TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií

Editor konfiguračních souborů Flow123d

Diplomová práce

Studijní program: N2612 – Elektrotechnika a informatika
Studijní obor: 1802T007 – Informační technologie

Autor práce: **Bc. Tomáš Křížek**
Vedoucí práce: doc. Ing. Jiřina Královcová, Ph.D.

Liberec 2016

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics
and Interdisciplinary Studies

Flow123d configuration file editor

Master thesis

Study programme: N2612 – Electrotechnology and informatics
Study branch: 1802T007 – Information technology

Author: **Bc. Tomáš Křížek**
Supervisor: doc. Ing. Jiřina Královcová, Ph.D.

Liberec 2016

Tento list nahraďte

originálem zadání.

Prohlášení

Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu TUL.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé diplomové práce a konzultantem.

Současně čestně prohlašuji, že tištěná verze práce se shoduje s elektronickou verzí, vloženou do IS STAG.

Datum:

Podpis:

Poděkování

Rád bych poděkoval rodičům za podporu při studiu. Zároveň bych chtěl poděkovat i mé vedoucí diplomové práce, paní doc. Jiřině Královcové, a mému konzultantovi, panu Ing. Pavlu Richterovi, za vedení práce a konzultace.

Abstrakt

Tato diplomová práce popisuje problematiku související s tvorbou editoru konfiguračních souborů pro simulátor Flow123d. Představuje formát YAML jako náhradu staršího souborového formátu CON a popisuje použití jeho syntaxe. Dále práce detailně popisuje autokonverze a proces validace datové struktury. Naposled jsou popsány i samotné komponenty vzniklého editoru ModelEditor.

Klíčová slova

editor, YAML, autokonverze, validace, Flow123d

Abstract

This paper describes the issues surrounding the development of the Flow123d configuration file editor. It introduces the YAML format as a replacement of the previously used CON format and explains the new syntax. This paper also describes the process of autoconversion and validation of the Flow123d data structure in detail. Finally, components of the developed editor, ModelEditor, are described.

Keywords

editor, YAML, autoconversion, validation, Flow123d

Obsah

Úvod **12**

1 Problematika **14**
1.1 Simulátor Flow123d 14
1.2 Konfigurační soubory 15
1.3 Formát pro výměnu dat 18

2 Analýza **22**
2.1 Formát YAML 22
2.2 Specifikace formátu konfiguračních souborů 26
2.3 Autokonverze 30

3 Návrh **36**
3.1 Použití syntaxe YAML 36
3.2 Datová struktura 39
3.3 Autokonverze 40
3.4 Validace 44
3.5 Vizualizace datové struktury 50
3.6 Textový editor 51
3.7 Kontextová nápověda 52

4 Implementace a testování **53**
4.1 Požadavky 54
4.2 Zpracování formátu YAML 54
4.3 Textový editor 55
4.4 Kontextová nápověda 56
4.5 Testování 57

Závěr **58**

Literatura **59**

Přílohy **61**

A Ukázky kódů konfiguračních souborů **61**

B Grafické rozhraní aplikace **65**

C Obsah přiloženého CD **69**

Seznam obrázků

1 Simulátor Flow123d a pomocný software 15
2 Příklad složeného datového typu s různými typy atributů 17
3 Ukázky různých formátů pro zápis konfiguračního souboru 19

4 Zpracování formátu YAML 23
5 Autokonverze na jednorozměrné pole 31
6 Autokonverze na vícerozměrné pole 32
7 Autokonverze na záznam 33
8 Autokonverze – transpozice 34
9 Ukázka použití transpozice 35

10 Abstraktní záznam DarcyFlow a jeho implementace 37
11 Získání datové struktury z konfiguračního souboru 39
12 UML diagram datové struktury 40
13 Proces autokonverze I. 41
14 Proces autokonverze II. 42
15 Proces autokonverze III. 43
16 Validace datového typu 44
17 Validace číselných datových typů – *Integer* a *Double* 45
18 Validace výčtového datového typu – *Selection* 46
19 Validace pole – *Array* 47
20 Validace záznamu – *Record* 48
21 Validace abstraktního záznamu – *Abstract* 49

A.1 Výběr implementace abstraktního záznamu ve formátu CON 61
A.2 Výběr implementace abstraktního záznamu ve formátu YAML 61
A.3 Použití reference ve formátu CON 62
A.4 Použití reference ve formátu YAML 62
A.5 Konfigurační soubor `flow_time_dep` ve formátu CON 63
A.6 Konfigurační soubor `flow_time_dep` ve formátu YAML 64

B.1 GUI – Přehled rozhraní aplikace 65
B.2 GUI – Komponenta stromu pro vizualizaci dat 66
B.3 GUI – Komponenta textového editoru s notifikacemi 67
B.4 GUI – Komponenta seznamu notifikací 67
B.5 GUI – Komponenta textového editoru s doplňováním textu 67
B.6 GUI – Kontextová nápověda pro záznam *Steady_MH* 68
B.7 GUI – Kontextová nápověda pro záznam *SoluteTransport_DG* 68
B.8 GUI – Kontextová nápověda pro záznam *OutputStream* 68

Seznam zkratek

API Application Programming Interface.

CON C++ Object Notation.
CSS Cascading Style Sheets.

DTD Document Type Definition.

FSF Free Software Foundation.

GNU GNU's Not Unix.
GPL General Public License.

HTML HyperText Markup Language.

ISO International Organization for Standardization.
IST Input Structure Tree.

JS JavaScript.
JSON JavaScript Object Notation.

OMG Object Management Group.

PSF Python Software Foundation.

RCL Riverbank Computing Limited.

SGML Standard Generalized Markup Language.

TDD Test Driven Development.

UML Unified Modeling Language.

W3C World Wide Web Consorcium.

XML Extensible Markup Language.

YAML YAML Ain't Markup Language.

Úvod

Existuje celá řada aplikací, která potřebuje pro zajištění požadované funkce správné nastavení, podle kterého pak daná aplikace přizpůsobí svoji činnost. Může se jednat o počáteční konfiguraci, jako je tomu například u serverových aplikací nebo e-mailových klientů. Tato konfigurace se zpravidla dále nemění, pokud nedojde k nějakým podstatným změnám.

Oproti tomu existují aplikace, od kterých se očekává, že budou spouštěny s širokou škálou různých nastavení. U těchto aplikací se typicky konfigurace předává při spuštění jako jeden ze vstupních parametrů. Činnost těchto aplikací se pak zásadně liší dle zvolené konfigurace.

Takovou aplikací je například simulátor Flow123d, který se používá pro modelování procesů v horninovém prostředí. Vstupem do této aplikace je výpočetní síť společně se zadáním úlohy. Konkrétní úloha je definovaná pomocí konfiguračního souboru, který vytváří uživatel. Po zadání vstupních dat provede simulátor výpočty na dané síti a výsledky uloží do datového souboru, který je výstupem z aplikace.

Simulátor Flow123d podporuje různé typy úloh. Konfigurace jednotlivých úloh vyžaduje odlišné nastavení a může tedy dojít k tomu, že uživatelem zadaná konfigurace je nevalidní – například kvůli tomu, že definice dané úlohy neobsahuje některé povinné parametry a je tedy neúplná. V souboru může vzniknout i syntaktická chyba, která způsobí, že zadaná data nelze správně interpretovat.

Specifikace formátu konfiguračních souborů pro Flow123d, která popisuje strukturu vstupních dat, je poměrně rozsáhlá. Tištěná referenční příručka, která obsahuje tuto specifikaci formátu, má několik desítek stran. To klade na uživatele velké nároky. Pokud chce například ověřit, že byly zadány všechny povinné parametry, buď musí mít se softwarem rozsáhlé zkušenosti, nebo musí trávit velké množství času studiem této dokumentace a zkoumáním souvislostí mezi parametry.

Celá situace je dále komplikována tím, že formát konfiguračních souborů se mění s tím, jak se vyvíjí nové a upřesňují stávající funkcionality simulátoru Flow123d. Může dojít k tomu, že některé změny ve formátu konfiguračních souborů nemusí být zpětně kompatibilní. Uživatel tedy potřebuje znovu prostudovat rozsáhlou referenční

dokumentaci, aby zjistil, jakým způsobem zadat dříve realizovanou úlohu pro novou verzi Flow123d.

Pokud se stane, že uživatel spustí Flow123d s nevalidní konfigurací, potom během inicializace dojde k chybě, o které se uživatel dozví pomocí textového rozhraní, ve kterém se Flow123d spouští. Jelikož se může jednat o výpočetně náročné úlohy, které se často pouští na vzdáleném výpočetním clusteru, je tento proces poměrně časově náročný a uživatelsky nepříjemný.

Při vzdáleném spouštění Flow123d se úloha zařadí do fronty na výpočetním clusteru, kde dále čeká na přidělení zdrojů. Ty se přidělují na základě aktuálního vytížení. Buď jsou k dispozici okamžitě, nebo je nutné čekat na dokončení některých předchozích úloh. Může tedy nastat situace, kdy uživatel zařadí úlohu do fronty a poté čeká na výsledky několik hodin nebo dokonce dní, a teprve potom zjistí, že v konfiguračním souboru, který vytvořil, byla chyba. Kvůli tomu nemohlo dojít k inicializaci úlohy a tím pádem ani neproběhla simulace.

Tyto důvody byly hlavní motivací ke vzniku speciálního editoru pro konfigurační soubory Flow123d, který práci s nimi značně zjednoduší a usnadní. Tvorba takového editoru je předmětem této diplomové práce. Editor zrychlí proces odstranění chyb tím, že je odhalí už v průběhu vytváření nebo upravování konfiguračních souborů. Uživatel tedy bude moci chyby odstranit ještě před tím, než předloží konfigurační soubor simulátoru Flow123d. Tím dojde ke značné časové úspoře obzvlášť v případech, kdy se výpočetní úloha spouští vzdáleně.

Součástí editoru má být grafické uživatelské rozhraní. Jedním z jeho hlavních přínosů bude zjednodušení přístupu k dokumentaci. Uživatel bude mít k dispozici tu část dokumentace, která bezprostředně souvisí s právě upravovanou částí konfiguračního souboru. Tato forma nápovědy by měla uživateli poskytnout alternativu k prohledávání rozsáhlé referenční dokumentace.

Dále bude editor umožňovat zobrazit datovou strukturu, která tvoří konfigurační soubor. Kromě toho bude editor poskytovat základní funkce pro práci s textovými soubory, jako je podpora operací se schránkou, možnost vrátit či opakovat změny, vyhledávání či nahrazení textu a další. Editor má podporovat platformy Windows a Linux.

Vlastní text diplomové práce kromě úvodu a závěru je členěn do čtyř hlavních kapitol. První kapitola slouží jako úvod do problematiky. Ve druhé kapitole je podrobně analyzován problém. Třetí kapitola navazuje na předchozí kapitolu a představuje konkrétní návrh řešení rozebraných problémů. V poslední kapitole jsou zmíněny implementační detaily.

1 Problematika

1.1 Simulátor Flow123d

Flow123d je simulátor, který slouží k výpočtu proudění v porézním médiu, transportu látek nebo transportu tepla. Jedná se o aplikaci, která je orientována na práci s daty, a vzhledem k tomu neobsahuje žádné grafické uživatelské rozhraní. Uživatel tedy s aplikací pracuje v textovém režimu prostřednictvím terminálu, kde může aplikaci předat vstupní soubory a případně další parametry.

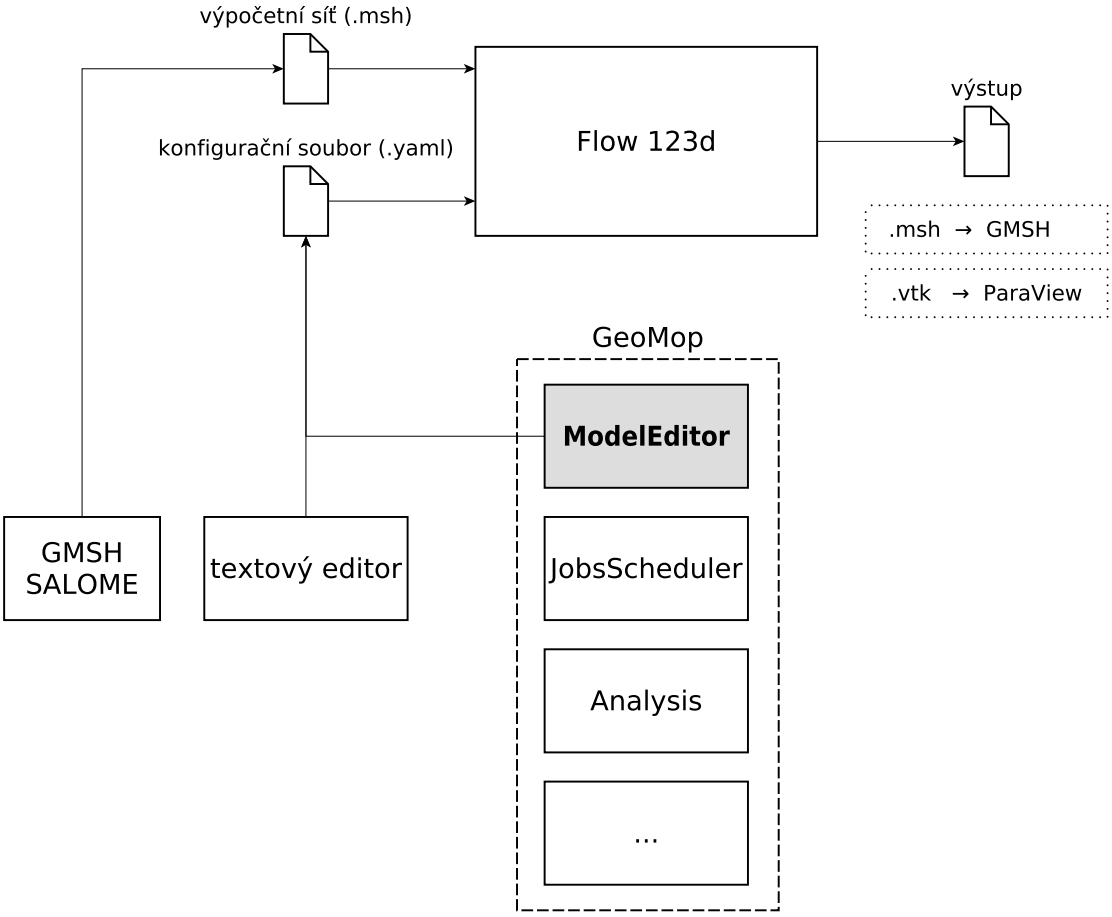
Na obrázku 1 jsou znázorněny vstupy a výstupy simulátoru Flow123d spolu s pomocnými aplikacemi, které uživatelé často používají. Vstupem do simulátoru Flow123d jsou dva soubory. První z těchto souborů popisuje výpočetní síť pomocí seznamu uzlů a elementů. Jedná se o textový soubor ve formátu `.msh`. Tuto síť generují softwary GMSH[1] nebo SALOME[2]. Druhým vstupním souborem je konfigurační soubor, který popisuje řešenou úlohu. Tento soubor si prozatím uživatelé tvořili sami pomocí obyčejných textových editorů.

Pro tento konfigurační soubor vzniká v rámci diplomové práce specializovaný editor s označením *ModelEditor*, který má oproti obyčejnému textovému editoru poskytnout např. validaci zadaných dat, zobrazení kontextové nápovědy nebo automatické doplňování textu. Vytváření a editace konfiguračních souborů se tak uživateli značně zjednoduší. *ModelEditor* je součástí softwarového balíku *GeoMop*, který obsahuje i další aplikace, které ovšem nejsou předmětem této práce.

GeoMop má sloužit jako nástroj, který usnadní práci se simulátorem Flow123d. Aplikace *JobsScheduler*, která je také součástí softwarového balíku *GeoMop*, bude zajišťovat vzdálené spouštění Flow123d na výpočetních clusterech. Softwarový balík *GeoMop* bude dále obsahovat aplikaci *Analysis*, která umožní parametrizaci úloh. *GeoMop* bude obsahovat i další aplikace, které jsou aktuálně ve vývoji. Je pravděpodobné, že se funkcionalita těchto aplikací bude dále rozšiřovat.

[1] `http://www.gmsh.info/`

[2] `http://www.salome-platform.org/`



Obrázek 1: Simulátor Flow123d a pomocný software

Výstupem ze simulátoru Flow123d je datový soubor, který obsahuje výsledky simulace. U tohoto souboru si uživatel může vybrat požadovaný formát, podle toho, kterou aplikaci chce použít pro zpracování výsledků. Typicky uživatelé používají buď opět GMSH nebo ParaView[3]. Existuje také celá řada jednoúčelových nástrojů, které si uživatelé často tvoří sami, nebo vznikají v rámci různých projektů.

1.2 Konfigurační soubory

V současné době (verze Flow123d 1.8.2), se pro zadání úlohy používá jeden konfigurační soubor, který obsahuje všechny potřebné údaje pro definici a inicializaci úlohy. Z pohledu Flow123d je úloha definovaná pomocí konkrétních objektů, které mají nastavené různé atributy na požadované hodnoty.

Vzhledem k tomu, že úlohy zadávají lidé, je potřeba určit nějaké společné roz-

[3] `http://www.paraview.org/`

hraní, pomocí kterého budou moci definovat tyto objekty a jejich obsah. Tato definice zároveň musí být strojově čitelná, aby ji simulátor Flow123d mohl zpracovat a nakonfigurovat se podle ní do správného počátečního stavu pro zahájení výpočtu.

Jelikož se pro předávání dat používají soubory, existují v principu dvě možnosti, jak předat tato data. Formát souboru může být buď binární, nebo textový. Vzhledem k tomu, že soubory mají vytvářet lidé, tak by bylo krajně nepraktické, kdyby se použil binární formát souboru.

Textová reprezentace konfiguračních souborů s sebou kromě čitelnosti přináší i další výhody. Oproti binárnímu formátu není závislá na architektuře, jelikož všechna data jsou kódována ve formě textu. Textový soubor navíc umožňuje kromě přenosu samotných dat i tato data nějakým způsobem popsat. Díky tomu se změny v interní struktuře Flow123d nemusí nutně projevit ve formátu konfiguračních souborů.

