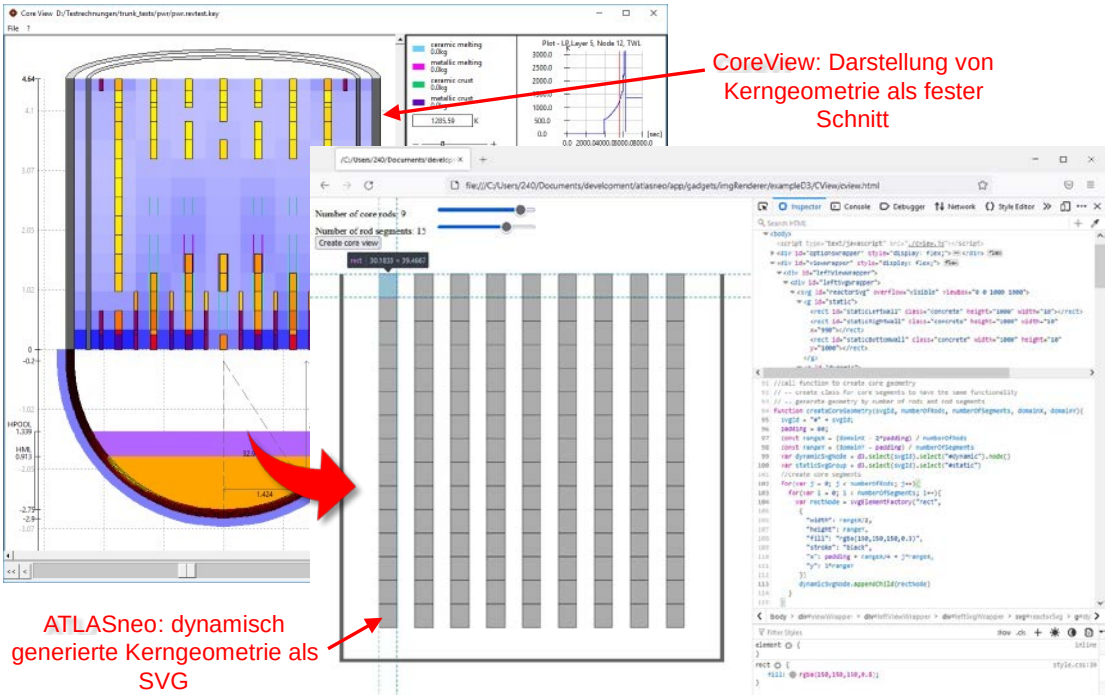

Abb. 4.45 Darstellung von ATHLET-CD-Ergebnissen in CoreView

CoreView erlaubt die Darstellung von ATHLET-CD Ergebnissen in Form von Schnitten durch den Reaktorkern**.** Diese Darstellungen werden in ATLASneo in Form von automatisch generierter SVG-Geometrie integriert.

4.4.5 Erweiterung der Topologie-Visualisierung

Für die Visualisierung topologischer Strukturen, wie GCSM-Netzwerken, wurde bereits im Vorgängerprojekt ein Funktionsmodul für ATLASneo exemplarisch realisiert. Inzwischen werden in ATHLET zusätzliche Metadaten zur Simulation bereitgestellt, die dazu genutzt werden können, um Prioritätsketten zu ermitteln und zu visualisieren. Da diese Funktionalität schon oft von Anwendern angefragt wurde, sollte das Funktionsmodul im Rahmen dieses Projektes entsprechend erweitert werden.

Grundlage für die Visualisierung von Topologien bildet in ATLASneo das in Abbildung 4.46 dargestellte Funktionsmodul GCSMViz. Im Projektzeitraum konnten verschiedene Wartungsarbeiten und die Aktualisierung der Baumdarstellung vorgenommen und die Handhabung der im Graph vorhandenen Nodes durch die Such- und Filterfunktion verbessert werden. Zwar wurde die typspezifische Behandlung unterschiedlicher Node-Arten um GCSM-Tabellen erweitert, jedoch fehlt noch die Unterstützung weiterer Knoten-Typen, wie sie z. B. in Topologien von Prioritätsketten vorkommen. Im aktuellen Stand wird der Graph bereits als SVG generiert und angezeigt. Im weiteren

Entwicklungsverlauf sollten allerdings sowohl die Generierung der SVG-Geometrie als auch deren Darstellung mit den bereits entwickelten Grundlagen zur Webbasierten Visualisierung (vgl. Abschnitt 4.4.3) zusammengeführt werden, um ein einheitliches Erscheinungsbild und die zeitgemäße Weiterentwicklung sicherzustellen.

