Docker Compose ファイルとUML 図の互換及び検証をサポートするastah* プラグインの実装

信州大学工学部電子情報工学科

16T2133H 野々川 麗樺

目次

- 背景・目的
- YAMLとUMLの互換
- 詳細表示と編集のためのUI
- 検査と検証
- 実装結果
- 互換機能実装
- UIと編集機能の実装
- 検査と検証機能実装
- まとめ・今後

背景・目的

- Docker は、コンテナ型の仮想環境を提供するプラットフォームであり、DevOps で盛んに取り 入れられている。
- Docker Compose は、Docker エンジンに基づいたマルチコンテナの管理ツールであり、コンテナの設定情報をあらかじめYAMLファイルに記述したうえで管理を行う。
  - YAMLによる記述の形式はサービス間の関係がわかりにくい。
  - YAMLに記述したリファレンスの検査は起動時に初めて行われ、妥当性についてチェックされない。

Docker Compose で用いるマルチコンテナ設計の効率向上及び正確性の向上をサポートするのが本研究の目的である。

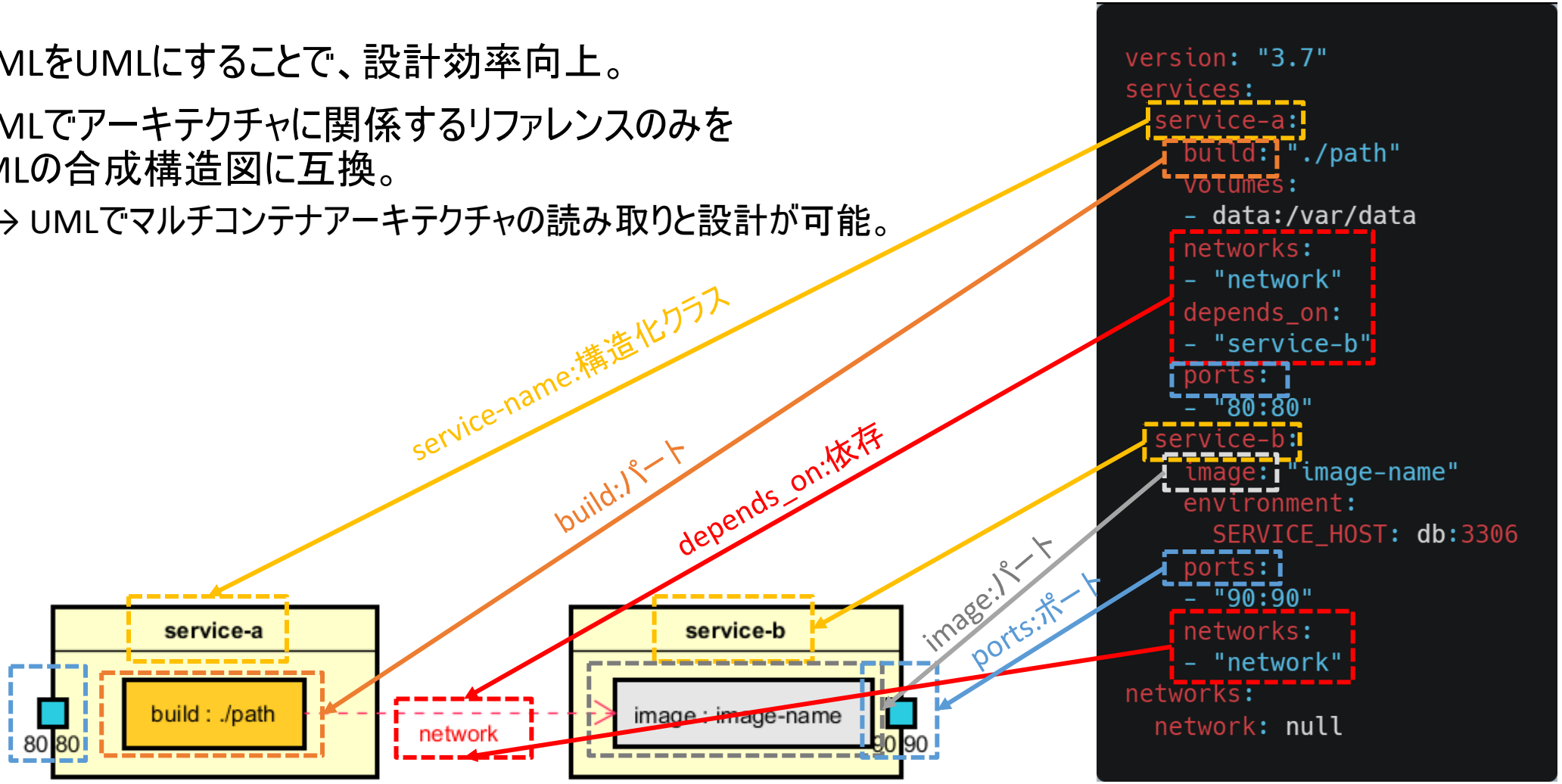
```yaml
version: "3.7"
services:

  serviceA:
    build: ./serviceA
    networks:
      - network1
    depends_on:
      - serviceB

  serviceB:
    build: ./serviceB
    networks:
      - network1
      - network2
```