1.2.1 Datová struktura

Použití textového formátu konfiguračních souborů s sebou však přináší otázku, jakým způsobem tato data v textu reprezentovat. Je důležité, aby pomocí vybraného formátu bylo možné inicializovat libovolnou datovou strukturu. Tato datová struktura se skládá z objektů, které mají různé atributy. Každý atribut má název (dále označován jako klíč), datový typ a hodnotu. Ve většině případů platí, že klíč jednoznačně implikuje datový typ. Potom je tedy dostačující ukládat dvojici klíč a hodnota.

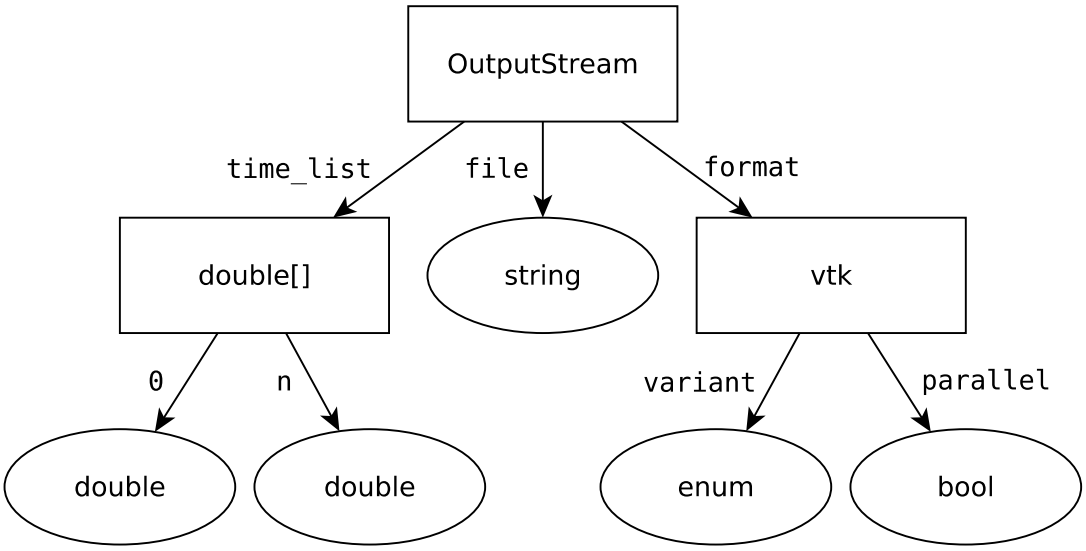
Existují i situace, kdy z názvu atributu nelze jednoznačně určit jeho datový typ. To je způsobené použitím polymorfismu. Z klíče lze tedy odvodit pouze jakého datového typu musí být předek. Pokud má tento předek více potomků, pak je nutné vybraný datový typ explicitně uvést. Tyto situace jsou v této kapitole zanedbány a jsou popsány samostatně v kapitole 3.1.2.

V konfiguračních souborech je tedy potřeba ukládat dvojice klíč a hodnota. Hodnota může být buď primitivního nebo složeného datového typu. Reprezentace primitivních datových typů je většinou triviální a spočívá pouze v převodu hodnoty na textový řetězec, pokud jím není. Podporované primitivní datové typy v rámci konfiguračních souborů jsou následující:

- booleovské hodnoty,
- celá čísla,
- desetinná čísla,
- hodnoty výčtového typu (tzv. enum),
- řetězce.

Složeným datovým typem z pohledu použitých konfiguračních souborů může být buď homogenní pole, nebo jiný objekt. Tím pádem vzniká hierarchická datová struktura, která může mít teoreticky nekonečný počet vnořených úrovní. V praxi je samozřejmě počet úrovní vždy konečný. Důležité ovšem je, aby použitý formát umožňoval reprezentovat libovolný počet vnoření.

Na obrázku 2 je znázorněna hierarchická struktura složeného datového typu *OutputStream*. Pro názornost jsou vynechány některé atributy. Datový typ *OutputStream* obsahuje řetězec `file`, což je primitivní datový typ. Dále obsahuje pole desetinných čísel `time_list`, které dále obsahuje konkrétní desetinná čísla. Posledním znázorněným atributem objektu *OutputStream* je `format`, který obsahuje referenci na objekt typu *vtk*. Objekt tohoto typu pak dále obsahuje atributy `variant` a `parallel`, což jsou primitivní datové typy.



Obrázek 2: Příklad složeného datového typu s různými typy atributů

1.3 Formát pro výměnu dat

1.3.1 Formát CON

Ve verzi Flow123d 1.8.2 se pro reprezentaci výše popsané datové struktury používá speciální formát C++ Object Notation (CON), který byl navržen vývojáři Flow123d pro účel zápisu konfiguračních souborů. Jedná se o formát, který vychází z JavaScript Object Notation (JSON), který je specifikován standardem ECMA-404 [1]. Oproti tomuto standardu se liší v několika detailech.

Příklad části konfiguračního souboru ve formátu CON je znázorněn na obrázku 3. Jednou z ihned zřejmých odlišností od formátu JSON je použití znaku „=“ místo „:“ pro oddělení klíče a hodnoty. Dále není nutné psát názvy klíčů do uvozovek. Další odlišnosti a kompletní specifikaci formátu CON lze nalézt v dokumentaci k Flow123d 1.8.2 [2].

Během používání tohoto formátu se ale jeho odlišnost od JSON projevila jako jeden z nedostatků. Kvůli nekompatibilitě s formátem JSON nelze použít pro zpracování formátu CON standardní knihovny. To je jeden z důvodů, proč bylo rozhodnuto, že ve verzi Flow123d 2.0 bude použit nějaký standardní formát pro výměnu dat.

To však nebylo jediným nedostatkem tohoto formátu. Uživatelé, kteří tento soubor upravovali, naráželi často na dva problémy. Bylo pro ně velice nepohodlné neustále kontrolovat správné uzávorkování objektů nebo zda na konci řádku nebyla vynechána čárka pro oddělení položek. To představovalo problém obzvlášť u rozsáhlejších konfiguračních souborů, které obsahují hodně úrovní vnoření. Jelikož formát JSON sdílí tyto nedostatky, tak byl zavržen jako možný nástupce formátu CON.

1.3.2 Formát XML

Extensible Markup Language (XML) je rozšiřitelný značkovací jazyk, který slouží pro popis dat. Tento jazyk vznikl z jazyka Standard Generalized Markup Language (SGML) [3], z kterého je odvozen i jazyk HyperText Markup Language (HTML). Všechny správně zformátované XML dokumenty jsou zároveň i SGML dokumenty. Popis a doporučení týkající se jazyka XML lze nalézt na webových stránkách World Wide Web Consorcium (W3C) [4].

Soubor ve formátu CON

```
output = {
  output_stream = {
    file = "./flow_test16.pvd",
    format = {
      TYPE = "vtk",
      variant = "ascii"
    },
    name = "flow_output_stream"
  },
  output_fields = [ "pressure_p0",
                    "pressure_p1",
                    "velocity_p0" ]
}
```

Soubor ve formátu XML

```
<output>
  <output_stream>
    <file>./flow_test16.pvd</file>
    <format type="vtk">
      <variant>ascii</variant>
    </format>
    <name>flow_output_stream</name>
  </output_stream>
  <output_fields>pressure_p0</output_fields>
  <output_fields>pressure_p1</output_fields>
  <output_fields>velocity_p0</output_fields>
</output>
```

Soubor ve formátu YAML

```
output:
  output_stream:
    file: ./flow_test16.pvd
    format: !vtk
      variant: ascii
    name: flow_output_stream
  output_fields:
    - pressure_p0
    - pressure_p1
    - velocity_p0
```

Obrázek 3: Ukázky různých formátů pro zápis konfiguračního souboru

Použití jazyka XML pro zápis konfiguračních souborů bylo jednou ze zvažovaných možností. Ukázku takového zápisu lze vidět na obrázku 3, na němž si lze zároveň všimnout odlišností zápisu formátu XML oproti formátům CON a YAML Ain't Markup Language (YAML).

Velkou výhodou tohoto jazyka je, že umožňuje nadefinovat si vlastní strukturu dokumentu. Lze tedy specifikovat kde se mohou vyskytnout jaké elementy, jaké mohou mít atributy a tak podobně. K tomu slouží Document Type Definition (DTD) nebo XML Schema, které má oproti DTD více funkcí, např. dokáže omezit počet výskytů elementů.

Použití XML Schema by velice usnadnilo validaci datové struktury a navíc existuje celá řada nástrojů, které jsou schopné ověřit, zda je daný XML dokument validní pro dané XML Schema. Úskalím použití této validace by však byly tzv. autokonverze, které jsou popsány v kapitole 2.3. Jedná se o speciální zápis, který lze použít v datové struktuře při zapisování polí nebo záznamů. V těchto speciálních případech lze místo pole či záznamu zapsat přímo hodnotu. To znemožňuje běžnou validaci pomocí XML Schema a bylo by nutné validaci XML upravit tak, aby byla schopná brát ohled na autokonverze. To znamená, že by bohužel nebylo možné použít univerzální nástroje pro validaci XML.

Nevýhodou formátu XML je jeho „výřečnost". Té si lze na první pohled všimnout na obrázku 3. Formát XML vyžaduje z uvedených možností pro zápis stejných dat nejvíce znaků. Hlavní nevýhodu v použití formátu XML pro zápis konfiguračních souborů ovšem viděli vývojáři Flow123d jinde. Dle jejich názoru by odlišnost v zápisu formátu XML od formátu CON byla pro uživatele Flow123d příliš velkou změnou.

1.3.3 Formát YAML

Poslední z uvažovaných možností bylo použití formátu YAML, který je zobecněním formátu JSON. Kterýkoliv JSON dokument je tím pádem i YAML dokumentem. Oproti formátu JSON ovšem formát YAML umožňuje syntaktický zápis, který byl speciálně navržen s ohledem na to, aby ho mohli jednoduše zapisovat lidé.

Toho si lze všimnout opět na obrázku 3. Ze všech uvedených možností je zápis v jazyce YAML nejkratší, a to jak počtem napsaných řádek, tak i počtem potřebných znaků. Zároveň i elegantně řeší problémy původně použitého formátu CON, resp. formátu JSON.

Jelikož se pro zápis vnořených dat používá odsazení, není nadále nutné používat závorky pro ohraničení záznamů a polí. Dále podporuje zápis polí pomocí odrážek, což je dobře čitelné a pohodlné pro zápis. Oproti formátu JSON také odpadá nut-

nost psaní čárek na koncích řádků pro oddělení klíčů v záznamech nebo položek v poli. Dále není nutné psát řetězce do uvozovek, protože se datové typy rozlišují implicitně.[4]

Všechna tato vylepšení vedou k tomu, že soubory zapsané ve formátu YAML jsou velice dobře čitelné a jednoduché pro zápis. Navíc se formát YAML od původního formátu CON neliší natolik, jako formát XML. To vedlo k rozhodnutí, že ve verzi Flow123d 2.0 bude použit jazyk YAML pro zápis konfiguračních souborů.

Aplikace editoru konfiguračních souborů vytvářená v rámci diplomové práce tedy bude pracovat s konfiguračními soubory napsanými ve formátu YAML. Aplikace bude zároveň podporovat i import dříve používaného formátu CON a jeho převod do formátu YAML.[5] Podpora exportu do formátu CON se neplánuje.

[4] V případě potřeby lze datový typ specifikovat i explicitně, viz kapitola 2.1.2.

[5] Import formátu CON je nad rámec této diplomové práce a v rámci projektu ho řešil Ing. Pavel Richter.

2 Analýza

Tato kapitola se skládá ze tří hlavních částí. V první z nich je popsán formát YAML, proces jeho zpracování a jeho základní syntaxe. Druhá část kapitoly rozebírá specifikaci konfiguračních souborů, která je stěžejní pro validaci datové struktury, generování kontextové nápovědy a funkci automatického doplňování textu. Poslední část popisuje autokonverze, což jsou speciální zkrácené typy zápisů, které se používají v konfiguračních souborech a mají zásadní dopad na zpracování datové struktury.

2.1 Formát YAML

Jak již bylo zmíněno v kapitole 1.3.3, pro zápis konfiguračních souborů se bude používat formát YAML. Jedná se o univerzální formát pro serializaci dat založený na kódování Unicode. Je navržen pro jednoduchý zápis polí, hashovacích tabulek a primitivních hodnot.

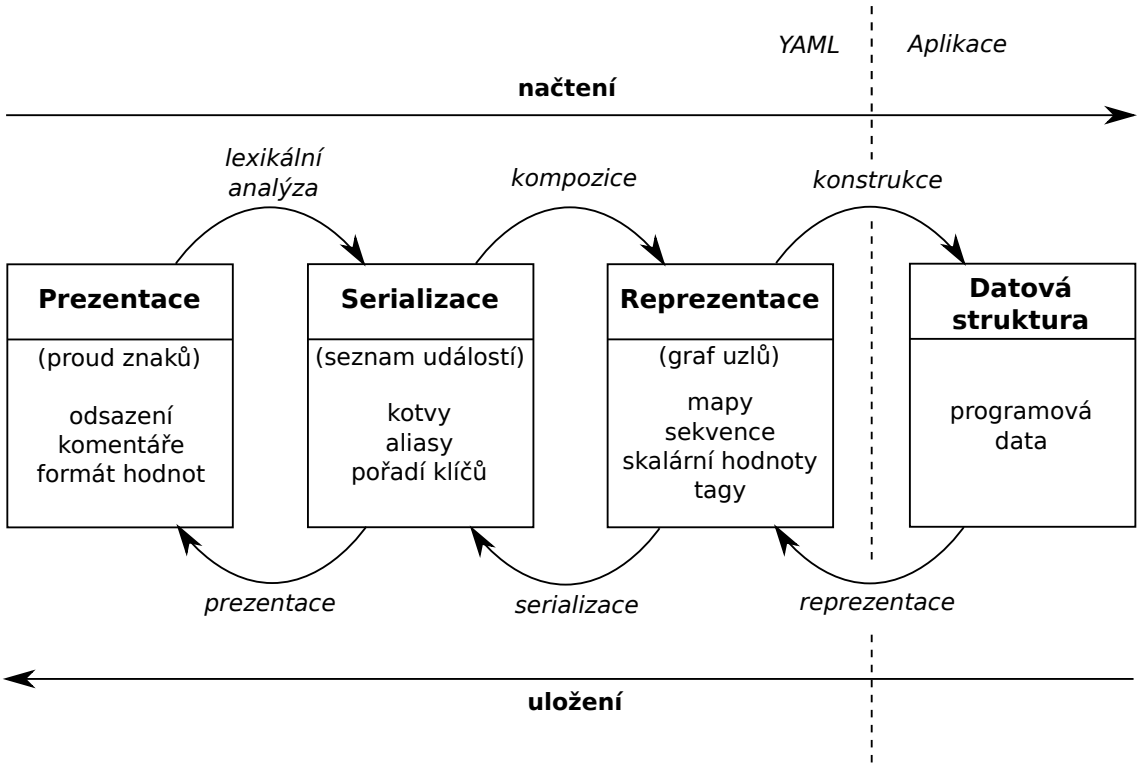
Mezi cíle formátu YAML patří mimo jiné jednoduchá čitelnost a přenositelnost, což jsou pro konfigurační soubory ideální vlastnosti. Dalším z cílů je čtení pomocí jediného průchodu, což s sebou oproti formátu CON přináší určité komplikace a změny, které jsou dále rozvedeny v kapitole 3.1.3.

2.1.1 Proces zpracování souboru

Jelikož je formát YAML navržen tak, aby k jeho přečtení a reprezentaci stačil jediný průchod, je možné ho zpracovávat dvěma způsoby. Buď je možné data zpracovávat proudově a generovat seznam událostí, nebo lze přečíst celý soubor a vytvořit z něj datovou reprezentaci.

Oficiální dokumentace formátu YAML [5] popisuje proces zpracování, ve kterém se využijí oba výše zmíněné způsoby, které se liší hlavně úrovní abstrakce. Na obrázku 4 lze vidět proces zpracování formátu YAML.

Proces načtení souboru ve formátu YAML prochází třemi fázemi. V první fázi proběhne lexikální analýza, která převede vstupní soubor na posloupnost událostí.



Obrázek 4: Zpracování formátu YAML

Lexikální analýza hledá symboly a hodnoty na základě syntaxe YAML, kontroluje správné odsazení a vypustí ze vstupu komentáře.

V druhé fázi proběhne kompozice, která převede posloupnost událostí na reprezentaci dat pomocí grafu. V této fázi se z dat odstraní kotvy a aliasy se nahradí příslušnými daty. Použití kotev a aliasů je dále vysvětleno v kapitole 3.1.3.

V poslední fázi se převede graf reprezentující data na datovou strukturu dané aplikace. Tato fáze představuje rozhraní mezi abstrakcí aplikace a procesem zpracování souboru v souborovém formátu YAML.

V každé fázi se ztratí část informace. Ve výsledné datové struktuře například nejsou žádné informace o jejich pozici v původním textovém souboru. Také nelze určit, zda byla konkrétní data v souboru zapsána, nebo zda vznikla pouhým odkazem na již existující data z jiné části souboru.

Formát YAML je takto navržen záměrně za účelem oddělení prezentace dat od jejich významu. Pro potřeby aplikace *ModelEditor* je ovšem tato vlastnost nežádoucí. Například informace o pozici hodnoty v konfiguračním souboru je podstatná a dále se využívá pro některé funkce. Touto problematikou se zabývá kapitola 4.2.

2.1.2 Zápis datových typů

Skalární hodnoty

Formát YAML podporuje několik skalárních datových typů, které jsou dále nedělitelné. Jedná se o celá a desetinná čísla, booleovské hodnoty a znakové řetězce. Zápis těchto hodnot je velice intuitivní.

