JAEA-Data/Code
2010-016

高速炉実機燃焼解析システムの高度化（その２）

Sophistication of Burnup Analysis System for Fast Reactor (2)

横山 賢治　平井 康志　巽 雅洋

Kenji YOKOYAMA, Yasushi HIRAI and Masahiro TATSUMI

原子力基礎工学研究部門
核工学・炉工学ユニット

Division of Nuclear Data and Reactor Engineering
Nuclear Science and Engineering Directorate

October 2010

Japan Atomic Energy Agency　日本原子力研究開発機構

JAEA-Data/Code

本レポートは独立行政法人日本原子力研究開発機構が不定期に発行する成果報告書です。
本レポートの入手並びに著作権利用に関するお問い合わせは、下記あてにお問い合わせ下さい。
なお、本レポートの全文は日本原子力研究開発機構ホームページ（http://www.jaea.go.jp）
より発信されています。

独立行政法人日本原子力研究開発機構　研究技術情報部　研究技術情報課
〒319-1195　茨城県那珂郡東海村白方白根 2 番地 4
電話 029-282-6387, Fax 029-282-5920, E-mail:ird-support@jaea.go.jp

This report is issued irregularly by Japan Atomic Energy Agency
Inquiries about availability and/or copyright of this report should be addressed to
Intellectual Resources Section, Intellectual Resources Department,
Japan Atomic Energy Agency
2-4 Shirakata Shirane, Tokai-mura, Naka-gun, Ibaraki-ken 319-1195 Japan
Tel +81-29-282-6387, Fax +81-29-282-5920, E-mail:ird-support@jaea.go.jp

JAEA-Data/Code 2010-016

高速炉実機燃焼解析システムの高度化（その２）

日本原子力研究開発機構
原子力基礎工学研究部門
核工学・炉工学ユニット
横山　賢治　平井　康志*　巽　雅洋*

（2010 年 8 月 5 日受理）

高速炉の実用化に向けて高速炉実機炉心の核特性予測精度を向上させることは、合理的で高性能な炉心を設計してプラントの経済性の向上を図る上でも信頼性および安全性の裕度をより高める上でも、重要な研究課題である。これまでの研究では炉定数調整法を適用することにより、JUPITER 等の臨界実験の成果を活用して核設計精度の向上を達成している。高速炉の炉心設計では燃焼核特性の精度向上も重要となるため、高速実験炉「常陽」、高速原型炉「もんじゅ」等の高速炉実機の燃焼関連データを有効に活用して、更なる精度向上を図る必要がある。特に近年ではマイナーアクチニド等の効率的な燃焼方式の検討も重要な課題となっている。

しかしながら従来の核特性解析システムは臨界実験体系を想定しているため、燃料組成の燃焼変化を考慮した計算機能は十分には整備されておらず、機能的な制限により解析作業が極めて非効率的になるという問題がある。そこで各計算機能を必要に応じて組み立てたり分解したりすることが可能な、高速炉実機の燃焼解析評価および予測精度向上に資する高速炉実機燃焼解析システム ORPHEUS を開発している。

これまでの「高速炉実機燃焼解析システムの高度化」において、簡便なユーザ入力で与えられた情報に基づき定数計算・炉心計算・燃焼計算を実行する機能を有し、さらに大規模な炉心解析に不可欠な燃料交換・シャッフリング機能を組み込んだプロトタイプシステムを開発した。本研究開発ではこれまでの成果を推し進め、制御棒を柔軟に取り扱うための機能や少数群への縮約計算機能などを設計・実装した。

実機燃焼解析システムによる解析結果はリスタートファイルに出力され、ユーザはファイルから詳細な計算結果を取り出すことができる。ただし、これまでの課題として、ファイルの取り扱い方法や目的とするデータへのアクセス方法が難解であるため、システムに精通した開発者でなければ事実上データ取得が困難であるという点があった。そこで本研究開発では、リスタートファイルの概念を拡張し、計算条件・計算結果を収めたデータベースとして取扱うための仕組みを設計・実装した。これによりユーザは様々な計算データに容易にアクセス可能となった。

原子力科学研究所（駐在）：〒311-1195 茨城県那珂郡東海村白方白根 2-4
＊原子燃料工業（株）

JAEA-Data/Code 2010-016

Sophistication of Burnup Analysis System for Fast Reactor (2)

Kenji YOKOYAMA, Yasushi HIRAI* and Masahiro TATSUMI*

Division of Nuclear Data and Reactor Engineering
Nuclear Science and Engineering Directorate
Japan Atomic Energy Agency
Tokai-mura, Naka-gun, Ibaraki-ken

(Received August 5, 2010)

Improvement on prediction accuracy for neutronics characteristics of fast reactor cores is one of the most important study domains in terms of both achievement of high economical plant efficiency based on reasonably advanced designs and increased reliability and safety margins. In former study, considerable improvement on prediction accuracy in neutronics design has been achieved in the development of the unified cross-section set as a fruit of a series of critical experiments such as JUPITER in application of the reactor constant adjustments. For design of fast reactor cores improvement of not only static characteristics but also burnup characteristics is very important. For such purpose, it is necessary to improve the prediction accuracy on burnup characteristics using actual burnup data of "JOYO" and "MONJU", experimental and prototype fast reactors. Recently, study on effective burnup method for minor actinides becomes important theme.

However, there is a problem that analysis work tends to become inefficient for lack of functionality suitable for analysis of composition change due to burnup since the conventional analysis system is targeted to critical assembly systems. Therefore development of burnup analysis system for fast reactors with modularity and flexibility is being done that would contribute to actual core design work and improvement of prediction accuracy.

In the previous study, we have developed a prototype system which has functions of performing core and burnup calculations using given constant files (PDS files) and information based on simple and easy user input data. It has also functions of fuel shuffling which is indispensable for power reactor analysis systems. In the present study, by extending the prototype system, features for handling of control rods and energy collapse of group constants have been designed and implemented.

＊Nuclear Fuel Industries, Ltd.

Computational results from the present analysis system are stored into restart files which can be accessible by users to retrieve detailed information. However, there have been difficulties in actual data management because of complex manner in data access; it was quite hard for anyone other than experts to retrieve data. In the present study, as a remedy for the difficulty, a mechanism for database management has been developed by extending the idea on restart files in order to help user easily access arbitrary data of the results.

Keywords: Fast Reactor, Neutronics Design, Burnup Calculation, Analysis Code System, Object-Oriented Technique, MARBLE, ORPHEUS

This is a blank page.

目次

Contents

表リスト

図リスト

1. はじめに

1.1 背景と目的

高速炉の実用化に向けて、高速炉実機炉心の核特性予測精度を向上させることは、合理的で高性能な炉心を設計してプラントの経済性の向上を図る上でも、信頼性および安全性の裕度をより高める上でも、極めて重要な研究課題となっている。これまでの研究では、炉定数調整法を適用することにより、JUPITER 等の臨界実験の成果を最大限有効に反映した統合炉定数を開発し、核設計精度の向上を達成している。しかし、高速炉の炉心設計においては、臨界性，反応率，制御棒価値等のいわゆる静核特性だけでなく、燃焼反応度損失、増殖比といった燃焼核特性の精度良い評価も重要である。このためには、高速実験炉「常陽」や高速原型炉「もんじゅ」等の実機燃焼データを有効に活用して燃焼核特性の精度向上を図る必要がある。特に近年では、マイナーアクチニド等の効果的な燃焼方式の検討も重要な課題となっている。

実機で測定された燃焼関連データを核設計精度向上に活用するには、実験データが持つ情報を最大限引き出す必要があるため、解析誤差を最小化しなければならない。このためには、実機燃焼解析に対して JUPITER 臨界実験解析等で適用したような最確モデルに基づく詳細な炉物理解析を実施する必要がある。しかしながら、これまで JUPITER 標準解析手法として整備されてきた核特性解析システムは、解析対象として臨界実験体系を想定しているため、燃料組成の燃焼変化を考慮した計算機能は十分には整備されてこなかった。したがって、実機燃焼解析を詳細に実施しようとすると様々な機能的な制限により、解析作業が極めて非効率的になるという問題があった。例えば、複数サイクルにまたがる燃焼計算を行う場合は、サイクル毎に手作業で処理する必要があり、さらに燃料交換，制御棒操作，出力変化に伴う体系サイズの変化等、実機では当然考慮されるべき現象についても、解析作業者が手作業で計算メッシュに変換しなければならなかった。このため、実機特有の問題を詳細に取り扱おうとすると作業量が多くなりすぎ、原理的には適用可能な解析モデルであっても簡略化せざるを得ないという問題があった。

一方、この実機燃焼解析では、定型の解析手順を繰り返せばよい訳ではなく、解析モデル誤差を評価する際には計算手順を変更し、物理的意味を分析する際には計算ステップの途中の結果を利用する必要がある。このため、各計算機能を単純に統合するだけでは不十分である。このため、定数計算や炉心計算といった各機能を部品として保持したまま、必要に応じて部品を組み立てたり分解したりできるような実機燃焼解析システム ORPHEUS を開発して、高速炉実機の燃焼解析評価及び予測精度向上に資する。

これまでの「高速炉実機燃焼解析システムの高度化」において、簡便なユーザ入力で与えられた情報に基づき定数計算・炉心計算・燃焼計算を実行する機能を有し、さらに大規模な炉心解析に不可欠な燃料交換・シャッフリング機能を組み込んだプロトタイプシステムを開発した。本研究開発ではこれまでの成果を推し進め、制御棒の柔軟な取り扱いや少数群への縮約計算機能などを設計・実装する。

実機燃焼解析システム ORPHEUS による解析結果はリスタートファイル[1]に出力され、ユーザはファイルから詳細な計算結果を取り出すことができる。ただし、これまでの課題として、ファイルの取り扱い方法や目的とするデータへのアクセス方法が難解であるため、システムに精通した開発者でなければ事実上データ取得が困難であるという点があった。そこで本研究開発では、ユーザが容易に解析結果にアクセスできるようにする仕組みを設計・実装する。

[1] 今回リスタートファイルの概念は、解析に関する様々な情報を収めたデータベースとして拡張される。詳細については第 4 章にて述べる。それまでは従来通り、「リスタートファイル」として記述する。

1.2 本報告書の内容

本研究開発では、主に、(1)制御棒取り扱い機能の高度化、(2)群縮約計算機能の追加、(3)計算結果取り扱い機能の高度化、の 3 つを実施した。

本報告書では第 2 章において、制御棒取り扱い機能に関する設計と実装の詳細を示す。第 3 章で群縮約計算機能に関して詳細を示し、第 4 章で計算結果取り扱い機能を高度化するに当たっての分析・検討から詳細な設計・実装について説明する。第 5 章では、第 2～4 章の内容を踏まえて、実機燃焼解析システム全体の制御を行う全体制御部の処理について昨年度までの実装から変更した部分を説明する。最後に第 6 章において本報告書を総括した。なお実装の詳細に関しては付録とした。

本報告の内容はこれまでの開発内容を踏まえたものであるため、参考文献[4]「高速炉実機燃焼解析システムの高度化」や参考文献[8]「次世代炉物理解析システムのためのフレームワーク開発（その４）」等に提示された概念や用語が適宜使用されている。したがって、参考文献[1]～[8]を本報告書と併読することが望ましい。

2. 制御棒取扱い機能の高度化

2.1 分析

2.1.1 従来の制御棒の取り扱い方法

これまでの ORPHEUS システムでは、各計算ステップで制御棒を上下動させる機能はサポートされておらず、計算開始時に構築された制御棒挿入状態を、計算ステップを通じて維持する仕様となっていた。すなわち、システム的には制御棒と他の燃料集合体との差異は事実上存在せず、単に特殊な組成を持つ燃料として取り扱われていた。

2.1.2 制御棒取り扱いに関する検討

各計算ステップにて制御棒を上下動させる場合、制御棒下端がマテリアルメッシュの軸方向境界にだけ位置するのであれば簡単であるが、実際はメッシュの間に位置する場合を考える必要がある。その場合、システムによる実現方法として大きく分けて次の 2 つが考えられる。

- 制御棒下端位置がメッシュの中間にある場合でも、計算体系のメッシュ構造は変更しない。当該メッシュの数密度や断面積は、制御棒と（制御棒の下側に存在する）冷却材・構造材等を、メッシュ内部で重み付け平均[2]して使用する。このメッシュでは制御棒と冷却材が混合して存在することになる。
- 制御棒下端位置でマテリアルメッシュを軸方向に分割する。この場合メッシュに制御棒と冷却材・構造材のマテリアルが混合することはなく、上で述べた平均化も不要となる。ただし、ステップ毎にメッシュ構造が変化することによる様々な影響を考慮する必要がある。

前者の方式はメッシュで平均化することによる不確かさが発生するが、実装上は現在のシステムの枠組みを大きく変更することなく対応可能であると考えられる。すなわち、各マテリアルメッシュ・ゾーンに管理されている数密度やミクロ断面積オブジェクトを、制御棒操作を行う度に平均化処理後の状態に適切に更新することで対応できる。

これに対して後者の方式は、まず計算ステップの途中でメッシュ構造が変更することによる不確かさの影響を考慮する必要がある。さらに実装上はメッシュ管理における困難な処理が発生する。解析開始時にシステムは計算体系の初期化を行うが、その時点でのマテリアルメッシュの構造やゾーンの定義を基にして、マテリアル番号のナンバリングやメッシュ番号・マテリアル番号・ゾーン番号それぞれの間の変換テーブルが作成される。したがって、マテリアルメッシュの構造が変化した場合、これらの情報も更新する必要があり、すなわち、その管理処理が発生する。また、メッシュ番号については当該メッシュのアドレスと軸方向プレーン番号[3]から算出されているが、軸方向メッシュの分割が変化した場合はプレーンのナンバリングも変化

[2] 中性子束と体積を重みとした加重平均とする。
[3] アドレス及びプレーン番号の定義については、参考文献[8]を参照することができる。

するため、同一のメッシュであっても制御棒移動前後でメッシュ番号が変化することになる。そのためメッシュ番号の変化を管理するための何らかの仕組みが必要になると考えられる。また、既存のメッシュのメッシュ番号は変えずに分割によって新たに生じたメッシュには新しい番号を割り当てるといった方法も考えられるが、メッシュ番号のナンバリング方法に一貫性がなくなる。いずれにしろ何らかの番号管理機構は必要になるため困難であることに変わりはない。

以上の考察から、まずは前者の方式による制御棒操作についてより検討を進めることとする。

簡単のため、図 1 制御棒操作の例のように炉心有効長を軸方向に 3 等分割して構成されたマテリアルメッシュに対して制御棒が上下するケースを考える。なお、以下では各マテリアルメッシュはプレーン番号を用いて、「プレーン 1 のメッシュ」等と表す。図で表されているように、制御棒は第 1 ステップではプレーン 1 のメッシュの途中まで挿入されているが、第 2 ステップでプレーン 1 とプレーン 2 の境界まで挿入されるケースを考える。ここで第 1 ステップのプレーン 1 のメッシュのマテリアルは制御棒と冷却材の重み付け平均となる。

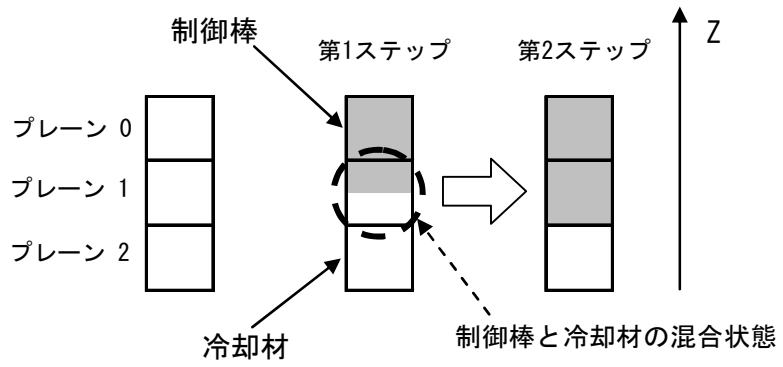

図 1 制御棒操作の例

まず、制御棒が燃焼するケースを考える。この場合、制御棒のミクロ断面積を更新するのであれば、計算ステップを通じて繰り返し定数計算を実行する必要がある。ここで制御棒がメッシュの途中まで挿入されているような状態を考える。通常、制御棒領域と冷却材領域では定数計算を行う際の集合体のモデル化に異なる方法を用いるが、現在の MARBLE フレームワークの枠組みではこれらが混在したような状態で定数計算用集合体モデルを構築することが困難である。したがって、定数計算を行う直前には制御棒を一旦メッシュ境界まで動かしておき、計算が終わった後に元の位置に戻す必要がある。すなわち、制御棒がメッシュ境界位置にある状態で定数計算を行い、算出されたミクロ断面積に対して重み付け平均化処理を行うことで、本来の制御棒位置にある場合のミクロ断面積を算出する。

ただし、この方式には問題が存在する。図 1の例をもとに、あるステップでメッシュの途中位置まで挿入されていた制御棒をメッシュの境界位置にまで戻すことで生じる問題を考える。第 1 ステップでプレーン 1 のメッシュの途中まで挿入されていた制御棒は、第 2 ステップでメッシュの境界位置まで挿入されるが、このとき単純にステップ 1 の状態のメッシュを基にステ

ップ 2 の状態のメッシュを構成すると、第 2 ステップのプレーン 1 のメッシュは第 1 ステップのプレーン 0 及びプレーン 1 のメッシュの重み付け平均となり、第 2 ステップのプレーン 2 のメッシュは第 1 ステップのプレーン 1 及び 2 のメッシュの重み付け平均となる。図からも明らかなように、本来第 2 ステップのプレーン 1 のメッシュは制御棒のみであり、プレーン 2 のメッシュは冷却材のみであるが、ここで述べた方式ではプレーン 1 のメッシュには冷却材が混入し、プレーン 2 のメッシュには制御棒が混入することになる（図 2）。これは制御棒の燃焼の有無に関わらずに生じる問題であり、要するに一度メッシュ内部で制御棒と冷却材を均質化してしまう以上、これらを再び完全に分離するのは不可能ということである。ここでの考察から、制御棒操作を実現するに当たってメッシュ内部で制御棒と冷却材のマテリアルを均質化する方式を取る場合、一度均質化処理を行ったメッシュに対して、次のステップで再度制御棒操作による均質化処理を行うのは適当でないことが分かる。

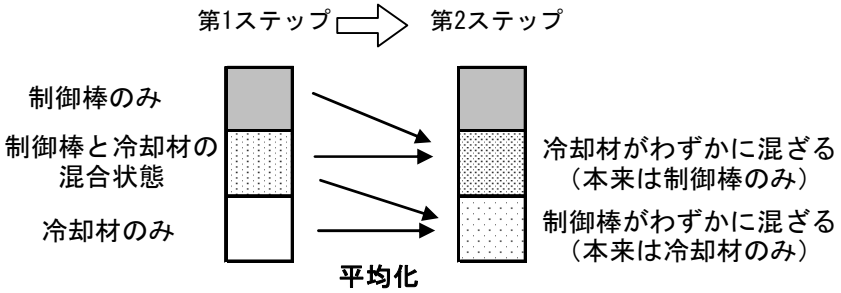

図 2 マテリアルメッシュで均質化することによる問題

ここで挙げた問題を回避する方法として、以下の 2 つが考えられる。

- メッシュ内部でマテリアルを均質化する方式を止め、最初の分析で挙げた制御棒下端位置で常にメッシュを分割する方式にする。
- 一度メッシュ内部で均質化した状態を次の計算ステップに引き継がないようにする。例えば制御棒全挿入状態を計算開始時点のメッシュ状態として保存しておき、計算ステップ毎にその状態を基に制御棒移動後のメッシュ状態を算出する。当然、制御棒は非燃焼という制限を設けて取扱うこととなる。

最初に述べた通り、前者については計算ステップ途中でメッシュ構造が変化することによる不確かさの影響もあり、望ましい手法ではない。基本的に既存システムによる従来の解析において、制御棒は非燃焼として取り扱われていることもあり、今回の実装では制御棒を非燃焼とする制約を設けて後者の方式を採用する[4]。

[4] この方式では制御棒操作と制御棒燃焼を同時に扱えないが、制御棒を特殊な組成を持つ燃料として扱うことで制御棒の燃焼効果を評価することはできる。

次に制御棒の上下動を実現するために、計算体系をどのように構築すべきかを考える。まず各計算ステップの初期では制御棒が仮想的に全挿入状態にあるとして各メッシュの数密度及びミクロ断面積を算出する。計算を行う際に制御棒を実際の位置まで移動させる必要があるが、その際、制御棒が移動した状態における各計算メッシュにおける数密度・断面積の算出を自然に行うために、炉心下端のさらに下側に仮想的なメッシュが存在すると仮定し、制御棒幾何形状等もそれに合わせてモデル化する。すなわち、以下のように考える。

- 制御棒幾何形状を定義する際に実際の炉心計算モデルの軸方向全長の 2 倍の長さでモデル化し、下側半分は冷却材（及び構造材等）領域として取り扱う。
- 制御棒集合体位置のマテリアルメッシュについては、炉心下端のさらに下側に仮想的なマテリアルメッシュが 1 メッシュ存在するものとする。このメッシュの軸方向長は炉心計算モデルの軸方向全長と同一であり、上で挙げた冷却材領域に対応するものとなる。
- 上記仮想マテリアルメッシュと同様に仮想的なゾーンを定義する。
- 制御棒集合体位置のゾーンの軸方向分割は、マテリアルメッシュの分割と同一となるように行う。