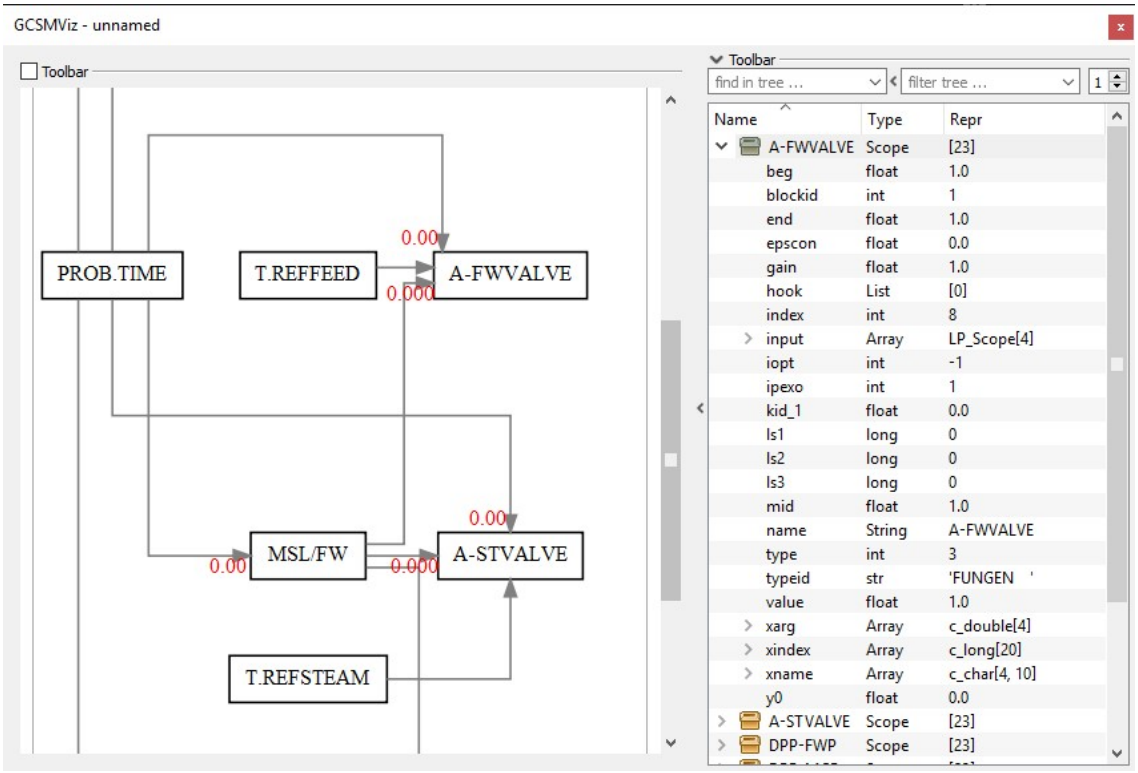

Abb. 4.46 GCSMViz in ATLASneo

Das Funktionsmodul bildet die Grundlage zur Visualisierung generischer Topologien, wie Prioritätsketten oder Netzwerken.

4.4.6 Ausbau der ATLASneo Anwender-Dokumentation

Im Projektverlauf sollte auch daran gearbeitet werden, die für die Anwender verfügbare Dokumentation auszubauen und leichter innerhalb der Anwendungen zugänglich zu machen. Hierzu muss das dafür am besten geeignete Ablage- und Textformat festgelegt werden, welches sowohl die einfache Erstellung und Versionierung der Hilfebeschreibungen erlaubt als auch gut in Standarddokumentformate zur weiteren Verwendung außerhalb der Anwendungen konvertiert werden kann.

Die Verfügbarkeit von Hilfetexten innerhalb einer Anwendung hängt sehr stark von deren Möglichkeiten zur Anzeige dieser Texte ab. Dadurch müssen z. B. die Beschreibungen zu den Startoptionen von ATLASneo, welche auf der Konsole dargestellt werden muss,

anders vorliegen als die interne Hilfe, die auch komplexe Formatierungen und Bilder enthalten kann. Somit können die Arbeiten zur Verbesserung der Anwender-Dokumentation in verschiedene Arten eingeteilt werden, wobei in jeder versucht wurde, Form und Format zu finden, welche pflegeleicht und portabel sind.

- **Startskripte:** Die hierin enthaltenen Texte beschreiben die verschiedenen Optionen zum Start der Anwendung und müssen, da sie auf der Konsole ausgegeben werden, als unformatierte Texte vorliegen. Durch die sehr beschränkten Möglichkeiten in Windows-Skripten (`*.bat`) wurden die Optionsbeschreibungen innerhalb der Skripte zusammengefasst, die bei Angabe des Arguments „`--help`“ ausgebeben werden.

  Aufgabe der Startskripte ist die Sicherstellung der grundlegenden Umgebung, die das Programm für seine Ausführung braucht. Im Fall von ATLASneo müssen die Skipte bei Bedarf eine Python-Installation durchführen und ein passendes Envionment einrichten. Die Einstellungsmöglichkeiten, die hierzu vom Anwender sinnvoll vorgenommen werden können, wie das Installationsverzeichnis oder die zu verendende Python-Version, wurden als Startoptionen mitaufgenommen und in ihrer Syntax und Wirkung beschrieben. Da die Skripte Teil der Anwendung sind, werden sie bislang zusammen mit den restlichen Quelldateien versioniert und bearbeitet.

- **Docopt:** Auch docopt-basierte Hilfetexte und die Verarbeitung von Startoptionen sind von der Konsole aus über das Argument „`-h`“ bzw. „`--help`“ verfügbar und unterstützen den Anwender beim Aufruf eines Programms. Anders als die Startskripte setzen diese allerdings schon eine passend eingerichtete Python-Umgebung (inkl. installiertem package docopt /DCO 22/) voraus und umfassen nur noch die für das Commandline-Interface vorgesehenen Optionen. Für ATLASneo sind das unter anderem die optionalen Angaben einer Session-Datei sowie die Kategorie der zu aktivierenden Gadgets.