Celá čísla se zapisují pomocí běžné konvence v desítkové soustavě. Dále je možné je zapsat v šestnáctkové nebo osmičkové soustavě pomocí prefixu `0x`, resp `0o`. Desetinná čísla se zapisují s desetinnou tečkou a dále je pro jejich zápis možné použít i vědeckou notaci. Booleovské hodnoty se zapisují jako `true` a `false`, kde první písmeno může být velké, případně mohou být velká i všechna písmena zároveň.

celá čísla ..... `0, 0o7, 0x3A, -19`
desetinná čísla ..... `0., -0.0, .5, +12e03, -2E+05, .inf, .NaN`
booleovské hodnoty ..... `true, True, false, FALSE`

Řetězce se zapisují jako sekvence znaků, které není třeba psát do uvozovek ani pokud obsahují mezery nebo zalomení řádků. Pokud by řetězec obsahoval speciální znak, jako je třeba dvojtečka, která se používá na oddělení klíče od hodnoty, tak musí být řetězec napsán do uvozovek.

Všechny tyto zápisy skalárních hodnot jsou popsány v dokumentaci YAML [5] pomocí regulárních výrazů. Během fáze kompozice se na základě těchto regulárních výrazů určí implicitní datový typ proměnné, který se použije, pokud není explicitně zadán jiný datový typ. Datový typ lze explicitně zadat pomocí tzv. tagu. Dokumentace YAML definuje následující tagy pro skalární datové typy.

celá čísla ..... `!!int`
desetinná čísla ..... `!!float`
booleovské hodnoty ..... `!!bool`
řetězec ..... `!!str`

Pokud je potřeba explicitně zadat datový typ, píše se tento tag před samotnou hodnotu a odděluje se od ní mezerou. Následuje seznam ukázek zápisu hodnot a příslušných datových typů, na které budou převedeny.

`0` ..... *celé číslo*
`!!float 0` ..... *desetinné číslo*
`"0"` ..... *řetězec*
`!!str 0` ..... *řetězec*

`true` ..... *booleovská hodnota*

`!!str true` ..... *řetězec*

Sekvence

Sekvence se používají pro reprezentaci polí. Formát YAML podporuje dva způsoby zápisu sekvencí. Lze je zapsat buď zjednodušeně[1] nebo blokově pomocí odrážek. Zjednodušený zápis sekvencí je totožný se zápisem polí ve formátu JSON a vypadá následovně.

```
[1, 2, 3]
```

Pro blokový zápis se používá znak `-` (spojovník) jako odrážka, která musí být následována alespoň jednou mezerou a poté samotnou položkou sekvence. Každá odrážka musí být na samostatném řádku a odsazení všech položek sekvence musí být stejné. Zápis sekvence pomocí odrážek může vypadat například následovně.

```
- 1
- 2
- 3
```

V rámci jednoho dokumentu lze používat oba typy zápisů sekvencí. V rámci jedné sekvence však musí být dodržena stejná syntaxe. Sekvence lze do sebe vnořovat do libovolné úrovně.

Mapy

Mapy se ve formátu YAML používají pro zápis dvojic klíč a hodnota – jedná se tedy o hashovací tabulku. Mapy lze obdobně jako sekvence zapsat buď zjednodušeně nebo blokově. Zjednodušený zápis se opět podobá formátu JSON.

```
{a: 1, b: 2}
```

Blokový zápis vyžaduje, aby každý klíč byl na novém řádku. Při použití blokového zápisu opět závisí na odsazení, které musí být jednotné v rámci celé mapy. Blokový zápis může vypadat například následovně.

```
a: 1
b: 2
```

V rámci dokumentu lze opět používat oba typy zápisů map a v rámci jedné mapy musí být dodržena jednotná syntaxe. Mapy i seznamy lze do sebe vzájemně vnořovat do libovolné úrovně.

[1] Angl. *flow style* – zde označován jako *zjednodušený zápis*.

2.2 Specifikace formátu konfiguračních souborů

Simulátor Flow123d umožňuje exportovat popis datové struktury konfiguračních souborů. Tento popis struktury bude v textu nadále označován jako *specifikace formátu*. Oproti tomu slovo *formát* udává použitou textovou reprezentaci pro zápis konfiguračního souboru – nejčastěji tedy označuje buď souborový formát YAML nebo starší formát CON.

Zatímco formát udává syntaktická pravidla pro vytváření konfiguračních souborů, specifikace formátu popisuje sémantický význam zapsaných dat, definuje použitelné uživatelské typy a klade různá omezení na vstupní datovou strukturu. Formát a specifikaci formátu lze považovat za analogii k XML, resp. XML Schema.

Mezi vývojáři Flow123d se specifikace formátu také označuje jako Input Structure Tree (IST). S tím, jak se vyvíjí nová a rozšiřuje stávající funkcionalita simulátoru Flow123d, se mění i tato specifikace formátu. Z toho plyne zásadní požadavek na validaci konfiguračních souborů – **proces validace musí být možné přizpůsobit konkrétní specifikaci formátu, která je závislá na verzi Flow123d**.

Specifikace formátu je generována simulátorem Flow123d na základě jeho interní datové struktury. Reprezentace této struktury je exportována do formátu JSON. Tato specifikace formátu obsahuje kromě požadavků na datovou strukturu i dokumentační řetězce, ze kterých se vytváří referenční dokumentace. Specifikace formátu je tedy úplným popisem datové struktury konfiguračních souborů, ze kterého vychází hlavní funkce *ModelEditoru*, jako je validace konfiguračních souborů, generování kontextové nápovědy nebo automatické doplňování textu.

2.2.1 Základní datové typy

Jak již bylo zmíněno v kapitole 1.2.1, konfigurační soubory obsahují hierarchickou datovou strukturu, která tvoří strom. Specifikace formátu definuje datový typ pro každý uzel stromu, ať už se jedná o interní uzel nebo list stromu. Datové typy definované ve specifikaci formátu mohou mít následující parametry:

`id` ..... unikátní řetězec označující datový typ
`input_type` .... základní typ, ze kterého je datový typ odvozen
`name` ..... název datového typu (nemusí být unikátní)
`description` ... dokumentační řetězec popisující datový typ
`attributes` .... pomocné atributy obsahující dodatečné informace

Základní typ `input_type` udává pouze typ, ze kterého je konkrétní datový typ odvozen. Konkrétní datové typy obsahují výše vyjmenované informace a případná další upřesnění či omezení. Specifikace formátu může takových datových typů definovat neomezený počet. Oproti tomu je možné využít pouze následující základní datové typy:

- *Integer* pro celá čísla,
- *Double* pro reálná čísla,
- *String* pro řetězce,
- *Filename* pro jména souborů,
- *Selection* pro výčtové typy,
- *Array* pro homogenní pole,
- *Record* pro záznamy,
- *Abstract* pro abstraktní záznamy.

2.2.2 Primitivní datové typy

Mezi tyto datové typy patří všechny typy odvozené z *Integer*, *Double*, *String*, *Filename* nebo *Selection*. Jedná se o datové typy, jejichž hodnota je z pohledu datové struktury konfiguračního souboru dále nedělitelná. Ve stromové datové struktuře se tedy vždy jedná o listy stromu.

Datové typy odvozené z *Filename* a *String* nemají v současné verzi Flow123d žádné další parametry, které by omezovaly jejich možné hodnoty (např. omezení délky řetězce). Datové typy, které slouží pro reprezentaci čísel, tedy typy odvozené od *Integer* a *Double*, mohou mít navíc omezený rozsah povolených hodnot pomocí zadání intervalu `range`.

`range` . . . . . dvojice čísel, která vymezuje rozsah platných hodnot

Datové typy odvozené z typu *Selection* slouží pro specifikaci výčtového typu. Tyto typy mají pevně definovanou množinu platných hodnot, která je zadána pomocí parametru `values`. Každý záznam v tomto poli je definován pomocí jejich názvu `name` a popisu `description`.

`values` . . . . . seznam přípustných hodnot včetně jejich popisu

2.2.3 Pole

Všechna pole ve specifikaci datové struktury jsou homogenní – obsahují tedy prvky, které jsou stejného datového typu (případně jsou odvozeny ze stejného abstraktního datového typu, viz dále). Datový typ potomků pole definuje parametr `subtype`, který obsahuje identifikátor datového typu. Pole dále mohou obsahovat omezení minimálního a maximálního počtu prvků, zadaného pomocí `range`.

`range` ..... dvojice čísel udávající minimální a maximální počet prvků
`subtype` ..... datový typ jednotlivých prvků pole

2.2.4 Záznamy

Záznamy slouží pro inicializaci tříd a jejich atributy tvoří dvojice klíč a hodnota. Každý datový typ odvozen z typu *Record* má v rámci specifikace formátu definován množinu klíčů `keys`. Dále mohou mít záznamy definován parametr `reducible_to_key`, který se používá pro autokonverzi na záznam (viz kapitola 2.3.2).

`keys` ..... definuje množinu klíčů daného záznamu
`reducible_to_key` ... obsahuje název klíče použitého pro autokonverzi

Definice každého klíče dále obsahuje další parametry. Mezi ně patří `key`, který udává název klíče, poté opět popis `description`, datový typ klíče `type` a parametr `default`.

`key` ..... název klíče
`description` ..... dokumentační řetězec popisují klíč záznamu
`type` ..... identifikátor očekávaného datového typu klíče
`default` ..... upřesňuje typ klíče a případně jeho výchozí hodnotu

Parametr `default` se skládá z dvojice parametrů `type` a `value`. Parametr `value` udává výchozí hodnotu klíče. Tato hodnota se použije v případě, že se v konfiguračním souboru nevyskytne daný klíč. Parametr `type` udává, zda je potřeba klíč explicitně zadat či nikoliv a může nabývat následujících hodnot.

obligatory ..... klíč je povinný a jeho hodnota musí být explicitně zadána
optional ..... klíč je nepovinný a nemá žádnou výchozí hodnotu
value at declaration .. výchozí hodnota je součástí deklarace typu
value at run time .... výchozí hodnota se načte při spuštění

Z pohledu aplikace *ModelEditor* je podstatná pouze hodnota *obligatory*, která udává, že klíč musí být v konfiguračním souboru uveden. Ostatní typy klíčů nemusí být v konfiguračním souboru explicitně uvedeny. Pokud se však objeví, provede se jejich validace jako stejně jako u ostatních klíčů.

2.2.5 Abstraktní záznamy

Posledním speciálním typem, který se používá ve struktuře konfiguračních souborů je abstraktní záznam. Jedná se o všechny typy, které jsou odvozené ze základního typu označovaného jako *Abstract*. Abstraktní záznamy slouží jako zástupný typ, který se ve specifikaci formátu datové struktury objevuje na místech, kde se mohou vyskytovat různé typy (ovšem ne zcela libovolné).

Jeden z příkladů využití je při výběru typu simulovaného procesu. Existuje několik typů procesů, které Flow123d dokáže řešit. Jednotlivé procesy se však liší a tím pádem i vyžadují jiné parametry a data mají odlišnou strukturu. V místě, kde se zadává proces očekává specifikace formátu abstraktní záznam. V konfiguračním souboru se na daném místě uvede konkrétní záznam, který implementuje očekávaný abstraktní záznam. U těchto záznamů je ve specifikaci formátu uvedeno, které abstraktní záznamy implementují pomocí klíče `implements`.

Použitá terminologie se rozchází s programátorskou terminologií, kde by se tento abstraktní záznam označil spíše jako rozhraní. Abstraktní záznamy totiž nemohou mít definované žádné klíče – tím pádem se nepoužívají pro dědičnost. Navíc jeden konkrétní typ může implementovat více abstraktních typů, což je z pohledu objektového programování typické spíše pro rozhraní. Pro zachování jednotné terminologie se specifikací formátu Flow123d a její dokumentací bude ale nadále tento typ označován jako abstraktní záznam.

`default_descendant` ..... výchozí typ abstraktního záznamu
`implementations` ..... seznam implementací abstraktního záznamu

Jak již bylo řečeno, oproti záznamům neobsahují abstraktní záznamy žádnou definici klíčů. Obsahují však navíc parametr `default_descendant`, který udává identifikátor datového typu, který se použije v případě, že záznam, který je uveden na místě, kde se očekává abstraktní záznam, nemá explicitně uvedený typ. Abstraktní záznam navíc obsahuje parametr `implementations`, který obsahuje seznam identifikátorů všech jeho implementací. Jedná se o duplicitní informaci a v rámci aplikace *ModelEditor* není využita, protože se využívá výše zmíněný parametr `implements`.

2.3 Autokonverze

V konfiguračních souborech pro Flow123d se používají speciální zápisy záznamů a polí, které vývojáři a uživatelé Flow123d označují jako automatické konverze, nebo zkráceně pouze autokonverze. Cílem autokonverzí je zjednodušení a zkrácení zápisu.

Pomocí autokonverzí lze například vynechat definici typu a klíče a místo toho zapsat pouze hodnotu. Zápis se tím výrazně zkrátí, ale zároveň to způsobí značné komplikace z pohledu validace konfiguračního souboru nebo určení přesné polohy v rámci datové struktury. Situace se dále komplikuje tím, že je možné použít vnořené autokonverze – tedy provést autokonverzi v rámci jiné autokonverze.

Logika autokonverzí je zhruba následující. Pokud datový typ neodpovídá očekávanému datovému typu dle specifikace formátu, zkusí se provést některá z autokonverzí, pokud je na tomto místě použitelná. V případě, že po autokonverzi již datový typ odpovídá očekávanému datovému typu, tak se datová struktura dále prochází do hloubky, kde může dojít k případným dalším autokonverzím.

Autokonverze jsou hlavní důvodem, proč není možné použít XML Schema a souborový formát XML pro validaci konfiguračních souborů. Kvůli autokonverzím by nebylo možné pouze triviálně použít nástroj pro validaci XML schématu k validaci konfiguračního souboru.

Autokonverze mohou působit poněkud chaoticky, protože datová struktura obsahuje ve skutečnosti něco jiného, než je zapsáno v konfiguračním souboru. Zkušení uživatelé Flow123d jsou ovšem zvyklí autokonverze používat a jsou pro ně přínosem. S přechodem na nový formát konfiguračního souboru se funkcionalita autokonverzí zachovává.

Aplikace *ModelEditor* tedy musí podporovat používání autokonverzí a je nutné je zohlednit při validaci konfiguračních souborů a určování pozice v rámci datového stromu, která je dále použita pro funkce kontextové nápovědy nebo automatického doplňování textu.

Používají se tři základní typy autokonverzí, které jsou dále popsány. Kromě doposud používaných autokonverzí na pole a na záznam přibyla i nová autokonverze, která se nazývá transpozice. Transpozice zobecňuje a rozšiřuje současně používanou autokonverzi na pole.

2.3.1 Autokonverze na pole

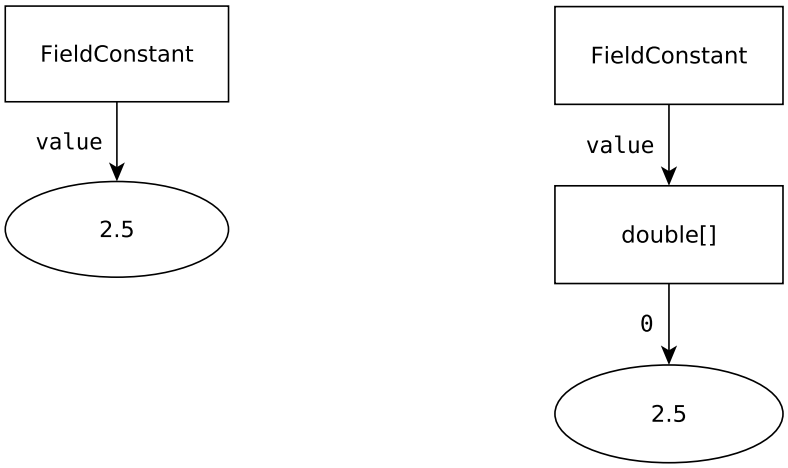
Autokonverze na pole je konverze, která se použije, pokud se dle specifikace formátu na dané pozici v datové struktuře očekává pole (typ *Array*) a místo něj je nalezen libovolný jiný datový typ – ať už se jedná o primitivní datový typ nebo o záznam.

Zapsaná datová struktura

```
value: 2.5
```

Skutečná datová struktura

```
value: [2.5]
```



Obrázek 5: Autokonverze na jednorozměrné pole

Obrázek 5 znázorňuje autokonverzi hodnoty na jednorozměrné pole. V klíči `value` v záznamu *FieldConstant* se očekává jednorozměrné pole, ale místo něj se na dané pozici vyskytla hodnota `2.5`. Provede se tedy autokonverze na pole a poté skutečná datová struktura obsahuje jednorozměrné pole o jednom prvku, kterým je právě tato hodnota.

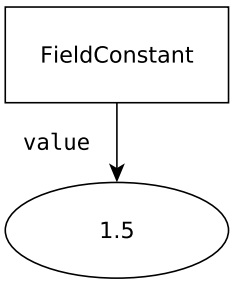
Autokonverze na pole funguje obdobně i pro vícerozměrná pole. Příklad této autokonverze je uveden na obrázku 6. Klíč `value` v záznamu *FieldConstant* má nyní být dvourozměrné pole.[2] Místo toho se v datové struktuře vyskytla hodnota. Ta se zkonvertuje na dvourozměrné pole, jehož jediným prvek je právě tato hodnota.

Autokonverze na vícerozměrné pole proběhne i v případě, pokud je v datové struktuře pole o menší dimenzi, než se očekává. Pokud takový případ nastane, tak se zadané pole zkonvertuje na pole vyšší dimenze obdobně jako to bylo popsáno u konverzí hodnot na pole.

[2]Přes to, že tento záznam má opět název *FieldConstant*, jedná se o odlišný typ záznamu od předchozího příkladu. První z nich byl implementací abstraktního záznamu *Field:R3* $\rightarrow$ *R*, zatímco druhý implementuje *Field:R3* $\rightarrow$ *R[3,3]*.