幾何形状・マテリアルメッシュ・ゾーンの定義の例を図 3に示す。またこれまでの検討をまとめた制御棒操作の概念図を図 4に示す。

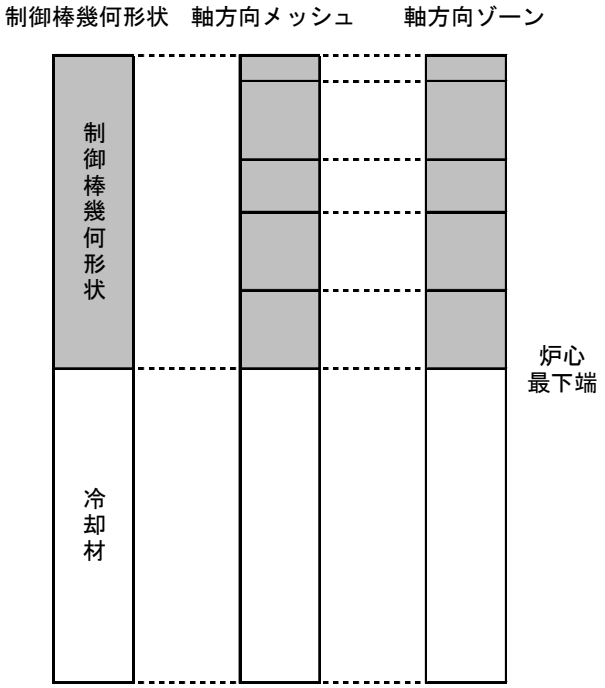

図 3 制御棒の取り扱い（幾何形状・メッシュ・ゾーン）

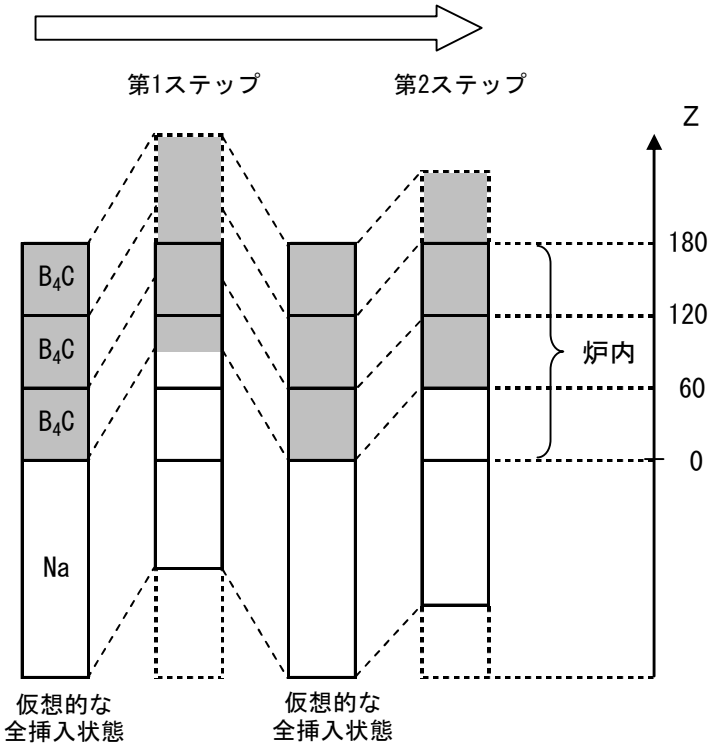

図 4 制御棒操作の概念図

2.2 設計

2.2.1 クラス設計

前節で検討したように、制御棒操作を実現するに当たっては、まず計算体系初期化の際に制御棒全挿入状態を仮定して各マテリアルメッシュの初期数密度を設定し、各ゾーンの実効ミクロ断面積オブジェクトを生成する。制御棒が引き抜かれた場合、マテリアルメッシュの軸方向位置を制御棒移動分だけスライドさせた状態を仮定し、それと本来のマテリアルメッシュの軸方向位置を比較する。制御棒移動後に、各メッシュにどのメッシュがどの程度オーバーラップうするかを評価して、最終的に各メッシュの数密度を決定する。断面積についても同様に、各ゾーンの軸方向のオーバーラップの度合い（ここではマテリアルメッシュのそれと同一になる）に応じて各断面積に重みを乗じて平均化する。各計算ステップにおける燃焼計算終了後には制御棒を再び全挿入状態に戻し、次ステップの計算で同様の処理を行う。

ここで述べた処理は引き抜きと全挿入状態へのリセットという 2 段階の手続きとしてまとめることができる。この手続きは MARBLE フレームワークの procedure パッケージにおいて、ControlRodOperationProcedure クラスとして実装する。

制御棒挿入位置（下端位置）は 3 次元メッシュ幾何形状を表す MeshGeometry オブジェクトにて管理する。ORPHEUS では中性子束メッシュとマテリアルメッシュで異なるメッシュ座標系を使用するため、それぞれで異なる MeshGeometry オブジェクトが存在するが、制御棒位置はマテリアルメッシュ用の MeshGeometry オブジェクトにて管理する。また前節で述べたように、制御棒集合体位置ではマテリアルメッシュは炉心下端側に伸びて本来の 2 倍の長さで構築されるため、計算体系として有効な範囲を明示的に管理するためのメソッドを MeshGeometry クラスに用意する。すなわち、MeshGeometry#setPlaneScope メソッドにて炉心有効長に含まれるプレーン番号のリストを与え、MeshGeometry#getPlaneScope メソッドで炉心に含まれるプレーン番号のリストを得る。

上で述べたように制御棒下端位置を MeshGeometry オブジェクトにて管理する結果、MeshGeometry オブジェクトの状態は計算ステップによって変化することになる。これまで ORPHEUS の「リスタートファイル」には、計算ステップを通じて変化しない情報をファイルのヘッダ部に、ステップ毎に変化する情報をボディ部に保存しており、MeshGeometry オブジェクトはヘッダ部に保存していた。今回の MeshGeometry オブジェクトにステップ毎の制御棒下端位置情報を持たせる結果、この点に関してリスタートファイルの設計変更が必要になる。詳細については第 4 章を参照のこと。

2.2.2 入力ファイル設計

入力については、まず以下の条件の下で既存の入力ファイルを定義する。

- 幾何形状ファイルにて制御棒集合体の詳細幾何形状を与える際、下側に冷却材・構造材の幾何形状を炉心有効長と同一長だけ定義する。

・ パターンファイルにてゾーンを定義する際、制御棒集合体位置のゾーンの軸方向分割をマテリアルメッシュのものと同一になるようにする。また制御棒は全挿入されているものとする。

ここで幾何形状ファイルとパターンファイルの例を図 5および図 6に示す。

実際の各計算ステップにおける制御棒位置は計算条件ファイルの“step”タグにて指定する。step タグの子要素として“contorol_rod”タグを定義し、このタグにて上下に動かす制御棒とその下端位置を指定する。下端位置の指定は、炉心下端を 0 としてそこからの移動量を長さで表す。計算条件ファイルの例を図 7に示す。

上下動させる制御棒の指定は、集合体位置ラベルによる個別指定と制御棒グループによる指定を可能とする。制御棒グループとは同時に操作する複数の制御棒をまとめてひとつのグループにしたものであり、炉心特性ファイルにてグループの定義を行うものとする。制御棒グループを定義した炉心特性ファイルの例を図 8に示す。

```
assembly:
  - name: fuel_assembly
    composition:
      - axial_range: [120.00, 180.00]
        segment    : reflector_segment
      - axial_range: [ 60.00, 120.00]
        segment    : fuel_segment
      - axial_range: [  0.00,  60.00]
        segment    : reflector_segment

  - name: blanket_assembly
    composition:
      - axial_range: [120.00, 180.00]
        segment    : reflector_segment
      - axial_range: [ 60.00, 120.00]
        segment    : blanket_segment
      - axial_range: [  0.00,  60.00]
        segment    : reflector_segment

  - name: control_rod_assembly
    composition:
      - axial_range: [   0.00, 180.00]
        segment    : b4c_segment
      - axial_range: [-180.00,   0.00]
        segment    : na_follower_segment

  - name: reflector_assembly
    composition:
      - axial_range: [  0.00, 180.00]
        segment    : reflector_segment
```

図 5 制御棒集合体詳細幾何形状定義の例（幾何形状ファイル）

（図の解説）

・ 炉心下端を 0 とし、軸方向位置 0 以上に制御棒セグメントを、0 以下に制御棒セグメントと同じだけの長さの冷却材セグメントを定義する。

```
zone_set:
  address:
    rid_core: [000, 1A1, 1C1, 1D1, 1E1, 1F1]
    rid_blkt: [2A1+, 2A2+]
    rid_refl: [3A1+, 3A2+, 3A3+, 4A2+, 4A3+, 4A4+]
    rid_cr:   [1B1]

  plane:
    zid_uref: [0, 0]
    zid_fuel: [1, 1]
    zid_lref: [2, 2]
    zid_all:  [0, 2]

  zone:
    1:  {address: rid_core, plane: zid_uref, cell: homo, burnable: False}  ## Upper Reflector
    2:  {address: rid_core, plane: zid_fuel, cell: ring, burnable: True }  ## Core Fuel
    3:  {address: rid_core, plane: zid_lref, cell: homo, burnable: False}  ## Lower Reflector

    4:  {address: rid_blkt, plane: zid_uref, cell: homo, burnable: False}  ## Upper Reflector
    5:  {address: rid_blkt, plane: zid_fuel, cell: homo, burnable: True }  ## Blanket Fuel
    6:  {address: rid_blkt, plane: zid_lref, cell: homo, burnable: False}  ## Lower Reflector

    7:  {address: rid_refl, plane: zid_all,  cell: homo, burnable: False}  ## Radial Reflector

    8:  {address: rid_cr,   plane: zid_uref, cell: rcc,  burnable: False}  ## B4C
    9:  {address: rid_cr,   plane: zid_fuel, cell: rcc,  burnable: False}  ## B4C
    10: {address: rid_cr,   plane: zid_lref, cell: rcc,  burnable: False}  ## B4C
```

図 6 制御棒領域のゾーン定義の例（パターンファイル）

（図の解説）

- 制御棒領域のゾーンは、制御棒が全挿入されているものとして定義する。
- 同ゾーンの軸方向分割は、マテリアルメッシュの軸方向分割と一致するようにする。（ここではマテリアルメッシュが軸方向に 3 分割されているものとする。）
- 制御棒領域のゾーンは非燃焼とする。
- ここでのゾーン定義では、制御棒集合体位置の炉心下側に仮定する仮想的なゾーンは含まない。（ユーザには仮想的なゾーンやマテリアルメッシュを直接意識させないようにする。）

```
 (略)
calc_system:
  coordinates:
    core  : triz
    burnup: hexz

  axial_mesh:
    core  : [[5, 60.0],
             [6, 60.0],
             [5, 60.0]]
    burnup: [[1, 60.0],
             [1, 60.0],
             [1, 60.0]]  ## necessary to match with zone
 (略)
step:
  - period:  25.00  ## day
    power : 100.00  ## %
    control_rod:
      all: 120.0

  - period:  25.00
    power : 100.00
    control_rod:
      bank1: 90.0
      1B1: 60.0

  - period:  25.00
    power : 100.00
    control_rod:
      bank1: 90.0
```

図 7 各ステップでの制御棒挿入位置定義の例（計算条件ファイル）

（図の解説）

- step タグの子要素である control_rod タグにて、ステップ毎に制御棒挿入下端位置を指定する。
- 制御棒はデフォルトで全挿入状態。
- 制御棒集合体位置で“all”を指定した場合、全ての制御棒を表す。集合体位置ラベル（1B1 等）を指定した場合は、当該制御棒のみを表す。バンクグループ名（bank1）を指定した場合、当該バンクグループに含まれる全ての制御棒を表す。
- マテリアルメッシュは軸方向に 3 分割。これは炉心下側の仮想マテリアルメッシュは含まない。

```
assembly:
  layer: 10
  except: [10A1+, 10A2+, 10A10+]

reflector:
  address: [5F2, 9A1+, 10A3+, 10A4+, 10A5+, 10A6+, 10A7+, 10A8+, 10A9+]

control_rod:
  address: [3A3+]
  bank_group:
    bank1: [3A3, 3C3, 3E3]
    bank2: [3B3, 3D3, 3F3]

source:
  address: [7F1]
```

図 8 バンクグループ定義の例（炉心特性ファイル）

（図の解説）

- control_rod タグの子要素である bank_group タグにて、バンクグループの定義を行う。
- バンクグループ名とそのグループに含まれる制御棒集合体位置ラベルのリストを指定する。
- ここで定義したバンクグループ名は、計算条件ファイルにおいて制御棒挿入量を指定する際に参照される。

2.2.3 実装

制御棒操作に伴うマテリアルやミクロ断面積の更新処理は、procedure パッケージの ControlRodOperation クラスにて行う。ControlRodOperationProcedure クラスの詳細については、参考文献[8]を参照することができる。ここでは、この Procedure クラスを用いた制御棒操作処理フローの実装について述べる。

ControlRodOperationProcedure クラスでは、制御棒を指定した位置まで引き抜いた際のマテリアルメッシュの数密度の更新とミクロ断面積の更新、及び制御棒を全挿入状態にリセットするという 2 つの処理を行う。全体の処理フローは図 9のようになる。

まず、制御棒全挿入状態という条件下で計算体系の初期化を行う。この際、以前に述べたとおり、制御棒集合体位置のマテリアルメッシュ及びゾーンに関しては、炉心下側に冷却材領域に相当する仮想的なマテリアルメッシュとゾーンが存在するものとして取り扱う。

制御棒操作については、まずマクロ定数計算前に制御棒引き抜きを行う。制御棒は非燃焼として取り扱うため、制御棒集合体に対する定数計算は最初のステップで 1 度行うだけである。したがって、第 2 ステップ以降の制御棒引き抜き操作は定数計算前後のいずれで実施しても変わりはないが、第 1 ステップにおいて制御棒引き抜きは定数計算実施後に行う必要があるため、図に示した処理フローで取り扱う。

第 2 ステップ以降のマクロ定数計算については、制御棒が不動の場合は燃焼対象のマテリアルメッシュのみ実施すれば良かったが、制御棒が上下動する今回のケースでは制御棒領域のマテリアルメッシュの数密度もステップ毎に変化することになるので、制御棒領域についてもマクロ断面積の再作成を行う必要がある。

制御棒状態の全挿入状態へのリセットはリスタートファイルへ各データを書き出した後に行う。なお、ここで書き出すデータには、各制御棒集合体の軸方向下端位置の情報も含まれる。最終ステップに到達した場合はそのまま反復計算を終了し、そうでない場合は最初に戻り再び定数計算を実行する。

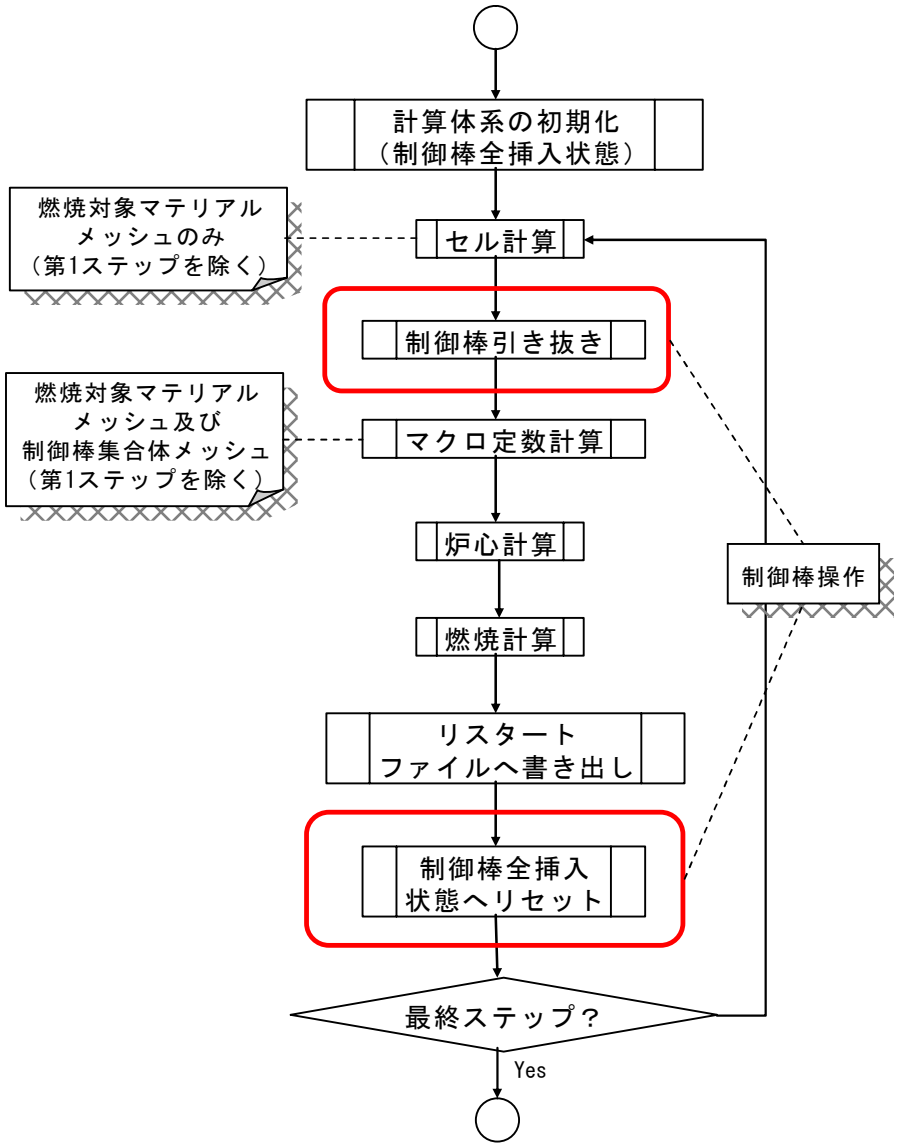

図 9 制御棒操作を含む燃焼解析フロー

3. 群縮約計算機能

3.1 分析

従来システムにおける燃焼計算では、実効ミクロ断面積は燃焼を通して固定するのが標準的な手法であったが、ORPHEUS では実効ミクロ断面積を各燃焼ステップで簡単に更新することができる。そのため、従来システムでは取り扱わなかった群縮約計算のユースケースが生じる。以下、群縮約計算のユースケースについてまとめる。

3.1.1 従来手法における群縮約計算

このユースケースにおける群縮約処理は、パラメータサーベイ等を行うために計算時間を短縮する、あるいは、大型炉心でメモリ使用量の制限から詳細群輸送計算ができない場合に対応する、といったことを目的とする。

まず、あらかじめ ORPHEUS 上で詳細群（通常は 70 群であるが、利用する炉定数によっては 175 群や 900 群等もある）による計算を実施し、その結果をリスタートファイルに保存しておく。次に、少数群による解析を行う計算ケースにおいて、このリスタートファイルに含まれる特定の時点の詳細群中性子束と詳細群ミクロ断面積を使って少数群（18 群や 7 群等）に縮約し、計算を行う。具体的には以下のような解析ケースが想定される。

例 1：平衡炉心に到達するまで詳細群で燃焼計算しておき、最後のサイクルの MOEC の中性子束とミクロ断面積で少数群に縮約して計算する。

例 2：燃焼中の平均燃料組成（例えば平衡炉心の燃料組成）が与えられた場合、縮約用のスナップショット計算を別ケースとして計算し、そのときの中性子束とミクロ断面積を使って縮約する。

注意点として、このユースケースでは、詳細群実効ミクロ断面積及び中性子束はあらかじめ実施済みの詳細群計算結果を保存したリスタートファイルに含まれるものを使用するため、燃焼計算の過程でミクロ断面積の更新は行われない。本ケースにおける処理フローを図 10に示す。

なお、従来のリスタートファイルは実効ミクロ／マクロ断面積の情報を含んでいなかったため、このユースケースを実現するには少なくとも実効ミクロ断面積を保存するようにリスタートファイルの設計を変更する必要がある。リスタートファイルの詳細については、第 4 章で述べる。

以降では、この手法のことを「ミクロ断面積の更新がない群縮約計算」と称する。

3.1.2 従来手法にない群縮約計算

先述したとおり、ORPHEUS では各燃焼ステップで実効ミクロ断面積を簡単に更新することができる。ただし、体系や燃焼度への依存性はミクロ断面積よりも中性子束の方が大きいと

考えられるので、異なる条件で計算されたミクロ断面積を持ってきて中性子束スペクトルだけを更新して縮約することはあっても、ミクロ断面積だけを更新して中性子束だけ他の条件で計算したものを持ってきて縮約することは考えにくい。したがって、ミクロ断面積を更新する場合はその時点で詳細群計算をやり直し、詳細群中性子束と詳細群ミクロ断面積を再計算することになる。

本ケースにおける処理フローは図 11のようになる。当該計算ステップにてミクロ断面積を更新する場合は、詳細群における定数計算及び炉心体系におけるスペクトル計算を行った後、その結果を用いて群縮約計算を行う。断面積を更新しない場合は、前ステップの少数群ミクロ断面積を用いて少数群による計算を行う。