  Da es aufgrund der besseren Wartbarkeit nicht sinnvoll ist, die Hilfetexte von der Verarbeitung der Optionen zu trennen, werden die Texte direkt im Dokumentationsblock `__doc__` des von der Konsole aufgerufenen Hauptskripts `__main__.py` abgelegt und zusammen mit dem Quellcode versioniert und bearbeitet.

- **Wiki:** Beschreibungen und Hilfetexte, die dem Benutzer einen Überblick über die Funktionsweise eines Programms liefern und ihn so bereits bei der Auswahl der Version bzw. bei den ersten Schritten in Installation und Anwendung

unterstützen sollen, können sehr gut auf den Wiki-Seiten des GitLab-Projekts abgelegt werden. Die Inhalte des Projekt-Wiki werden in einem zum Projekt verknüpften Git-Repository verwaltet und können daher neben den gewöhnlichen Bearbeitungsmöglichkeiten im Browser auch lokal geklont und in jedem beliebigen Texteditor bearbeitet werden. Ein großer Vorteil dieser zentralen Ablageart ist, dass vorgenommene Änderungen sofort auf den Wiki-Seiten des Projekts sichtbar sind und somit allen internen Anwendern als Referenz zur Verfügung stehen.

Die im Wiki erstellten Hilfeseiten werden als Markdown-Textdateien gespeichert und bieten dadurch den weiteren Vorteil der Konvertierbarkeit in andere Dokumentformate. Unter Verwendung von pandoc /PAD 22/ können die einzelnen Seiten leicht innerhalb eines PDF- oder Word-Dokuments zusammengefasst werden, welches dann sowohl innerhalb der Anwendung angezeigt als auch außerhalb zur Verfügung gestellt werden kann.

4.5 AP 5: Querschnittsaufgaben

Abgesehen von den geplanten Arbeiten und den konkreten Entwicklungsaufgaben im Projekt müssen weitere Maßnahmen unternommen werden, um möglichst hohe Qualität und Nachhaltigkeit der entwickelten Funktionalität zu erreichen. Diese müssen kontinuierlich betrieben werden, auch um die Entwicklungen in den einzelnen Arbeitspunkten aufeinander abzustimmen. Die hierzu im Projektverlauf durchgeführten Arbeiten und Weiterentwicklungen werden in den folgenden Abschnitten beschrieben.

4.5.1 Qualitätssicherung und Programmwartung

Durch neu einzuarbeitende Anforderungen ergab sich immer wieder die Notwendigkeit auch bestehende Softwareteile zu überarbeiten und Änderungen vorzunehmen. Hierunter fielen Fehlerbehebungen sowie Anpassungen in deren Schnittstellen, um auch diese Komponenten mit neu entwickelten Programmstrukturen zusammenarbeiten zu lassen. Des Weiteren mussten hierfür oft Programmteile neu aufgeteilt werden, um deren Einsatz auch unter den erweiterten Bedingungen zu erlauben. Die durchgeführten Maßnahmen steigerten die Modularität und Wartbarkeit der Software und trugen so zur Sicherung ihrer Qualität bei.

Im August 2020 wurde eine neue Version des AC² Programmpakets für das Jahr 2021 angekündigt. Um die Handhabung für Anwender zu verbessern, sollten alle zugehörigen Programme einschließlich ATLAS in einem einzigen Gesamtpaket enthalten sein. Bis dahin wurde ATLAS allerdings unabhängig vom AC² Programmpaket erstellt und vertrieben. Folglich mussten zur Integration in das AC² Programmpaket, verschiedene Modernisierungsarbeiten am Repository und Buildsystem von ATLAS durchgeführt werden.

Die bisher von der GRS verwendete Entwicklungsplattform Teamforge wurde im Jahr 2020 von GitLab abgelöst wurde. Um weiterhin auf ATLAS zugreifen zu können, musste deshalb auch das ATLAS Repository zuerst von Subversion nach Git bzw. von Teamforge nach GitLab umgezogen werden. Wichtig war es, dass anschließend wieder alle notwendigen Daten zum Erstellen von ATLAS in einem GitLab Repository verfügbar sind. Dieses sollte kompakt und möglichst keine unnötigen oder gar abgeleiteten Daten enthalten, um eine schnelle und zuverlässige Handhabung zu gewährleisten. Umfangreiche Aufräumarbeiten der 25 Jahre langen Versionshistorie sollten vermieden werden. Zu diesem Zwecke wurde das alte Repository archiviert und nur noch für lesenden Zugriff freigegeben. Die letzte Version davon wurde anschließend als Basis für das neue GitLab Repository verwendet. Bei der Übernahme wurde berücksichtigt, dass nur die Dateien übernommen werden, die tatsächlich zum Kompilieren von ATLAS benötigt werden - alle weiteren Daten wurden weggelassen. Im Ergebnis ist das neue Repository deutlich übersichtlicher. Gleichzeitig konnte die Größe des Quellcode Repositories von bisher 350 MB auf 5 MB reduziert werden, wodurch das Arbeiten damit deutlich effizienter wurde.