YAMLとUMLの互換：リファレンス対応付け

- YAMLをUMLにすることで、設計効率向上。
- YAMLでアーキテクチャに関係するリファレンスのみを
  UMLの合成構造図に互換。
  → UMLでマルチコンテナアーキテクチャの読み取りと設計が可能。



YAMLとUMLの互換：UMLでのサービス配置

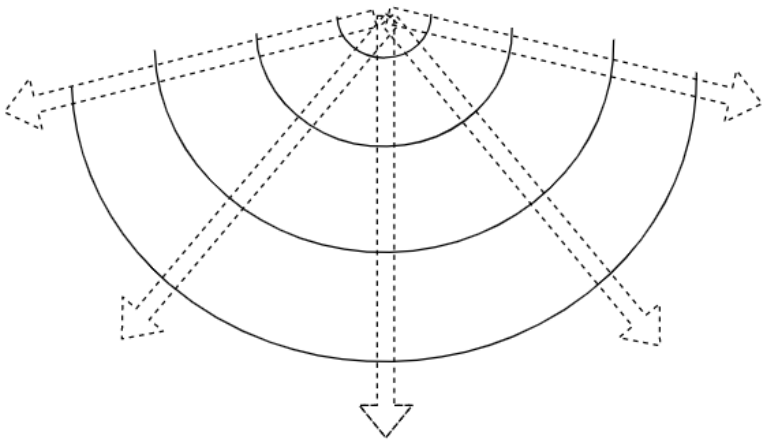
理解しやすい扇円配置決め

システム構造の関係で決める → 依存数

- 上位のサービス（ゲートウェイなど）が他のサービス多く依存 → 扇円の中央
- 下位のサービス（データベースなど）が他のサービスに多く依存される → 扇円の外側