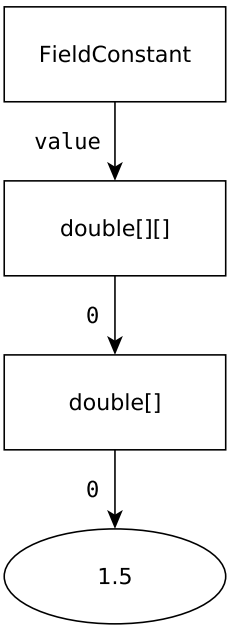
Zapsaná datová struktura

```
value: 1.5
```



Skutečná datová struktura

```
value: [[1.5]]
```



Obrázek 6: Autokonverze na vícerozměrné pole

2.3.2 Autokonverze na záznam

Autokonverze na záznam se provádí v případě, že se v datové struktuře na dané pozici očekává záznam[3] a místo něj se v datové struktuře nachází pole či primitivní datový typ. Oproti autokonverzi na pole, která proběhne vždy, musí být autokonverze na záznam explicitně povolena ve specifikaci formátu pro daný záznam.

Záznam musí mít ve specifikaci formátu uvedený parametr `reducible_to_key`, který udává, do kterého klíče se má nalezená datová struktura vložit. Pokud tento parametr není uveden, tak záznam autokonverzi nepodporuje a nelze ji provést.

Příklad autokonverze na záznam je znázorněn na obrázku 7. Záznam *Steady_MH* obsahuje klíč `balance`. V zapsané datové struktuře je pro tento klíč uvedena hodnota `true`, tedy datový typ *Boolean*. Ovšem podle specifikace formátu se zde očekává záznam typu *Balance*. Jelikož má datový typ *Balance* uveden klíč `reducible_to_key`, který je nastaven na hodnotu `balance_on`, tak dojde k autokonverzi na záznam. Ve skutečné datové struktuře je tedy v klíči `balance` záznam typu *Balance*, jehož klíč `balance_on` je nastaven na původně uvedenou hodnotu `true`.

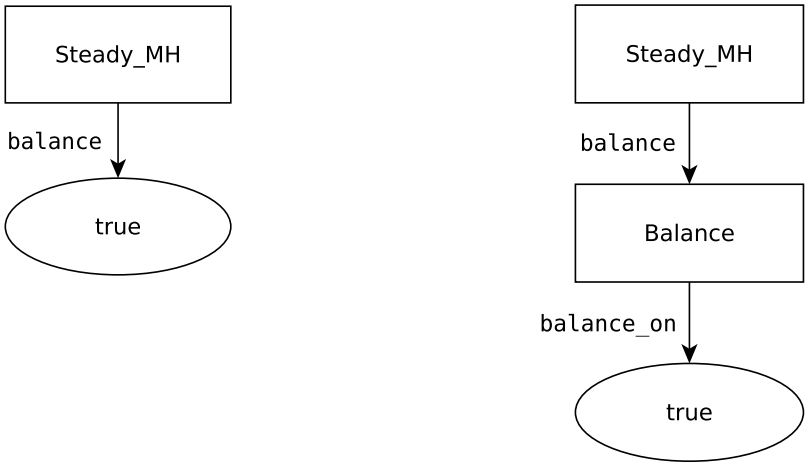
[3]Záznam je datový typ, který je odvozený ze základního typu *Record* nebo *Abstract*, viz dále.

Zapsaná datová struktura

```
balance: true
```

Skutečná datová struktura

```
balance:
  balance_on: true
```



Obrázek 7: Autokonverze na záznam

U záznamů, které podporují tuto autokonverzi, jsou typicky předdefinované výchozí hodnoty klíčů. Uživatel přitom nejčastěji mění právě jeden z těchto klíčů. Tento klíč je pak ve specifikaci formátu uveden jako `reducible_to_key`. Díky autokonverzi na záznam pak uživatel může měnit hodnotu tohoto klíče bez nutnosti psát název tohoto klíče.

Existuje speciální případ této autokonverze, kdy se podle specifikace formátu očekává v datové struktuře abstraktní záznam. Abstraktní záznamy samy o sobě nepodporují autokonverzi na záznam. Pokud ovšem mají definovaný klíč `default_descendant`, který udává výchozí typ záznamu, je možné provést autokonverzi, pokud ji daný záznam podporuje.

2.3.3 Transpozice

Transpozice je nově přidaná autokonverze, která umožňuje definovat pole záznamů najednou pomocí speciálního zápisu pro jeden záznam. Transpozice může výrazně zkrátit zápis pole záznamů obzvlášť v případech, kdy záznamy obsahují více klíčů a mění se pouze hodnoty některých klíčů, zatímco hodnoty ostatních klíčů zůstávají totožné.

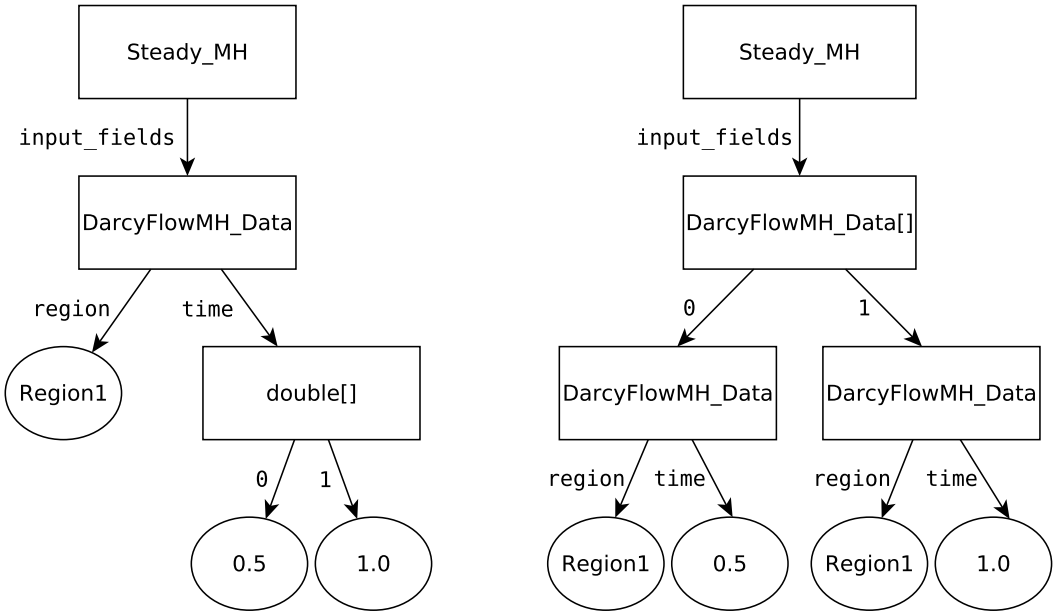
Transpozice je speciálním případem autokonverze na pole. Provádí se v případě, kdy se v datové struktuře očekává pole záznamů, ale místo něj je nalezen pouze

Zapsaná datová struktura

```
input_fields:
  region: Region1
  time:
    - 0.5
    - 1.0
```

Skutečná datová struktura

```
input_fields:
  - region: Region1
    time: 0.5
  - region: Region1
    time: 1.0
```



Obrázek 8: Autokonverze – transpozice

jeden záznam. Pokud nastane tato situace, proběhnou další kontroly, zda je možné transpozici provést (viz kapitola 3.3), a pokud je to možné, tak se zapsaná datová struktura převede následujícím způsobem.

Nejprve se vytvoří pole záznamů, které se očekávalo podle specifikace formátu. Dále se provede expanze polí, kdy se pro každou hodnotu v poli původně zapsaného záznamu vytvoří samostatný záznam, který se uloží do nově vytvořeného pole záznamů. Během této expanze se do nově vytvořených záznamů vkládají konkrétní hodnoty místo původních polí hodnot. Pokud v daném klíči v původním záznamu nebylo pole, ale hodnota či jiný záznam, tak se tato hodnota zkopíruje.

Nejjednodušší případ této konverze je znázorněn na obrázku 8. Zde se pomocí transpozice definuje pole `input_fields`, které obsahuje záznamy typu *Steady_MH*. Tento záznam obsahuje klíče `region` a `time`. Zapsaná datová struktura se transponuje tak, že se místo jediného uvedeného záznamu vytvoří pole záznamů o dvou prvcích, protože pole `time` obsahuje dvě hodnoty. Každý z těchto záznamů bude mít

klíč `time` i `region`. U prvního záznamu bude klíč `time` nastaven na hodnotu `0.5`, zatímco u druhého záznamu bude mít klíč `time` hodnotu `1.0`. Klíč `region` bude v obou záznamech nastaven na totožnou hodnotu `Region1`.

Pokud zapsaný záznam obsahuje více polí, tak jejich expanze probíhá souběžně. První záznam bude tedy obsahovat vždy první prvky z polí, druhý záznam druhé prvky z polí atd. To je znázorněno na obrázku 9. Dále si lze všimnout, že vhodně použitá transpozice může výrazně zkrátit zápis.

Zapsaná datová struktura

```
input_fields:
  region: MyRegion
  bc_type: neumann
  time: [0.5, 1.0, 1.5]
  anisotropy: [2.5, 1.5, 1.8]
```

Skutečná datová struktura

```
input_fields:
  - region: MyRegion
    bc_type: neumann
    time: 0.5
    anisotropy: 2.5
  - region: MyRegion
    bc_type: neumann
    time: 1.0
    anisotropy: 1.5
  - region: MyRegion
    bc_type: neumann
    time: 1.5
    anisotropy: 1.8
```

Obrázek 9: Ukázka použití transpozice

3 Návrh

3.1 Použití syntaxe YAML

V této kapitole je upřesněno použití syntaxe YAML pro zápis konfiguračních souborů včetně speciálních případů, které se používaly ve formátu CON, jako jsou specifikace typu záznamu a použití referencí. Dále je popsán proces získání datové struktury z konfiguračního souboru, který zahrnuje provedení autokonverzí. Kapitola se dále zabývá validací načtené datové struktury. V poslední části kapitoly jsou navrženy základní komponenty uživatelského rozhraní a je popsána jejich funkce.

3.1.1 Základní použití syntaxe

Pro reprezentace primitivních datových typů, polí a záznamů v konfiguračních souborech lze využít základní syntaxi formátu YAML popsanou v kapitole 2.1.2. Primitivní datové typy lze zapsat s využitím syntaxe pro skalární hodnoty, pole je možné reprezentovat pomocí syntaxe pro sekvence a záznamy lze zapsat pomocí syntaxe pro mapy.

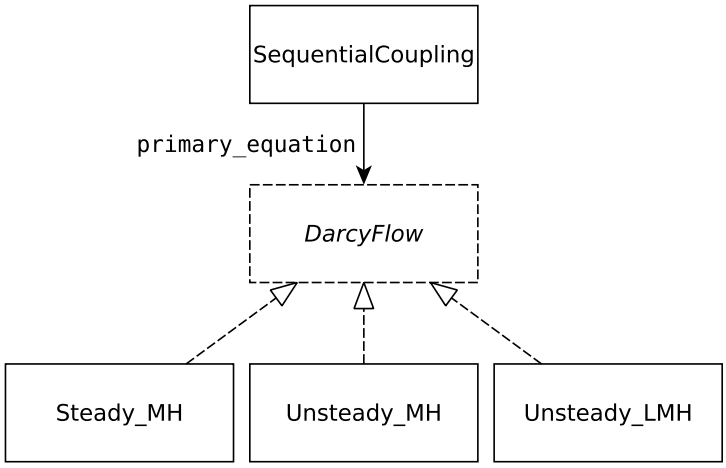
Pro reprezentaci polí se doporučuje použít syntaxe pro blokový zápis, kromě případů, kdy je zjednodušený zápis výrazně kratší bez újmy na čitelnosti (např. zápis pole celočíselných hodnot). Pro reprezentaci záznamů se doporučuje používat výhradně syntaxe pro blokový zápis. Zjednodušený zápis záznamů působí chaoticky a editor nemusí podporovat některé funkce při používání tohoto zápisu.

Formát YAML nepředepisuje, jaké by mělo být odsazení při vnořování blokových zápisů, ale pouze vyžaduje, aby bylo v rámci daného bloku jednotné. V rámci konfiguračních souborů Flow123d se ale doporučuje používat odsazení pomocí dvou mezer, aby byl styl použitý pro zápis konfiguračních souborů jednotný. Editor je optimalizován na použití odsazení pomocí dvou mezer a znaky tabulátor automaticky převádí na toto odsazení.

3.1.2 Abstraktní záznamy

Abstraktní záznamy jsou datovým typem, který definuje specifikace formátu. Tyto záznamy mohou mít více různých implementací. V konfiguračním souboru, kde se podle specifikace formátu očekává abstraktní záznam, musí být uvedena konkrétní implementace. Implementace abstraktního záznamu je záznam, který má v jeho specifikaci uvedeno, že implementuje daný abstraktní záznam. To bylo popsáno v kapitole 2.2.5.

Příkladem je třeba klíč `primary_equation` v datovém typu *SequentialCoupling*. Typ klíče `primary_equation` je *DarcyFlow*, což je abstraktní záznam. V současné verzi Flow123d existují tři implementace tohoto záznamu – *Steady_MH*, *Unsteady_MH* a *Unsteady_LMH*. To znázorňuje obrázek 10.



Obrázek 10: Abstraktní záznam DarcyFlow a jeho implementace

V konfiguračním souboru je nutné nějakým způsobem určit, jaká implementace byla zvolena. Formát CON toto řešil pomocí speciálního klíče `TYPE`. Tento klíč obsahoval výběrový typ, který mohl nabývat hodnot, které odpovídaly názvům jednotlivých implementací. Nevýhoda tohoto řešení byla, že klíč `TYPE` byl uveden v záznamu jako běžný klíč i přes to, že pouze určoval datový typ záznamu a měl jiný význam než ostatní klíče. Ukázka výběru konkrétní implementace ve formátu CON je na obrázku A.1 na straně 61.

Ve formátu YAML existují takzvané tagy, které slouží pro určení datového typu. Jsou definovány některé základní tagy, které se používají pro odlišení typů, které jsou univerzální pro všechny YAML dokumenty. Tyto předdefinované tagy byly popsány v kapitole 2.1.2 a jejich kompletní popis lze nalézt v dokumentaci formátu YAML [5]. Všechny předdefinované tagy začínají `!!` (dvěma vykřičníky).

Lze ovšem využít i uživatelsky definované tagy. Ty začínají pouze jedním znakem `!` (vykřičník) a poté jsou následovány uživatelsky definovaným názvem. Zpracování těchto tagů je potom aplikačně specifické a řeší se v rámci procesu načtení YAML dokumentu. Použití uživatelsky definovaného tagu může vypadat následovně.

```
solver: !Petsc
  a_tol: 1e-12
  r_tol: 1e-12
```

Uživatelsky definované tagy jsou ideální pro výběr konkrétní implementace abstraktního záznamu. Jejich zápis je jednoznačně odlišuje od ostatních dat v záznamu a nejsou potom součástí samotných dat záznamu. Na obrázcích A.1 a A.2 na straně 61 lze porovnat způsob výběru implementace abstraktního záznamu ve formátech CON a YAML.

3.1.3 Reference

Ve formátu CON bylo možné se pomocí referencí odkázat na libovolnou část datové struktury. Tento odkaz byl pak při zpracování nahrazen daty, na které odkazoval. Dalo se tak předejít duplicitě dat. Pokud byla některá data totožná s daty v jiné části datové struktury, stačilo použít referenci.

Reference se ve formátu CON používaly tak, že se do speciálního klíče `REF` uvedla libovolná cesta do jiné části datové struktury. Ta mohla být buď absolutní nebo relativní a mohla být v libovolné části konfiguračního souboru. Ukázka použití reference ve formátu CON je na obrázku A.6 na straně 64.

Formát YAML poskytuje podobnou funkcionalitu v podobě kotev a aliasů. V libovolné části souboru je možné definovat tzv. kotvu, na kterou je pak možné se odkázat ve zbývající části souboru pomocí aliasu. Kotva začíná znakem `&` (ampersand) a za ním následuje její název, který si uživatel může zvolit. Alias začíná znakem `*` (hvězdička) a za ní je bezprostředně uveden název již existující kotvy. Definice kotvy a použití aliasu může vypadat například následovně.

```
primary_equation: !Unsteady_LMH
  time: &time
    end_time: 0.5
    max_dt: 0.01
    min_dt: 0.01
time: *time
```

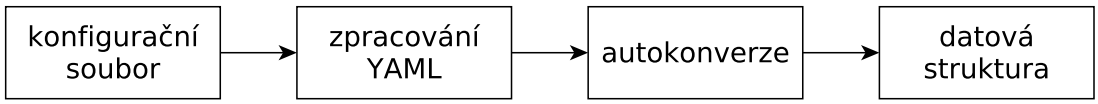
Použití kotev a aliasů bylo navrženo ve formátu YAML za stejným účelem jako reference ve formátu CON – ke zrychlení zápisu duplicitních dat. Formát YAML byl ovšem navržen tak, aby byl schopný zpracovávat i proud dat, nikoli pouze kompletní soubory. To vedlo k tomu, že kotvy je nutné vždy definovat dříve, než je možné je použít jako alias.

Tím se použití kotev a aliasů liší od referencí ve formátu CON. V něm bylo možné se odkázat i na datovou strukturu, která v dané fázi zpracování souboru ještě nebyla definovaná. Nicméně vzhledem k účelu referencí – předejití duplicitě dat – není tato funkcionalita zásadní.

3.2 Datová struktura

3.2.1 Získání datové struktury

V rámci aplikace je zapotřebí s daty, která jsou zapsaná v konfiguračním souboru, dále pracovat. Z konfiguračního souboru je tedy potřeba získat vhodnou datovou strukturu, která bude podporovat operace, které potom využijí jiné části aplikace. Proces získání datové struktury z konfiguračního souboru je znázorněn na obrázku 11.