Weiterhin wurde das ATLAS Buildsystem, bestehend aus vielen einzelnen Visual Studio Projekten und Makefiles, auf das in AC² verwendete CMake umgestellt. Mithilfe von CMake kann eine Vielzahl von unterschiedlichen Projekten flexibel miteinander kombiniert werden und anschließend zu einem Gesamtpaket zusammengefasst werden. Bereits bestehende komplexe Abhängigkeiten zwischen Projektkomponenten werden automatisch von CMake transitiv aufgelöst und müssen beim Kompilieren nicht mehr manuell berücksichtigt werden. Zusätzlich ist mit CMake der Kompilierprozess standardisiert und kann ohne vorhergehende Einarbeitung von jedem Entwickler bedient werden, der allgemein mit CMake vertraut ist. Erhebliche Erleichterungen ergeben sich auch im Wartungsaufwand, da CMake plattformübergreifend ist und gängige Compiler unterstützt werden. Es muss also beispielsweise nicht mehr für jede Plattform und Compiler ein separates Projekt definiert werden.

Die konkrete Umstellung erfolgte auf Basis eines minimalen CMake-Skriptes und wurde iterativ ausgebaut, bis alle notwendigen Komponenten fehlerfrei kompiliert und benutzt werden konnten. Die resultierenden CMake-Skripte sind daher so simpel gehalten wie möglich. Gleichzeitig konnten mittlerweile nicht mehr benötigte Aspekte aus dem Build-System aussortiert werden. Abbildung 4.47 zeigt als Graph die in CMake implementierten Komponenten und deren Abhängigkeiten zueinander.