以降では、この手法のことを「ミクロ断面積の更新を含む群縮約計算」と称する。

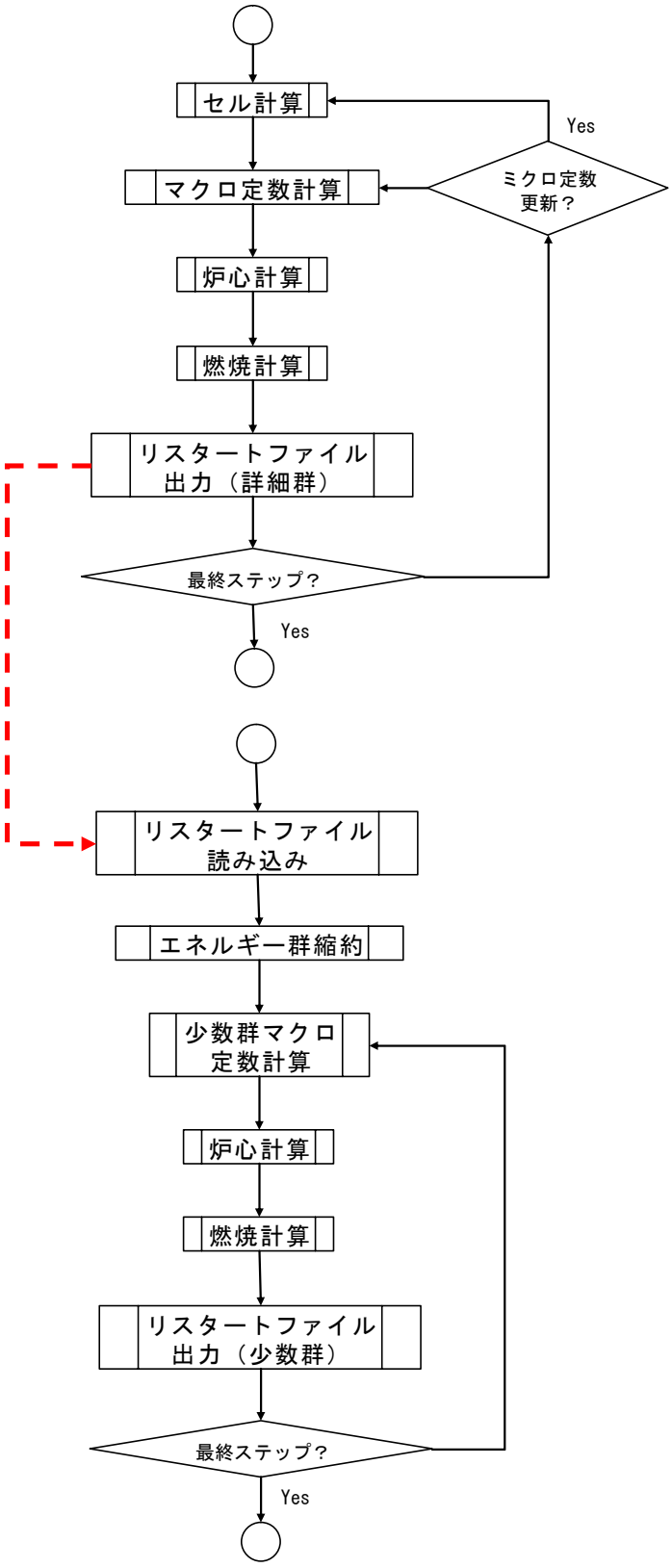

図 10 従来手法における群縮約計算
（ミクロ断面積の更新がない群縮約計算）

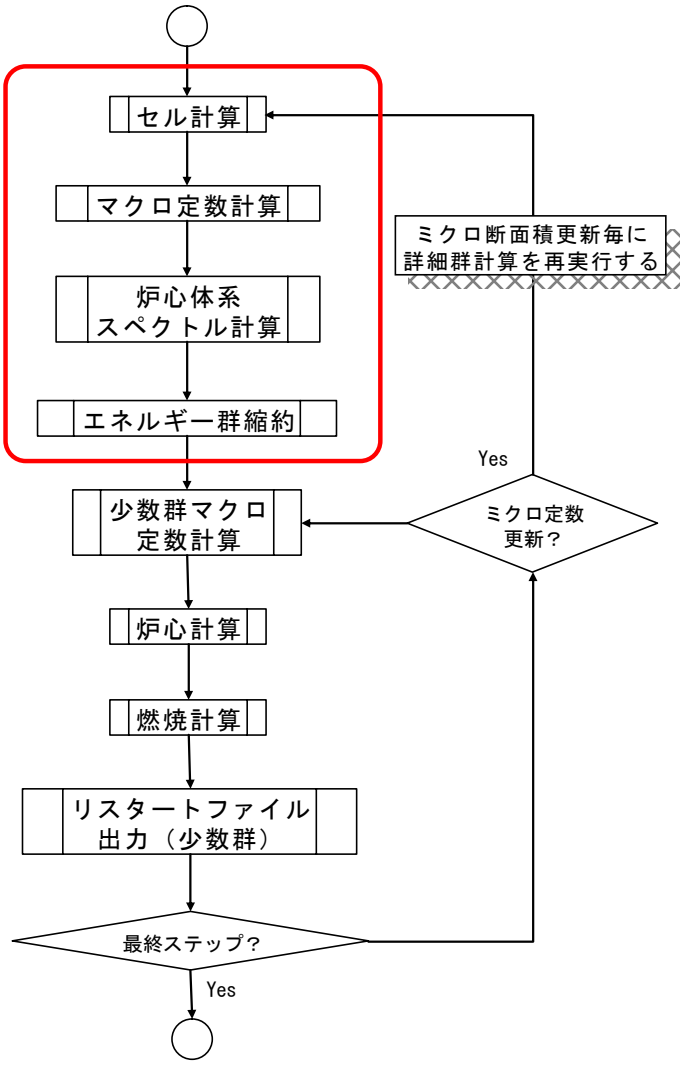

図 11 従来手法にない群縮約計算フロー
（ミクロ断面積の更新を含む群縮約計算）

3.2 設計

3.2.1 クラス設計

群縮約計算に関する処理は、procedure パッケージの CollapseGroupCalculationProcedure クラスにて行うものとする。このクラスにて実装されている run メソッドを呼び出すと、CoreFluxInfo オブジェクトにて管理されている呼び出し時点での中性子束分布を用いて、CrossSectionInfo オブジェクトにて管理されている詳細群実効ミクロ断面積を縮約し、算出された少数群ミクロ断面積をもって詳細群ミクロ断面積を置き換える。この一連の処理で必要になる情報は縮約する少数群のエネルギー構造だけであり、後述する入力ファイルにて与える。procedure パッケージ及び CollapseGroupCalculationProcedure クラスの詳細については参考文献[8]を参照することができる。

3.2.2 入力ファイル設計

(1) ミクロ断面積の更新がない群縮約計算の場合

このケースの場合、ユーザが与えるべき情報は以下の 3 つである。

- 詳細群計算結果が保存されたリスタートファイル名
- 詳細群計算結果のなかで、縮約に使用する時点（サイクル・ステップ）の指定
- 縮約する少数群のエネルギー構造

詳細群計算結果を保存するリスタートファイルの指定については、通常、リスタートファイルを指定する時と同様に、計算条件ファイルの file タグにて行う。さらに縮約に使用する詳細群計算結果の指定及び少数群エネルギー構造については、同ファイルの condensed_group タグにて行う。ファイルの入力例を図 12に示す。

(2) ミクロ断面積の更新を含む群縮約計算の場合

このケースの場合、ユーザが新たに与えるべき情報は縮約する少数群のエネルギー構造だけである。ミクロ断面積を更新するタイミングに合わせて、その時点で算出された詳細群計算結果を指定された少数群構造に縮約する。群構造の指定は、計算条件ファイルの condensed_group タグにて行う。ファイルの入力例を図 13に示す。

```
file:
  input:
    database[5]:
      0: $MARBLE_TEST_DATA_PATH/orpheus/minimal/0cyc.out.odb
      1: $MARBLE_TEST_DATA_PATH/orpheus/minimal/1cyc.out.odb
      2: $MARBLE_TEST_DATA_PATH/orpheus/minimal/2cyc.out.odb
 (略)
condensed_group:
  boundary: [4, 8, 19, 28, 37, 46 ,70]
  target:
    cycle: 2
    step: 5
```

図 12 ミクロ断面積の更新がない群縮約計算（計算条件ファイル）

（図の解説）

- boundary タグにて縮約する際の少数群の構造を指定する。この例では 70 群の詳細群構造を 1～4／5～8／9～19／20～28／29～37／38～46／47～70 の 7 群に縮約している。
- 詳細群の計算結果ファイルは、通常のリスタート計算と同様に file タグにて指定する。
- target タグにて縮約に使用する詳細群計算結果の任意の時点を指定する。この例では、サイクル 2 の計算ステップ 5 の結果を使用する。

```
condensed_group:
  boundary: [4, 8, 19, 28, 37, 46 ,70]
```

図 13 ミクロ断面積の更新を含む群縮約計算（計算条件ファイル）

（図の解説）

- boundary タグにて縮約する際の少数群の構造を指定する。
- その他の指定は不要。

[5] 従来のタグ名は“restart”であったが、“database”に名称変更する。理由については4.3.1を参照することができる。

3.3 実装

3.2.1 述べた通り、群縮約計算処理は procedure パッケージに含まれる CollapseGroupCalculationProcedure クラスにて行う。このクラスは各ゾーンの実効ミクロ断面積を与えられた少数群エネルギー構造に合わせて縮約処理を行い、得られた少数群断面積を各ゾーンに再設定するものである。縮約に使用する中性子束スペクトルは、その時点の中性子束メッシュオブジェクトに設定されている値が使用される。したがって、CollapseGroupCalculationProcedure クラスの処理を呼び出す時点で計算体系のデータを適切に設定しておくことで、ミクロ断面積更新がない群縮約計算とミクロ断面積更新を行う群縮約計算のいずれでも、本クラスを利用することができる。

ミクロ断面積の更新を含む群縮約計算処理の場合、上記の Procedure クラスを呼び出す際に中性子束メッシュには詳細群による計算結果が保存されている。したがって、単純に詳細群による炉心計算後に上記の Procedure クラスの処理を呼び出す実装とする。ミクロ断面積を更新する際は再度詳細群による定数計算を行い、炉心体系スペクトル計算を行う。なお、縮約の前後で中性子束メッシュが保持する中性子束ベクトルの配列数を変更する必要があるが、この処理は CollapseGroupCalculationProcedure クラスや CoreCalculationProcedure クラスの処理の中で自動的に行われるため、これらの Procedure クラスを利用する場合、特に処理を行う必要はない。

この解析処理は5.2で後述するように、ORPHEUS の計算シナリオのひとつとして実装される。

ミクロ断面積の更新がない群縮約計算処理の場合、リスタートファイルに含まれる詳細群解析結果を読み出して、現在の計算体系に設定する。計算体系のセットアップが完了した後に、単純に CollapseGroupCalculationProcedure クラスの処理を呼び出して縮約処理を行う実装とする。

この解析処理も上と同様に、ORPHEUS の計算シナリオのひとつとして実装される。

4. 計算結果処理部の高度化

4.1 分析

4.1.1 従来の実装方法における問題点

これまでの ORPHEUS の開発において、計算結果をリスタートファイルに保存する機能、及びリスタートファイルから必要なデータを取り出す機能が整備された。実装方法としては、MARBLE フレームワークの各モデルや Python ディクショナリをシリアライズし、所定の順序でファイルに書き出したものをリスタートファイルとしていた。この方式は単純なためファイル自体の取り扱いが簡単であるという利点があるが、オブジェクトをシーケンシャルにファイルに保存するため、目的のデータを取り出すためにはファイルの構造を理解した上で一定の手順を踏む必要がある。また、中性子束分布や数密度分布に関しては、特定メッシュにおける値を直接指定して取り出すことができずに、一旦全メッシュのデータをメモリ上にロードしなければならないという問題があった。さらにリスタートファイルに格納される情報は基本的にメッシュ毎の一次データのみであるため、ユーザが通常利用する「1A1」等の集合体アドレスやプレーン番号を使ったアクセスを行うことができず、システム内部にある程度通じた開発者でなければ目的の計算結果にアクセスすることは困難であった。

4.1.2 解決案の検討

既存のリスタートファイルの利点を生かしつつ上で述べた問題点を解決するには、以下の要件を満たす方式を考える必要がある。

① ファイルの取り扱い、及びファイルへのデータの書き出し／読み出し処理の実装が簡単であること。
② シリアライズしたオブジェクトを保存可能であること。
③ ファイル内部の詳細構造を知らなくても、目的とするデータにダイレクトにアクセス可能であること。
④ 特定のメッシュだけのデータを容易に取り出すことが可能であること。
⑤ 集合体アドレスやプレーン番号等、ユーザにとって分かりやすい指標を用いたデータアクセスが可能であること。
⑥ 実用上、十分なパフォーマンスが確保されること。

ここで上記要件のうち、特に③④に着目して、リスタートファイルとして Key/Value 型データストア（以下、KVS）を用いることを検討する。KVS はいわば永続化されるディクショナリ（ハッシュ）のようなものである。標準の Python ディクショナリのように任意のキーワードに値を対応付ける形でファイルに格納することができる。また Python オブジェクトでもシリアライズすることで値として格納することが可能である。したがって、所定のキーワードを使ってオブジェクトを格納したり取り出したりすることが可能であり、上記要件に挙げられた②③④については容易に満たされることが分かる。また KVS は Python コード側からは、ほ

ぼ通常のディクショナリの延長として取り扱うことができるため[6]、実装も比較的容易である（要件①）。

要件⑤については、ユーザが使用しやすい形でのデータアクセス方法を提供するユーティリティクラスを実装し、通常はこのユーティリティクラスの API を介して KVS にアクセスする方式をとることを考える。

要件⑥については、一般に KVS 自体は実装がシンプルなこともあり（RDB 等の複雑なデータベースと比較して）高速に動作する。ただし、Pyhton から使用するに当たっては、Python の処理速度やオブジェクトのシリアライズ／デシリアライズ処理のオーバーヘッドがネックになる可能性がある。パフォーマンスに問題が生じる場合は、KVS ファイルに直接アクセスするレイヤーの実装を C/C++で置き換えて高速化を図ることも考えられる。

[6] KVS 固有のメソッドを用いてより高度な処理を行うことも可能である。

4.2 KVS の検討

4.2.1 代表的な KVS

リスタートファイルとして KVS を用いることを前提として、いくつかの KVS から候補を検討する。以下では候補となる KVS をリストアップし、その特徴について簡単に整理する。

(1) GDBM

GNU による DBM（データベースライブラリ）の実装である。

GNU DBM 自体は標準的な Linux システムであればデフォルトでインストールされる。また、GDBM 用のインターフェイスは Python では gdbm モジュールとして標準的に提供されており、Python ディクショナリと同様の取り扱いで、ファイル上にキーと値のペアを作って高速にアクセスできるようにする。

ライセンスは GPL（Gnu Public Lisence）である。

(2) BerkeleyDB

GDBM と同様のデータベースライブラリの実装である。

Berkeley DB ライブラリ自体は、標準的な Linux システムであれば多くの場合インストールされている。Python では bsddb モジュールとして標準でインターフェイスが提供されており、gdbm モジュールと同様に、bsddb モジュールもまた Python 標準のディクショナリと同様のインターフェイスによる取り扱いを可能とする。

レコードはファイル上にハッシュもしくは B 木形式で記録される。Berkeley DB 自体はトランザクション／ロック／レプリケーションといった処理にも対応し、GDBM より高機能である。

ライセンスは独自のオープンソースライセンスである。

(3) QDBM

GDBM と同様のデータベースライブラリである。GDBM や Berkeley DB 等と比較して、以下に主眼が置かれている。

- 処理が高速であること[7]
- データベースファイルのサイズが小さいこと
- インターフェイスが単純であること

[7] QDBM は Quick Database Manager の略である。

QDBM は Python 標準ではサポートされていないが、qdbm-python という Python 用のインターフェイスモジュールが開発されており、これを導入することで Python から GDBM や Berkeley DB 等と同様の簡便さで利用することが可能である。

ライセンスは QDBM 及びその Python 用インターフェイスモジュールともに GPL である。

(4) Tokyo Cabinet

QDBM の後継として開発されたものであり[8]、QDBM と比較して更なる高速性・より小さなファイルサイズ・マルチスレッド環境への最適化・データベースの堅牢性等を実現している。

レコードはハッシュ・B+木もしくは固定長配列で記録される。

Python 標準ではサポートされていないが、PyTC という Python 用インターフェイスモジュールが開発されており、これを導入することで Python からの利用も可能となる。使用感は GDBM 等とほぼ同様である。

ライセンスは GPL である。また PyTC は BSD ライセンスで配付されている。

(5) Redis

永続化に対応したオンメモリ DB である。メモリ上に記録したデータを非同期でディスクに書き出すことで永続性を実現する。これまでに挙げた DBM とは異なり、使用するに当たっては Redis サーバプロセスを立ち上げる必要がある。

キーに対応する値として、単純な値の他にリストやセットといった複雑なデータ構造を利用可能である。また値のインクリメント／デクリメントといった操作を atomic に実行するための特別なインターフェイスが提供されている。

Python 標準ではサポートされていないが、Redis の配付パッケージには Python 用インターフェイスモジュールも添付されており、これを用いることで Python での利用も可能である。

ライセンスは BSD ライセンスである。

4.2.2 ORPHEUS での利用における要件

4.2.1で列挙した KVS から ORPHEUS のリスタートファイルとして使用するものを選択するにあたって、リスタートファイルに必要とされる項目を検討する。

基本的な前提として、KVS に値として格納されるものはシリアライズした Python オブジェクトとなる。これは開発者／ユーザの立場からは、リスタートファイルから Python オブジェクトを直に取り出せる方が利便性が高いと考えられるからである。ファイルにはデータを適当に分解した形で保存し、ファイルから取り出す際に適切にオブジェクト化する O/R（Object / Relational）マッピング方式も考えられるが、実装の手間とそこから得られるメリットを考慮すると、今回はオブジェクトを直接シリアライズ／デシリアライズする方式が適当と考えられ

[8] QDBM と同一の開発者。

る。格納するオブジェクトの粒度を適切に設計し、シリアライズ／デシリアライズするデータ量を適正なものとすることで、十分なパフォーマンスを確保することは可能であろうと思われる。

上記を念頭に入れて ORPHEUS のリスタートファイルとして利用するに当たって特に重要となる要件を考えると、以下が挙げられる。

- Python で簡便に利用可能なインターフェイスを有していること。
- シリアライズした Python オブジェクトを高速に読み出し／書き出しできること。
- 安定していること（十分な実績があること）

4.2.3 機能・性能の比較検討

(1) パフォーマンス評価

4.2.2で見た通り、KVS を選択するに当たって、今回の要件ではシリアライズした Python オブジェクトの読み出し／書き出し処理のパフォーマンスが重要な要素となる。そこで簡単なベンチマークテストによって実際の読み出し／書き出し処理速度の測定を評価した。測定は以下のように実施した。

- 2 つの Material オブジェクト[9]（一方は 9 個の Nuclide オブジェクトを含み、他方は 4 つの Nuclide オブジェクトを含む）からなる MaterialSet オブジェクトをデータベースファイルに 10 万回書き出すのに要した時間を測定する。
- 上記の処理で生成されたデータベースファイルから MaterialSet オブジェクトを 10 万回読み出すのに要した時間を測定する。

なお、MaterialSet オブジェクトは書き出し／読み出し毎に Python 標準の cPickle モジュールを用いてシリアライズ／デシリアライズしている。

測定結果を表 1に示す。なお処理時間は上記の測定を 5 回ずつ行ったものを平均した値である。

[9] MARBLE フレームワークの material パッケージにて実装されているクラス。Nuclide クラスや MaterialSet クラスも同様。

表 1 KVS のパフォーマンス比較

KVS	書き出し [sec]	読み出し [sec]	ファイルサイズ [MB]
GDBM（同期）	35.87	9.05	47.9
GDBM（非同期）	12.19	8.95	47.9
BerkeleyDB（ハッシュ）	23.20	7.85	83.4
BerkeleyDB（B 木）	9.21	4.37	90.9
QDBM	10.77	9.92	45.6
TokyoCabinet（ハッシュ）	9.62	3.94	46.1
TokyoCabinet（B+木）	5.65	4.15	44.4
Redis	27.94	－[10]	36.9

※　BerkeleyDB 及び TokyoCabinet では、データ記録方式としてハッシュ／B 木の双方を用いたケースを検討している。但し B 木の結果はデータを記録するキーの構成方法によって変わりうるため、ハッシュ方式の結果と合わせて総合的に評価する必要がある。

(2) KVS の総合的な比較

パフォーマンスや機能面など、今回検討した KVS を総合的に比較した結果は表 2の通りである。

表 2 KVS の比較まとめ

	Python 対応	性能	使い易さ	ファイルサイズ	高機能性	情報の入手性
GDBM	◎	△	◎	○	×	◎
BerkeleyDB	◎	△	◎	×	○	◎
QDBM	○	○	◎	○	△	○
TokyoCabinet	○	◎	◎	○	○	○
Redis	○	△	△	◎	◎	△

特にパフォーマンス面での評価が高く、また使い易さ等に大きな問題もないことから、今回 ORPHEUS で使用するリスタートファイル用 KVS として TokyoCabinet を採用する。

[10] cPickle にてシリアライズした値を格納したデータベースから正常にデシリアライズできなかったため、読み出しテストは実施していない。

4.3 設計

4.3.1 データベースファイル設計

(1) データベースファイルの機能要件

リスタートファイルとして使用する KVS のデータ構造が満たすべき基本的な要件として、以下が挙げられる。

- 1 サイクル分の情報が一つのファイルに全て含まれること（計算開始前の状態も含む）
- 全ての一次情報に対してダイレクトに読み出し／書き出しができること

本 KVS ファイルには計算結果のみならず、計算体系を構築するために必要な各種の情報（オブジェクト）や計算コードが出力した計算結果ファイルに関する情報など、ある解析処理にまつわる種々の情報を格納する。したがって、以降はその呼称をリスタートファイルではなく ORPEHUS データベースファイル（ODB ファイル）と改め、汎用的な解析情報データベースとして取り扱うこととする。

(2) 基本構造

ODB ファイルの構造として、KVS のキー構造を考える。

ODB ファイルに格納する情報は、基本的には現行のリスタートファイルに準じるもので、以下の通りとなる。

【計算ステップを通じて変わらない情報】

- 計算条件等の計算全体に関わるサマリ情報
- CoreProperty オブジェクト
- ZoneSet オブジェクト
- Pattern オブジェクト
- MeshGeometry オブジェクト（中性子束メッシュ用）
- MeshGeometry オブジェクト（マテリアルメッシュ用）
- メッシュ番号等の変換テーブル

【計算ステップ毎に保存する情報】

- 計算ステップ毎のサマリ情報
- ゾーン毎の CellModelConverter オブジェクト
- メッシュ毎の FluxRegionInfo オブジェクト（中性子束）
- メッシュ毎の MaterialRegionInfo オブジェクト（マテリアル）
- ゾーン毎の MicroscopicEffectiveCrossSectionSet オブジェクト（ミクロ断面積）
- 制御棒下端位置情報

【その他】

- 計算コードが出力した計算結果ファイル（ファイルパス）

ここでオブジェクトに関しては、これまでの検討で出てきたようにシリアライズしてKVS に値として格納する。シリアライズした Python オブジェクトを格納した場合、オブジェクトの属性値へのアクセスは基本的にはオブジェクトを一旦デシリアライズしてからアクセスすることになる。すなわち、各オブジェクトの属性値へ直接アクセスするようなキーは考えないものとする。

中性子束メッシュやマテリアルメッシュといった情報へのアクセスは、これまでのリスタートファイルの実装では全メッシュをまとめて取り扱うことしかできなかったが、(1)の要件で挙げたように、メッシュ単位で読み出し／書き出しできるようにする。これにより、例えば集合体のシャッフリング処理を行う際、過去サイクルのメッシュオブジェクトを全てメモリ上に展開する必要なく、目的とするメッシュオブジェクトのみを取り出す実装が可能となる。

中性子束がある値からある値の範囲にあるオブジェクトを全て取り出したいといった逆引きのニーズに対して、KVS 側で対応する場合は、一般的には逆引き用の無数のキーを設定する必要がある。（つまり中性子束の値をキーとして、対応する値にその中性子束を表す中性子束オブジェクトを設定しておく。）KVS に格納するデータの種類によっては採用される手法であるが、今回のようなデータを格納するケースでは現実的ではない。したがって、ここで挙げたような逆引きのニーズに対しては、KVS 自体で対処するのではなく、KVS を管理する上位層のロジックにて対処するものとする。

計算条件等の計算全体に関わるサマリ情報や計算ステップ毎のサマリ情報に関しては、各項目を所定のキーを用いてアクセス可能とする。

以上を考慮して ODB ファイルに格納するデータのキーを検討した結果を表 3から表5に示す。ここでキーが“STEP:cell_model:ZONENO”等となる場合、STEP は計算ステップ番号、ZONENO はゾーン番号を表している。すなわち、実際のキーは“2:cell_model:15”等である。なお cell_model の部分を主キー、ZONENO の部分を副キー、STEP を STEP 修飾子と呼称する。