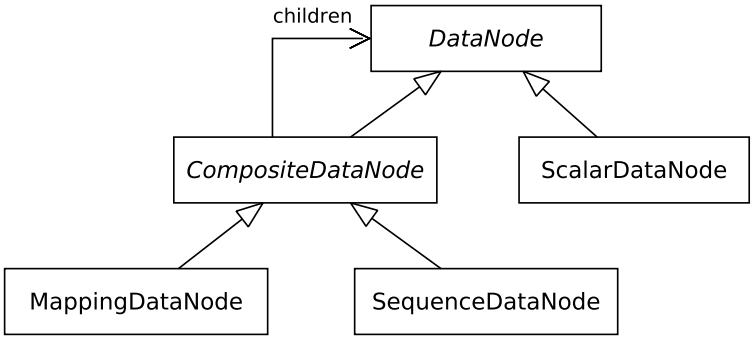
Obrázek 11: Získání datové struktury z konfiguračního souboru

Na začátku řetězce je vstupní konfigurační soubor. Z toho se získá obsah pomocí lexikální a syntaktické analýzy jazyka YAML. Tato data se dále zpracovávají speciální vrstvou, která provede potřebné autokonverze. Výstupem z celého tohoto procesu je výsledná datová struktura, se kterou se v aplikaci dále pracuje.

3.2.2 Návrh datové struktury

Datová struktura z pohledu konfiguračních souborů, resp. Flow123d, byla popsána v kapitole 1.2.1. Jak bylo řečeno, datová struktura tvoří strom. Jednotlivé uzly tedy mohou být buď interní, což znamená, že obsahují další potomky, nebo mohou být koncové a jsou to tedy tzv. listy. Na obrázku 12 je znázorněna navržená datová

struktura pomocí diagramu Unified Modeling Language (UML). Více o modelovacím jazyku UML je možné se dočíst v příslušné normě [6].



Obrázek 12: UML diagram datové struktury

Obecná abstrakce datového uzlu je reprezentována třídou *DataNode*. Od této třídy jsou odvozeny implementace *ScalarDataNode* a *CompositeDataNode*. Třída *ScalarDataNode* reprezentuje list stromu a tím pádem obsahuje nějakou skalární hodnotu. Třída *CompositeDataNode* reprezentuje interní uzel stromu, který obsahuje další uzly *DataNode*. Abstraktní třída *CompositeDataNode* má dvě odvozené třídy – *MappingDataNode* pro záznamy a *SequenceDataNode* pro pole. Popsaná a znázorněná reprezentace datové struktury odpovídá návrhovému vzoru *Composite*. Více o tomto a dalších návrhových vzorech se lze dočíst v knize Design Patterns [7].

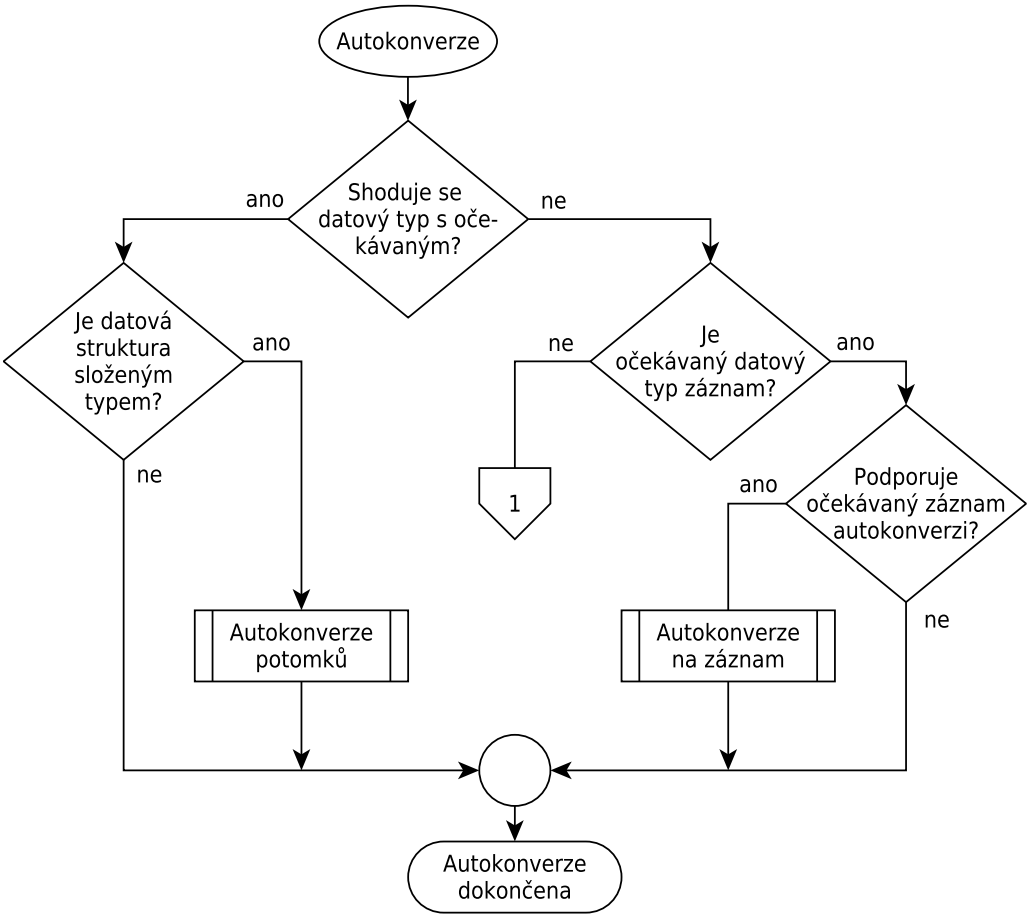
V aplikaci se pracuje s abstrakní třídou *DataNode*, která reprezentuje obecný uzel a poskytuje všechny potřebné informace o datové struktuře. Typicky se v rámci aplikace pracuje s kořenovým uzlem stromu, ze kterého je možné se dostat do všech ostatních uzlů.

3.3 Autokonverze

V konfiguračních souborech se často používají zkrácené zápisy, které je potřeba zpracovat pomocí autokonverzí, které byly podrobně popsány v kapitole 2.3. Proces autokonverzí, obdobně jako třeba validace, potřebuje ke své funkci ne pouze konfigurační soubor, ale i specifikaci formátu, která předepisuje očekávané datové typy.

Pro zpracování autokonverzí byl v rámci aplikace navržen samostatný modul, který zkonvertuje vstupní datovou strukturu spolu se specifikací formátu dle dříve zmíněných pravidel pro autokonverze. Autokonverze pracuje vždy s nějakým uzlem

v datové struktuře a funguje rekurzivně. Pokud se tedy proces autokonverze spustí na kořen stromu, dojde ke všem potřebným autokonverzím. Kompletní logika procesu autokonverzí je znázorněna na obrázcích 13, 14 a 15 pomocí vývojových diagramů.

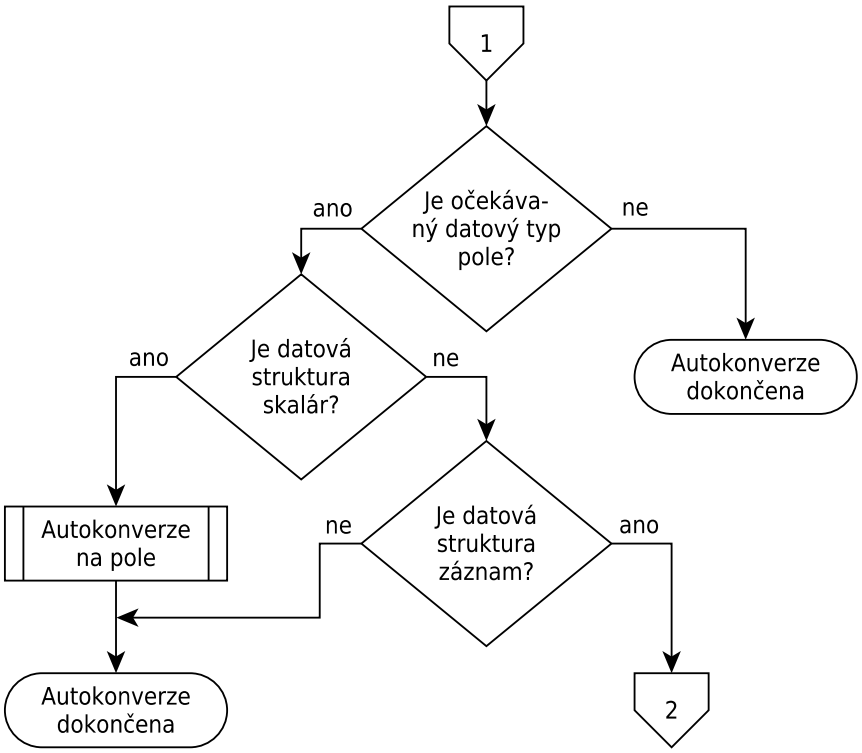


Obrázek 13: Proces autokonverze I. – Počáteční stav, kdy se ověřuje shoda datových typů, a případné provedení autokonverze na záznam

Začátek celého procesu se nachází na obrázku 13. Nejprve se zkontroluje, zda datový typ uzlu odpovídá očekávanému datovému typu. Pokud je datový typ správný, tak se pro danou datovou strukturu žádná autokonverze neprovede, ale rekurzivně se stejný proces opakuje i pro všechny přímé potomky této datové struktury.

K autokonverzi může dojít v případě, že se datový typ daného uzlu neshoduje s datovým typem, který se na tomto místě v datové struktuře očekává. Pokud tato situace nastane, tak se nejprve ověří, zda se očekává záznam. Pokud tomu tak je a zároveň tento záznam podporuje funkci autokonverze, tak dojde k autokonverzi na záznam, jak bylo popsáno v kapitole 2.3.2.

Další fáze pokračuje na obrázku 14. Zde se rozhoduje, zda je očekávaným datovým typem pole. Pokud tomu tak je, tak záleží, jakého datového typu je aktuální uzel. Pokud se jedná o skalární hodnotu, tak proběhne autokonverze na pole, která je schopná provést konverzi i na vícerozměrné pole. Funkce autokonverze na pole byla popsána v kapitole 2.3.1.



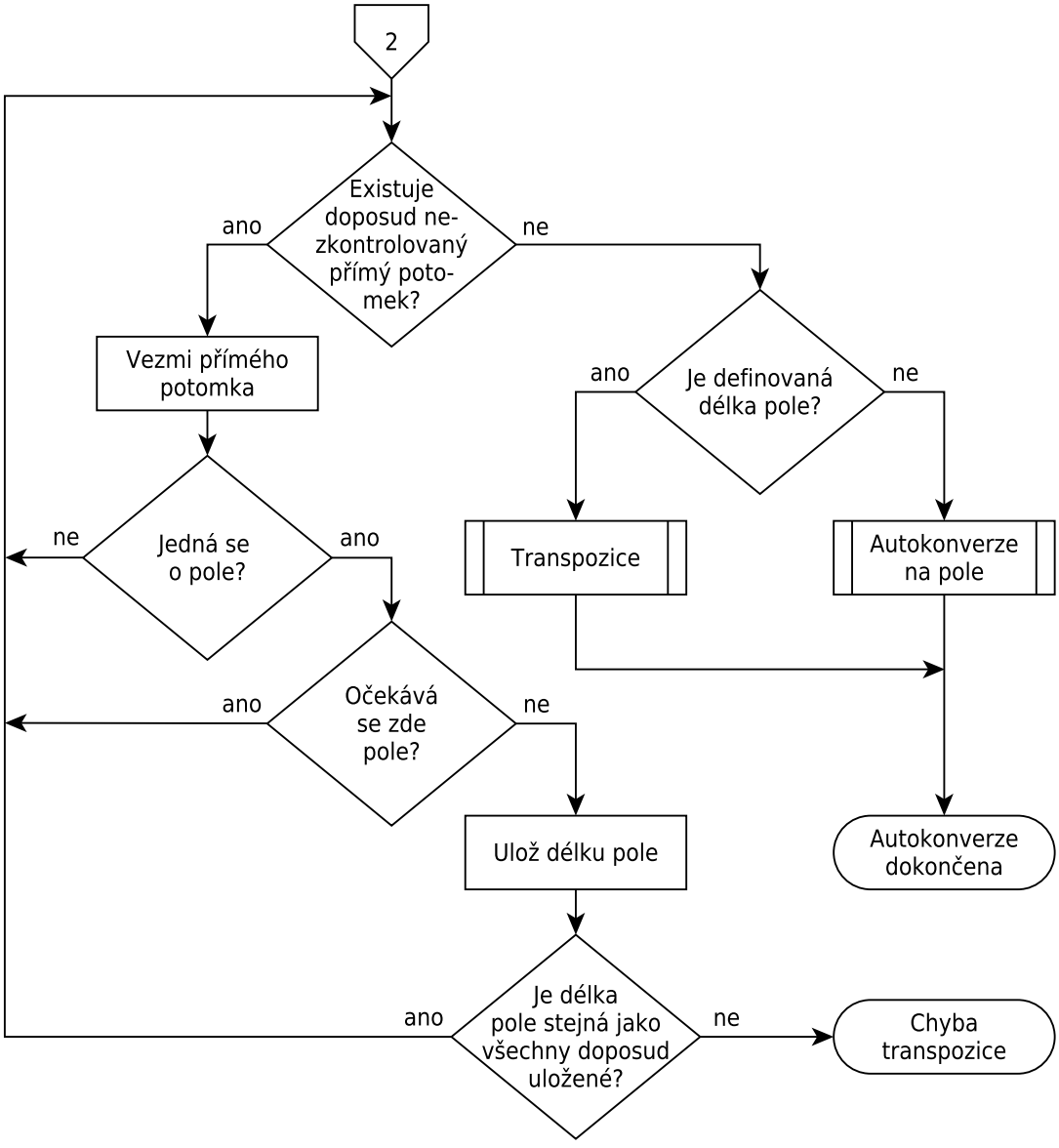
Obrázek 14: Proces autokonverze II. – Rozhodovací logika, která určuje, zda se má provést autokonverze na pole

Pokud se očekává pole a místo toho je uzel typu záznam, tak situace je komplikovanější, protože je potřeba určit, zda stačí provést autokonverzi na pole, nebo zda-li se jedná o transpozici. Tento proces popisuje poslední obrázek znázorňující proces autokonverzí, obrázek 15.

Detekce transpozice spočívá v následujících krocích. Za prvé se již počítá s faktem, že v této fázi se musí provést buď transpozice nebo autokonverze na pole (pokud nedojde k chybě transpozice díky špatným vstupním datům). Na základě toho lze odvodit datový typ očekávaného záznamu, který se má po autokonverzi získat, a tím pádem i očekávaný datový typ jeho potomků.

U aktuálního uzlu se postupně projdou všichni přímí potomci a dojde ke kontrole jejich datového typu oproti očekávanému datovému typu. Pokud se jedná o pole,

které se zde ovšem neočekává, signalizuje to použití transpozice. V tuto chvíli se délka tohoto pole uloží a zkontroluje. Aby transpozice mohla proběhnout, musí souhlasit délka všech neočekávaných polí – pokud nesouhlasí, jedná se o chybně zadaný vstup. Po zkontrolování všech přímých potomků se dorazí do větve, která rozhoduje, zda se má provést transpozice nebo autokonverze na pole. Pokud byla během procesu kontroly potomků definována délka pole a nedošlo k chybě, tak se provede transpozice tak, jak byla popsána v kapitole 2.3.3. Pokud se v záznamu nevyskytla žádná neočekávaná pole, tak se provede autokonverze na pole.

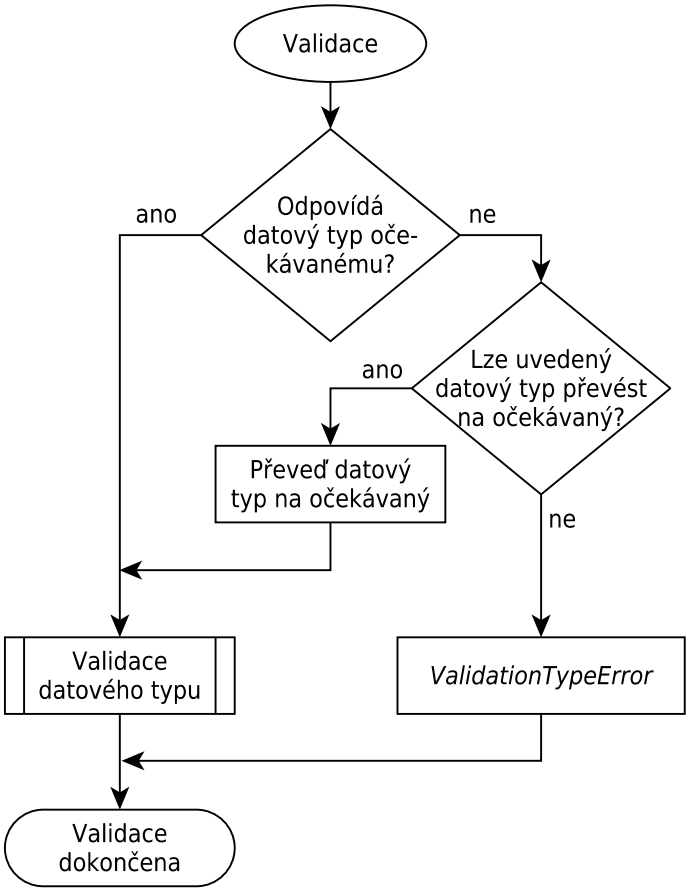


Obrázek 15: Proces autokonverze III. – Rozhodovací logika, která určuje, zda lze provést transpozice

3.4 Validace

Jednou z nejužitečnějších funkcí editoru je validace, která umožňuje odhalit chyby a možné problémy se zadanou konfigurací již během vytváření konfiguračních souborů. Vstupem do validace je datová struktura, která vznikla z dat konfiguračního souboru po provedení výše popsaných autokonverzí. Během procesu validace se tato datová struktura rekurzivně prochází a zaznamenávají se detekované chyby. Po dokončení procesu validace je uživatel upozorněn na tyto chyby pomocí grafického uživatelského rozhraní aplikace.

Validace ke své funkci potřebuje specifikaci formátu, na základě které ověřuje, zda mají vstupní data předepsanou strukturu a splňují všechna omezení. Jelikož validace pracuje s dynamicky zadanou specifikací formátu, tak je možné validaci provést pro libovolnou verzi Flow123d, kterou si uživatel může zvolit v uživatelském rozhraní.