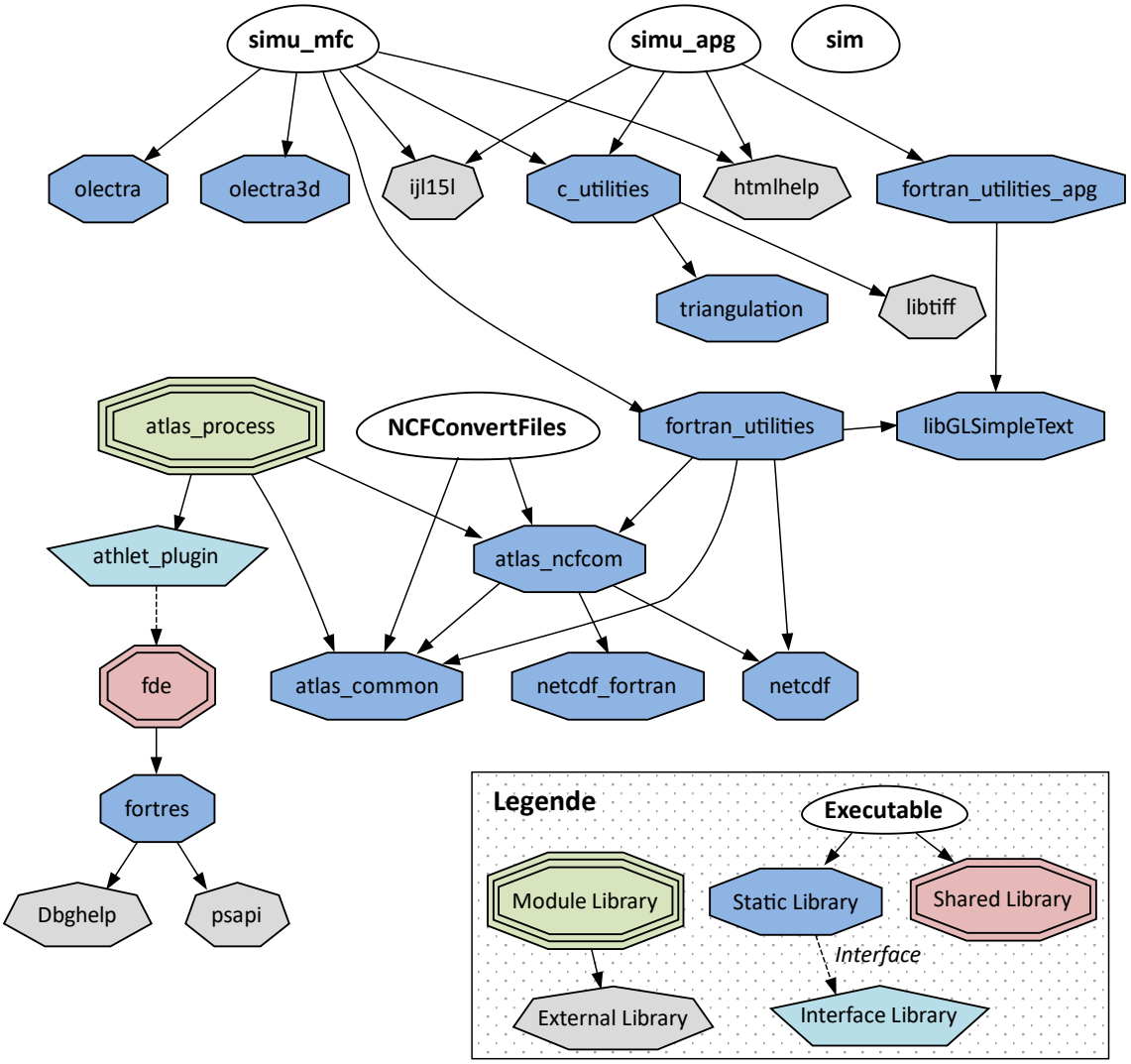

Abb. 4.47 In CMake implementierte Softwarekomponenten und ihre Abhängigkeiten

Zur Qualitätssicherung wurde eine GitLab-CI Pipeline implementiert, die bei jeder Veränderung im Repository automatisch alle enthaltenen Projekte kompiliert und das fertige Programm in Form von Artefakten zur Verfügung stellt. Dadurch ist sichergestellt, dass ATLAS aus technischer Sicht jederzeit erstellt und veröffentlicht werden kann. Gleichzeitig dienen das Skript und die zugehörigen Pipelines als Dokumentation. Es zeigt

welche Schritte konkret ausgeführt wurden, um das finale Paket zur Integration in AC² zu erhalten.

4.5.2 Entwicklungsdokumentation

Neben der allgemeinen Zielsetzung, durch die Anwendung bewährter Verfahren in der Softwareentwicklung möglichst selbstbeschreibenden Code zu schaffen, ist die Dokumentationserstellung eine wichtige Aufgabe in der Programmentwicklung. Hierunter fallen eine verständnisfördernde Kommentierung direkt im Quellcode sowie die Dokumentation der Softwarearchitektur, welche auch zukünftigen Entwicklern die Einarbeitung erleichtert.

Im vorliegenden Projekt wurden die hierfür erstellten Dokumente nach Möglichkeit durch geeignete Verfahren automatisch aus dem Quellcode generiert. Hierfür mussten in dessen Struktur und Kommentierung die dazu notwendigen Standards angewendet werden. Ergänzend zur so erstellten Programmdokumentation kommen die Kurzbeschreibungen, welche beim Einstellen von Codeänderungen in die Versionsverwaltung verfasst werden und die so die einzelnen Entwicklungsfortschritte im Projektverlauf nachvollziehbar machen. Des Weiteren werden sowohl die Planung zu Implementierungsdetails als auch der Stand der geplanten Entwicklungen durch sog. Issues auf der Projektverwaltung GitLab festgehalten, was den Entwicklungsverlauf während eines Projekts dokumentiert.

Gegenstand dieses Arbeitspunkts waren darüber hinaus auch alle fachlichen Aufgaben bzgl. der Dokumentation der Ergebnisse in Halbjahres- und Jahresberichten und die Erstellung des Abschlussberichts sowie die fachliche Leitung des Projektes.

5 Zusammenfassung und Ausblick

Das wesentliche Ziel der Arbeiten im vorliegenden Projekt war, die für den Systemcode ATHLET vorhandene Plattform ATLAS-GRAMOVIS so weiterzuentwickeln, dass sie alle Schritte der Simulation, wie

- Erstellung der Eingabedaten für die Modelle,
- Simulationsdurchführung,
- Ergebnisauswertung und Visualisierung,

insbesondere auch für ATHLET-CD und COCOSYS, mit zeitgemäßen Methoden unterstützt. Die dafür entwickelten Werkzeuge haben einen Stand erreicht, der sie teilweise bereits für die Anwendung in der Praxis einsetzbar macht. Trotzdem besteht aber in Teilbereichen noch Bedarf für Erweiterungen und Verbesserungen, die in den folgenden Abschnitten beschrieben sind. Neben einer Umsetzung dieser Punkte sollen auch künftig die in GRAMOVIS vorhandenen Werkzeuge durch Generalisierung und Modularisierung speziell auch für ATHLET-CD und COCOSYS ausgebaut werden.

5.1 AC² Design Modeller

Die interaktive Erstellung der ATHLET-Eingabe ist mit dem erreichten Entwicklungsstand des ATHLET Thermohydraulic Modeller (ATM) und des ATHLET GCSM Modeller (AGM) in weiten Bereichen möglich. AGM wird bereits in vielen Anwendungen zur Modellierung von realen Anlagensystemen eingesetzt und deckt viele der bestehenden Nutzer-Anforderungen ab. Dessen Verfügbarkeit wird über einen separaten Wartungs-Branch sichergestellt. Der bereits in der Master-Entwicklungslinie (ADM) enthaltene ATM wurde bisher nur in geringem Umfang praktisch verwendet, so dass weitere Erfahrungen aus der Anwendung einfließen müssen. Deshalb ist geplant, die repräsentativen ATHLET-Beispieldatensätze des GRS Softwarepakets AC² in enger Zusammenarbeit mit den Anwendern als Beispielprojekte im AC² Design Modeller (ADM) vollständig abzubilden. Die dabei gewonnenen Erkenntnisse können in der Entwicklung direkt umgesetzt werden, um derzeitig noch nicht entdeckte Fehler zu identifizieren und zu beseitigen. Auch die Handhabung für Anwender soll so weiter vereinfacht und ein geschlossener Workflow erzielt werden.

Die Arbeiten zur Eingabeerstellung in ADM für die Simulationscodes ATHLET-CD und COCOSYS konnten nur sehr eingeschränkt durchgeführt werden. Insbesondere die notwendige spezielle Implementierung des Datenimports für viele ATHLET-Modelle als auch der generische Ansatz durch Stub-Objekte erzeugten erheblichen Zusatzaufwand.