サービスのつながりで決める → 共有ネットワーク

- サービスが同じネットワークを共有 → 同じ縦ブロック



依存数で扇円での行、共有ネットワークで扇円での列を決める

詳細表示と編集のためのUI

UML互換では一部のリファレンスしか互換されない

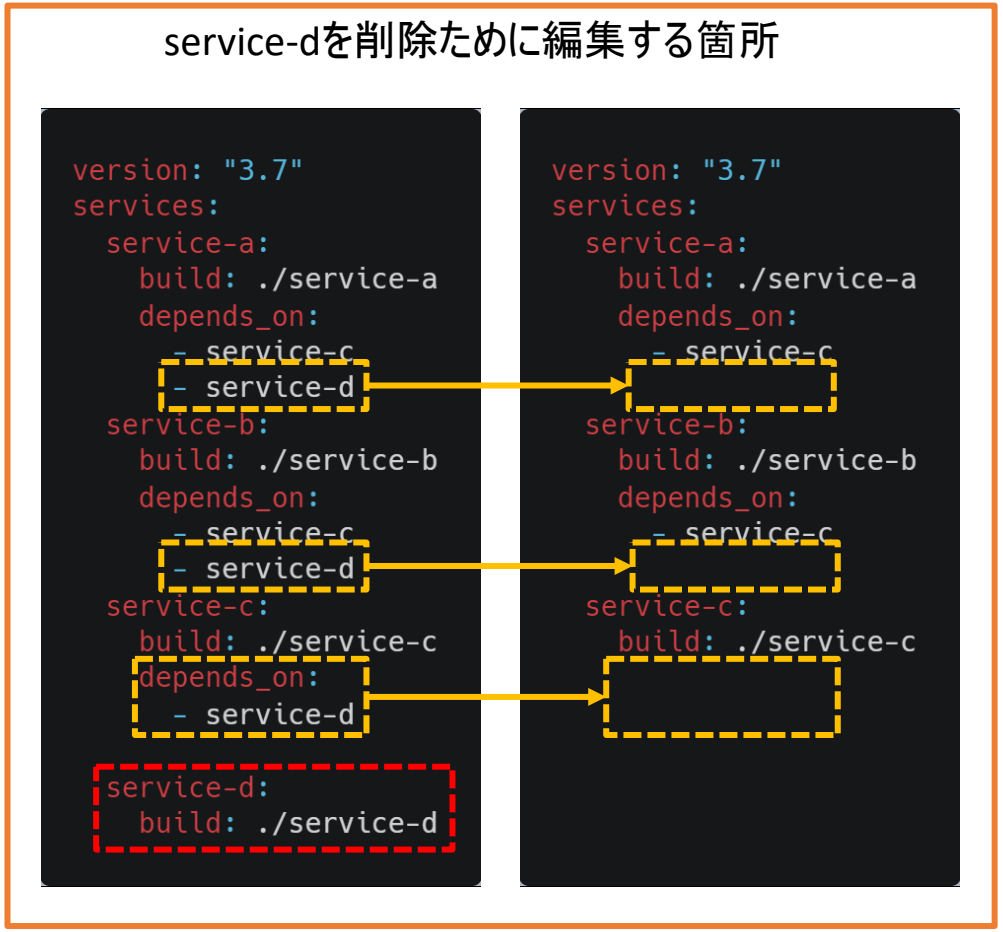
→ すべてのリファレンスを格納できるUIが必要である

一か所の編集を行うのに引用箇所の変更も必要

→ 編集箇所を全体に反映できる機能が必要である

UIで必要な機能：

- 指定箇所のリファレンス情報をYAMLテキストの状態で閲覧できる
- サービスについてYAMLテキストで編集できる
- YAMLテキストによるサービスの追加ができる
- 指定サービスの削除ができる
- UIからの編集、削除、追加を全体に反映することができる



検査と検証 : リファレンス検査

UMLに互換されたDocker Compose YAMLが、
実際にDocker Composeで動作するYAMLであるかを確かめる必要がある。

本来はDocker Compose起動時に行われる検査を、
UML互換の過程やUIによる編集の際に行うことで、早い段階でのエラー発見が可能になる。

リファレンス検査を行うタイミング

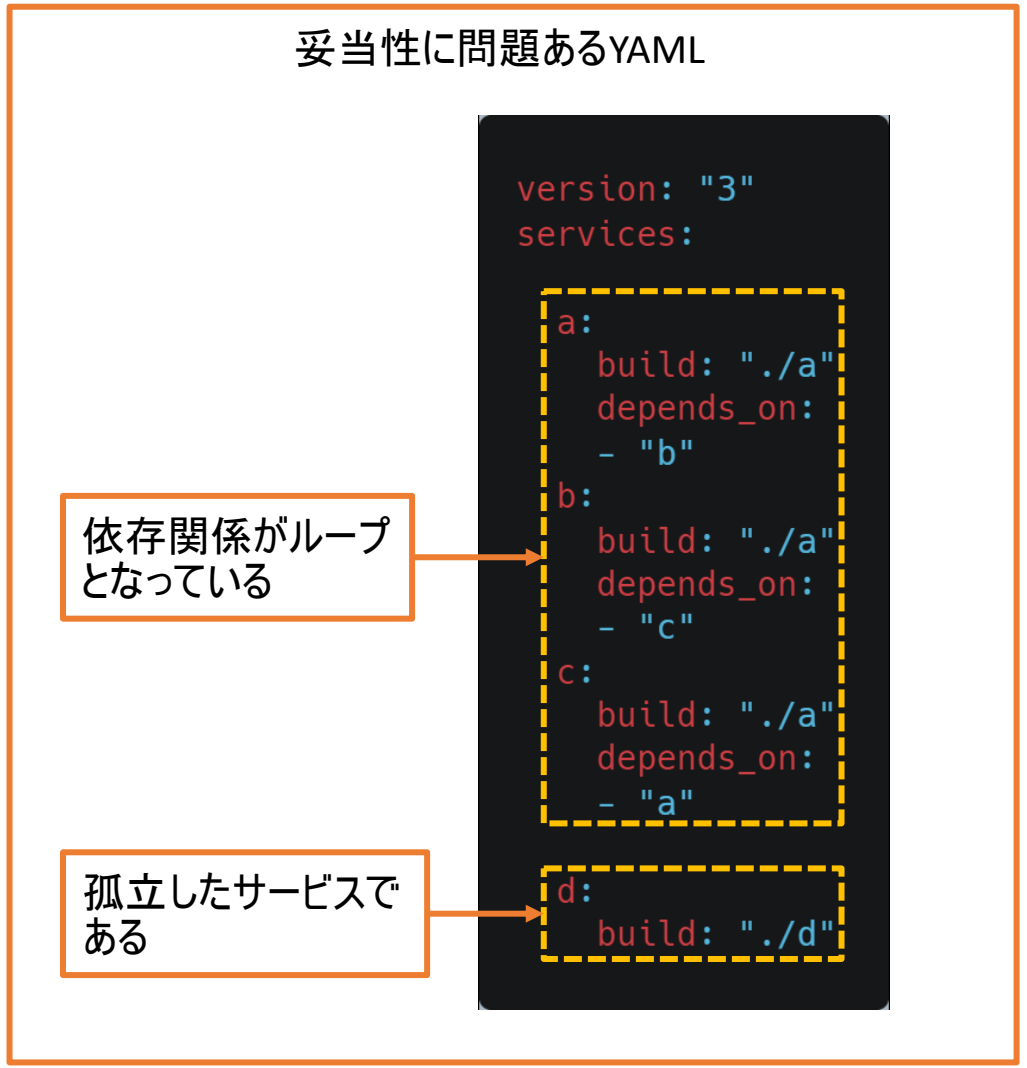
- YAMLファイルをUMLに互換する時
- UMLからYAMLファイルに互換する時
- UIでサービスの編集、追加、削除などを行ったとき