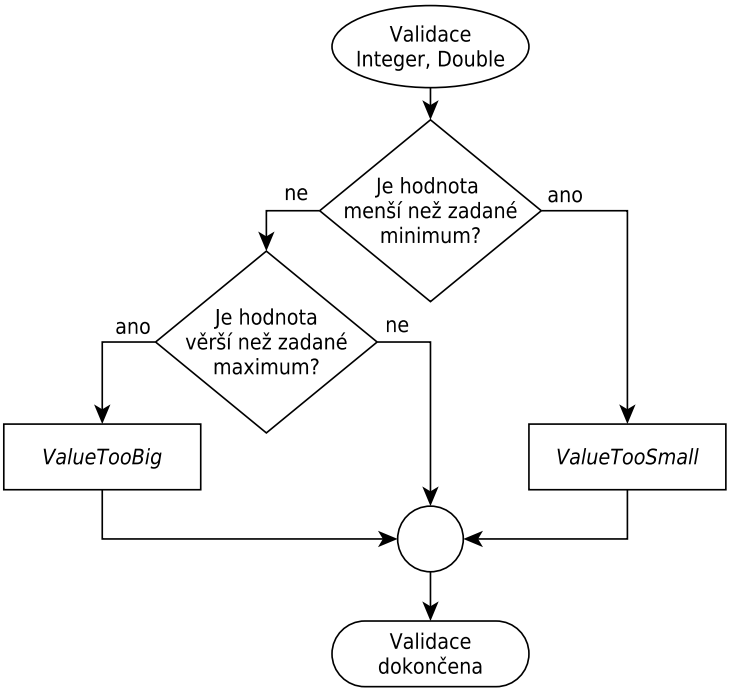


Obrázek 16: Validace datového typu

Prvním krokem validace je detekce a ověření zadaného datového typu. Tento proces je znázorněn na obrázku 16. Pokud neodpovídá zadaný datový typ očekávanému, aplikace se pokusí převést datový typ na očekávaný, pokud je to možné. Poté se provede validace, která závisí na datovém typu (tyto validace jsou postupně popsány dále). Pokud datový typ nelze převést na očekávaný, dojde k chybě *ValidationTypeError*.

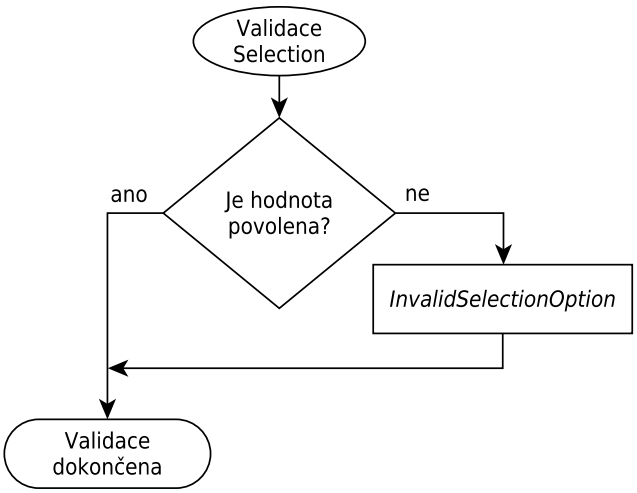
3.4.1 Validace primitivních datových typů

Některé z primitivních datových typů nemají žádnou typově specifickou validaci. Aktuálně se jedná o řetězce, názvy souborů a booleovské hodnoty (*String*, *Filename* a *Boolean*). U těchto datových typů se provede pouze kontrola datového typu a případná konverze, jak bylo popsáno výše.



Obrázek 17: Validace číselných datových typů – *Integer* a *Double*

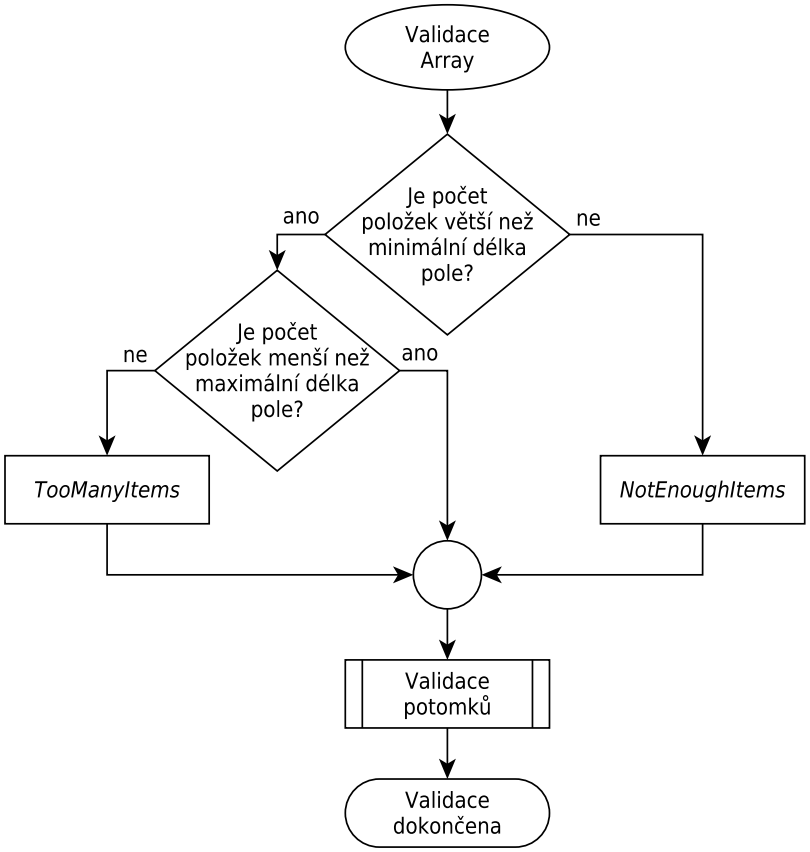
Číselné datové typy pro reprezentaci celých a desetinných čísel (*Integer* a *Double*) mohou mít ve specifikaci formátu zadaný interval platných hodnot, který je shora i zdola uzavřený. Provádí se tedy kontrola, zda se zadané číslo nachází v tomto intervalu. Pokud je hodnota menší než požadované minimum, dojde k chybě *ValueTooSmall*. V opačném případě, kdy je hodnota větší než požadované maximum, dojdě k chybě *ValueTooBig*. Validaci číselných datových typů znázorňuje obrázek 17.



Obrázek 18: Validace výčtového datového typu – *Selection*

Posledním primitivním datovým typem, který má speciální validaci, je výčtový datový typ (*Selection*). U tohoto datového typu se ověří, zda je zadaná hodnota jednou z možných hodnot, která je uvedena ve specifikaci formátu. Pokud zadaná hodnota není uvedena ve specifikaci formátu, dojde k chybě *InvalidSelectionOption*. Validaci výčtového datového typu znáznorňuje obrázek 18.

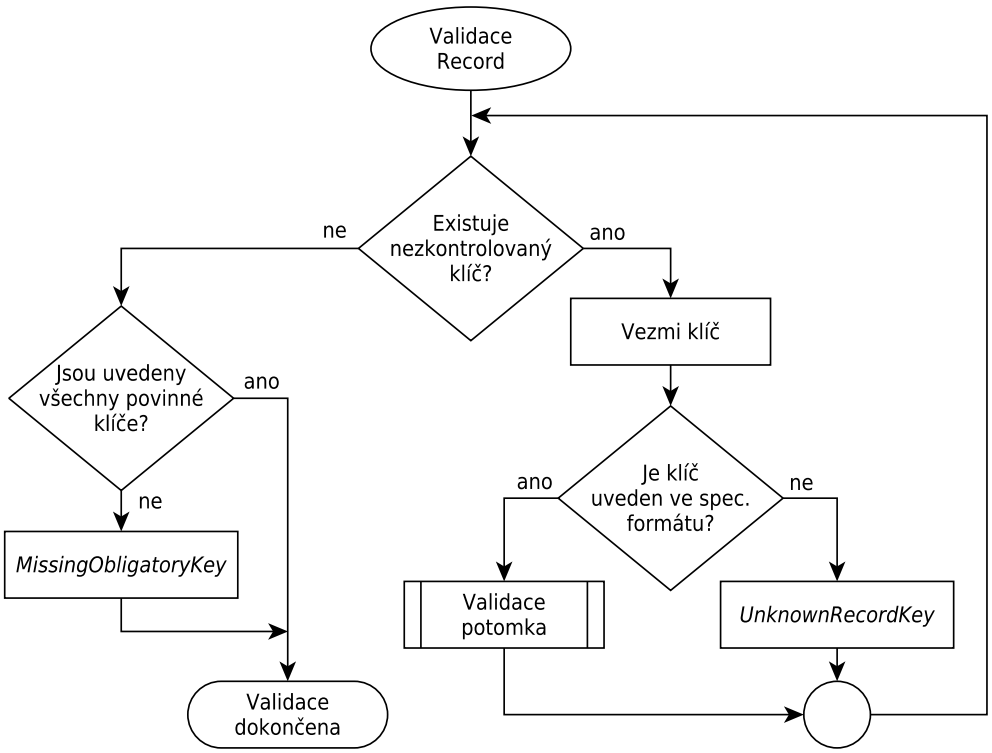
3.4.2 Validace pole



Obrázek 19: Validace pole – *Array*

U polí může být omezen počet prvků, které mohou obsahovat. Toto omezení je zadáno pomocí intervalu, který je z obou stran uzavřený. Proces validace tedy probíhá obdobně jako při validaci číselných datových typů s tím rozdílem, že se kontroluje počet prvků pole. Případná chybová hlášení se také liší. Pokud je počet prvků v poli menší než zadané minimum, dojde k chybě *NotEnoughItems*. V opačném případě dojde k chybě *TooManyItems*. Po kontrole počtu prvků pole se postupně prochází jeho potomci, u kterých se opět provádí validace. Validace pole je znázorněna na obrázku 19.

3.4.3 Validace záznamu



Obrázek 20: Validace záznamu – *Record*

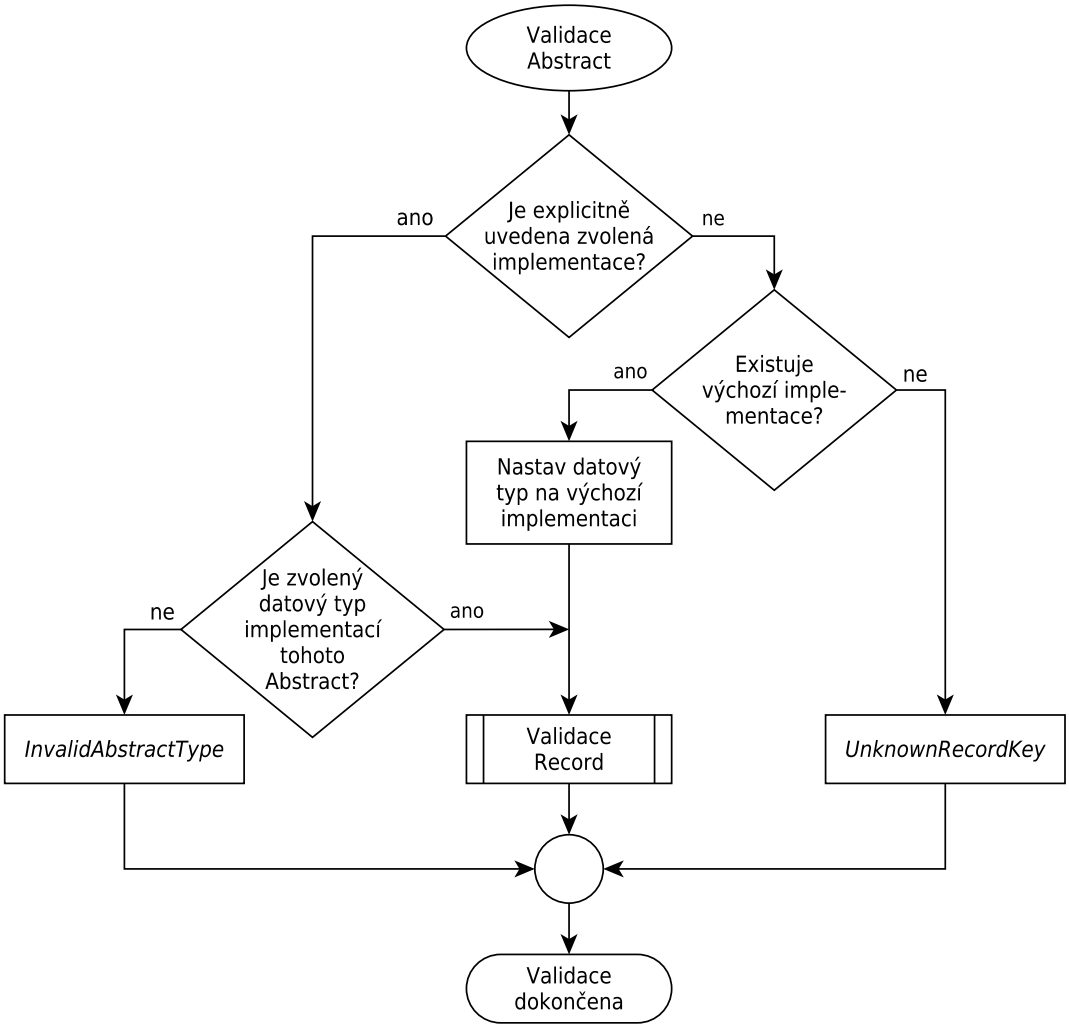
Záznamy jsou opět složeným typem, takže součástí procesu je opět validace potomků. Průběh validace záznamu je znázorněn na obrázku 20. Během tohoto procesu se postupně ověřují všechny klíče záznamu, které obsahují jeho potomky. Pro každý z těchto klíčů se najde jeho odpovídající datový typ ve specifikaci formátu. Pokud v ní není daný klíč uveden, dojde k upozornění *UnknownRecordKey* a pokračuje se dalším klíčem. Po validaci všech klíčů, resp. potomků, se ještě ověří, ze jsou v záznamu uvedeny všechny povinné klíče. Pokud tomu tak není, je uživatel na chybějící klíče upozorněn pomocí chyby *MissingObligatoryKey*.

Validace umožňuje různé úrovně chybových hlášení – informativní zprávy, upozornění, chyby a fatální chyby. Ve výše popsané validaci může kromě chyb vzniknout i upozornění *UnknownRecordKey*. K tomu dojde tehdy, když se v záznamu vyskytne klíč, který není uveden ve specifikaci formátu. Z hlediska Flow123d se nejedná o chybu. V praxi se však v těchto případech často jedná o překlep nebo špatný název nepovinného klíče, na který potom Flow123d nijak nereaguje. Uživatel pak musí pracně zjišťovat, proč se simulátor chová jinak, než očekává. Z toho důvodu validace upozorňuje i na tyto případy.

3.4.4 Validace abstraktního záznamu

Posledním datovým typem, který se může na vstupu očekávat, je abstraktní záznam (*Abstract*). Proces validace abstraktního záznamu je znázorněn na obrázku 21. Nejprve se určí, zda je explicitně zadán typ abstraktního záznamu (viz kapitola 3.1.2). Pokud je uveden, tak se ověří, ze tento záznam je implementací očekávaného abstraktního záznamu. Pokud by nebyl, validace skončí chybou *InvalidAbstractType*.

Jestliže není explicitně uvedený typ, tak se použije výchozí datový typ implementace, pokud je uveden ve specifikaci formátu. Pokud ovšem abstraktní záznam nemá výchozí implementaci, validace skončí chybou *UnknownRecordKey*. Ve chvíli, kdy je určen konkrétní datový typ záznamu, se může provést jeho validace tak, jak byla výše popsaná.



Obrázek 21: Validace abstraktního záznamu – *Abstract*

3.5 Vizualizace datové struktury

Jednou z komponent editoru je grafický strom, který slouží pro vizualizaci datové struktury. V prvotní fázi plánování editoru se uvažovalo o tom, že by se pomocí tohoto grafického stromu kompletně zadávala celá konfigurace a editor by neměl žádnou komponentu pro přímou úpravu zdrojového textu konfiguračního souboru. Nakonec se ovšem od tohoto návrhu upustilo z několika důvodů. Jednou z nevýhod je například to, že takový editor by uživateli neposkytoval žádnou kontrolu nad formátováním textu v konfiguračních souborech. S tím souvisí i problém komentářů, které by nebylo možné do konfiguračního souboru jednoduše zapsat na požadované místo. Dalším z důvodů bylo i to, že jsou modeláři zvyklí editovat konfigurační soubor ve formě textu.

Komponenta stromu má ovšem oproti pouhému editoru textu i určité výhody a proto byla v aplikaci nakonec použita. Zatímco zápis a všechny úpravy datové struktury se provádí editací textu, komponenta stromu slouží primárně jako vizualizace zadaných dat. Strom slouží k rychlé orientaci v konfiguračním souboru, označování hodnot a poskytuje náhled, jak vypadá datová struktura po provedení autokonverzí.

Komponenta stromu je rozdělena do dvou sloupců. V levém se nachází klíče nebo indexy pole a v pravém sloupci jsou příslušné hodnoty. Stromová struktura je znázorněna pomocí odsazení v levém sloupci. Jednotlivé větve stromu je možné zabalit pro přehlednější výpis požadovaných informací. Komponentu stromu je možné vidět na obrázku B.2 na straně 66.

Po kliknutí na libovolnou hodnotu či klíč dojde k jejímu označení v textu konfiguračního souboru. To má v zásadě dvě využití. Pokud je konfigurační soubor delší, než se vejde do okna editoru, tak se pozice okna textového editoru upraví tak, aby byla hodnota viditelná, takže uživatel se může rychle zorientovat. Díky tomu, že se zároveň označí, tak je možné ji rychle a pohodlně přepsat.