Die Möglichkeit zur benutzerseitigen Modifikation von ADM-Modellen, konnte mit den vom Basissystem „SimInTech“ zur Verfügung gestellten Methoden („Scripting“) nicht zufriedenstellend realisiert werden. Auch lässt die derzeitige Implementierung von ADM dies nicht zu, da der Datenexport im ADM-Quellcode die explizite Angabe der zu exportierenden Parameter erfordert. Leider stellte sich beim Test der Einzelkomponenten des Bindings „python4delphi“ die Einbindung in die Applikation als schwieriger heraus als angenommen und konnte über diesen Weg nur unzufriedenstellend realisiert werden.

Es hat sich gezeigt, dass der Ansatz unter Verwendung von „SimInTech“ als Basissystem hohe Lizenzkosten bei gleichzeitig geringer Zukunftsfähigkeit und erheblichem Eigenanteil für Entwicklung und Wartung erfordert. Deshalb muss die weitere Entwicklung von ADM neu ausgerichtet werden. Hierfür sollten gemäß dem aktuellen Stand von Wissenschaft und Technik die mittlerweile etablierten OpenSource-Entwicklungen (vgl. Abschnitt 3.1), welche in vergleichbaren Eingabe-Systemen zum Einsatz kommen, berücksichtigt werden. Im Rahmen eines Folgevorhabens sollte deshalb die Anwendbarkeit des aktuellen Entwicklungsstands von ADM geprüft werden. Die hierin zu erarbeitenden Funktionalitäten sollten so konzeptioniert sein, dass diese bei einer möglichen Neuausrichtung der Entwicklung, auch unabhängig vom „SimInTech“-Basissystem, für die Modellierung und Input-Erstellung einsetzbar bleiben.

5.2 ATLASneo

Eine weitere Zielsetzung des Projekts war der Ausbau von ATLASneo und dessen Unterstützung bei der Durchführung von Simulationsläufen mit AC²-Rechencodes und der Auswertung ihrer Ergebnisse. Um den Anforderungen des sehr breiten Aufgabenspektrums nachzukommen, war eine Weiterentwicklung sowohl des Anwendungsrahmens und dessen Architektur als auch der in ATLASneo enthaltenen Funktionsmodule notwendig. Zusätzlich sorgte die mittlerweile dringend erforderliche Herstellung der Kompatibilität zu Python 3 für weitere wichtige Anpassungen und Paket-Umstellungen. Dadurch konnte zum Projektabschluss für die Kernfunktionalität von ATLASneo eine stabile Milestone-Version erreicht werden, welche gleichermaßen unter Python 2 und Python 3 genutzt werden kann.

Im Framework des Anwendungsrahmens wurden einige zentrale Konzepte weiterentwickelt, wodurch viele Verbesserungen erzielt werden konnten. Durch die hierarchische Konfigurationsstruktur wurde die Grundlage zur Speicherung von Anwendungszuständen geschaffen, welche getrennt nach Installation, Benutzer und Sitzung abgelegt und später wieder eingeladen werden können. Die Konfiguration deckt hierbei explizit nicht nur den aktuellen Zustand einer ATLASneo-Sitzung ab, sondern auch die Präferenzen des Benutzers und die Default-Einstellungen der gesamten Anwendung. Die Einstellungen können über das Funktionsmodul Config-Gadget hierarchisch gruppiert angezeigt und angepasst werden. Hierbei leistet der Ausbau des Qt-Delegate-basierten Konzepts der typspezifischen Anzeige und interaktiven Bearbeitung einen sehr wichtigen Beitrag, um die Menge an unterschiedlichen Einstellungsgrößen einheitlich handhaben zu können. Insbesondere für die Speicherung und Wiederherstellung komplexer Qt-Datentypen musste die JSON-Serialisierung ausgebaut werden. Hierdurch konnten auch die Möglichkeiten zur Speicherung von funktionsmodulinternen Daten erweitert werden, was z. B. dem HDF5-Panel erlaubt, eine hierin angezeigte HDF5-Datei beim Wiedereinladen einer ATLASneo-Sitzung automatisch neu zu öffnen. Diese Möglichkeiten sollten in der weiteren Entwicklung der Funktionsmodule genutzt werden, um ihren Zustand beim Laden einer Sitzung möglichst vollständig wiederherstellen zu können.

Auch die Online-Simulation in ATLASneo wurde weiterentwickelt und die Stabilität des MonitorWidgets, dem Funktionsmodul zur Simulationssteuerung, konnte vor allem durch die Arbeiten an den Schnittstellen der Rechencodes (vgl. Abschnitt 4.1.2) deutlich verbessert werden. Auch konnten bereits viele Ergebnisse aus den Arbeiten in AP 3 (siehe Abschnitt 4.3.2) einfließen, die eine Verwendung von Protokollen zur Ablaufüberwachung und automatisierten Durchführung von Handmaßnahmen ermöglichen. Die noch bevorstehende Einbindung extern ablaufender Simulationen in das Funktionsmodul wurde in den vorbereitenden Anpassungen bereits berücksichtigt. Diese kann aber erst im weiteren Entwicklungsverlauf erfolgen, da durch die geplante Nutzung von IPython-Kernels zur Simulationsauslagerung (vgl. Abschnitt 4.3.1) noch mit einigen Anpassungen in der Netzwerkschnittstelle zu rechnen ist.