STEP 修飾子は 0 以上の整数である。1 以上の値は個別の計算ステップを表し、0 は計算開始直前の時点を表す。副キーとしては ZONENO の他に、メッシュ番号を表す MESHNO、マテリアル番号を表す MATNO、セルアドレスを表す CELLNO、計算コード名と使用ファイル機番を表す CODENAME 及び UNITNO を定義する。

表 3 ODB ファイルのデータアクセス用キーの一覧（1/3）

キー	値
code_name	コード名（ORPHEUS）
code_version	コードバージョン （例：1.0）
user_name	計算実行ユーザアカウント名
case_name	解析ケース名
calc_mode	計算モード名 standard：燃焼解析 constant_micro：燃焼解析（ミクロ更新なし） collapse：群縮約計算（ミクロ更新なし） collapse_micro_update：群縮約計算 control_rod：制御棒価値計算
core_name	炉心名
cycle_name	サイクル名
cycle_number	積算サイクル数
num_steps	計算ステップ数
ng	エネルギー群数
library	炉定数ライブラリ名 （例：JENDL-3.2）
xs_solver	定数計算ソルバー名 （例：slarom_uf）
core_solver	炉心計算ソルバー名 （例：marble_citation）
burnup_solver	燃焼計算ソルバー名 （例：burnup）
calc_option:CODENAME	計算コード毎の詳細オプション （ディクショナリ）

calc_option では、計算条件ファイルで指定した各計算コードの詳細オプションを設定する。CODENAME には対象とするソルバー名を与える。これはキーxs_solver, core_solver 及び burnup_solver で与えられる名称と同一のものである。

表 4 ODB ファイルのデータアクセス用キーの一覧（2/3）

キー	値
core_property	CoreProperty オブジェクト
zone_set	ZoneSet オブジェクト
pattern	Pattern オブジェクト
flux_mesh_geometry	MeshGeometry オブジェクト （中性子束メッシュ用）
material_mesh_geometry	MeshGeometry オブジェクト （マテリアルメッシュ用）
material_number_table	マテリアルメッシュ番号からマテリアル番号への変換テーブル
zone_number_table	マテリアル番号からゾーン番号への変換テーブル
mesh_number_table	マテリアル番号からマテリアルメッシュ番号への変換テーブル

表 5 ODB ファイルのデータアクセス用キーの一覧（3/3）

キー	値
STEP:generated_time	日付情報 （例：Tue, 26 Feb 2010 09:22:46 +0900）
STEP:burnup_period	燃焼期間［日］
STEP:power	炉心相対出力［%］
STEP:asy_axial_pos:CELLNO	（制御棒）集合体軸方向下端位置
STEP:eigenvalue	固有値
STEP:cell_model:ZONENO	CellModelConverter オブジェクト
STEP:flux_info:MESHNO	FluxRegionInfo オブジェクト
STEP:material_info:MATNO	MaterialRegionInfo オブジェクト
STEP:fission_spectrum:ZONENO	核分裂スペクトル （numpy.array）
STEP:micro_cross_section:ZONENO	実効ミクロ断面積オブジェクト （MicroscopicEffectiveCrossSectionSet）
STEP:nuclides:ZONE_NO	核種（Nuclide オブジェクトのリスト）
STEP:code:CODENAME:UNITNO: ZONE_NO	計算コードの入出力ファイルのパス （定数計算）
STEP:code:CODENAME:UNITNO	計算コードの入出力ファイルのパス （炉心計算）
STEP:code:CODENAME:UNITNO: MATNO	計算コードの入出力ファイルのパス （燃焼計算）

4.3.2 クラス設計

ODB ファイルの取り扱いは大きく分けて 2 つの層で行う。ひとつはデータベースを直接操作する“低レベル層”であり、データベース操作用の Python インターフェイスを駆使してデータの書き出し／読み出しを行う。もうひとつはこの低レベル層の上位に位置する“高レベル層”であり、データベースのデータ構造に直接依存しないデータアクセス用のインターフェイスを提供する。すなわち、集合体位置ラベル等のより分かりやすい指標を基にしたデータアクセスを可能とするインターフェイスを実装する。

以上の点を考慮して ODB ファイルを管理するためのクラス構造を検討した結果を図 14に示す。また、クラスの一覧を表 6に示す。

“低レベル層”に相当するクラスは、OrpheusDatabase クラスである。このクラスは抽象クラスであり、実際の実装はサブクラスである KvsOrpheusDatabase クラスや RdbOrpheusDatabase クラスにて行われる。これらのクラスには実際に ODB ファイルにアクセスして、必要なデータの読み出しや書き出しを行うインターフェイスが定義される。今回の検討では ODB ファイルのストレージエンジンとして KVS を採用しているが、今後の改良の結果、場合によっては sqlite のようなリレーショナルデータベース（RDB）をストレージエンジンに採用する可能性もある[11]。そこで ODB ファイルへのアクセス用インターフェイスは抽象化しておき、実際の実装は変更可能なように設計しておく[12]。

“高レベル層”に相当するクラスとして OrpheusDatabaseHandler クラスを定義する。これはこれまでの実装における RestartFileManager と同等の位置づけのものである。このクラスは抽象クラスであり、低レベル層に対応するサブクラスとして KvsOrpheusDatabaseHandler クラスや RdbOrpehusDatabaseHandler クラスが定義される。これらはそれぞれ KVS 及び RDB 形式のデータベースに対応するものである[13]。

KVS に格納されたデータにアクセスする場合、4.3.1のデータベース設計で見たとおり、例えば中性子束データ等はステップ単位・メッシュ単位でしか取得できない。そこであるメッシュの集合全体に対して平均化された中性子束を求める場合などでは、KVS から得られたメッシュ毎のデータを基に適当な演算を行う必要がある。OrpheusDatabaseHandler クラスではこの演算処理が管理される。すなわち、OrpheusDatabaseHandler の edit メソッドに対して、集合体位置情報を表す PositionInfo オブジェクト、“FLUX（中性子束）”“ND（数密度）”等の編集対象を表すキーワード及び“GET（取得）”“AVE（平均化）”等の対象に対する処理を表すキーワードの 3 つを与えると、これらの情報から、指定された場所の指定された対象に対する指定された処理を行い、その結果が返される。

edit メソッドに対して与える集合体位置情報を表す PositionInfo オブジェクトは、セルアドレス・軸方向範囲を表す Zrange オブジェクト・集合体内のピン位置を表すピン番号で構成さ

11 RDB を採用する利点は、データ集合に対する平均化や最大値の算出といった処理をストレージエンジン上で実行できる点である。対象とするデータ量が増えれば、同処理を Python ロジックで実行するよりも格段に高いパフォーマンスが得られると期待される。
12 今回は KVS 用の OrpheusDatabase のみ実装する。
13 低レベル層と同様に、KVS 用の OrpheusDatabaseHandler のみ実装する。

れる。これらの情報から中性子束や数密度といった一次情報にアクセスするために必要なメッシュ番号等への変換は LocationToMeshInterpreter クラスにより行われる。PositionInfo オブジェクトへのデータの指定例や定義について図 15及び図 16に示す。また対象とするデータの種類及び操作について、現時点で実装されているものを図 17に示す。

なお、燃料ピン位置を定義する番号は図 18のように定義される。集合体の回転位置については、MAX KEY（常陽）もしくはハンドリングヘッドキー（もんじゅ）が図の位置にある場合が基準となる。各集合体についてこの基準となるキーが実際にどこにあるか（各集合体がどのような回転状態で装荷されているか）については、pattern パッケージの CellData オブジェクトの maxKeyPosition 属性として保持されている。この属性値は 0 から 5 の整数値をとり、0 を基準として集合体が反時計回りに 60 度回転するごとに 1, 2, 3...となる。maxKeyPosition 属性が 0 となる集合体回転位置の定義について図 19に示す。

表 6 ORPHEUS データベース関連クラス一覧

クラス名	説明
OrpheusDatabase	“低レベル層”の ODB ファイル操作用クラス。 抽象クラス。
KvsOrpheusDatabase	OrpheusDatabase のサブクラス。KVS 形式の DB を処理する。
RdbOrpheusDatabase	OrpheusDatabase のサブクラス。RDB 形式の DB を処理する。
OrpheusDatabaseHandler	“高レベル層”の ODB ファイル操作用クラス。
KvsOrpheusDatabaseHandler	OrpheusDatabaseHandler のサブクラス。KVS 形式の DB に対応する。
RdbOrpheusDatabaseHandler	OrpheusDatabaseHandler のサブクラス。RDB 形式の DB に対応する。
LocationToMeshInterpreter	抽象的なデータ位置指定情報から具体的なメッシュ番号等の情報への変換を行う。
PositionInfo	ステップ番号・集合体位置・軸方向位置等の位置情報をカプセル化したクラス。

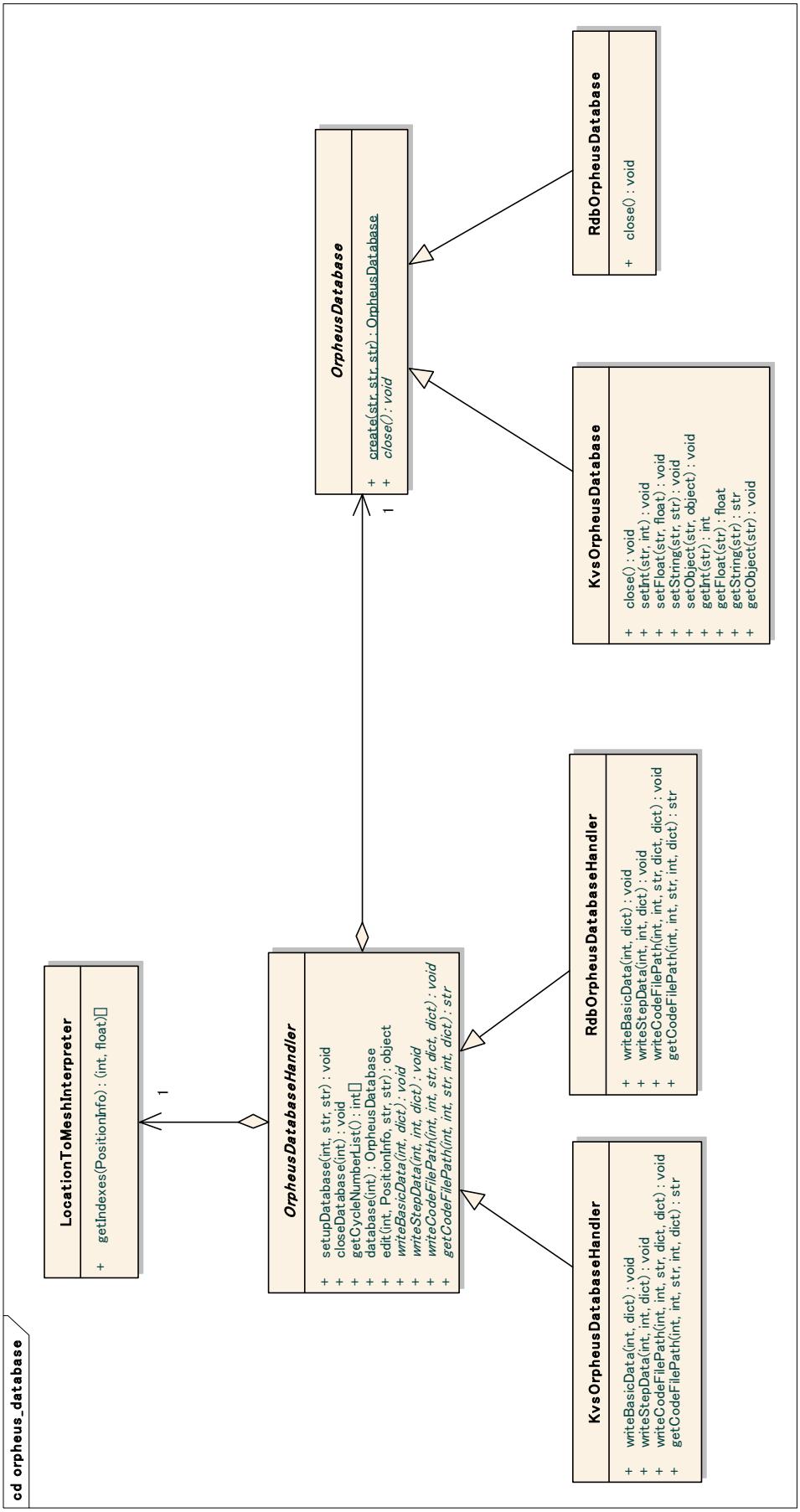

図 14 ORPHEUS データベース関連クラス図

【PositionInfo への時間及び位置情報の指定の例】
○ ステップ番号 1、集合体 1A1、プレーン番号 1
positionInfo = PositionInfo(1, “1A1”, 1)
positionInfo = PositionInfo(step=1, asyIndex=”1A1”, zIndex=1)

○ 全ステップ、全集合体、軸方向全プレーン
positionInfo = PositionInfo(“ALL”, “ALL”, “ALL”)
positionInfo = PositionInfo(step=”ALL”, asyIndex=”ALL”, zIndex=”ALL”)

○ ステップ番号 1、集合体ラベル PFD001、軸方向全プレーン
positionInfo = PositionInfo(1, “PFD001”, “ALL”)
positionInfo = PositionInfo(step=1, asyIndex=”PFD001”, zIndex=”ALL”)

○ ステップ番号 2、集合体アドレス 2A1、プレーン番号 (1, 2, 3)
positionInfo = PositionInfo(2, “2A1”, [1, 2, 3])
positionInfo = PositionInfo(step=2, asyIndex=”2A1”, zIndex=[1, 2, 3])

○ ステップ番号 2、集合体アドレス 2A1 及び 2A2 の対称位置、軸方向全プレーン
positionInfo = PositionInfo(2, “2A1+, 2A2+”, “ALL”)
positionInfo = PositionInfo(step=2, asyIndex=”2A1+, 2A2+”)

○ ステップ番号 2、集合体アドレス (2, 3, 4, 5, 6, 7)、軸方向範囲 Zrange オブジェクト
positionInfo = PositionInfo(2, [2, 3, 4, 5, 6, 7], Zrange(1.5, 3.5))
positionInfo = PositionInfo(step=2, asyIndex=[2, 3, 4, 5, 6, 7], zIndex=Zrange(1.5, 3.5))

○ ステップ番号 2、集合体アドレス (PFD002, PFD003, PFD004)、プレーン番号 2
positionInfo = PositionInfo(2, [“PFD002”,”PFD003”,”PFD004”], 2)
positionInfo = PositionInfo(step=2, asyIndex=[“PFD002”,”PFD003”,”PFD004”], zIndex=2)

○ ステップ番号 2、集合体アドレス 000 及び 1A1 の対称位置、軸方向全プレーン
positionInfo = PositionInfo(2, [“000”, “1A1+”], “ALL”)
positionInfo = PositionInfo(step=2, asyIndex=[“000”, “1A1+”], zIndex=”ALL”)

○ ステップ番号 2、メッシュ番号 (100001, 100002, 100003)
positionInfo = PositionInfo(2, [100001, 100002, 100003])
positionInfo = PositionInfo(step=2, meshIndex=[100001, 10002, 10003])

図 15 PositionInfo オブジェクトへの時間及び位置情報の指定例

【PositionInfo の詳細】
PositionInfo オブジェクトは step, asyIndex, pinIndex, zIndex, meshIndex の 5 つの属性値をもつ。但し meshIndex が指定されている場合は asyIndex/pinIndex/zIndex の指定値は無視される。各属性が取りうる値は次の通り。

（属性 step）
整数もしくは”ALL”。”ALL”は全ステップを表す。

（属性 asyIndex）
集合体位置ラベル，集合体ラベル、セルアドレス及びこれらのリスト。集合体位置ラベルについては、”+”を付加することにより対称位置を表す。”ALL”を指定した場合、全集合体を表す。

（属性 pinIndex）
燃料ピン番号及びこれらのリスト。

（属性 zIndex）
軸方向プレーン番号、Zrange オブジェクト及びこれらのリスト。

（属性 meshIndex）
メッシュ番号及びそのリスト。

図 16 PositionInfo による位置情報指定の詳細

【対象データ】
KEFF：実効増倍率
MICROXS：実効ミクロ断面積
FLUX：中性子束
ND：数密度
FLUX_VOL：中性子束メッシュ体積
MATERIAL_VOL：マテリアルメッシュ体積

【操作】
GET：取得
AVE：平均化
SUM：総和

図 17 edit メソッドで取り扱うデータ種類及び操作

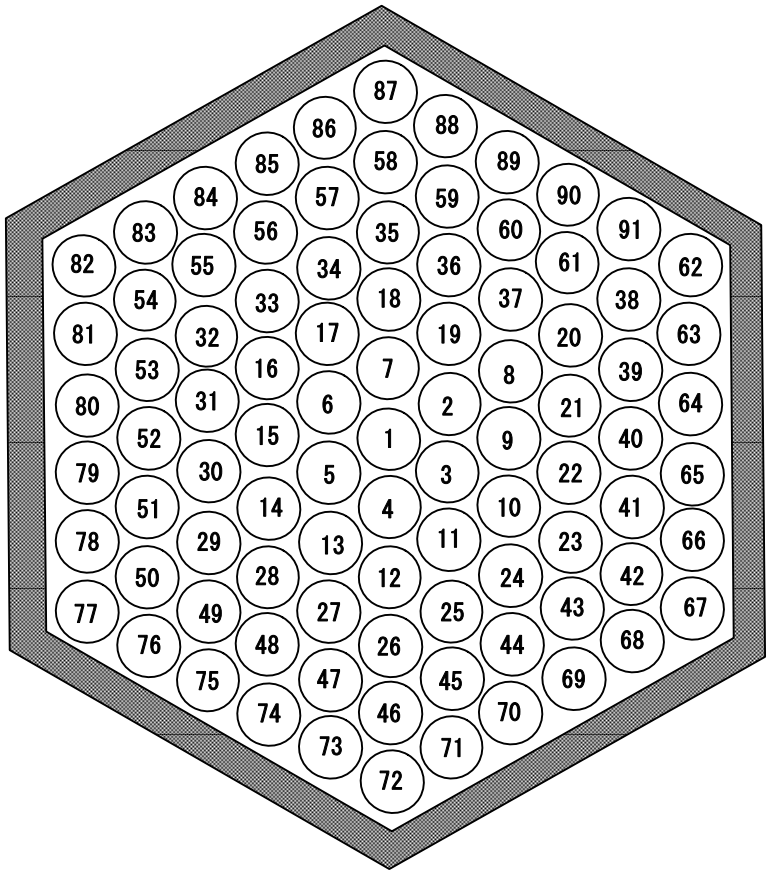

図 18 燃料ピン番号の定義

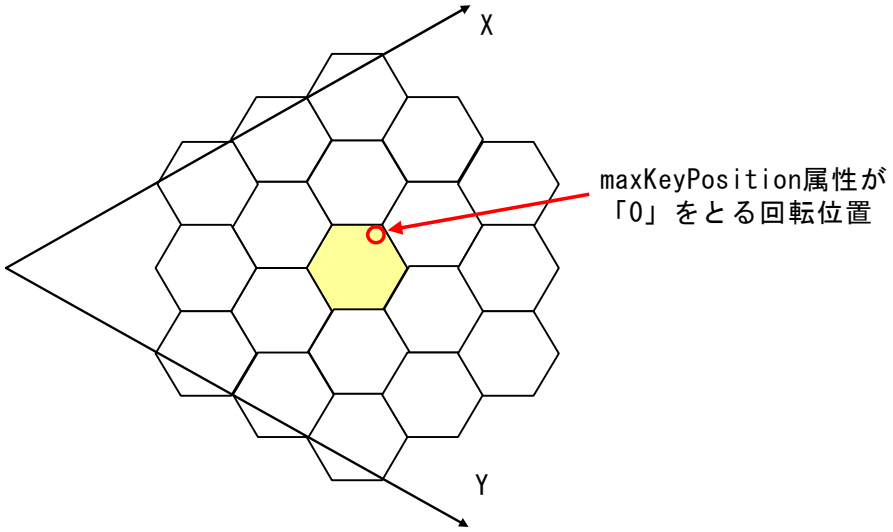

図 19 集合体回転位置の定義

5. ORPHEUS の全体制御処理

5.1 全体の構成

ORPHEUS の全体的な処理フローの制御は、全体を司る Controller オブジェクトと Controller の配下で各部分を管理する複数の Manager オブジェクトによって行われる。各クラスの一覧を表 7に示す。

表 7 全体制御部のクラス一覧

クラス名	説明
Contoroller	全体をトップレベルで司るクラス。各 Manager オブジェクトを管理する。
InputManager	各種のユーザ入力ファイルの読み込みを管理するクラス。
InputCheckManager	ユーザ入力ファイル間の不整合をチェックするためのクラス。
CoreManager	計算体系を構築するオブジェクト群の生成・初期化を管理するクラス。
ListManager	処理過程やエラーメッセージ等の帳票出力を管理するクラス。
OrpheusDatabaseHandler	ORPHEUS データベースファイルを管理するクラス。
Scenario	後述する計算シナリオを管理するクラス。

これまでに実装したクラスに加えて、今回新たに InputCheckManager クラスを追加している。このクラスはユーザが与える各種ファイルを読み込んだ後でその内容についてチェックを行い、データの不整合があった場合には例外（OrpheusStandardError）を発生させる。シンタックスエラーやごく単純なエラー（例えば正数値を指定すべき箇所で負数を指定するなど）についてはファイルを読み込む際に各パーサー部にてエラーチェックされるので、このクラスでは複数の入力ファイルにまたがる不整合など、ファイル単体のチェックでは判定が困難なエラーについて確認を行う。

また、従来は計算フローの制御は CalculationManager 及び CrossSectionCaclulator / CoreCalculator / BurnupCalculator 等のクラスにて実施していたが、それらを廃止し、後述する「計算シナリオ」を扱う Scenario クラスを追加している。

さらに、これまでのリスタートファイルを廃止して、新たに ORPHEUS データベースファイル（ODB ファイル）を導入したため、従来の RestartFileManager クラスを廃止し、4.3.2 で説明した OrpheusDatabaseHandler クラスを新たに使用する。