Komponenta stromu aktuálně podporuje zvýraznění typů abstraktních záznamů. V rámci budoucích rozšíření aplikace by mohla podporovat i další zvýraznění některých hodnot či datových typů. Kromě toho by bylo pohodlné, kdyby strom dokázal lépe zobrazovat dvourozměrná pole hodnot, například pomocí matice. To je ovšem otázkou budoucích vylepšení aplikace *ModelEditor*, nikoli předmětem této diplomové práce.

3.6 Textový editor

Hlavní komponentou aplikace *ModelEditor* je textový editor, který umožňuje upravovat text konfiguračního souboru. Editor byl optimalizován pro vytváření a upravování konfiguračních souborů Flow123d ve formátu YAML.

Rozhraní editoru zobrazuje čísla řádků, poskytuje základní zvýraznění syntaxe a zobrazuje vodící čáry pro lepší orientaci v úrovni odsazení. Přímo v okně textového editoru je na příslušných řádcích uživatel upozorněn na případné chyby pomocí ikony, která odpovídá závažnosti chyby. Po kliknutí na tuto ikonu se uživateli zobrazí podrobný popis chyby a označí se její lokace v textu. Ukázku grafického rozhraní komponenty textového editoru a seznamu notifikací je možné najít na obrázku B.3, resp. B.4 na straně 67.

Editor také podporuje funkci automatického doplňování textu, které značně urychluje zápis konfiguračních souborů. Na obrázku B.5 na straně 67 je znázorněn textový editor s vyvolanou nabídkou pro automatické doplňování textu. Pomocí funkce automatického doplňování je možné vybrat hodnotu z nabízeného seznamu místo jejího celého zápisu. Tato funkce pracuje na základě specifikace formátu a je citlivá na pozici v datové struktuře. To znamená, že nabízí k doplnění pouze ty hodnoty, které mají na současné pozici smysl. Například při doplňování názvu klíče jsou nabízené možnosti závislé na tom, ve kterém záznamu je aktuálně umístěn kurzor. Funkce automatického doplňování umožňuje v konfiguračních souborech doplňovat:

- názvy klíčů v záznamech,
- datové typy záznamů,
- hodnoty výčtových typů,
- uživatelsky definované kotvy[1].

Textový editor podporuje i běžné operace pro manipulaci s textem, které se od textového editoru očekávají. Editor podporuje následující operace:

- operace vracení změn (mechanismus undo/redo);
- práce se schránkou – vložení, vyjmutí, kopírování;
- nalezení a nahrazení[2];
- změna odsazení bloku;
- zakomentování nebo odkomentování označeného bloku.

[1] Uživatelsky definované kotvy slouží jako reference, viz kapitola 3.1.3.
[2] Funkce nalezení a nahrazení podporuje i regulární výrazy.

3.7 Kontextová nápověda

Poslední z hlavních komponent editoru je kontextová nápověda. Tato komponenta slouží pro zobrazení dokumentace k aktuálně upravované části datové struktury. Obsah této nápovědy je totožný s kompletním referenčním manuálem, který má několik desítek stran. Hlavním rozdílem je ovšem to, že komponenta kontextové nápovědy zobrazuje uživateli pouze informace, které bezprostředně souvisí s právě editovanou částí konfiguračního souboru. Kontextová nápověda tedy poskytuje mnohem pohodlnější a příjemnější rozhraní k již existující dokumentaci.

Všechny informace potřebné pro generování dokumentace se opět nachází ve specifikaci formátu. Tím pádem i funkce kontextové dokumentace je schopná se přizpůsobit vybrané verzi Flow123d.

Kontextová dokumentace zároveň pracuje s pozicí kurzoru v okně editoru, na základě které se určuje aktuální pozice v datové struktuře, podle které se zobrazí příslušná část dokumentace.

Struktura konfiguračních souborů je taková, že všechna data[3] mají nějaký nadřazený záznam a jsou zapsána v nějakém klíči. Z toho vychází, jaké informace se zobrazují v kontextové nápovědě. V té jsou zobrazeny následující informace:

- název a popis nadřazeného záznamu,
- seznam všech možných klíčů tohoto záznamu,
- popis aktuálně vybraného klíče,
- datový typ klíče a jeho popis.

Pokud je nadřazený záznam implementací některého abstraktního záznamu, zobrazí se navíc ještě název a popis tohoto abstraktního záznamu včetně seznamu jeho možných implementací, které se skládají z názvu a popisu.

Všechny názvy složených datových typů a klíčů záznamů jsou odkazy, na které je možné kliknout. Dokumentaci zobrazenou v kontextové nápovědě je tak možné procházet. Dále je možné pro navigaci použít tlačítka zpět, vpřed a domů, podobně jako v internetových prohlížečích.

Vzhled komponenty v grafickém rozhraní aplikace a ukázky zobrazovaných dat je možné vidět na obrázcích B.6, B.7 a B.8 na straně 68.

[3] Kromě samotného kořenového uzlu celé datové struktury.

4 Implementace a testování

Aplikace *ModelEditor* je součástí aplikačního balíku *GeoMop*, za jehož vývoj je zodpovědný Ing. Pavel Richter. V rámci této diplomové práce byly v aplikaci *ModelEditor* implementovány následující funkce:

- datová struktura pro práci s konfiguračními soubory a specifikací formátu,
- zpracování konfiguračního souboru ve formátu YAML,
- proces autokonverzí,
- proces validace,
- generování kontextové dokumentace,
- automatické doplňování textu,
- grafické rozhraní aplikace včetně jeho komponent.

Kromě těchto funkcí ovšem *ModelEditor* obsahuje i části, na kterých pracoval Ing. Richter a nevznikly tedy v rámci této diplomové práce. Konkrétně se jedná o následující funkce:

- rozhodovací logika, která určuje, kdy má dojít k opětovnému načtení datové struktury,
- import formátu CON,
- transformace.

Nejužší souvislost s touto diplomovou prací má rozhodovací logika pro opětovné načtení datové struktury. Výše popsaný proces zpracování formátu YAML, autokonverzí, validací, generování dokumentace a reprezentace dat musí probíhat neustále dokola. Když uživatel upravuje text souboru, mění se data i pozice kurzoru. Na těchto informacích závisí všechny zmíněné funkce. Pokud by ovšem celý proces měl probíhat při každém napsaném znaku, vedlo by to ke zbytečnému vytížení procesoru a znatelnému zpomalení aplikace. Bylo tedy nutné implementovat funkcionalitu, která rozhodne, kdy má dojít k opětovnému načtení datové struktury a aktualizaci všech komponent, které na ní závisí.

Dále *ModelEditor* obsahuje doplňkové funkce jako import formátu CON, ve kterém se dříve psaly konfigurační soubory, nebo transformace, které umožňují upra-

vovat datovou strukturu (a tedy i text konfiguračního souboru) pomocí předdefinované sady pravidel. Transformace je možné využít například pro konverzi konfiguračních souborů mezi různými verzemi Flow123d, kde došlo ke změně specifikace formátu.

4.1 Požadavky

Na aplikaci *ModelEditor* se vztahují určité požadavky, které vychází z požadavků na celý aplikační balík *GeoMop*. Jedná se především o následující požadavky:

- Aplikace je vyvíjena jako open-source pod licencí GNU General Public License (GPL) v3.0 [8].
- Aplikace musí být multiplatformní a má podporovat systémy Windows (XP a novější) a Linux.
- Aplikace je napsaná v jazyce Python 3 [9].
- Aplikace používá knihovnu PyQt 5 [10] pro grafické rozhraní.
- Pro vývoj aplikace se používá verzovací systém Git. Aktuální verze zdrojových kódů je veřejně dostupná na adrese: `https://github.com/GeoMop/GeoMop`.
- Pro dokumentaci zdrojových kódů se používá nástroj Sphinx [11].

4.2 Zpracování formátu YAML

V kapitole 2.1.1 byl popsán proces zpracování formátu YAML, který je rozdělen do čtyř vrstev – prezentace, serializace, reprezentace a datová struktura. Uvedený postup zpracování, který je původně definován v dokumentaci YAML [5], se většinou dodržuje i v konkrétních implementacích knihoven, které se používají pro zpracování YAML. V jazyce Python je dostupná knihovna *PyYAML* [12], která až na některé detaily odpovídá dokumentaci.

Rozhraní nejvyšší úrovně poskytuje funkci, jejímž vstupem je text obsahující data ve formátu YAML a výstupem jsou nativní datové struktury v jazyce Python – slovníky, seznamy a hodnoty. Vrstva datové struktury je tedy kompletně oddělená od její prezentace[1], což je v souladu s dokumentací YAML.

Pro potřeby aplikace *ModelEditor* je toto chování nevyhovující. V aplikaci je zásadní pozice symbolů v textu, aby bylo možné s textovým editorem interaktivně pracovat. Tato informace je dostupná pouze nižším vrstvám v rámci zpracování souboru. Dále je potřeba v rámci aplikace rozeznávat reference – což také není

[1] Prezentací se rozumí zápis formátu YAML v podobě textového souboru.

možné, protože data jsou ve výsledné struktuře pouze zkopírovaná a nelze tím pádem určit, která data byla zdrojová. Posledním úskalím pak byl zápis typu abstraktních záznamů pomocí tagů. Knihovna *PyYAML* se pokouší interpretovat tagy jako názvy tříd v Pythonu, což je pro účely aplikace *ModelEditor* opět nežádoucí.

Z knihovny *PyYAML* byla tedy využita pouze nejnižší úroveň zpracování, která provede pouze lexikální analýzu. Výstupem z tohoto procesu je seznam událostí. Tento seznam událostí je dále zpracováván aplikací *ModelEditor*, který na jejím základě vytváří aplikačně specifickou datovou strukturu, která obsahuje všechny potřebné informace. Výsledkem je tedy modul, který vychází z knihovny *PyYAML* a na nejvyšší úrovni poskytuje funkci, která umožňuje načíst vstupní text ve formátu YAML a výstupem z této funkce je datová struktura aplikace.

4.3 Textový editor

Pro komponentu textového editoru byla použita knihovna *QScintilla 2* [13] pro Python, což je port knihovny *Scintilla* [14] do grafického prostředí *Qt*. *Scintilla* poskytuje multiplatformní komponentu, která podporuje mnoho funkcí, které se používají při editaci zdrojového kódu. Podporuje například zobrazení čísel řádků, zobrazení indikátorů chyb, zvýraznění syntaxe, doplňování textu a další.

QScintilla se snaží většinu funkcí z knihovny *Scintilla* obalit a poskytnout k nim vlastní Application Programming Interface (API), které by mělo zjednodušit jejich použití. Při implementaci se ovšem ukázalo, že pro některé funkce je nutné použít nízkoúrovňový přístup, při kterém se volaly přímo funkce editoru *Scintilla* pomocí zasílání zpráv, které není příliš dobře zdokumentované. Pomocí tohoto přístupu bylo potřeba zajistit funkci automatického doplňování a použití správných znaků pro ukončení řádky podle platformy.

Automatické doplňování je poskytováno i pomocí funkcí z *QScintilla* API. Ovšem poskytovaná funkčnost je značně omezená a neumožňovala by tak například vhodně filtrovat nabízené možnosti podle kontextu. Oproti tomu přímý přístup k rozhraní *Scintilla* poskytuje plnou kontrolu nad komponentou automatického doplňováním i nad samotným textem editoru. Díky tomu bylo možné tuto funkcionalitu realizovat tak, aby splňovala všechny požadavky.

Další implementační zajímavostí byl mechanismus vracení provedených změn. API poskytuje funkce, které toto umožňují implementovat. Ovšem v jejich nezměněné podobě probíhá vracení operací po nejmenší možné jednotce – tedy po napsaných znacích. Z uživatelského hlediska není takové chování žádoucí, protože zachycené

změny jsou příliš drobné a často nepodstatné. Tento mechanismus byl proto rozšířen tak, aby podporoval vrácení změn po větších částech – konkrétně po částech, které vyvolají opětovné načtení datové struktury. Typicky se jedná o třeba dokončení řádky nebo vložení textu ze schránky.

4.4 Kontextová nápověda

Při tvorbě komponenty pro zobrazování kontextové nápovědy bylo vzato v úvahu následující:

- grafický vzhled nemusí být finální a měl by jít upravovat,
- položky nápovědy by mezi sebou měly být propojené pomocí odkazů,
- v rámci dokumentačních řetězců se mohou vyskytnout rovnice zapsané pomocí syntaxe `math` prostředí TEX,
- v současnosti existuje knihovna, která generuje ze specifikace formátu dokumentaci ve formátu HTML.

Vzhledem k těmto skutečnostem byla kontextová nápověda realizována prostřednictvím komponenty *QWebView*, která funguje jako primitivní internetový prohlížeč s omezenou podporou HTML, Cascading Style Sheets (CSS) a JavaScript (JS). Ze specifikace formátu, která obsahuje dokumentační řetězce a další informace o datové struktuře, se generuje webová stránka, která se poté pomocí této komponenty zobrazí.

Pro generování HTML byla využita stávající knihovna *Flow123d-python-utils*[2], jejíž autorem je Ing. Jan Hybš. Z této knihovny byla použita odpovídající část, která slouží pro generování dokumentace, a ta byla dále upravena pro potřeby aplikace *ModelEditor*. Knihovna řeší i vykreslení rovnic pomocí nástroje KaTeX[3].

Během implementace bylo nutné nahradit či obejít některé vlastnosti HTML, CSS a JS, protože jejich podpora v komponentě *QWebView* je v porovnání s moderními webovými prohlížeči značně omezená. Použití této komponenty bylo ovšem výhodné v tom, že je přímo součástí *Qt* a nebyla tak zapotřebí žádná další externí knihovna, která by poskytovala komponentu okna webového prohlížeče.

[2] `https://github.com/x3mSpeedy/Flow123d-python-utils`
[3] `https://github.com/Khan/KaTeX`

4.5 Testování

V rámci vývoje byly psány testy pro ověření funkčnosti aplikace. Snahou ovšem nebylo dosáhnout 100% pokrytí kódu testy. Potřebná časová investice by byla naprosto neúměrná reálnému přínosu. Spíše byl kladen důraz na psaní testů pro moduly, které se často měnily a upravovaly, nebo se očekává, že by v se v budoucnu mohly měnit. Dále byly vytvořeny testy pro ty části aplikace, které jsou jednoznačné a snadno testovatelné a dalo by se říct, že jejich funkcionalita je pomocí testů nadefinovaná. U těchto částí aplikace byl v rámci vývoje použit přístup Test Driven Development (TDD).

Bylo napsáno celkem 206 testů. Většinou se jedná o jednotkové testy, které testují nějakou konkrétní funkci, metodu, třídu či komponentu. Jsou mezi nimi ovšem i testy, které by se daly považovat za integrační testy, protože testují více komponent najednou a interakce mezi nimi. Z těchto 206 testů bylo napsáno:

- 72 testů pro datovou strukturu a operace, které s ní souvisí – načtení souboru YAML, autokonverze a validace;
- 115 testů pro pomocné třídy, které zajišťují především analýzu textu;
- 5 testů pro uživatelské rozhraní;
- 14 testů pro serializaci a ukládání nastavení aplikace.

Kromě jednotkových a integračních testů bylo součástí testování i manuální ověření funkčnosti aplikace a její odladění. Manuálně bylo testováno především uživatelské rozhraní a interakce mezi komponentami. Většina chyb, které byly odhaleny během testování, byla odstraněna. Některé chyby, které nebyly zásadní a jednalo se spíše o nedostatky, byly pouze zaznamenány. Jejich odstranění je záležitostí dalšího vývoje aplikace *ModelEditor* v rámci projektu *GeoMop*. Aplikace byla testována i několika cílovými uživateli a jejich připomínky a náměty vedly k některým změnám a vylepšením.

Závěr

V rámci této diplomové práce vznikl editor konfiguračních souborů pro Flow123d. Jedná se o samostatně použitelnou aplikaci, která je ovšem navržena s ohledem na její integraci do softwarového balíku GeoMop, který obsahuje další nástroje, které usnadňují práci uživatelům Flow123d.

Editor uživatelům zjednodušuje vytváření a upravování konfiguračních souborů. Umožňuje ověřit správnost zadané konfigurace pro zvolenou verzi Flow123d a upozorňuje uživatele na detekované chyby. Tato funkce přináší uživateli časovou úsporu a uživatelsky příjemnější rozhraní pro odhalení a odstranění chyb.

Editor dále uživatelům poskytuje kontextovou nápovědu, která zobrazuje dokumentaci, které bezprostředně souvisí s částí konfiguračního souboru, kterou právě upravuje. Obsah nápovědy se interaktivně mění v závislosti na tom, jakou část konfiguračního souboru uživatel právě edituje, aby byla zobrazena vždy relevantní dokumentace. Pro uživatele to představuje značné zjednodušení, jelikož může využít tuto kontextovou nápovědu místo prohledávání rozsáhlé referenční dokumentace.

V neposlední řadě editor obsahuje komponentu pro grafické znázornění datové struktury, která poskytuje alternativní pohled na zadaná data, a umožňuje rychlejší orientaci v rozsáhlých konfiguračních souborech. Kromě těchto stěžejních funkcí poskytuje editor uživateli i běžné nástroje pro manipulaci s textem, jako jsou například operace se schránkou, možnost vracení provedených změn, vyhledávání a nahrazení nebo změna úrovně odsazení.

Aplikace je multiplatformní a podporuje systémy Windows (XP nebo novější) a Linux. Pro vývoj použit jazyk Python 3 a grafická knihovna PyQt 5. K aplikaci byly vytvořeny instalační balíčky pro Windows a Debian.

V rámci budoucího vývoje jsou plánovány další dodatečné funkce. Jedná se např. o zlepšení zvýraznění syntaxe, které by mohlo zohledňovat datovou strukturu konfiguračních souborů Flow123d. Další možné vylepšení spočívá v rozšíření funkcionality komponenty pro vizualizaci datové struktury, která by mohla v budoucnu podporovat kromě i editaci dat nebo vylepšené zobrazení speciálních datových typů.