検査と検証：妥当性検証

Docker Composeでの起動後、
システムとしての正確性を高めるために妥当性を検証する。

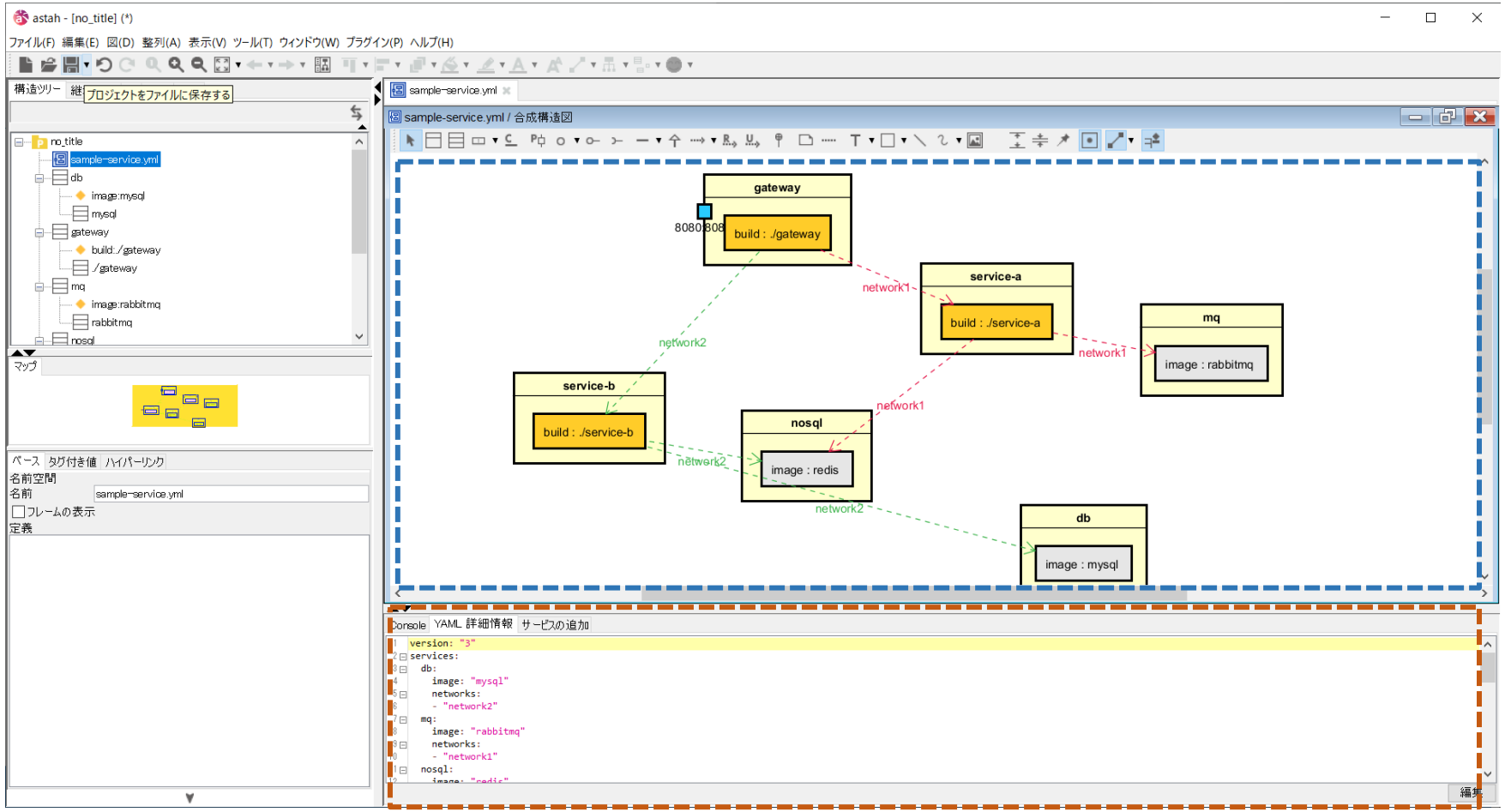
起動後にエラーによるコンテナ停止のリスクを下げる。

- ワーニングを示す箇所
  - サービスが孤立している
  - サービス間の依存がループになっている
- エラーを示す箇所
  - 同じ名前のサービスが存在する
  - 他のサービスとポートが被っている
  - 既存でないネットワークに依存している