5.2 計算シナリオ

従来の ORPHEUS は定数計算・炉心計算・燃焼計算という基準解析ケースを処理するだけであったが、今回新たに群縮約計算のサポートを追加した。特に、計算ステップの途上でミクロ断面積を更新しつつ、少数群に縮約して炉心計算・燃焼計算を行うような計算フローは、従来のORPHEUSの計算処理を取り扱うCalculationManagerクラスの実装では対処できない。また、将来の ORPHEUS の機能拡張を考えると、今後も新たな計算フローを取り扱う必要性に対応できるような実装であることが望まれる。そこで様々な計算フローを計算シナリオクラスとしてカプセル化し、入力ファイルにて与えられる計算条件から適切な計算シナリオオブジェクトを生成・実行する仕組みを取り入れる。これらの計算シナリオオブジェクトは必要に応じて交換可能とし、新たな計算フローに対しては新たな計算シナリオオブジェクトを定義することで対応できるようにする。

計算シナリオを実装するクラスとして Scenario クラスを定義する。このクラスは共通インターフェイスを定める抽象クラスであり、計算シナリオ毎に異なる Scenario クラスを実装する。Scenario クラスの共通インターフェイスとして定義されるメソッドは perform のみであり、計算シナリオを実行する際には Scenario クラスのオブジェクトを生成した後、perform メソッドを発行する。以降、実際の計算処理は Scenario オブジェクトに引き継がれる。なお Scenario オブジェクトの生成及び実行管理は Controller にて行うものとする。

Scenario クラスの実装では、MARBLE フレームワークの procedure パッケージ[14]を利用する。例えば基準解析シナリオの場合は、定数計算を処理する CellCalculationProcedure，マクロ定数計算を処理する MacroXSCalculationProcedure，炉心計算を処理する CoreCalculationProcedure，燃焼計算を処理する BurnupCalculationProcedure さらに制御棒操作のための ControlRodOperationProcedure を用いる。ミクロ断面積更新を含む群縮約計算シナリオの場合は、これらに加えて CollapseGroupCalculationProcedure クラスを活用して実装する。

今回は計算シナリオの実装として、基準解析ケースを表す ScenarioStandard クラス、ミクロ断面積を固定した炉心計算・燃焼計算を行う ScenarioConstantMicro クラス、詳細群計算結果を基に少数群縮約処理を行い一連の解析を実施する ScenarioCollapseGroup クラス、同様の群縮約解析ではあるものの適宜詳細群計算を再実行し、ミクロ断面積を更新しつつ一連の解析を行う ScenarioCollapseGroupWithMicroUpdate クラス、さらに制御棒価値計算等に資するようにブランチ計算[15]を行う ScenarioControlRod クラスを用意する。計算シナリオクラスの実装については、今後必要に応じて随時追加することで、様々な解析ケースに対応することができる。

計算シナリオ関連クラスのクラス図を図 20に示す。またクラス一覧を表 8に示す。

14 詳細は参考文献[8]を参照することができる。
15 既存の計算結果を引き継いだうえで、ステップ毎に炉心計算のみを実行する。

表 8 計算シナリオクラス一覧

クラス名	説明
Scenario	計算シナリオを表す抽象クラス。
ScenarioStandard	基準解析ケースを実装した Scenario クラス。
ScenarioConstantMicro	実効ミクロ断面積を固定したまま炉心計算・燃焼計算を行う処理を実装した Scenario クラス。
ScenarioCollapseGroup	ミクロ断面積更新を行わない群縮約計算解析ケースを実装した Senario クラス。
ScenarioCollapseGroup-WithMicroUpdate	ミクロ断面積更新を含む群縮約計算解析ケースを実装した Scenario クラス。
ScenarioControlRod	ブランチ計算処理を実装した Scenario クラス。

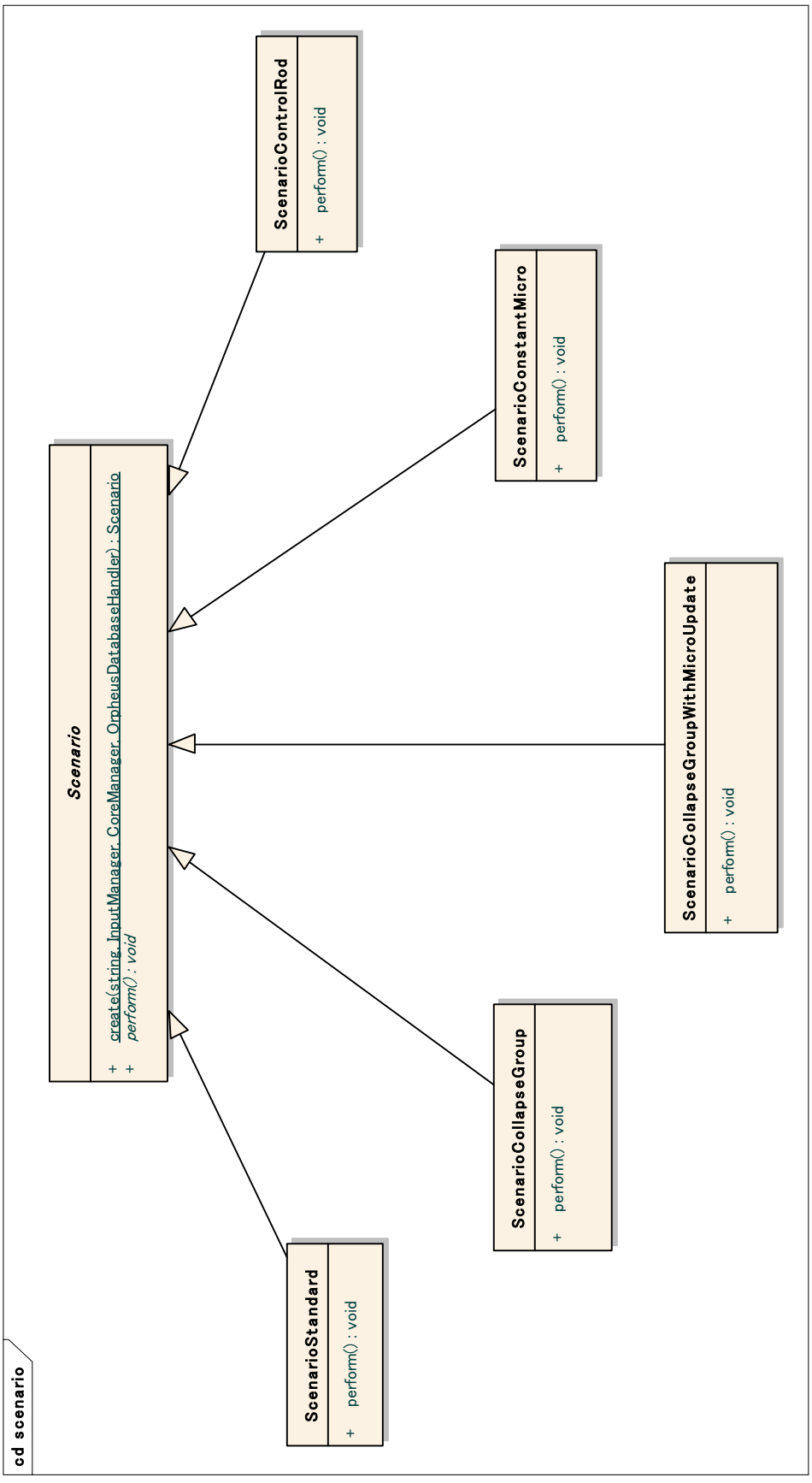

図 20 計算シナリオ関連クラス図

5.3 計算モード

標準解析手法以外の解析フローをサポートするに当たって、計算条件ファイルにて計算モードを明示的に指定するようにする。すなわち、計算条件ファイルに“calc_mode”タグを新設し、ここで与えられる値に応じて処理フローを切り替えるものとする。システムは計算モードに応じて計算体系を構築するオブジェクト群の初期化を行い、各計算モードに対応した計算シナリオを実行する。calc_mode タグに与える値の一覧は表 9の通り。また各計算モードに対応する計算シナリオクラスを表 10に示す。

なお、計算モードに応じて、計算条件ファイルで与える必要がある入力データが変更になる。必須の入力データが未指定の場合は例外（OrpheusStandardError）が発生する。

表 9 計算モード一覧

計算モード	意味
standard	標準解析モード。
constant_micro	ユーザ指定のミクロ断面積を用いた解析を実行。
collapse	群縮約計算。（ミクロ断面積更新なし）
collapse_micro_update	群縮約計算。（ミクロ断面積更新を含む）
control_rod	制御棒価値計算。

表 10 計算モードと計算シナリオの対応

計算モード	対応する計算シナリオ
standard	ScenarioStandard
constant_micro	ScenarioConstantMicro
collapse	ScenarioCollapseGroup
collapse_micro_update	ScenarioCollapseGroupWithMicroUpdate
control_rod	ScenarioControlRod

6. まとめ

高速炉の実用化に向けて高速炉実機炉心の核特性予測精度を向上させることは、合理的で高性能な炉心を設計してプラントの経済性の向上を図る上でも信頼性および安全性の裕度をより高める上でも、重要な研究課題である。これまでの研究では、炉定数調整法を適用することにより、JUPITER 等の臨界実験の成果を活用して核設計精度の向上を達成しているが、高速炉の炉心設計では燃焼核特性の精度向上も重要である。そこで、高速炉実機解析を効率的かつ柔軟に実施するためのシステムの開発を実施している。

本研究開発では、これまでに実施した「高速炉実機燃焼解析システムの高度化」の内容をさらに発展させ、制御棒操作機能の高度化や群縮約計算機能の実装を行った。さらにリスタートファイルの概念を拡張したデータベースを設計し、様々な解析データを効率的に取り扱うための仕組みを実装した。

参考文献

1. 兵頭 秀昭，巽 雅洋：「高速炉実機燃焼解析システムの開発」，JAEA-Data/Code 2006-018 (2006)
2. 平井 康志，兵頭 秀昭，巽 雅洋：「高速炉実機燃焼解析システムの開発（その２）」，JAEA-Data/Code 2007-019 (2007)
3. 平井 康志，兵頭 秀昭，巽 雅洋，横山 賢治：「高速炉実機燃焼解析システムの開発（その３）」，JAEA-Data/Code 2008-021 (2008)
4. 横山 賢治，平井 康志，兵頭 秀昭，巽 雅洋：「高速炉実機燃焼解析システムの高度化」，JAEA-Data/Code 2009-016（2010）
5. 巽 雅洋，横山 賢治：「次世代炉物理解析システムのためのフレームワーク開発」，JAEA-Data/Code 2007-020 (2007)
6. 平井 康志，兵頭 秀昭，巽 雅洋，神 智之，横山 賢治：「次世代炉物理解析システムのためのフレームワーク開発（その２）」，JAEA-Data/Code 2008-020 (2008)
7. 横山 賢治，平井 康志，兵頭 秀昭，巽 雅洋：「次世代炉物理解析システムのためのフレームワーク開発（その３）」，JAEA-Data/Code 2009-012（2010）
8. 横山 賢治，平井 康志，巽 雅洋：「次世代炉物理解析システムのためのフレームワーク開発（その４）」，JAEA-Data/Code 2010-015（2010）.[To be published]

付録

This is a blank page.

A. モジュール一覧

ORPHEUS の実装コードについて詳細を示す。

A.1 全体制御部

ORPHEUS の全体処理フローの制御及び入出力や計算処理の制御を行っているモジュールは以下のとおりである。

- Controller
- InputManager
- InputCheckManager
- ListManager
- CoreManager
- ObjectBuilder
- ObjectReloader
- Scenario
- ScenarioStandard
- ScenarioConstantMicro
- ScenarioCollapseGroup
- ScenarioCollapseGroupWithMicroUpdate
- ScenarioControlRod

実装の詳細については後述する。

A.2 入力処理

以下のモジュールは、InputManager による制御の下、各種の入力ファイルの取り扱いを行っている。

- InputInterface
- CalcConditionFile
- CorePropertyFile
- GeometryFile
- MaterialFile
- PatternFile
- CrossSectionFile
- Parser
- YamlParser
- Validator
- YamlValidator

実装の詳細については後述する。

A.3 ユーティリティ関連

以下のモジュールは、ORPHEUS 外部の関連ツール（計算結果編集ツール等）から利用されることを想定されたものである。主に ORPHEUS データベースを取り扱うクラスで構成される。

- LocationToMeshInterpreter
- OrpheusDatabase
- OrpheusDatabaseHandler
- OrpheusException

実装の詳細については後述する。

This is a blank page.

B. 実装の詳細

実装の詳細を次頁以降に示す。

This is a blank page.

Controller.py


```
Help on module Controller:

NAME
    Controller - Controller

FILE
    marble/orpheus/main/Controller.py

CLASSES
    Controller

    class Controller
     |  This class controlls ORPHEUS analysis flow.
     |
     |  Methods defined here:
     |
     |  __init__(self)
     |      Constructor.
     |
     |  execute(self)
     |      Execute lattice/core/burnup calculations.
     |
     |  finalize(self)
     |      Finalize system.
     |
     |  initialize(self, fileName)
     |      Initialize calculation system.
     |
     |      Parameters:
     |        fileName -- calc condition file name.

DATA
    CODE_NAME = 'ORPHEUS'
    CODE_VERSION = '1.0'
    ERROR = 3
    NOTE = 1
    PLAIN = 0
    WARNING = 2
```

InputManager.py Page 1/2

```
Help on module InputManager:

NAME
    InputManager - InputManager

FILE
    marble/orpheus/main/InputManager.py

CLASSES
    InputManager

    class InputManager
     |  This class manages ORPHEUS input files.
     |
     |  Methods defined here:
     |
     |  __init__(self)
     |      Constructor.
     |
     |  calcConditionFileName(self)
     |      Get calc condition file name.
     |
     |      Return:
     |        calc condition file name.
     |
     |  checkCalculationMode(self)
     |      Check and set calculation mode.
     |      The calculatin mode is as follows.
     |
     |        standard: depletion calculation.
     |        constant_micro: using pre-calculated cross section.
     |        collapse: energy-group collapse calculation.
     |        collapse_micro_update: energy-group collapse calculation. (micro update)
     |        control rod: control rod operation.
     |
     |  getCalcCondition(self)
     |      Get a dictionary that stores calculation conditions.
     |
     |      Return:
     |        calculation condition. (dict)
     |
     |  getCorePropertyData(self)
     |      Get data for initialization of core property object.
     |
     |      Return:
     |        core property initialization data. (dict)
     |
     |  getCrossSectionData(self)
     |      Get data for initialization of cross section objects.
     |
     |      Return:
     |        user defined microscopic cross section data. (dict)
     |
     |  getGeometryData(self)
     |      Get data for initialization of geometry objects.
     |
     |      Return:
     |        geometry initialization data. (dict)
     |
     |  getMaterialData(self)
     |      Get data for initialization of material objects.
     |
     |      Return:
     |        material initialization data. (dict)
     |
     |  getPatternData(self)
     |      Get data for initailization of pattern object.
     |
     |      Return:
     |        pattern initialization data. (dict)
     |
     |  initialize(self, calcConditionFileName)
     |      Read calc condition file and initialize CalcConditionFile object.
     |
     |      Parameters:
     |        calcConditionFileName -- calc condition file name.
     |
     |      Exceptions:
     |        OrpheusStandardError for invalid calc condition file I/O error.
     |
     |  readCorePropertyFile(self)
     |      Read core property file.
     |
     |      Exceptions:
     |        OrpheusStandardError for file I/O error.
     |
     |  readCrossSectionFile(self)
     |      Read cross section file.
     |
     |      Exceptions:
     |        OrpheusStandardError for file I/O error.
     |
     |  readGeometryFile(self)
     |      Read geometry file.
```

InputManager.py Page 2/2

```
        Exceptions:
          OrpheusStandardError for file I/O error.

    readMaterialFile(self)
        Read material file.

        Exceptions:
          OrpheusStandardError for file I/O error.

    readPatternFile(self)
        Read pattern file.

        Exceptions:
          OrpheusStandardError for file I/O error.

    secondInitialize(self, coreMgr)
        Initialize CalcConditionFile object. (2nd initialization stage)

        Parameters:
          coreMgr -- CoreManager object.

    writeRayTraceInfo(self, coreMgr)
        Dump information on raytracing geometries.

        Parameters:
          coreMgr -- CoreManager object.

DATA
    CODE_NAME = 'ORPHEUS'
    CODE_VERSION = '1.0'
    ERROR = 3
    NOTE = 1
    PLAIN = 0
    WARNING = 2
```

InputCheckManager.py Page 1/1

```
Help on module InputCheckManager:

NAME
    InputCheckManager - InputCheckManager

FILE
    marble/orpheus/main/InputCheckManager.py

CLASSES
    InputCheckManager

    class InputCheckManager
     |  This class checks wheter user input files adjusts logically or not.
     |
     |  Methods defined here:
     |
     |  __init__(self)
     |      Constructor.
     |
     |  checkUserInputs(self, coreMgr)
     |      Check user input data and raise exceptions if error is detected.
     |
     |      Parameters:
     |        coreMgr -- CoreManager object.
     |
     |      Exceptions:
     |        OrpheusStandardError for illegal input information.

DATA
    CODE_NAME = 'ORPHEUS'
    CODE_VERSION = '1.0'
    CONTROL_ROD = 2
    ERROR = 3
    FUEL = 0
    NOTE = 1
    PLAIN = 0
    REFLECTOR = 1
    SOURCE = 3
    WARNING = 2
```

ListManager.py Page 1/2

```
Help on module ListManager:

NAME
    ListManager - ListManager

FILE
    marble/orpheus/main/ListManager.py

CLASSES
    marble.base.Singleton(__builtin__.object)
        ListManager

    class ListManager(marble.base.Singleton)
     |  This class manages ORPHEUS list output. This class is singleton.
     |
     |  Method resolution order:
     |      ListManager
     |      marble.base.Singleton
     |      __builtin__.object
     |
     |  Methods defined here:
     |
     |  elapsedTime(self, err=False)
     |      Output elapsed time.
     |
     |      Parameters:
     |        err -- True or False. If True, output device for debug is used.
     |
     |  init(self)
     |      Constructor.
     |
     |  initialize(self, inpMgr)
     |      Initialize list output device.
     |
     |      Parameters:
     |        inpMgr -- InputManager object.
     |
     |  outCalcConditionFile(self, inpMgr)
     |      Output calc condition file.
     |
     |      Parameters:
     |        inpMgr -- InputManager object.
     |
     |  outCodeInfo(self)
     |      Output code information.
     |
     |  outCorePropertyInfo(self, coreProperty)
     |      Output core information.
     |
     |      Parameters:
     |        coreProperty -- CoreProperty object.
     |
     |  outDebugMessage(self, message, time_reset=False)
     |      Output message to output device for debug.
     |
     |      Parameters:
     |        message -- debug message.
     |        time_reset -- Trur or False. If true, elapsed_time is reset.
     |
     |  outEndMark(self)
     |      Output calculation end mark.
     |
     |  outMeshGeometryInfo(self, domainType, meshGeometry)
     |      Output flux or material mesh geometry information.
     |
     |      Parameters:
     |        domainType -- FLUX_REGION or MATERIAL_REGION
     |        meshGeometry -- MeshGeometry object.
     |
     |  outMessage(self, level, message, time_reset=False)
     |      Output various message.
     |
     |      Parameters:
     |        level -- message level. (PLAIN/NOTE/WARNING/ERROR)
     |        message -- output message
     |        time_reset -- True or False. If true, elapsed time is reset.
     |
     |  outNumberDensityInfo(self, coreMaterialInfo)
     |      Output number densities for each material mesh.
     |
     |      Parameters:
     |        coreMaterialInfo -- CoreMaterialInfo object.
     |
     |  outOrpheusDatabaseInfo(self, dbhandler)
     |      Output orpheus database information.
     |
     |      Parameters:
     |        dbhandler -- OrpheusDatabaseHandler object.
     |
     |  outPatternInfo(self, pattern)
     |      Output pattern information.
     |
     |      Parameters:
     |        pattern -- Pattern object.
```

```
    outZoneAveragedNumberDensityInfo(self, coreMaterialInfo)
        Output zone-average number densities for all zones.

        Parameters:
          coreMaterialInfo -- CoreMaterialInfo object.

    outZoneInfo(self, zoneSet)
        Output zone information.

        Parameters:
          zoneSet -- ZoneSet object.

    ----------------------------------------------------------------------
    Static methods inherited from marble.base.Singleton:

    __new__(cls, *args, **kwds)

    ----------------------------------------------------------------------
    Data descriptors inherited from marble.base.Singleton:

    __dict__
        dictionary for instance variables (if defined)

    __weakref__
        list of weak references to the object (if defined)

DATA
    CODE_NAME = 'ORPHEUS'
    CODE_VERSION = '1.0'
    ERROR = 3
    FLUX_REGION = 0
    MATERIAL_REGION = 1
    NOTE = 1
    PLAIN = 0
    WARNING = 2
```

CoreManager.py Page 1/2

```
Help on module CoreManager:

NAME
    CoreManager - CoreManager

FILE
    marble/orpheus/main/CoreManager.py

CLASSES
    CoreManager

    class CoreManager
     |  This class manages building and initializing model objects.
     |
     |  Methods defined here:
     |
     |  __init__(self)
     |      Constructor.
     |
     |  fluxInfoID(self)
     |      Get ID number of CoreFluxInfo objects.
     |
     |      Return:
     |        ID of CoreFluxInfo object. (int)
     |
     |  getCoreInfo(self)
     |      Get CoreInfo object.
     |
     |      Return:
     |        CoreInfo object.
     |
     |  getCoreProperty(self)
     |      Get CoreProperty object.
     |
     |      Return:
     |        CoreProperty object.
     |
     |  getInitialComposition(self)
     |      Get InitialComposition object.
     |
     |      Return:
     |        InitialComposition object.
     |
     |  getMaterialSet(self)
     |      Get MaterialSet object.
     |
     |      Return:
     |        MaterialSet object.
     |
     |  getOverlapRegionSetPool(self)
     |      Get list of OverlapRegionSet objects.
     |
     |      Return:
     |        list of OverlapRegionSet objects.
     |
     |  getPattern(self)
     |      Get Pattern object.
     |
     |      Return:
     |        Pattern object.
     |
     |  getSegmentSetGeometryPool(self)
     |      Get SegmentSetGeometryPool object.
     |
     |      Return:
     |        SegmentSetGeometryPool object.
     |
     |  getZoneSet(self)
     |      Get ZoneSet object.
     |
     |      Return:
     |        ZoneSet object.
     |
     |  initializeCollapseMode(self, inpMgr, dbhandler)
     |      Build and initialize model objects for energy-group collapse calculation.
     |
     |      Parameters:
     |        inpMgr -- InputManager object.
     |        dbhandler -- OrpheusDatabaseHandler object.
     |
     |  initializeConstantMicroMode(self, inpMgr, dbhandler)
     |      Build and initialize model objects for using pre-calculated cross section calculation.
     |
     |      Parameters:
     |        inpMgr -- InputManager object.
     |        dbhandler -- OrpheusDatabaseHandler object.
     |
     |  initializeControlRodMode(self, inpMgr, dbhandler)
     |      Build and initialize model objects for control rod calculation.
     |
     |      Parameters:
     |        inpMgr -- InputManager object.
     |        dbhandler -- OrpheusDatabaseHandler object.
     |
```

```
      |  initializeDepletionMode(self, inpMgr, dbhandler)
      |      Build and initialize model objects for standard depletion calculation.
      |
      |      Parameters:
      |        inpMgr -- InputManager object.
      |        dbhandler -- OrpheusDatabaseHandler object.
      |
      |  materialInfoID(self)
      |      Get ID number of CoreMaterialInfo objects.
      |
      |      Return:
      |        ID of CoreMaterialInfo object. (int)

DATA
    CODE_NAME = 'ORPHEUS'
    CODE_VERSION = '1.0'
    ERROR = 3
    FLUX_REGION = 0
    MATERIAL_REGION = 1
    NOTE = 1
    PLAIN = 0
    WARNING = 2
```