Literatura

[1] Ecma International. *Standard ECMA-404: The JSON Data Interchange Format*. [online]. Geneva: Ecma International, 2013. [cit. 2016-02-23] Dostupné na: <http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf>

[2] Březina, Jan et al. *Flow123d version 1.8.2: Documentation of file formats and brief user manual*. [online]. Liberec, 2015 [cit. 2016-02-24]. Dostupné na: <http://flow.nti.tul.cz/packages/1.8.2_release/flow123d_1.8.2_doc.pdf>

[3] ISO. *ISO 8879*. Amd.1:1988. International Organization for Standardization (ISO), 1988. [cit. 2016-02-26].

[4] W3C. *Extensible Markup Language (XML) 1.0*. Fifth Edition. [online]. World Wide Web Consorcium (W3C), 2008. [cit. 2016-02-26]. Dostupné na: <http://www.w3.org/TR/2008/REC-xml-20081126>

[5] Ben-Kiki, Oren et al. *YAML Ain't Markup Language (YAML^{TM}) Version 1.2*. 3rd Edition. [online]. 2009. [cit. 2016-03-06]. Dostupné na: <http://www.yaml.org/spec/1.2/spec.html>

[6] OMG. *ISO/IEC 19505-1: Information technology - Object Management Group Unified Modeling Language (OMG UML), Infrastructure*. [online]. Object Management Group (OMG), 2012. [cit. 2016-03-21]. Dostupné na: <http://www.omg.org/spec/UML/ISO/19505-1/PDF>

[7] Gamma, Erich et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. 1st Edition. Addison-Wesley Professional, 1994. ISBN 978-0201633610.

[8] FSF. *GNU Project: The GNU General Public License v3.0*. [online]. Free Software Foundation (FSF), 2007. [cit. 2016-03-26]. Dostupné na: <http://www.gnu.org/copyleft/gpl.html>

[9] PSF. *Python 3.4.4 Documentation*. [online]. Python Software Foundation (PSF), 2016. [cit. 2016-03-26]. Dostupné na: <https://docs.python.org/3.4/>

[10] RCL. *PyQt 5 Reference Guide*. [online]. Riverbank Computing Limited (RCL), 2016. [cit. 2016-03-26]. Dostupné na: <http://pyqt.sourceforge.net/Docs/PyQt5/>

[11] Brandl, Georg et al. *Sphinx 1.3.7+ Documentation*. [online]. Georg Brandl and the Sphinx team, 2016. [cit. 2016-03-26]. Dostupné na: <http://www.sphinx-doc.org/en/stable/>

[12] Kirill, Simonov. *PyYAML*. [online]. 2014. [cit. 2016-04-12]. Dostupné na: <http://pyyaml.org/wiki/PyYAML>

[13] RCL. *QScintilla: QScintilla – a Port to Qt v4 and Qt v5 of Scintilla* . [online]. Riverbank Computing Limited (RCL), 2016. [cit. 2016-04-11]. Dostupné na: <http://pyqt.sourceforge.net/Docs/QScintilla2/>

[14] Scintilla. *Scintilla Documentation*. [online]. 2016. [cit. 2016-04-11]. Dostupné na: <http://www.scintilla.org/ScintillaDoc.html>

A Ukázky kódů konfiguračních souborů

```
{
  solver = {
    TYPE = "Petsc",
    a_tol = 1e-12,
    r_tol = 1e-12
  }
}
```

Obrázek A.1: Výběr implementace abstraktního záznamu ve formátu CON

```
solver: !petsc
  a_tol: 1e-12
  r_tol: 1e-12
```

Obrázek A.2: Výběr implementace abstraktního záznamu ve formátu YAML

```
{
  primary_equation = {
    TYPE = "Unsteady_LMH",
    ...
    time = {
      end_time = 0.5,
      max_dt = 0.01,
      min_dt = 0.01
    }
  },

  time = {
    REF = "../primary_equation/time"
  }
}
```

Obrázek A.3: Použití reference ve formátu CON

```
primary_equation: !Unsteady_LMH
  ...
  time: &time
    end_time: 0.5
    max_dt: 0.01
    min_dt: 0.01

time: *time
```

Obrázek A.4: Použití reference ve formátu YAML

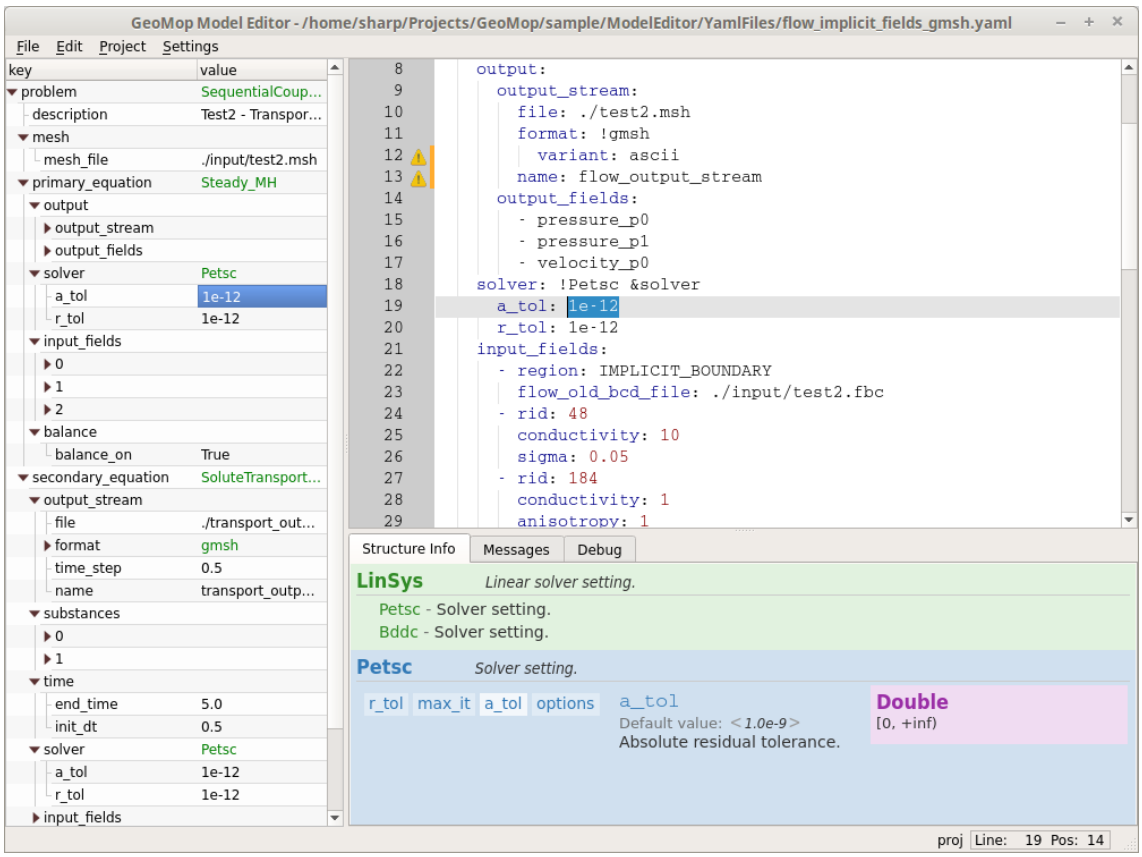
```
{
  problem = {
    TYPE = "SequentialCoupling",
    description = "Unsteady flow in 2D, Mixed Hybrid method",
    mesh = {
      mesh_file = "${INPUT}/square_2d.msh"
    },
    primary_equation = {
      TYPE = "Steady_MH",

      input_fields= [
        { region = "plane",
          conductivity = 1
        },
        { r_set = "BOUNDARY",
          time=0,
          bc_type = "dirichlet",
          bc_pressure = {TYPE="FieldFormula", value="0"}
        },
        { r_set = "BOUNDARY",
          time=1,
          bc_type = "dirichlet",
          bc_pressure = {TYPE="FieldFormula", value="x"}
        },
        { r_set = "BOUNDARY",
          time=2,
          bc_type = "dirichlet",
          bc_pressure = {TYPE="FieldFormula", value="2*x"}
        }
      ],

      n_schurs = 2,
      output = {
        output_stream = {
          add_input_times=true,
          file = "./test09.pvd",
          format = {
            TYPE = "vtk",
            variant = "ascii"
          },
          name = "flow_output_stream"
        },
        output_fields = [ "pressure_p0", "pressure_p1", "
    velocity_p0" ]
      },

      balance = true,

      solver = {
        TYPE = "Petsc",
        a_tol = 1e-07
      }
    }
  }
}
```

Obrázek A.5: Konfigurační soubor `flow_time_dep` ve formátu CON

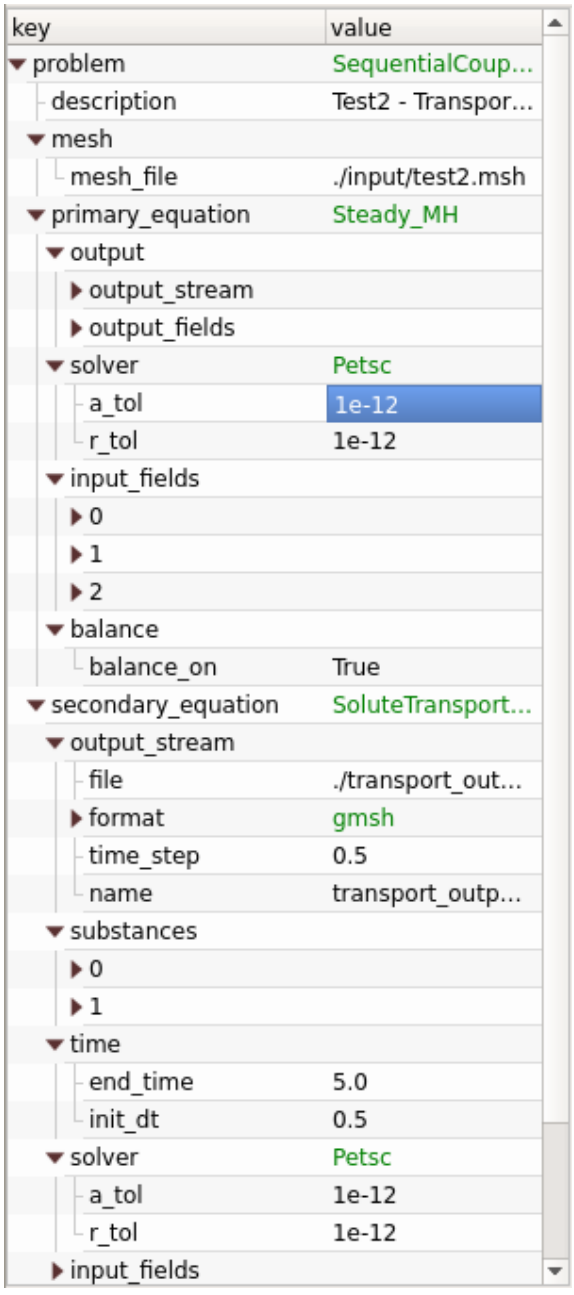
```
problem: !SequentialCoupling
  description: Unsteady flow in 2D, Mixed Hybrid method
  mesh:
    mesh_file: <input>/square_2d.msh
  primary_equation: !Steady_MH

    input_fields:
      - region: plane
        conductivity: 1
      - r_set: BOUNDARY
        time: 0
        bc_type: dirichlet
        bc_pressure: !FieldFormula 0
      - r_set: BOUNDARY
        time: 1
        bc_type: dirichlet
        bc_pressure: !FieldFormula x
      - r_set: BOUNDARY
        time: 2
        bc_type: dirichlet
        bc_pressure: !FieldFormula 2*x

    n_schurs: 2

    output:
      output_stream:
        add_input_times: true
        file: ./test09.pvd
        format: !vtk
          variant: ascii
        name: flow_output_stream
      output_fields:
        - pressure_p0
        - pressure_p1
        - velocity_p0

    balance: true

    solver: !Petsc
      a_tol: 1.0e-07
```

Obrázek A.6: Konfigurační soubor `flow_time_dep` ve formátu YAML

B Grafické rozhraní aplikace

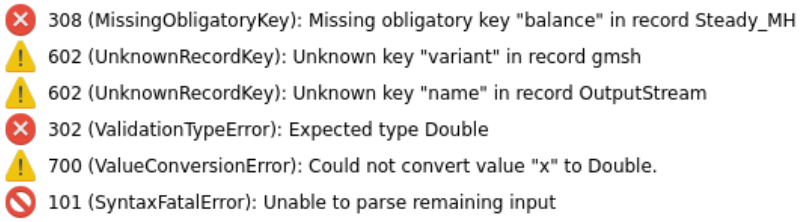


Obrázek B.1: GUI – Přehled rozhraní aplikace



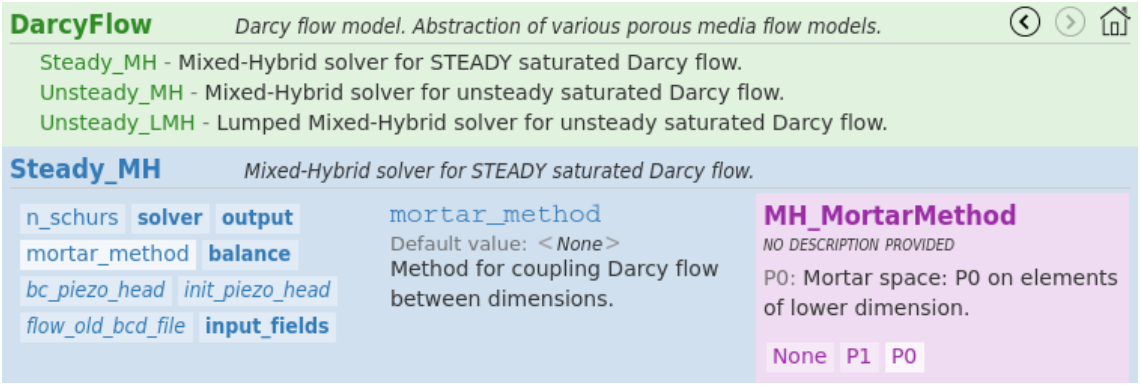
Obrázek B.2: GUI – Komponenta stromu pro vizualizaci dat

Obrázek B.3: GUI – Komponenta textového editoru s notifikacemi

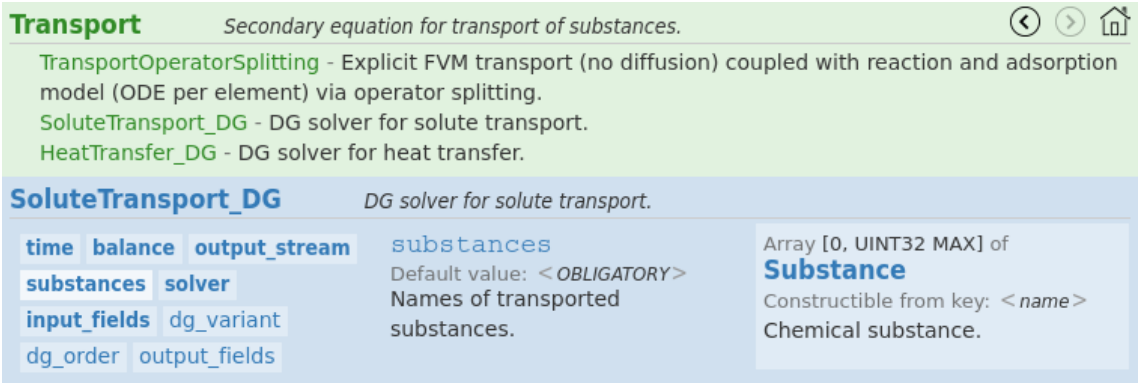


Obrázek B.4: GUI – Komponenta seznamu notifikací

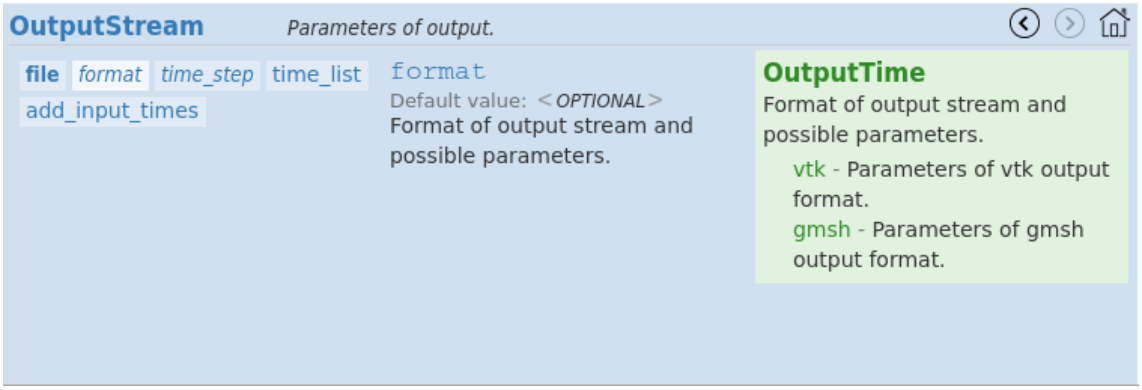
Obrázek B.5: GUI – Komponenta textového editoru s doplňováním textu



Obrázek B.6: GUI – Komponenta kontextové nápovědy zobrazující dokumentaci k hodnotě `P0` výčtového typu *MH_MortartMethod* v klíči `mortar_method` v záznamu *Steady_MH*, který implementuje abstraktní záznam *DarcyFlow*



Obrázek B.7: GUI – Komponenta kontextové nápovědy zobrazující dokumentaci k poli záznamů *Substance* v klíči `substances` v záznamu *SoluteTransport_DG*, který implementuje abstraktní záznam *Transport*



Obrázek B.8: GUI – Komponenta kontextové nápovědy zobrazující dokumentaci k abstraktnímu záznamu *OutputTime* v klíči `format` v záznamu *OutputStream*

C Obsah přiloženého CD

/
- `img` ..... Obrázky použité v diplomové práci.
- `git` ..... Zdrojové kódy a testy (git commit `234f0da`).
- `dp_krizek_2016.pdf` ..... Text diplomové práce ve formátu pdf.
- `modeleditor_234f0da.deb` ..... Instalační balíček pro Debian.
- `modeleditor_234f0da.exe` ..... Instalační soubor pro systémy Windows.