Zur Auswertung von Simulationsergebnissen die nicht als HDF5-Datei vorliegen wurde der Datenkonverter für Ergebnisdaten deutlich ausgebaut und unterstützt jetzt neben dem Datenformat von MELCOR auch grundlegend das COCOSYS-Ausgabeformat. Durch das Einladen der so erzeugten HDF5-Dateien können jetzt auch diese Rechencodes als Datenquellen in ATLASneo verwendet werden. Das parallele Öffnen von

HDF5-Dateien, welche von den Rechencodes direkt erstellt werden, ist jetzt durch die Reload-Funktion des HDF5-Panels möglich. Der Ausbau der Konvertiermöglichkeiten für AC²-Rechencodes sowie der in den HDF5-Dateien enthaltenen Daten ist vorbereitet und für Entwicklungen im Rahmen von nachfolgenden Projekten geplant.

Darüber hinaus wurden die Konzepte für die Visualisierung dynamischer Bilder deutlich weiterentwickelt und erlauben jetzt die Verwendung moderner, webbasierter Technologien für die Darstellung und Dynamisierung von SVG-Geometrien. Vergleichbar mit dem in ATLAS genutzten APG-Format können damit statische Bildteile in SVG-Dateien entworfen, in ATLASneo eingeladen und durch JavaScript-Codes dynamisiert werden. Zusätzlich sind die Bilder jetzt auch durch generierte Geometrien automatisch erweiterbar. Der dazu notwendige JavaScript-Code kann aus Bibliotheken wie D3 eingebunden werden, was auch die Entwicklung spezieller Visualisierungslösungen, wie sie z. B. für ATHLET-CD benötigt werden, deutlich erleichtert. Mittlerweile konnte auch das React-Framework eingebunden werden, welches die zukünftige Entwicklung dynamischer Visualisierungen nicht nur durch Skript-Abstraktion, sondern auch durch Objekt-orientierte Modularität und Wiederverwendbarkeit dynamischer Komponenten deutlich erleichtert.

5.3 Wartung bisheriger Entwicklungen und Verbesserung der Wartbarkeit

Neben den konkreten Weiterentwicklungen und in den Arbeitspunkten geplanten Aufgaben mussten im Projektzeitraum auch einige allgemeine Wartungsaufgaben angegangen werden. Diese sind nicht nur essenziell, um die bestehenden Entwicklungen zu erhalten, sondern bieten auch immer Gelegenheit, die grundlegenden Entwicklungsprozesse zu modernisieren, aktuelle und zukünftige Arbeiten zu erleichtern und die Qualität zu steigern.

So war für den Umstieg der Codeverwaltungsplattform nach GitLab zwar die Konvertierung aller Code-Repositories nach Git und damit einhergehend die Umstellung von Client-Programmen in der Entwicklung notwendig. Allerdings zahlte sich dieser Schritt bereits nach Kurzem durch eine deutliche Steigerung der Produktivität aus, da in den vorbereiteten Entwicklungsprozessen von GitLab viele Schritte unterstützt und wiederkehrende Aufgaben automatisiert werden konnten. Insbesondere die Automatisierung der Buildprozesse, welche mittlerweile in GitLab integriert werden konnten, wirkten sich sehr positiv auf die Wartbarkeit der Entwicklungen aus. Unter Verwendung von CMake konnten bereits viele der Build-Konfigurationen Plattform- und Compiler-übergreifend

spezifiziert werden und sichern somit auch für zukünftige Compilerversionen und Entwicklungsumgebungen die Erstellbarkeit der Programmcodes und Bibliotheken. Auch durch den Einsatz von Docker-Images, welche eine virtuelle, genau festgelegte Systemumgebung bereitstellen, wurde eine wichtige Voraussetzung geschaffen. Innerhalb dieser standardisierten Umgebungen wird nicht nur die Automatisierung der Erstellung von Programm-Binaries, sondern auch die anschließende Durchführung von Integrations- und Regressionstests (Continuous Integration) möglich.

Durch die Flexibilität von GitLab und den mittlerweile erarbeiteten Buildprozessen können auch Abhängigkeiten zwischen verschiedenen Projekten berücksichtigt werden. Hierdurch wird in der automatischen Erstellung auch die Bereitstellung aller erforderlichen Komponenten, wie Bibliotheken, in den zur geforderten Konfiguration passenden Versionen sichergestellt. Diese Komplexität kann manuell kaum bewältigt werden und war früher oft der Grund dafür, dass Abhängigkeiten, wie Bibliotheken, nicht in Quellform, sondern als vorkompilierte Binaries in den Repositories mitabgelegt wurden. Dieser spontane Lösungsansatz beschränkte die Lebensdauer und Wartbarkeit von Software-Entwicklungen sehr deutlich, da die abgelegten Binaries früher oder später nicht mehr mit neueren Versionen von Compilern oder Buildumgebungen kompatibel waren.