ObjectBuilder.py Page 1/3

```
Help on module ObjectBuilder:

NAME
    ObjectBuilder - ObjectBuilder

FILE
    marble/orpheus/main/ObjectBuilder.py

CLASSES
    ObjectBuilder

    class ObjectBuilder
        This class handles functions for building MARBLE objects based on user input information.

        Methods defined here:

        __init__(self, dbhandler)
            Constructor.

            Parameters:
              dbhandler -- OrpheusDatabaseHandler object.

        getAveragedFissionSpectrum(self, zone, preZoneSet, preCrossSectionInfo)
            Get zone-averaged fission spectrum.
            Convert data between different zone systems.

            Parameters:
              zone -- Zone object.
              preZoneSet -- ZoneSet object of origin system.
              preCrossSectionInfo -- CrossSectionInfo object of origin system.

            Return:
              fission spectrum. (numpy.array)

        getAveragedMicroXSSet(self, zone, meshGeometry, preZoneSet, preCrossSectionInfo, preCoreFluxInfo)
            Get zone-averaged MicroscopicEffectiveCrossSectionSet object.
            Convert data between different zone systems.

            Parameters:
              zone -- Zone object.
              meshGeometry -- MeshGeometry object.
              preZoneSet -- ZoneSet object of origin system.
              preCrossSectionInfo -- CrossSectionInfo object of origin system.
              preCoreFluxInfo -- CoreFluxInfo object of origin system.

            Return:
              MicroscopicEffectiveCrossSectionSet object.

        getCoreInfo(self, ng, fluxMeshGeometry, materialMeshGeometry, zoneSet)
            Create and get CoreInfo object.

            Parameters:
              ng -- the number of energy groups
              fluxMeshGeometry -- MeshGeometry object for flux domain.
              materialMeshGeometry -- MeshGeometry object for material domain.
              zoneSet -- ZoneSet object.

            Return:
              tuple of CoreInfo object, ID of CoreFluxInfo object and ID of CoreMaterialInfo object.

        getCoreProperty(self, corePropertyData)
            Create and get CoreProperty object.

            Parameters:
              corePropertyData -- data for initialization of CoreProperty object. (dict)

            Return:
              CoreProperty object.

        getInitialComposition(self, rayTraceOrder, pattern, materialSet, geometryPool, meshGeometry)
            Create and get InitialComposition object.

            Parameters:
              rayTraceOrder -- integral order of ray trace.
              pattern -- Pattern object.
              materialSet -- MaterialSet object.
              geometryPool -- SegmentSetGeometryPool object.
              meshGeometry -- MeshGeometry object.

            Return:
              InitialComposition object.

        getInitialCompositionFromRayTraceFile(self, rayTraceFile, pattern, materialSet, geometryPool, meshGeomet
ry)
            Create and get InitialComposition object using pre-calculated ray trace result.

            Parameters:
              rayTraceFile -- pre-calculated ray trace result file.
              pattern -- Pattern object.
              materialSete -- MaterialSet object.
              geometryPool -- SegmentSetGeometryPool object.
              meshGeometry -- MeshGeometry object.

            Return:
```

```
      InitialComposition object.

  getMaterialSet(self, materialData)
      Create and get MaterialSet object.

      Parameters:
        materialData -- data for initialization of MaterialSet object. (dict)

      Return:
        MaterialSet object.

  getMeshGeometry(self, coreProperty, coordinates, geometryPool, pattern, axialMeshPitches)
      Create and get MeshGeometry object.

      Parameters:
        coreProperty -- CoreProperty object.
        geometryPool -- SegmentSetGeometryPool object.
        pattern -- Pattern object.
        axialMeshPitches -- list of axial mesh pitches.

      Return:
        MeshGeometry object.

  getPattern(self, cycleName, cycleNumber, patternData, coreProperty)
      Create and get Pattern object.

      Parameters:
        cycleName -- cycle name.
        cycleNumber -- cumulative cycle number.
        patternData -- assembly loading data. (dict)
        coreProperty -- CoreProperty object.

      Return:
        Pattern object.

  getSegmentSetGeometryPool(self, geometryData, geometryTypeSet)
      Create and get SegmentSetGeometryPool object.

      Parameters:
        geometryData -- data for initialization of GeometryBuildContext object. (dict)
        geometryTypeSet -- list of geometry type names.

      Return:
        SegmentSetGeometryPool object.

  getSystemCoordinates(self, coordinateType, coreProperty)
      Create and get SystemCoordinates object.

      Parameters:
        coordinateType -- coordinate type (HEXZ/TRIZ/P_XYZ)
        coreProperty -- CoreProperty object.

      Return:
        SystemCoordinates object.

  getZoneSet(self, zoneBuildData, coreProperty, meshGeometry)
      Create and get ZoneSet object.

      Parameters:
        zoneBuildData -- data of zones. (dict)
        coreProperty -- CoreProperty object.
        meshGeometry -- MeshGeometry object.

      Return:
        ZoneSet object.

  setCellModel(self, crossSectionInfo, zoneSet, pattern, materialSet, geometryPool, meshGeometry)
      Set cell model objects for CrossSectionInfo object.

      Parameters:
        crossSectionInfo -- CrossSectionInfo object.
        zoneSet -- ZoneSet object.
        pattern -- Pattern object.
        materialSet -- MaterialSet object.
        geometryPool -- SegmentSetGeometry Pool object.
        meshGeometry -- MeshGeometry object.

  setInitialComposition(self, coreMaterialInfo, coreProperty, pattern, composition)
      Set initial composition for each meshes of CoreMaterialInfo object.

      Parameters:
        coreMaterialInfo -- CoreMaterialInfo object.
        coreProperty -- CoreProperty object.
        pattern -- Pattern object.
        composition -- InitialComposition object.

  setMicroXSData(self, zoneSet, zoneXSSet, zoneFissionSpectrum, crossSectionInfo)
      Set corss section data for CrossSectionInfo object.

      Parameters:
        zoneSet -- ZoneSet object.
        zoneXSSet -- MicroscopicEffectiveCrossSectionSet objects for each zones. (dict)
        zoneFissionSpectrum -- fission spectrum for each zones. (dict)
        crossSectionInfo -- CrossSectionInfo object.
```

ObjectBuilder.py Page 3/3

```
DATA
    CONTROL_ROD = 2
    HEXZ = 0
    P_XYZ = 3
    TRIZ = 1
```

ObjectReloader.py Page 1/1

```
Help on module ObjectReloader:

NAME
    ObjectReloader - ObjectReloader

FILE
    marble/orpheus/main/ObjectReloader.py

CLASSES
    ObjectReloader

    class ObjectReloader
     |  This class handles deserialization of objects from database files.
     |
     |  Methods defined here:
     |
     |  __init__(self, dbhandler)
     |      Constructor.
     |
     |      Parameters:
     |        dbhandler -- OrpheusDatabaseHandler object.
     |
     |  getCoreInfo(self, cycleNumber, step=None)
     |      Get CoreInfo object of target cycle and step.
     |
     |      Parameters:
     |        cycleNumber -- cycle number.
     |        step -- step number (default is None, and means last step.)
     |
     |      Return:
     |        tuple of CoreInfo object, ID of CoreFluxInfo object and ID of CoreMaterialInfo object.
     |
     |  getCoreProperty(self, cycleNumber)
     |      Get CoreProperty object of target cycle.
     |
     |      Parameters:
     |        cycleNumber -- cycle number.
     |
     |      Return:
     |        CoreProperty object.
     |
     |  getMeshGeometry(self, domain, cycleNumber, step=None)
     |      Get MeshGeometry object of target cycle and step.
     |
     |      Parameters:
     |        domain -- flux/material region. [FLUX_REGION/MATERIAL_REGION]
     |        cycleNumber -- cycle number.
     |        step -- step number (default is None, and means last step.)
     |
     |      Return:
     |        MeshGeometry object.
     |
     |  getPattern(self, cycleNumber)
     |      Get Pattern object of target cycle.
     |
     |      Parameters:
     |        cycleNumber -- cycle number.
     |
     |      Return:
     |        Pattern object.
     |
     |  getZoneSet(self, cycleNumber)
     |      Get ZoneSet object of target cycle.
     |
     |      Parameters:
     |        cycleNumber -- cycle number.
     |
     |      Return:
     |        ZoneSet object.
     |
     |  initializeControlRods(self, cycleNumber, coreProperty, zoneSet, coreMaterialInfo, crossSectionInfo)
     |      Set control rods to fully inserted state.
     |
     |      Parameters:
     |        cycleNumber -- cycle number.
     |        coreProperty -- CoreProperty object.
     |        zoneSet -- ZoneSet object.
     |        coreMaterialInfo -- CoreMaterialInfo object.
     |        crossSectionInfo -- CrossSectionInfo object.

DATA
    CONTROL_ROD = 2
    FLUX_REGION = 0
    MATERIAL_REGION = 1
```

Scenario.py Page 1/1

```
Help on module Scenario:

NAME
    Scenario - Scenario

FILE
    marble/orpheus/main/Scenario.py

CLASSES
    Scenario

    class Scenario
     |  This class encapsulates calculation scenario.
     |  This class is abstract, and concrete calculation scenario is implemented by sub-classes.
     |
     |  Methods defined here:
     |
     |  perform(self)
     |      Start calculation scenario.
     |      This method is abstract.
     |
     |  ----------------------------------------------------------------------
     |  Class methods defined here:
     |
     |  create(cls, type, inpMgr, coreMgr, dbhandler) from __builtin__.classobj
     |      Create Scenario object. (factory method)
     |
     |      Parameters:
     |        type -- scenario type (string)
     |        inpMgr -- InputManager object.
     |        coreMgr -- CoreManager object.
     |        dbhandler -- OrpheusDatabaseHandler object.
     |
     |      Return:
     |        Scenario object.
     |
     |      Exceptions:
     |        OrpheusStandardError for invalid scenario type.

DATA
    CODE_NAME = 'ORPHEUS'
    CODE_VERSION = '1.0'
    CONTROL_ROD = 2
    ERROR = 3
    NOTE = 1
    PLAIN = 0
    WARNING = 2
```

ScenarioStandard.py Page 1/1

```
Help on module ScenarioStandard:

NAME
    ScenarioStandard - ScenarioStandard

FILE
    marble/orpheus/main/ScenarioStandard.py

CLASSES
    marble.orpheus.main.Scenario.Scenario
        ScenarioStandard

    class ScenarioStandard(marble.orpheus.main.Scenario.Scenario)
     |  This class encapsulates standard calculation flow.
     |  (cell/core/burnup calculation including micro cross section updates.)
     |
     |  Methods defined here:
     |
     |  __init__(self, inpMgr, coreMgr, dbhandler)
     |      Constructor.
     |
     |      Parameters:
     |        inpMgr -- InputManager object.
     |        coreMgr -- CoreManager object.
     |        dbhandler -- OrpheusDatabaseHandler object.
     |
     |  perform(self)
     |      Start calculation scenario.
     |
     |  ----------------------------------------------------------------------
     |  Class methods inherited from marble.orpheus.main.Scenario.Scenario:
     |
     |  create(cls, type, inpMgr, coreMgr, dbhandler) from __builtin__.classobj
     |      Create Scenario object. (factory method)
     |
     |      Parameters:
     |        type -- scenario type (string)
     |        inpMgr -- InputManager object.
     |        coreMgr -- CoreManager object.
     |        dbhandler -- OrpheusDatabaseHandler object.
     |
     |      Return:
     |        Scenario object.
     |
     |      Exceptions:
     |        OrpheusStandardError for invalid scenario type.

DATA
    CODE_NAME = 'ORPHEUS'
    CODE_VERSION = '1.0'
    ERROR = 3
    NOTE = 1
    PLAIN = 0
    WARNING = 2
```

ScenarioConstantMicro.py Page 1/1

```
Help on module ScenarioConstantMicro:

NAME
    ScenarioConstantMicro - ScenarioConstantMicro

FILE
    marble/orpheus/main/ScenarioConstantMicro.py

CLASSES
    marble.orpheus.main.Scenario.Scenario
        ScenarioConstantMicro

    class ScenarioConstantMicro(marble.orpheus.main.Scenario.Scenario)
     |  This class encapsulates using user defined micro cross section scenario.
     |
     |  Methods defined here:
     |
     |  __init__(self, inpMgr, coreMgr, dbhandler)
     |      Constructor.
     |
     |      Parameters:
     |        inpMgr -- InputManager object.
     |        coreMgr -- CoreManager object.
     |        dbhandler -- OrpheusDatabaseHandler object.
     |
     |  perform(self)
     |      Start calculation scenario.
     |
     |  ----------------------------------------------------------------------
     |  Class methods inherited from marble.orpheus.main.Scenario.Scenario:
     |
     |  create(cls, type, inpMgr, coreMgr, dbhandler) from __builtin__.classobj
     |      Create Scenario object. (factory method)
     |
     |      Parameters:
     |        type -- scenario type (string)
     |        inpMgr -- InputManager object.
     |        coreMgr -- CoreManager object.
     |        dbhandler -- OrpheusDatabaseHandler object.
     |
     |      Return:
     |        Scenario object.
     |
     |      Exceptions:
     |        OrpheusStandardError for invalid scenario type.

DATA
    CODE_NAME = 'ORPHEUS'
    CODE_VERSION = '1.0'
    ERROR = 3
    NOTE = 1
    PLAIN = 0
    WARNING = 2
```

ScenarioCollapseGroup.py


```
Help on module ScenarioCollapseGroup:

NAME
    ScenarioCollapseGroup - ScenarioCollapseGroup

FILE
    marble/orpheus/main/ScenarioCollapseGroup.py

CLASSES
    marble.orpheus.main.Scenario.Scenario
        ScenarioCollapseGroup

    class ScenarioCollapseGroup(marble.orpheus.main.Scenario.Scenario)
     |  This class encapsulates energy-group collapesd calculation flow.
     |
     |  Methods defined here:
     |
     |  __init__(self, inpMgr, coreMgr, dbhandler)
     |      Constructor.
     |
     |      Parameters:
     |        inpMgr -- InputManager object.
     |        coreMgr -- CoreManager object.
     |        dbhandler -- OrpheusDatabaseHandler object.
     |
     |  perform(self)
     |      Start calculation scenario.
     |
     |  ----------------------------------------------------------------------
     |  Class methods inherited from marble.orpheus.main.Scenario.Scenario:
     |
     |  create(cls, type, inpMgr, coreMgr, dbhandler) from __builtin__.classobj
     |      Create Scenario object. (factory method)
     |
     |      Parameters:
     |        type -- scenario type (string)
     |        inpMgr -- InputManager object.
     |        coreMgr -- CoreManager object.
     |        dbhandler -- OrpheusDatabaseHandler object.
     |
     |      Return:
     |        Scenario object.
     |
     |      Exceptions:
     |        OrpheusStandardError for invalid scenario type.

DATA
    CODE_NAME = 'ORPHEUS'
    CODE_VERSION = '1.0'
    ERROR = 3
    NOTE = 1
    PLAIN = 0
    WARNING = 2
```

ScenarioCollapseGroupWithMicroUpdate.py


```
Help on module ScenarioCollapseGroupWithMicroUpdate:

NAME
    ScenarioCollapseGroupWithMicroUpdate - ScenarioCollapseGroupWithMicroUpdate

FILE
    marble/orpheus/main/ScenarioCollapseGroupWithMicroUpdate.py

CLASSES
    marble.orpheus.main.Scenario.Scenario
        ScenarioCollapseGroupWithMicroUpdate

    class ScenarioCollapseGroupWithMicroUpdate(marble.orpheus.main.Scenario.Scenario)
     |  This class encapsulates energy-group collapesd calculation flow.
     |  (including micro cross section updates.)
     |
     |  Methods defined here:
     |
     |  __init__(self, inpMgr, coreMgr, dbhandler)
     |      Constructor.
     |
     |      Parameters:
     |        inpMgr -- InputManager object.
     |        coreMgr -- CoreManager object.
     |        dbhandler -- OrpheusDatabaseHandler object.
     |
     |  perform(self)
     |      Start calculation scenario.
     |
     |  ----------------------------------------------------------------------
     |  Class methods inherited from marble.orpheus.main.Scenario.Scenario:
     |
     |  create(cls, type, inpMgr, coreMgr, dbhandler) from __builtin__.classobj
     |      Create Scenario object. (factory method)
     |
     |      Parameters:
     |        type -- scenario type (string)
     |        inpMgr -- InputManager object.
     |        coreMgr -- CoreManager object.
     |        dbhandler -- OrpheusDatabaseHandler object.
     |
     |      Return:
     |        Scenario object.
     |
     |      Exceptions:
     |        OrpheusStandardError for invalid scenario type.

DATA
    CODE_NAME = 'ORPHEUS'
    CODE_VERSION = '1.0'
    ERROR = 3
    NOTE = 1
    PLAIN = 0
    WARNING = 2
```

ScenarioControlRod.py Page 1/1

```
Help on module ScenarioControlRod:

NAME
    ScenarioControlRod - ScenarioControlRod

FILE
    marble/orpheus/main/ScenarioControlRod.py

CLASSES
    marble.orpheus.main.Scenario.Scenario
        ScenarioControlRod

    class ScenarioControlRod(marble.orpheus.main.Scenario.Scenario)
     |  This class encapsulates branch calculation scenario.
     |  (Control rod worth calculation etc.)
     |
     |  Methods defined here:
     |
     |  __init__(self, inpMgr, coreMgr, dbhandler)
     |      Constructor.
     |
     |      Parameters:
     |        inpMgr -- InputManager object.
     |        coreMgr -- CoreManager object.
     |        dbhandler -- OrpheusDatabaseHandler object.
     |
     |  perform(self)
     |      Start calculation scenario.
     |
     |  ----------------------------------------------------------------------
     |  Class methods inherited from marble.orpheus.main.Scenario.Scenario:
     |
     |  create(cls, type, inpMgr, coreMgr, dbhandler) from __builtin__.classobj
     |      Create Scenario object. (factory method)
     |
     |      Parameters:
     |        type -- scenario type (string)
     |        inpMgr -- InputManager object.
     |        coreMgr -- CoreManager object.
     |        dbhandler -- OrpheusDatabaseHandler object.
     |
     |      Return:
     |        Scenario object.
     |
     |      Exceptions:
     |        OrpheusStandardError for invalid scenario type.

DATA
    CODE_NAME = 'ORPHEUS'
    CODE_VERSION = '1.0'
    ERROR = 3
    NOTE = 1
    PLAIN = 0
    WARNING = 2
```

InputInterface.py Page 1/1

```
Help on module InputInterface:

NAME
    InputInterface - InputInterface

FILE
    marble/orpheus/input/InputInterface.py

CLASSES
    InputInterface

    class InputInterface
     |  This class abstracts system boundary for a text input.
     |  This class emulates a container, and input data is accessed with keywords.
     |
     |  Methods defined here:
     |
     |  __init__(self)
     |      Constructor.
     |
     |  data(self)
     |      Get input data.
     |
     |      Return:
     |        input data dictionary.
     |
     |  read(self, input, output=<open file '<stderr>', mode 'w' at 0x2ab85d47e210>)
     |      Read a text input. --- parse and validate a text, and construct input data.
     |
     |      Parameters:
     |        input -- input device.
     |        output -- output device. (default is sys.stderr)
     |
     |      Return:
     |        True or False.
     |        If format of input data is invalid, return False.
     |
     |      Exceptions:
     |        OrpheusStandardError for illegal input data.
     |
     |  ----------------------------------------------------------------------
     |  Class methods defined here:
     |
     |  create(self, type) from __builtin__.classobj
     |      Create an InputInterface object.(factory method)
     |
     |      Parameters:
     |        type -- target file type.
     |
     |      Return:
     |        InputInterface object.
     |
     |      Exceptions:
     |        OrpheusStandardError for invalid type name.
     |
     |  ----------------------------------------------------------------------
     |  Data and other attributes defined here:
     |
     |  CALC_CONDITION_FILE = 0
     |
     |  CORE_PROPERTY_FILE = 1
     |
     |  CROSS_SECTION_FILE = 5
     |
     |  GEOMETRY_FILE = 2
     |
     |  MACRO_CROSS_SECTION_FILE = 6
     |
     |  MATERIAL_FILE = 4
     |
     |  PATTERN_FILE = 3
```