実装結果

astahでプラグインとして実装をした結果である



UML互換結果表示フォーム

詳細表示と編集のためのUI

互換機能実装：YAMLからUMLにリバース

- YAMLからUMLを生成するリバースの過程はYAML検査の後に行われる。
- 新規作成の場合はテンプレートYAMLからのリバースが行われる。

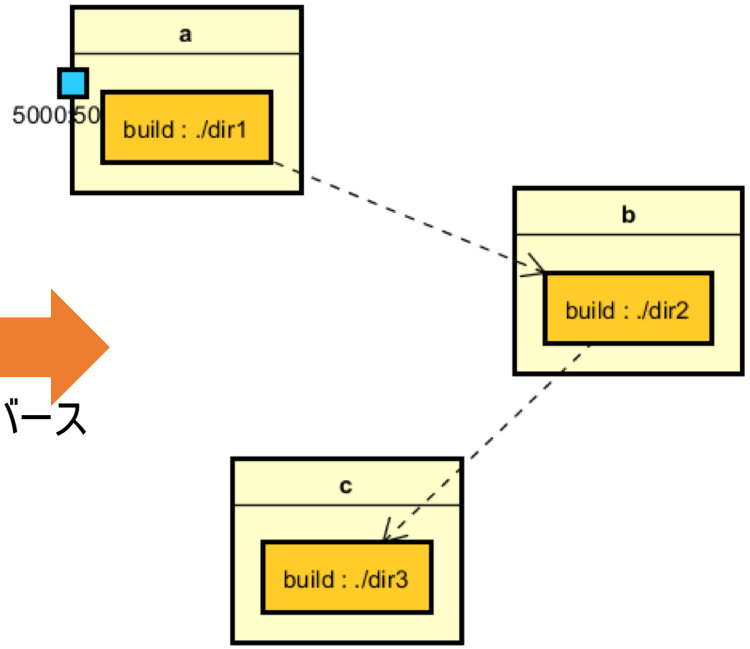
リバース実行フローは右図である。

リバースの例

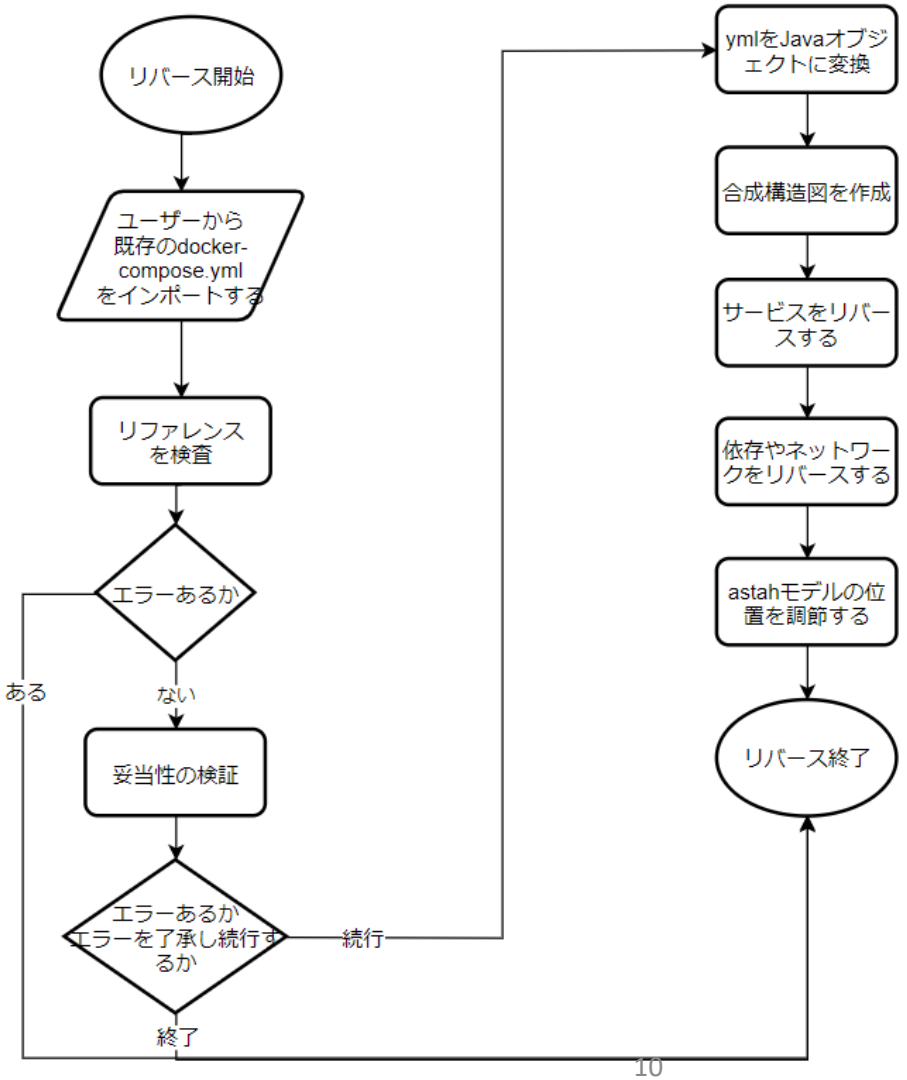
```yaml
version: "3.7"
services:
  a:
    build: "./dir1"
    ports:
    - "5000:5000"
    command: "python app.py"
    depends_on:
    - "b"
  b:
    build: "./dir2"
    depends_on:
    - "c"
  c:
    build: "./dir3"
```