Die in GRAMOVIS enthaltenen Entwicklungen profitieren enorm vom mittlerweile erreichten Grad an Build-Automatisierung, da auch komplexe Entwicklungsprojekte wie ATLAS, trotz der Vielzahl ihrer Komponenten und ihrer teilweise bewegten Entwicklungsgeschichte, auf die automatisierte Erstellung und Bereitstellung umgebaut wurden. Hierdurch wird die langfristige Wartbarkeit von ATLAS überhaupt erst möglich, und neuere Entwicklungen mit noch höherer Komplexität, wie ATLASneo, können direkt mit allen Vorkehrungen für die Wartbarkeit (CI/CD) entwickelt werden.

Literaturverzeichnis

/AC2 19/ Weyermann, F. et al., "Development of AC^2 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

/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 Laboratory, 2012

/DCO 22/ docopt, Command-line interface description language,
Stand: 08. März 2022, URL: http://docopt.org/

/EMB 15/ Embarcadero Technologies, Delphi XE 7, Stand: 15. Februar 2015,
URL: http://www.embarcadero.com/de/products/delphi

/FAH 21/ FastHelp is a Windows Help File Generator, Stand: 06. Dezember 2021,
URL: https://www.fast-help.com/

/FDE 21/ Scheuer J. et al., Fortran Development Extensions (libfde),
URL: https://gitlab.com/Zorkator/libfde, 2021, Rev. 2.8.0.

/GIL 21/ GitLab, The DevOps Platform, Stand: 06. Dezember 2021,
URL: https//about.gitlab.com

/GIT 21/ Git is a free and open source distributed version control system,
Stand: 06. Dezember 2021, URL: https://git-scm.com/

/HDF 17/ HDF5, a technology suite that makes possible the management of extremely large and complex data collections,
Stand: 10. Juli 2017, URL: https://support.hdfgroup.org/HDF5/

/HTM 08/ Wikipedia, HTML5, the hypertext markup language, ver. 5,
Stand: 03. März 2022, URL: https://de.wikipedia.org/wiki/HTML5

/INK 17/ Inkscape – Open Source Vector Graphics Editor using SVG as
the native format, Stand: 7. Juli 2017, URL: https://inkscape.org/

/IRS 14/ Institute da Radioprotection et de Sûreté Nucléaire, XASTEC Data Set
Editor, User's Guide, Draft Version, 2014

/JSC 22/ Wikipedia, JavaScript (kurz JS),
Stand: 03. März 2022, URL: https://de.wikipedia.org/wiki/JavaScript

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

/LER 16/ Lerchl, G.; et al., ATHLET Mod 3.1 Cycle A User's Manual,
GRS-P-1/Vol. 1 Rev. 7, 2016

/MAT 22/ Simulink, Mathworks
URL: https://de.mathworks.com/products/simulink.html

/MID 22/ pyscripter/MultiInstaller, installing Delphi packages from git repositories,
Stand: 03. März 2022, URL: https://github.com/pyscripter/MultiInstaller

/PAD 22/ Pandoc, a universal document converter,
Stand: 03. März 2022, URL: https://pandoc.org/

/PLY 21/ Python Lex-Yacc, "A pure-Python implementation of the compiler construction tools lex and yacc"
Stand: 20. Juli 2021, URL: https://ply.readthedocs.io/en/latest/index.html

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

/PYS 22/ PySide2, "Python bindings for the Qt cross-platform application and UI framework",
Stand: 03. März 2022, URL: https://pypi.org/project/PySide2/

/PYT 22/ Python Software Foundation,
Stand: 03. März 2022, URL: https://www.python.org/

/QTC 22/ The Qt Company, C++ library suite for user interfaces and applications,
Stand: 03. März 2022, URL: https://www.qt.io/#

/QWE 17/ QWebView, a widget provided by WebKit in Qt used to view and edit web documents, Stand: 10. Juli 2017,
URL: https://wiki.qt.io/Open_Web_Page_in_QWebView

/SCA 11/ Scalable Vector Graphics (SVG) 1.1 (Second Edition),
Stand: 07. Juli 2017, URL: https://www.w3.org/TR/SVG11/

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

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

/VOG 18a/ Voggenberger, T., 2018, ATHLET-Input-Modeller User Manual

/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

/WEB 22/ WebKit – Open Source Web Browser Engine,
Stand: 07. Juli 2022, URL: https://webkit.org/

Abbildungsverzeichnis

Abkürzungsverzeichnis

ADM	AC² Design Modeller
AGM	ATHLET GCSM Modeller
AIG	ATHLET Input Graphics
AIM	ATHLET Input Modeller
APG	ATLAS Picture Generator
AST	Abstract Syntax Tree
ATHLET	Analyse der Thermohydraulik von Lecks und Transienten
ATLAS	ATHLET Analysesimulator
ATM	ATHLET Thermohydraulic Modeller
CCO	Cross Connection Object
CI/CD	Continuous Integration / Continuous Deployment
COCOSYS	Containment Code System
DLL	Dynamic Link Library (dynamische Bibliothek)
GRAMOVIS	grafische Modellierung und Visualisierung
GUI	Bedienoberflächenelement
HCO	Heat Conduction Object (Wärmeleitungsobjekt)
QUABOX/CUBBOX	3D-Neutronen-Kinetik-Kernmodell
SVG	Scalable Vector Graphics
TFO	Thermo-Fluid-Obje

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 978-3-949088-55-1