CalcConditionFile.py


```
Help on module CalcConditionFile:

NAME
    CalcConditionFile - CalcConditionFile

FILE
    marble/orpheus/input/CalcConditionFile.py

CLASSES
    marble.orpheus.input.InputInterface.InputInterface
        CalcConditionFile

    class CalcConditionFile(marble.orpheus.input.InputInterface.InputInterface)
     |  This class handles and encapsulates a calculation condition file.
     |  Some methods used to parse a file depends on the format and structure of it.
     |
     |  Information on the calculation condition file is converted into a 'dictionary'.
     |
     |  Methods defined here:
     |
     |  secondInitialize(self, **objects)
     |      Initialize using pattern/geometry/etc. models.
     |
     |      Parameters:
     |        objects -- Marble objects.
     |
     |  ----------------------------------------------------------------------
     |  Methods inherited from marble.orpheus.input.InputInterface.InputInterface:
     |
     |  __init__(self)
     |      Constructor.
     |
     |  data(self)
     |      Get input data.
     |
     |      Return:
     |        input data dictionary.
     |
     |  read(self, input, output=<open file '<stderr>', mode 'w' at 0x2b6d0cebc210>)
     |      Read a text input. --- parse and validate a text, and construct input data.
     |
     |      Parameters:
     |        input -- input device.
     |        output -- output device. (default is sys.stderr)
     |
     |      Return:
     |        True or False.
     |        If format of input data is invalid, return False.
     |
     |      Exceptions:
     |        OrpheusStandardError for illegal input data.
     |
     |  ----------------------------------------------------------------------
     |  Class methods inherited from marble.orpheus.input.InputInterface.InputInterface:
     |
     |  create(self, type) from __builtin__.classobj
     |      Create an InputInterface object.(factory method)
     |
     |      Parameters:
     |        type -- target file type.
     |
     |      Return:
     |        InputInterface object.
     |
     |      Exceptions:
     |        OrpheusStandardError for invalid type name.
     |
     |  ----------------------------------------------------------------------
     |  Data and other attributes inherited from marble.orpheus.input.InputInterface.InputInterface:
     |
     |  CALC_CONDITION_FILE = 0
     |
     |  CORE_PROPERTY_FILE = 1
     |
     |  CROSS_SECTION_FILE = 5
     |
     |  GEOMETRY_FILE = 2
     |
     |  MACRO_CROSS_SECTION_FILE = 6
     |
     |  MATERIAL_FILE = 4
     |
     |  PATTERN_FILE = 3

DATA
    CODE_NAME = 'ORPHEUS'
    CODE_VERSION = '1.0'
    CONTROL_ROD = 2
    ERROR = 3
    HEXZ = 0
    NOTE = 1
    PLAIN = 0
    P_XYZ = 3
    TRIZ = 1
```

CalcConditionFile.py Page 2/2

```
WARNING = 2
```

CorePropertyFile.py


```
Help on module CorePropertyFile:

NAME
    CorePropertyFile - CorePropertyFile

FILE
    marble/orpheus/input/CorePropertyFile.py

CLASSES
    marble.orpheus.input.InputInterface.InputInterface
        CorePropertyFile

    class CorePropertyFile(marble.orpheus.input.InputInterface.InputInterface)
     |  This class handles and encapsulates a core property input file.
     |  Some methods used to parse a file depends on the format and structure of it.
     |
     |  Information on the core property file is converted into a 'dictionary'.
     |
     |  Methods inherited from marble.orpheus.input.InputInterface.InputInterface:
     |
     |  __init__(self)
     |      Constructor.
     |
     |  data(self)
     |      Get input data.
     |
     |      Return:
     |        input data dictionary.
     |
     |  read(self, input, output=<open file '<stderr>', mode 'w' at 0x2aac48884210>)
     |      Read a text input. --- parse and validate a text, and construct input data.
     |
     |      Parameters:
     |        input -- input device.
     |        output -- output device. (default is sys.stderr)
     |
     |      Return:
     |        True or False.
     |        If format of input data is invalid, return False.
     |
     |      Exceptions:
     |        OrpheusStandardError for illegal input data.
     |
     |  ----------------------------------------------------------------------
     |  Class methods inherited from marble.orpheus.input.InputInterface.InputInterface:
     |
     |  create(self, type) from __builtin__.classobj
     |      Create an InputInterface object.(factory method)
     |
     |      Parameters:
     |        type -- target file type.
     |
     |      Return:
     |        InputInterface object.
     |
     |      Exceptions:
     |        OrpheusStandardError for invalid type name.
     |
     |  ----------------------------------------------------------------------
     |  Data and other attributes inherited from marble.orpheus.input.InputInterface.InputInterface:
     |
     |  CALC_CONDITION_FILE = 0
     |
     |  CORE_PROPERTY_FILE = 1
     |
     |  CROSS_SECTION_FILE = 5
     |
     |  GEOMETRY_FILE = 2
     |
     |  MACRO_CROSS_SECTION_FILE = 6
     |
     |  MATERIAL_FILE = 4
     |
     |  PATTERN_FILE = 3
```

GeometryFile.py


```
Help on module GeometryFile:

NAME
    GeometryFile - GeometryFile

FILE
    marble/orpheus/input/GeometryFile.py

CLASSES
    marble.orpheus.input.InputInterface.InputInterface
        GeometryFile

    class GeometryFile(marble.orpheus.input.InputInterface.InputInterface)
     |  This class handles and encapsulates a geometry input file.
     |  Some methods used to parse a file depends on the format and structure of it.
     |
     |  Information on the geometry file is converted into a 'dictionary',
     |  and the data structure of that dictionary is equal to that of 'GeometryBuildContext' class,
     |  excluding 'MeshSection'.
     |
     |  Methods inherited from marble.orpheus.input.InputInterface.InputInterface:
     |
     |  __init__(self)
     |      Constructor.
     |
     |  data(self)
     |      Get input data.
     |
     |      Return:
     |        input data dictionary.
     |
     |  read(self, input, output=<open file '<stderr>', mode 'w' at 0x2b41b3d3c210>)
     |      Read a text input. --- parse and validate a text, and construct input data.
     |
     |      Parameters:
     |        input -- input device.
     |        output -- output device. (default is sys.stderr)
     |
     |      Return:
     |        True or False.
     |        If format of input data is invalid, return False.
     |
     |      Exceptions:
     |        OrpheusStandardError for illegal input data.
     |
     |  ----------------------------------------------------------------------
     |  Class methods inherited from marble.orpheus.input.InputInterface.InputInterface:
     |
     |  create(self, type) from __builtin__.classobj
     |      Create an InputInterface object.(factory method)
     |
     |      Parameters:
     |        type -- target file type.
     |
     |      Return:
     |        InputInterface object.
     |
     |      Exceptions:
     |        OrpheusStandardError for invalid type name.
     |
     |  ----------------------------------------------------------------------
     |  Data and other attributes inherited from marble.orpheus.input.InputInterface.InputInterface:
     |
     |  CALC_CONDITION_FILE = 0
     |
     |  CORE_PROPERTY_FILE = 1
     |
     |  CROSS_SECTION_FILE = 5
     |
     |  GEOMETRY_FILE = 2
     |
     |  MACRO_CROSS_SECTION_FILE = 6
     |
     |  MATERIAL_FILE = 4
     |
     |  PATTERN_FILE = 3

DATA
    ShapeID = {'circle': 'CIRCLE', 'hex': 'HEX', 'square': 'SQUARE', 'tria...
```

MaterialFile.py Page 1/1

```
Help on module MaterialFile:

NAME
    MaterialFile - MaterialFile

FILE
    marble/orpheus/input/MaterialFile.py

CLASSES
    marble.orpheus.input.InputInterface.InputInterface
        MaterialFile

    class MaterialFile(marble.orpheus.input.InputInterface.InputInterface)
     |  This class handles and encapsulates material file.
     |  Some methods used to parse input file depends on format and the structure of input file.
     |
     |  Information on the material file is converted into a 'dictionary',
     |  and the structure of dictionary is as follows.
     |      material_name:
     |          type: material type (string)
     |          component:
     |              nuclide_name: number density
     |
     |  Ex:
     |      aDictionary = {"fuel1": {"type":Material.FUEL, "component":{"U-235":4.09565e-03, "U-238":1.35459e-02
, ...}},
     |                     "clad1": {"type":Material.SUS , "component":{"Cr-nat":1.72871e-02, "Fe-nat":6.29425e-
02, ...}},
     |                     ...}
     |
     |  Methods inherited from marble.orpheus.input.InputInterface.InputInterface:
     |
     |  __init__(self)
     |      Constructor.
     |
     |  data(self)
     |      Get input data.
     |
     |      Return:
     |        input data dictionary.
     |
     |  read(self, input, output=<open file '<stderr>', mode 'w' at 0x2b349c9ef210>)
     |      Read a text input. --- parse and validate a text, and construct input data.
     |
     |      Parameters:
     |        input -- input device.
     |        output -- output device. (default is sys.stderr)
     |
     |      Return:
     |        True or False.
     |        If format of input data is invalid, return False.
     |
     |      Exceptions:
     |        OrpheusStandardError for illegal input data.
     |
     |  ----------------------------------------------------------------------
     |  Class methods inherited from marble.orpheus.input.InputInterface.InputInterface:
     |
     |  create(self, type) from __builtin__.classobj
     |      Create an InputInterface object.(factory method)
     |
     |      Parameters:
     |        type -- target file type.
     |
     |      Return:
     |        InputInterface object.
     |
     |      Exceptions:
     |        OrpheusStandardError for invalid type name.
     |
     |  ----------------------------------------------------------------------
     |  Data and other attributes inherited from marble.orpheus.input.InputInterface.InputInterface:
     |
     |  CALC_CONDITION_FILE = 0
     |
     |  CORE_PROPERTY_FILE = 1
     |
     |  CROSS_SECTION_FILE = 5
     |
     |  GEOMETRY_FILE = 2
     |
     |  MACRO_CROSS_SECTION_FILE = 6
     |
     |  MATERIAL_FILE = 4
     |
     |  PATTERN_FILE = 3
```

PatternFile.py Page 1/1

```
Help on module PatternFile:

NAME
    PatternFile - PatternFile

FILE
    marble/orpheus/input/PatternFile.py

CLASSES
    marble.orpheus.input.InputInterface.InputInterface
        PatternFile

    class PatternFile(marble.orpheus.input.InputInterface.InputInterface)
     |  This class handles and encapsulates pattern file, which defines fuel loading information.
     |  Some methods used to parse input file depends on the format and structure of a input file.
     |
     |  Information on the pattern file is converted into a 'dictionary'.
     |
     |  Methods inherited from marble.orpheus.input.InputInterface.InputInterface:
     |
     |  __init__(self)
     |      Constructor.
     |
     |  data(self)
     |      Get input data.
     |
     |      Return:
     |        input data dictionary.
     |
     |  read(self, input, output=<open file '<stderr>', mode 'w' at 0x2b938deee210>)
     |      Read a text input. --- parse and validate a text, and construct input data.
     |
     |      Parameters:
     |        input -- input device.
     |        output -- output device. (default is sys.stderr)
     |
     |      Return:
     |        True or False.
     |        If format of input data is invalid, return False.
     |
     |      Exceptions:
     |        OrpheusStandardError for illegal input data.
     |
     |  ----------------------------------------------------------------------
     |  Class methods inherited from marble.orpheus.input.InputInterface.InputInterface:
     |
     |  create(self, type) from __builtin__.classobj
     |      Create an InputInterface object.(factory method)
     |
     |      Parameters:
     |        type -- target file type.
     |
     |      Return:
     |        InputInterface object.
     |
     |      Exceptions:
     |        OrpheusStandardError for invalid type name.
     |
     |  ----------------------------------------------------------------------
     |  Data and other attributes inherited from marble.orpheus.input.InputInterface.InputInterface:
     |
     |  CALC_CONDITION_FILE = 0
     |
     |  CORE_PROPERTY_FILE = 1
     |
     |  CROSS_SECTION_FILE = 5
     |
     |  GEOMETRY_FILE = 2
     |
     |  MACRO_CROSS_SECTION_FILE = 6
     |
     |  MATERIAL_FILE = 4
     |
     |  PATTERN_FILE = 3

DATA
    CODE_NAME = 'ORPHEUS'
    CODE_VERSION = '1.0'
    ERROR = 3
    NOTE = 1
    PLAIN = 0
    WARNING = 2
```

CrossSectionFile.py


```
Help on module CrossSectionFile:

NAME
    CrossSectionFile - CrossSectionFile

FILE
    marble/orpheus/input/CrossSectionFile.py

CLASSES
    marble.orpheus.input.InputInterface.InputInterface
        CrossSectionFile
        MacroCrossSectionFile

    class CrossSectionFile(marble.orpheus.input.InputInterface.InputInterface)
     |  Methods inherited from marble.orpheus.input.InputInterface.InputInterface:
     |
     |  __init__(self)
     |      Constructor.
     |
     |  data(self)
     |      Get input data.
     |
     |      Return:
     |        input data dictionary.
     |
     |  read(self, input, output=<open file '<stderr>', mode 'w' at 0x2b9d92d4f210>)
     |      Read a text input. --- parse and validate a text, and construct input data.
     |
     |      Parameters:
     |        input -- input device.
     |        output -- output device. (default is sys.stderr)
     |
     |      Return:
     |        True or False.
     |        If format of input data is invalid, return False.
     |
     |      Exceptions:
     |        OrpheusStandardError for illegal input data.
     |
     |  ----------------------------------------------------------------------
     |  Class methods inherited from marble.orpheus.input.InputInterface.InputInterface:
     |
     |  create(self, type) from __builtin__.classobj
     |      Create an InputInterface object.(factory method)
     |
     |      Parameters:
     |        type -- target file type.
     |
     |      Return:
     |        InputInterface object.
     |
     |      Exceptions:
     |        OrpheusStandardError for invalid type name.
     |
     |  ----------------------------------------------------------------------
     |  Data and other attributes inherited from marble.orpheus.input.InputInterface.InputInterface:
     |
     |  CALC_CONDITION_FILE = 0
     |
     |  CORE_PROPERTY_FILE = 1
     |
     |  CROSS_SECTION_FILE = 5
     |
     |  GEOMETRY_FILE = 2
     |
     |  MACRO_CROSS_SECTION_FILE = 6
     |
     |  MATERIAL_FILE = 4
     |
     |  PATTERN_FILE = 3

    class MacroCrossSectionFile(marble.orpheus.input.InputInterface.InputInterface)
     |  Methods inherited from marble.orpheus.input.InputInterface.InputInterface:
     |
     |  __init__(self)
     |      Constructor.
     |
     |  data(self)
     |      Get input data.
     |
     |      Return:
     |        input data dictionary.
     |
     |  read(self, input, output=<open file '<stderr>', mode 'w' at 0x2b9d92d4f210>)
     |      Read a text input. --- parse and validate a text, and construct input data.
     |
     |      Parameters:
     |        input -- input device.
     |        output -- output device. (default is sys.stderr)
     |
     |      Return:
     |        True or False.
     |        If format of input data is invalid, return False.
     |
```

```
    Exceptions:
      OrpheusStandardError for illegal input data.

----------------------------------------------------------------------
Class methods inherited from marble.orpheus.input.InputInterface.InputInterface:

create(self, type) from __builtin__.classobj
    Create an InputInterface object.(factory method)

    Parameters:
      type -- target file type.

    Return:
      InputInterface object.

    Exceptions:
      OrpheusStandardError for invalid type name.

----------------------------------------------------------------------
Data and other attributes inherited from marble.orpheus.input.InputInterface.InputInterface:

CALC_CONDITION_FILE = 0

CORE_PROPERTY_FILE = 1

CROSS_SECTION_FILE = 5

GEOMETRY_FILE = 2

MACRO_CROSS_SECTION_FILE = 6

MATERIAL_FILE = 4

PATTERN_FILE = 3
```

Parser.py Page 1/1

```
Help on module Parser:

NAME
    Parser - Parser

FILE
    marble/orpheus/input/Parser.py

CLASSES
    Parser

    class Parser
     |  This class abstracts a parser of a text file, and defines some abstract methods.
     |
     |  Ex:
     |      parser = Parser.create(Parser.YAML_FORMAT)
     |      if parser.read(input, output):
     |          document = parser.getDocument()
     |
     |  Methods defined here:
     |
     |  getDocument(self)
     |      Return a parsed document.
     |      This method is abstract.
     |
     |      Return:
     |        parsed document. (dict)
     |
     |  parse(self, input, output)
     |      Parse a input text file and return True or False.
     |      This method is abstract.
     |
     |      Parameters:
     |        input -- input file handler.
     |        output -- message output handler. (ex. sys.std.err)
     |
     |      Return:
     |        True or Fasle.
     |        If given file format is invalid, return False.
     |
     |  ----------------------------------------------------------------------
     |  Class methods defined here:
     |
     |  create(self, type) from __builtin__.classobj
     |      Create a Parser object. (factory method)
     |
     |      Parameters:
     |        type -- file format type. (YAML_FORMAT)
     |
     |      Return:
     |        Parser object.
     |
     |  ----------------------------------------------------------------------
     |  Data and other attributes defined here:
     |
     |  YAML_FORMAT = 0
```

YamlParser.py Page 1/1

```
Help on module YamlParser:

NAME
    YamlParser - YamlParser

FILE
    marble/orpheus/input/YamlParser.py

CLASSES
    marble.orpheus.input.Parser.Parser
        YamlParser

    class YamlParser(marble.orpheus.input.Parser.Parser)
     |  This class parses a YAML text file.
     |  In parsing, check a syntax error, and write error messages to an output device.
     |
     |  Methods defined here:
     |
     |  __init__(self)
     |      Constructor.
     |
     |  getDocument(self)
     |      Return a parsed document, the data structure of that is a dictionary.
     |
     |      Return:
     |        parsed document. (dict)
     |
     |  parse(self, input, output)
     |      Parse a YAML file.
     |
     |      Parameters:
     |        input -- input file handler.
     |        output -- message output handler. (ex. sys.std.err)
     |
     |      Return:
     |        True or Fasle.
     |        If given file format is invalid, return False.
     |
     |  ----------------------------------------------------------------------
     |  Class methods inherited from marble.orpheus.input.Parser.Parser:
     |
     |  create(self, type) from __builtin__.classobj
     |      Create a Parser object. (factory method)
     |
     |      Parameters:
     |        type -- file format type. (YAML_FORMAT)
     |
     |      Return:
     |        Parser object.
     |
     |  ----------------------------------------------------------------------
     |  Data and other attributes inherited from marble.orpheus.input.Parser.Parser:
     |
     |  YAML_FORMAT = 0
```

Validator.py


```
Help on module Validator:

NAME
    Validator - Validator

FILE
    marble/orpheus/input/Validator.py

CLASSES
    Validator

    class Validator
     |  This class abstracts a validator of a text file, and defines some abstract methods.
     |  The validity of input data is verified based on a given schema.
     |
     |  Ex:
     |      validator = Validator.create(Validator.YAML_FORMAT)
     |      if validator.validate(document, schema, output) == False:
     |          error()
     |
     |  Methods defined here:
     |
     |  validate(self, document, output)
     |      Validate a text file.
     |      This method is abstract.
     |
     |      Parameters:
     |        document -- parsed document. (dict)
     |        output -- message output handler. (ex. sys.std.err)
     |
     |      Return:
     |        True or False.
     |
     |  ----------------------------------------------------------------------
     |  Class methods defined here:
     |
     |  create(self, type) from __builtin__.classobj
     |      Create a Validator object. (factory method)
     |
     |      Parameters:
     |        type -- file format type. (YAML_FORMAT)
     |
     |      Return:
     |        Validator object.
     |
     |  ----------------------------------------------------------------------
     |  Data and other attributes defined here:
     |
     |  YAML_FORMAT = 0
```

YamlValidator.py


```
Help on module YamlValidator:

NAME
    YamlValidator - YamlValidator

FILE
    marble/orpheus/input/YamlValidator.py

CLASSES
    marble.orpheus.input.Validator.Validator
        YamlValidator

    class YamlValidator(marble.orpheus.input.Validator.Validator)
     |  This class validates a YAML text file.
     |
     |  Methods defined here:
     |
     |  __init__(self)
     |      Constructor.
     |
     |  validate(self, document, output)
     |      Validate a YAML file.
     |
     |      Parameters:
     |        document -- parsed document. (dict)
     |        output -- message output handler. (ex. sys.std.err)
     |
     |      Return:
     |        True or False.
     |
     |  ----------------------------------------------------------------------
     |  Class methods inherited from marble.orpheus.input.Validator.Validator:
     |
     |  create(self, type) from __builtin__.classobj
     |      Create a Validator object. (factory method)
     |
     |      Parameters:
     |        type -- file format type. (YAML_FORMAT)
     |
     |      Return:
     |        Validator object.
     |
     |  ----------------------------------------------------------------------
     |  Data and other attributes inherited from marble.orpheus.input.Validator.Validator:
     |
     |  YAML_FORMAT = 0
```

LocationToMeshInterpreter.py


```
Help on module LocationToMeshInterpreter:

NAME
    LocationToMeshInterpreter - LocationToMeshInterpreter

FILE
    marble/orpheus/util/LocationToMeshInterpreter.py

CLASSES
    LocationToMeshInterpreter
    PositionInfo

    class LocationToMeshInterpreter
     |  This class gives the function of conversion of location information.
     |  This class converts PositionInfo object to list of mesh numbers.
     |
     |  Methods defined here:
     |
     |  __init__(self, stepNumSet, meshGeometry, pattern)
     |      Constructor.
     |
     |      Parameters:
     |        stepNumSet -- list of step numbers.
     |        meshGeometry -- MeshGeometry object.
     |        pattern -- Pattern object.
     |
     |  getIndexes(self, positionInfo)
     |      Get mesh numbers and its weights.
     |      Returned data is the dictionary that the key is step number
     |      and the value is list of tuples of mesh numbers and its weights.
     |
     |      Parameters:
     |        positionInfo -- PositionInfo object.
     |
     |      Exceptions:
     |        OrpheusStandardError for PositionInfo object with invalid location information.

    class PositionInfo
     |  This class encapsulates location information about assemblies and fuel pins.
     |
     |  Methods defined here:
     |
     |  __init__(self, *args, **keywords)
     |      Constructor.
     |
     |      Parameters:
     |        args -- step/asyIndex/pinIndex/zIndex or meshIndex
     |        keywords -- step/asyIndex/pinIndex/zIndex or meshIndex

DATA
    CODE_NAME = 'ORPHEUS'
    CODE_VERSION = '1.0'
    ERROR = 3
    NOTE = 1
    PLAIN = 0
    WARNING = 2
```

OrpheusDatabase.py Page 1/2

```
Help on module OrpheusDatabase:

NAME
    OrpheusDatabase - OrpheusDatabase

FILE
    marble/orpheus/util/OrpheusDatabase.py

CLASSES
    OrpheusDatabase
        KvsOrpheusDatabase

    class KvsOrpheusDatabase(OrpheusDatabase)
     |  This class is modeling of KVS type's ORPHEUS database file.
     |
     |  Methods defined here:
     |
     |  __init__(self, databaseFileName, mode='w')
     |      Constructor.
     |
     |      Parameters:
     |        databaseFileName -- database file name.
     |        mode -- access mode (r/w), default is write mode.
     |
     |  close(self)
     |      Close database.
     |
     |  getFloat(self, key)
     |      Get float data.
     |
     |      Parameters:
     |        key -- keyword. (str)
     |
     |      Return:
     |        float data.
     |
     |  getInt(self, key)
     |      Get integer data.
     |
     |      Parameters:
     |        key -- keyword. (str)
     |
     |      Return:
     |        integer data.
     |
     |  getObject(self, key)
     |      Get Python object.
     |
     |      Parameters:
     |        key -- keyword. (str)
     |
     |      Return:
     |        object.
     |
     |  getString(self, key)
     |      Get string data.
     |
     |      Parameters:
     |        key -- keyword. (str)
     |
     |      Return:
     |        string data.
     |
     |  setFloat(self, key, val)
     |      Set float data.
     |
     |      Parameters:
     |        key -- keyword. (str)
     |        val -- value. (float)
     |
     |  setInt(self, key, val)
     |      Set integer data.
     |
     |      Parameters:
     |        key -- keyword. (str)
     |        val -- value. (int)
     |
     |  setObject(self, key, val)
     |      Set Python object.
     |
     |      Parameters:
     |        key -- keyword. (str)
     |        val -- object.
     |
     |  setString(self, key, val)
     |      Set string data.
     |
     |      Parameters:
     |        key -- keyword. (str)
     |        val -- value. (str)
     |
     |  ----------------------------------------------------------------------
     |  Class methods inherited from OrpheusDatabase:
     |
```