リバースされない
リファレンス

リバース



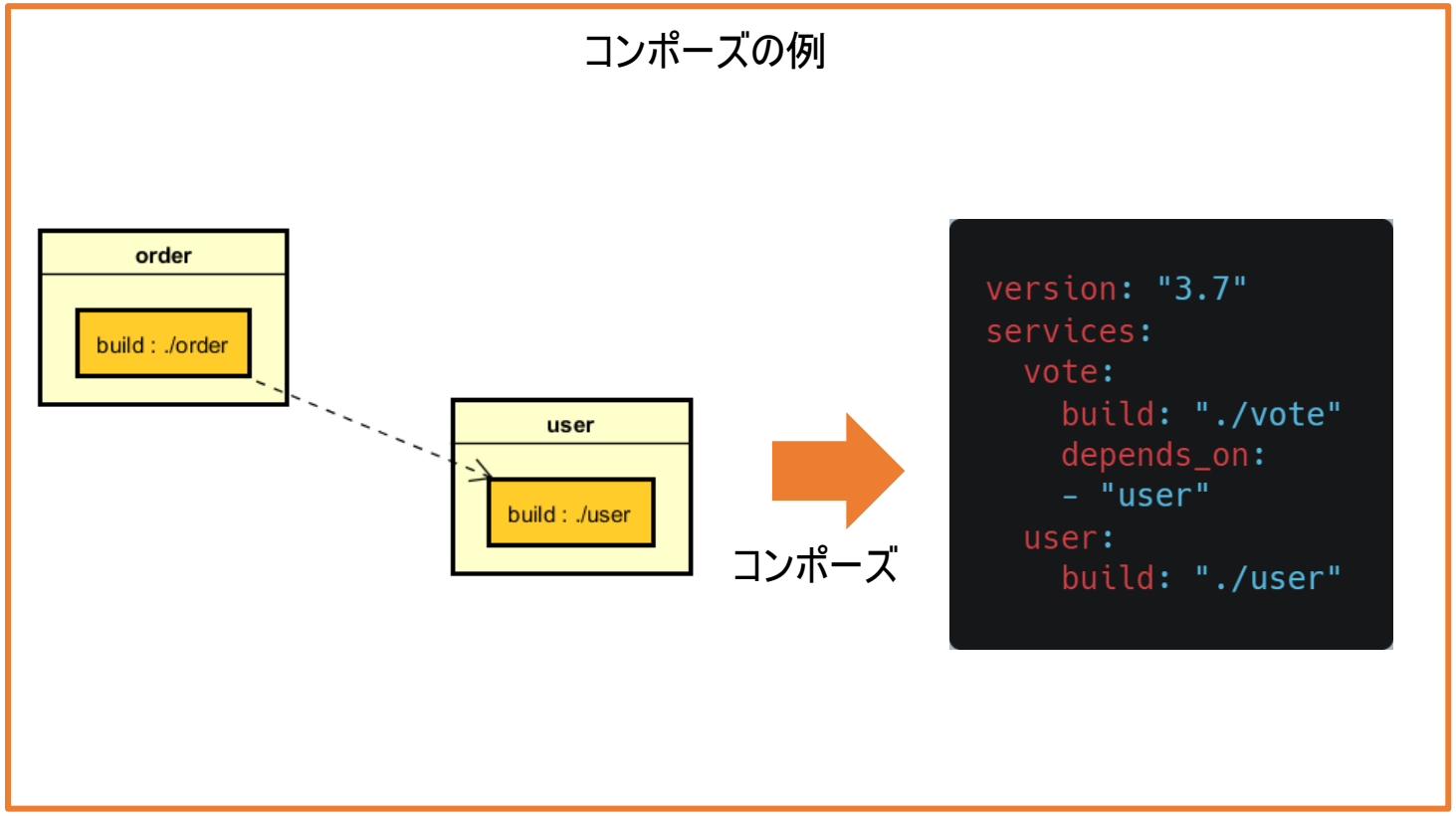
リバース実行フロー



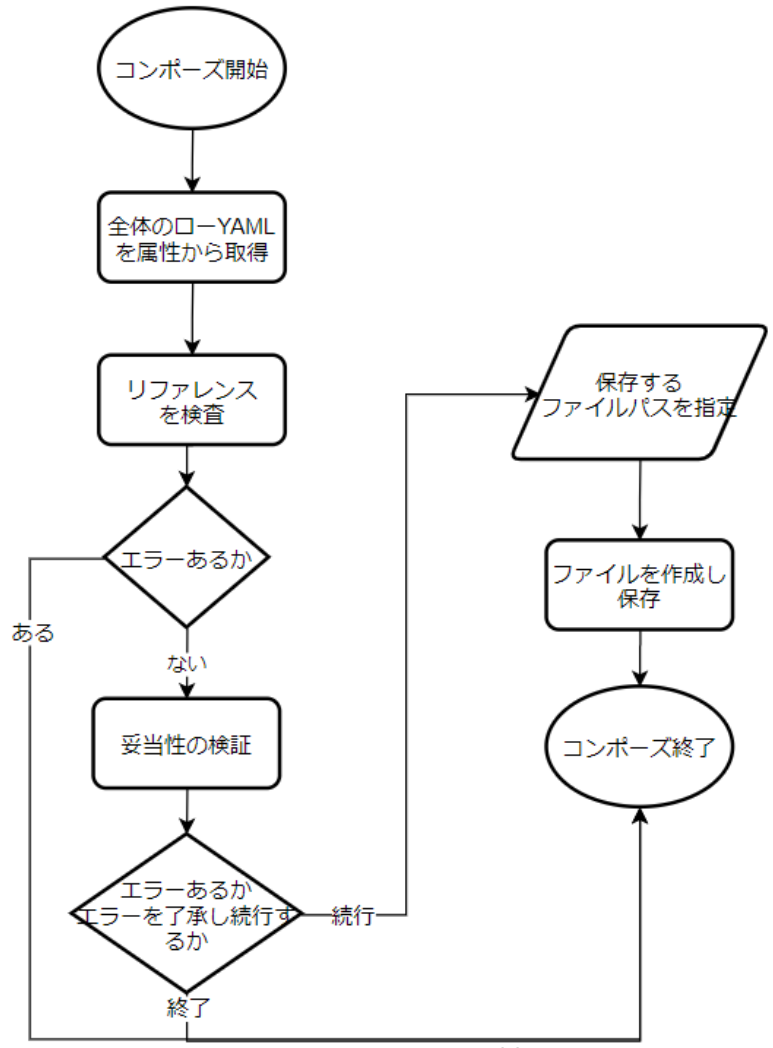
互換機能実装：UMLからYAMLにコンポーズ

- UMLからYAMLを生成するコンポーズの過程はYAML検査の後に行われる。
- 実装クラスで保持している現在図を示すYAMLテキストを出力する。

コンポーズ実行フローは右図である。



コンポーズ実行フロー



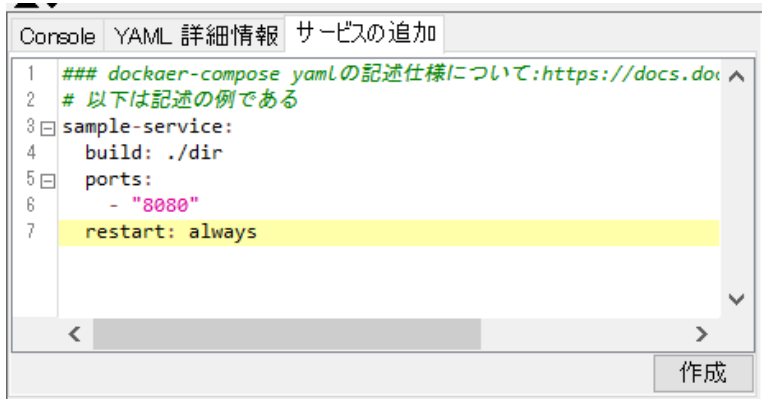
UIと編集機能の実装

- astahの拡張ビューにYAMLでの詳細表示と編集のためのUIを実装した。
- 全体と連動し、サービスの編集、追加、削除が可能である。
- 実際は検査後に編集済みYAMLが再リバースされ、UMLが更新される。