OrpheusDatabase.py

```
    create(cls, type, databaseFileName, mode='w') from __builtin__.classobj
        Create OrpheusDatabase object. (factory method)

        Parameters:
          type -- database type.
          databaseFileName -- database file name.
          mode -- access mode (r/w), default is write mode.

        Return:
          OrpheusDatabase object.

class OrpheusDatabase
    This class is modeling of ORPHEUS database file.

    Methods defined here:

    close()
        Close database.
        This method is abstract.

    ----------------------------------------------------------------------
    Class methods defined here:

    create(cls, type, databaseFileName, mode='w') from __builtin__.classobj
        Create OrpheusDatabase object. (factory method)

        Parameters:
          type -- database type.
          databaseFileName -- database file name.
          mode -- access mode (r/w), default is write mode.

        Return:
          OrpheusDatabase object.
```

OrpheusDatabaseHandler.py


```
Help on module OrpheusDatabaseHandler:

NAME
    OrpheusDatabaseHandler - OrpheusDatabaseHandler

FILE
    marble/orpheus/util/OrpheusDatabaseHandler.py

CLASSES
    OrpheusDatabaseHandler
        KvsOrpheusDatabaseHandler

    class KvsOrpheusDatabaseHandler(OrpheusDatabaseHandler)
     |  This class manages a set of orpheus database files.
     |
     |  Methods defined here:
     |
     |  __init__(self, cycleNumber=None, databaseFileName=None)
     |      Constructor.
     |
     |      Parameters:
     |        cycleNumber -- cycle number. (default is None.)
     |        databaseFileName -- database file name. (default is None.)
     |
     |  getCodeFilePath(self, cycleNumber, step_no, code_name, unitno, **args)
     |      Get file path of calculation code.
     |
     |      Parameters:
     |        cycleNumber -- cycle number.
     |        step_no -- step number.
     |        code_name -- code name.
     |        unitno -- unit number.
     |        args -- zone number or material number.
     |                  - zoneNum/materialNum.
     |
     |      Return:
     |        file path that is used by specified code and unit number. (str)
     |
     |  writeBasicData(self, cycleNumber, **objects)
     |      Write basic information.
     |
     |      Parameters:
     |        cycleNumber -- cycle number.
     |        objects -- MARBLE objects.
     |                     - calcConition/coreProperty/zoneSet/pattern/coreFluxInfo/coreMaterialInfo.
     |
     |  writeCodeFilePath(self, cycleNumber, step_no, code_name, solverFileInfo, **args)
     |      Write file path that is used by calculation code.
     |
     |      Parameters:
     |        cycleNumber -- cycle number.
     |        step_no -- step number.
     |        code_name -- code name.
     |        solverInfo -- file path information. (dict)
     |        args -- zone number or material number.
     |                  - zoneNum/materialNum.
     |
     |  writeInitialCrossSectionData(self, cycleNumber, **objects)
     |      Write cross section data at calculation starting point.
     |
     |      Parameters:
     |        cycleNumber -- cycle number.
     |        step -- step number.
     |        objects -- MARBLE objects.
     |                     - zoneSet/crossSectionInfo.
     |
     |  writeStepData(self, cycleNumber, step, **objects)
     |      Write calculation step information.
     |
     |      Parameters:
     |        cycleNumber -- cycle number.
     |        step -- step number.
     |        objects -- MARBLE objects.
     |                     - calcCondition/coreProperty/zoneSet/coreFluxInfo/coreMaterialInfo/crossSectionInfo.
     |
     |  ----------------------------------------------------------------------
     |  Methods inherited from OrpheusDatabaseHandler:
     |
     |  closeDatabase(self, cycleNumber=None)
     |      Close all database.
     |      If cycle number is specified, close only its database.
     |
     |      Parameters:
     |        cycleNumber -- cycle number. (defalult is None)
     |
     |  database(self, cycleNumber)
     |      Get OrpheusDatabase object.
     |
     |      Parameters:
     |        cycleNumber -- cycle number.
     |
     |      Return:
     |        OrpheusDatabase object.
     |
```

```
        Exceptions:
          OrpheusStandardError for invalid cycle number.

    edit(self, cycleNumber, positionInfo, target, operation)
        Get the result that is edited. -- keff/flux/material/... etc.

        Parameters:
          cycleNumber -- cycle number.
          positionInfo -- PositionInfo object.
          target -- data target keyword (KEFF/MICROXS/FLUX/ND/FLUX_VOL/MATERIAL_VOL)
          operation -- data operation keyword (GET/AVE/SUM)

        Return:
          edited result.

        Exceptions:
          OrpheusStandardError for unsupported targets/operations.

    getCycleNumberList(self)
        Get list of cycle numbers.

        Return:
          list of cycle numbers.

    setupDatabase(self, cycleNumber, databaseFileName, dbmode='w')
        Setup ORPHEUS database for read/write.

        Parameters:
          cycleNumber -- cycle number.
          databaseFileName -- database file (path).
          dbmode -- access mode (r/w).

        Exceptions:
          OrpheusStandardError for invalid database file.

class OrpheusDatabaseHandler
    This class handles a set of orpheus database files.

    Methods defined here:

    __init__(self, databaseType, cycleNumber=None, databaseFileName=None)
        Constructor.

        Parameters:
          databaseType -- database type.
          cycleNumber -- cycle number. (default is None.)
          databaseFileName -- database file name. (default is None.)

    closeDatabase(self, cycleNumber=None)
        Close all database.
        If cycle number is specified, close only its database.

        Parameters:
          cycleNumber -- cycle number. (defalult is None)

    database(self, cycleNumber)
        Get OrpheusDatabase object.

        Parameters:
          cycleNumber -- cycle number.

        Return:
          OrpheusDatabase object.

        Exceptions:
          OrpheusStandardError for invalid cycle number.

    edit(self, cycleNumber, positionInfo, target, operation)
        Get the result that is edited. -- keff/flux/material/... etc.

        Parameters:
          cycleNumber -- cycle number.
          positionInfo -- PositionInfo object.
          target -- data target keyword (KEFF/MICROXS/FLUX/ND/FLUX_VOL/MATERIAL_VOL)
          operation -- data operation keyword (GET/AVE/SUM)

        Return:
          edited result.

        Exceptions:
          OrpheusStandardError for unsupported targets/operations.

    getCodeFilePath(self, cycleNumber, step_no, code_name, unitno, **args)
        Get file path of calculation code.
        This method is abstract.

        Parameters:
          cycleNumber -- cycle number.
          step_no -- step number.
          code_name -- code name.
          unitno -- unit number.
          args -- zone number or material number.

        Return:
```

```
        file path that is used by specified code and unit number. (str)

    getCycleNumberList(self)
        Get list of cycle numbers.

        Return:
          list of cycle numbers.

    setupDatabase(self, cycleNumber, databaseFileName, dbmode='w')
        Setup ORPHEUS database for read/write.

        Parameters:
          cycleNumber -- cycle number.
          databaseFileName -- database file (path).
          dbmode -- access mode (r/w).

        Exceptions:
          OrpheusStandardError for invalid database file.

    writeBasicData(self, cycleNumber, **objects)
        Write basic information.
        This method is abstract.

        Parameters:
          cycleNumber -- cycle number.
          objects -- MARBLE objects. (Pattern, ZoneSet etc.)

    writeCodeFilePath(self, cycleNumber, step_no, code_name, solverFileInfo, **args)
        Write file path that is used by calculation code.
        This method is abstract.

        Parameters:
          cycleNumber -- cycle number.
          step_no -- step number.
          code_name -- code name.
          solverInfo -- file path information. (dict)
          args -- zone number or material number.

    writeInitialCrossSectionData(self, cycleNumber, **objects)
        Write cross section data at calculation starting point.
        This method is abstract.

        Parameters:
          cycleNumber -- cycle number.
          step -- step number.
          objects -- MARBLE objects. (CoreFluxInfo, CoreMaterialInfo etc.)

    writeStepData(self, cycleNumber, step, **objects)
        Write calculation step information.
        This method is abstract.

        Parameters:
          cycleNumber -- cycle number.
          step -- step number.
          objects -- MARBLE objects. (CoreFluxInfo, CoreMaterialInfo etc.)

DATA
    CODE_NAME = 'ORPHEUS'
    CODE_VERSION = '1.0'
    CONTROL_ROD = 2
    ERROR = 3
    NOTE = 1
    PLAIN = 0
    WARNING = 2
```

OrpheusException.py


```
Help on module OrpheusException:

NAME
    OrpheusException - OrpheusStandardError

FILE
    marble/orpheus/util/OrpheusException.py

CLASSES
    exceptions.StandardError(exceptions.Exception)
        OrpheusStandardError

    class OrpheusStandardError(exceptions.StandardError)
     |  Method resolution order:
     |      OrpheusStandardError
     |      exceptions.StandardError
     |      exceptions.Exception
     |      exceptions.BaseException
     |      __builtin__.object
     |
     |  Methods defined here:
     |
     |  __init__(self, level, message)
     |      Constructor.
     |
     |      Parameters:
     |        level -- error level. (NOTE/WARNING/ERROR)
     |        message -- error message.
     |
     |  __str__(self)
     |      Get error message.
     |
     |      Return:
     |        error message.
     |
     |  ----------------------------------------------------------------------
     |  Data descriptors defined here:
     |
     |  __weakref__
     |      list of weak references to the object (if defined)
     |
     |  ----------------------------------------------------------------------
     |  Data and other attributes inherited from exceptions.StandardError:
     |
     |  __new__ = <built-in method __new__ of type object at 0x719400>
     |      T.__new__(S, ...) -> a new object with type S, a subtype of T
     |
     |  ----------------------------------------------------------------------
     |  Methods inherited from exceptions.BaseException:
     |
     |  __delattr__(...)
     |      x.__delattr__('name') <==> del x.name
     |
     |  __getattribute__(...)
     |      x.__getattribute__('name') <==> x.name
     |
     |  __getitem__(...)
     |      x.__getitem__(y) <==> x[y]
     |
     |  __getslice__(...)
     |      x.__getslice__(i, j) <==> x[i:j]
     |
     |      Use of negative indices is not supported.
     |
     |  __reduce__(...)
     |
     |  __repr__(...)
     |      x.__repr__() <==> repr(x)
     |
     |  __setattr__(...)
     |      x.__setattr__('name', value) <==> x.name = value
     |
     |  __setstate__(...)
     |
     |  ----------------------------------------------------------------------
     |  Data descriptors inherited from exceptions.BaseException:
     |
     |  __dict__
     |
     |  args
     |
     |  message
     |      exception message
```

国際単位系（SI）

表１．SI 基本単位

基本量	SI 基本単位	
	名称	記号
長さ	メートル	m
質量	キログラム	kg
時間	秒	s
電流	アンペア	A
熱力学温度	ケルビン	K
物質量	モル	mol
光度	カンデラ	cd

表２．基本単位を用いて表されるSI組立単位の例

組立量	SI 基本単位	
	名称	記号
面積	平方メートル	m^2
体積	立法メートル	m^3
速さ，速度	メートル毎秒	m/s
加速度	メートル毎秒毎秒	m/s^2
波数	毎メートル	m^{-1}
密度，質量密度	キログラム毎立方メートル	kg/m^3
面積密度	キログラム毎平方メートル	kg/m^2
比体積	立方メートル毎キログラム	m^3/kg
電流密度	アンペア毎平方メートル	A/m^2
磁界の強さ	アンペア毎メートル	A/m
量濃度(a)，濃度	モル毎立方メートル	mol/m^3
質量濃度	キログラム毎立法メートル	kg/m^3
輝度	カンデラ毎平方メートル	cd/m^2
屈折率(b)	（数字の） 1	1
比透磁率(b)	（数字の） 1	1

(a) 量濃度（amount concentration）は臨床化学の分野では物質濃度（substance concentration）ともよばれる。
(b) これらは無次元量あるいは次元１をもつ量であるが、そのことを表す単位記号である数字の１は通常は表記しない。

表３．固有の名称と記号で表されるSI組立単位

組立量	SI 組立単位			
	名称	記号	他のSI単位による表し方	SI基本単位による表し方
平面角	ラジアン(b)	rad	1 (b)	m/m
立体角	ステラジアン(b)	sr(c)	1 (b)	m^2/m^2
周波数	ヘルツ(d)	Hz		s^{-1}
力	ニュートン	N		m kg s^{-2}
圧力，応力	パスカル	Pa	N/m^2	m^{-1} kg s^{-2}
エネルギー，仕事，熱量	ジュール	J	N m	m^2 kg s^{-2}
仕事率，工率，放射束	ワット	W	J/s	m^2 kg s^{-3}
電荷，電気量	クーロン	C		s A
電位差（電圧），起電力	ボルト	V	W/A	m^2 kg s^{-3} A^{-1}
静電容量	ファラド	F	C/V	m^{-2} kg^{-1} s^4 A^2
電気抵抗	オーム	Ω	V/A	m^2 kg s^{-3} A^{-2}
コンダクタンス	ジーメンス	S	A/V	m^{-2} kg^{-1} s^3 A^2
磁束	ウエーバ	Wb	Vs	m^2 kg s^{-2} A^{-1}
磁束密度	テスラ	T	Wb/m^2	kg s^{-2} A^{-1}
インダクタンス	ヘンリー	H	Wb/A	m^2 kg s^{-2} A^{-2}
セルシウス温度	セルシウス度(e)	℃		K
光束	ルーメン	lm	cd sr(c)	cd
照度	ルクス	lx	lm/m^2	m^{-2} cd
放射性核種の放射能(f)	ベクレル(d)	Bq		s^{-1}
吸収線量，比エネルギー分与，カーマ	グレイ	Gy	J/kg	m^2 s^{-2}
線量当量，周辺線量当量，方向性線量当量，個人線量当量	シーベルト(g)	Sv	J/kg	m^2 s^{-2}
酸素活性	カタール	kat		s^{-1} mol

(a)SI接頭語は固有の名称と記号を持つ組立単位と組み合わせても使用できる。しかし接頭語を付した単位はもはやコヒーレントではない。
(b)ラジアンとステラジアンは数字の１に対する単位の特別な名称で、量についての情報をつたえるために使われる。実際には、使用する時には記号rad及びsrが用いられるが、習慣として組立単位としての記号である数字の１は明示されない。
(c)測光学ではステラジアンという名称と記号srを単位の表し方の中に、そのまま維持している。
(d)ヘルツは周期現象についてのみ、ベクレルは放射性核種の統計的過程についてのみ使用される。
(e)セルシウス度はケルビンの特別な名称で、セルシウス温度を表すために使用される。セルシウス度とケルビンの単位の大きさは同一である。したがって、温度差や温度間隔を表す数値はどちらの単位で表しても同じである。
(f)放射性核種の放射能（activity referred to a radionuclide）は、しばしば誤った用語で"radioactivity"と記される。
(g)単位シーベルト（PV,2002,70,205）についてはCIPM勧告2（CI-2002）を参照。

表４．単位の中に固有の名称と記号を含むSI組立単位の例

組立量	SI 組立単位		
	名称	記号	SI 基本単位による表し方
粘度	パスカル秒	Pa s	m^{-1} kg s^{-1}
力のモーメント	ニュートンメートル	N m	m^2 kg s^{-2}
表面張力	ニュートン毎メートル	N/m	kg s^{-2}
角速度	ラジアン毎秒	rad/s	m m^{-1} s^{-1}=s^{-1}
角加速度	ラジアン毎秒毎秒	rad/s^2	m m^{-1} s^{-2}=s^{-2}
熱流密度，放射照度	ワット毎平方メートル	W/m^2	kg s^{-3}
熱容量，エントロピー	ジュール毎ケルビン	J/K	m^2 kg s^{-2} K^{-1}
比熱容量，比エントロピー	ジュール毎キログラム毎ケルビン	J/(kg K)	m^2 s^{-2} K^{-1}
比エネルギー	ジュール毎キログラム	J/kg	m^2 s^{-2}
熱伝導率	ワット毎メートル毎ケルビン	W/(m K)	m kg s^{-3} K^{-1}
体積エネルギー	ジュール毎立方メートル	J/m^3	m^{-1} kg s^{-2}
電界の強さ	ボルト毎メートル	V/m	m kg s^{-3} A^{-1}
電荷密度	クーロン毎立方メートル	C/m^3	m^{-3} sA
表面電荷	クーロン毎平方メートル	C/m^2	m^{-2} sA
電束密度，電気変位	クーロン毎平方メートル	C/m^2	m^{-2} sA
誘電率	ファラド毎メートル	F/m	m^{-3} kg^{-1} s^4 A^2
透磁率	ヘンリー毎メートル	H/m	m kg s^{-2} A^{-2}
モルエネルギー	ジュール毎モル	J/mol	m^2 kg s^{-2} mol^{-1}
モルエントロピー，モル熱容量	ジュール毎モル毎ケルビン	J/(mol K)	m^2 kg s^{-2} K^{-1} mol^{-1}
照射線量（X線及びγ線）	クーロン毎キログラム	C/kg	kg^{-1} sA
吸収線量率	グレイ毎秒	Gy/s	m^2 s^{-3}
放射強度	ワット毎ステラジアン	W/sr	m^4 m^{-2} kg s^{-3}=m^2 kg s^{-3}
放射輝度	ワット毎平方メートル毎ステラジアン	$W/(m^2$ sr)	m^2 m^{-2} kg s^{-3}=kg s^{-3}
酵素活性濃度	カタール毎立方メートル	kat/m^3	m^{-3} s^{-1} mol

表５．SI 接頭語

乗数	接頭語	記号	乗数	接頭語	記号
10^{24}	ヨタ	Y	10^{-1}	デシ	d
10^{21}	ゼタ	Z	10^{-2}	センチ	c
10^{18}	エクサ	E	10^{-3}	ミリ	m
10^{15}	ペタ	P	10^{-6}	マイクロ	μ
10^{12}	テラ	T	10^{-9}	ナノ	n
10^9	ギガ	G	10^{-12}	ピコ	p
10^6	メガ	M	10^{-15}	フェムト	f
10^3	キロ	k	10^{-18}	アト	a
10^2	ヘクト	h	10^{-21}	ゼプト	z
10^1	デカ	da	10^{-24}	ヨクト	y

表６．SIに属さないが、SIと併用される単位

名称	記号	SI 単位による値
分	min	1 min=60s
時	h	1h =60 min=3600 s
日	d	1 d=24 h=86 400 s
度	°	1°=(π/180) rad
分	’	1’=(1/60)°=(π/10800) rad
秒	”	1”=(1/60)’=(π/648000) rad
ヘクタール	ha	$1ha=1hm^2=10^4m^2$
リットル	L, l	$1L=11=1dm^3=10^3cm^3=10^{-3}m^3$
トン	t	$1t=10^3$ kg

表７．SIに属さないが、SIと併用される単位で、SI単位で表される数値が実験的に得られるもの

名称	記号	SI 単位で表される数値
電子ボルト	eV	$1eV=1.602\ 176\ 53(14)\times10^{-19}J$
ダルトン	Da	$1Da=1.660\ 538\ 86(28)\times10^{-27}kg$
統一原子質量単位	u	1u=1 Da
天文単位	ua	$1ua=1.495\ 978\ 706\ 91(6)\times10^{11}m$

表８．SIに属さないが、SIと併用されるその他の単位

名称	記号	SI 単位で表される数値
バール	bar	$1\ bar=0.1MPa=100kPa=10^5Pa$
水銀柱ミリメートル	mmHg	1mmHg=133.322Pa
オングストローム	Å	$1\ Å=0.1nm=100pm=10^{-10}m$
海里	M	1 M=1852m
バーン	b	$1\ b=100fm^2=(10^{-12}cm)2=10^{-28}m^2$
ノット	kn	1 kn=(1852/3600)m/s
ネーパ	Np	SI単位との数値的な関係は、対数量の定義に依存。
ベル	B	
デジベル	dB	

表９．固有の名称をもつCGS組立単位

名称	記号	SI 単位で表される数値
エルグ	erg	$1\ erg=10^{-7}$ J
ダイン	dyn	$1\ dyn=10^{-5}$N
ポアズ	P	1 P=1 dyn s cm^{-2}=0.1Pa s
ストークス	St	1 St =$1cm^2$ s^{-1}=$10^{-4}m^2$ s^{-1}
スチルブ	sb	1 sb =1cd cm^{-2}=10^4cd m^{-2}
フォト	ph	1 ph=1cd sr cm^{-2} 10^4lx
ガル	Gal	1 Gal =1cm s^{-2}=$10^{-2}ms^{-2}$
マクスウェル	Mx	1 Mx = 1G cm^2=10^{-8}Wb
ガウス	G	1 G =1Mx cm^{-2} =10^{-4}T
エルステッド(c)	Oe	1 Oe ≙ $(10^3/4\pi)$A m^{-1}

(c) ３元系のCGS単位系とSIでは直接比較できないため、等号「 ≙ 」は対応関係を示すものである。

表10．SIに属さないその他の単位の例

名称	記号	SI 単位で表される数値
キュリー	Ci	$1\ Ci=3.7\times10^{10}Bq$
レントゲン	R	$1\ R=2.58\times10^{-4}C/kg$
ラド	rad	$1\ rad=1cGy=10^{-2}Gy$
レム	rem	$1\ rem=1\ cSv=10^{-2}Sv$
ガンマ	γ	1γ=1 nT=10-9T
フェルミ		1フェルミ=1 fm=10-15m
メートル系カラット		1メートル系カラット = 200 mg = 2×10-4kg
トル	Torr	1 Torr = (101 325/760) Pa
標準大気圧	atm	1 atm = 101 325 Pa
カロリー	cal	1cal=4.1858J（「15℃」カロリー），4.1868J（「IT」カロリー）4.184J（「熱化学」カロリー）
ミクロン	μ	1 μ =1μm=10^{-6}m

（第8版，2006年改訂）

この印刷物は再生紙を使用しています