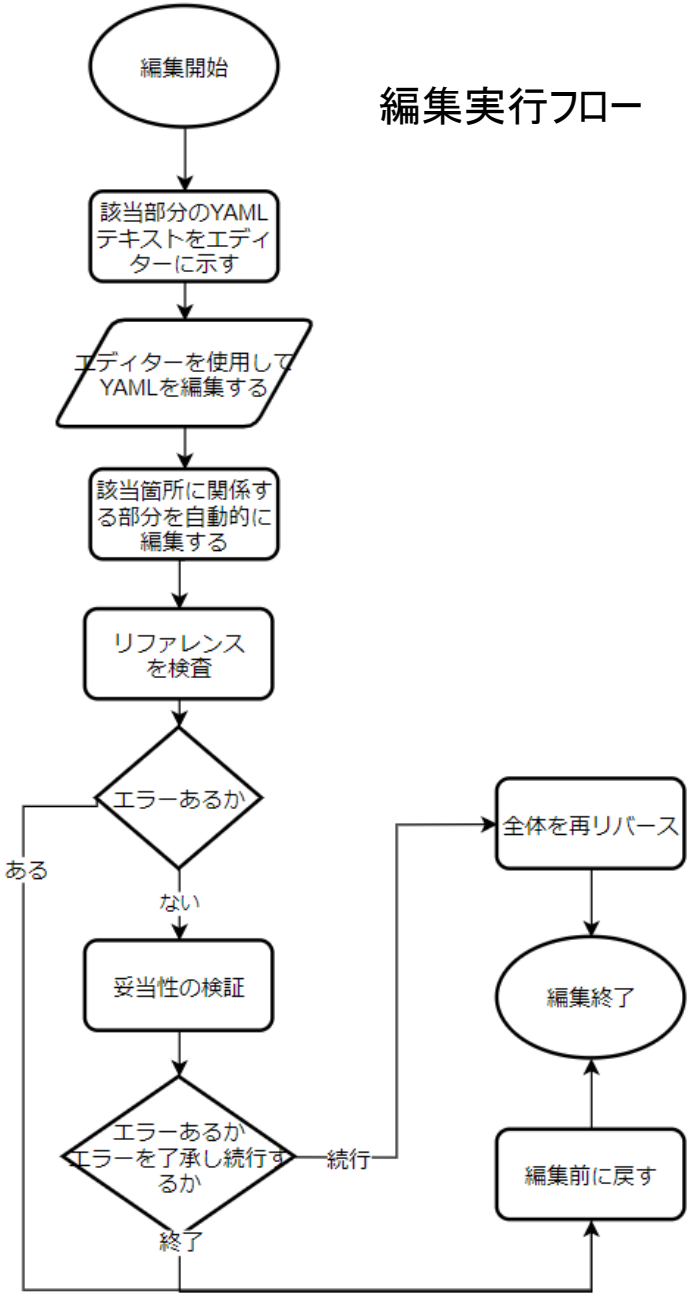
編集実行フローは右図である。

編集と削除を行うUI

テンプレートによる
サービスの追加を行うUI



編集実行フロー



検査と検証機能実装：リファレンス検査

Docker Composeではリファレンス検査の機能をすでに備わっている。

→ JythonでDocker Composeソースコードをプラグインに移植した。

実装クラスでは、
リファレンス検査該当部分を全体からの剥離は行っていない。

→ 検査実行の過程は素早いが、セットアップ過程は重い。

CPU Intel Core i5-8350U、RAM 8GBのマシンで実験した結果が以下

- プラグインビルド時間 ：1分
- 生成されたプラグインJarの大きさ：60MB
- 初回の検査を行うときのセットアップ時間：10秒
- 実際の検査にかかる時間：20ms~300ms

検査と検証機能実装：妥当性検証

クラスComposeValidationCheckerにてJavaで実装を行った。

妥当性検証はリファレンス検査の後に実行される。

メソッド名	検証項目	レベル
checkServiceIsolation()	サービスが孤立している	ワーニング
checkServiceDependsLoop()	サービス間の依存がループになっている	
checkDuplicatedServiceName()	同じ名前のサービスが存在する	エラー
checkDuplicatedServicePorts()	他のサービスとポートが被っている	
checkUndifinedNetwork()	既存でないネットワークに依存している	

リファレンス検査でエラーが検出された場合、直ちに現在の操作をキャンセルするが、妥当性の検証については続行することが可能である。

まとめ・今後

- UMLとの互換で、コンテナ間の依存関係が理解しやすくなった。
- UI を実装したことで、astah のみで外部に頼らない設計手法が可能になった。
- 検査と検証で、早い段階でのエラー発見が可能になった。

考察・今後

- 互換実験を繰り返したうち、エイリアスによる互換失敗の例を発見した。
  → エイリアスでの到達を可能にする。
- リファレンス検査のためにDocker Composeソースコード全体を内蔵した。
  → 該当部分を全体から剥離し、動作を軽くする
- 妥当性強化のため、さらに検証項目が必要である。

参考文献

[1] kotaro-dr:【図解】Docker の全体像を理解する（https://qiita.com/kotarodr/items/b1024c7d200a75b992fc）

[2] 株式会社チェンジビジョン: astah*professional（http://astah.changevision.com/ja/product/astah-professional.html）

[3] Docker:（https://docs.docker.com/compose/compose-file/）

[4] 畑瀬尚之，和﨑克己（2019）:UML 上位設計からの自動コード生成を対象とした整合性検査; 2019 年度電子情報通信学会信州大学Student Branch 論文発表会講演論文集，（A-1）

[5] IBM: コンポジット構造図（https://www.ibm.com/support/knowledgecenter/ja/SS5JSH_9.1.2/com.ibm.xtools.modeler.doc/topics/ccompstruc.html）

[6] Jython:（https://www.jython.org/）