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

# Editor for Flow123d configuration files

## 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:

# Abstrakt

Český abstrakt

# Abstract

English abstract

# Poděkování

# Obsah

# Seznam obrázků

# Seznam zkratek

**CON** C++ Object Notation.

**DTD** Document Type Definition.

**HTML** HyperText Markup Language.

**IST** Input Structure Tree.

**JSON** JavaScript Object Notation.

**SGML** Standard Generalized Markup Language.

**UML** Unified Modeling Language.

**W3C** World Wide Web Consorcium.

**XML** Extensible Markup Language.

**YAML** YAML Ain't Markup Language.

# Úvod

Existuje celá řada softwarů, která pro zajištění požadované funkce potřebuje správné nastavení, podle kterého pak daný program 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 programy, 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 programů 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.

Software 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 softwaru 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. Editor zrychlí proces odstranění chyb tím, že je odhalí už v průběhu vytváření nebo upravování konfiguračních souborů. To uživateli umožní chyby odstranit ještě před tím, než předloží konfigurační soubor softwaru 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.

# 1 Problematika

## 1.1 Software Flow123d

Flow123d je software, 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.

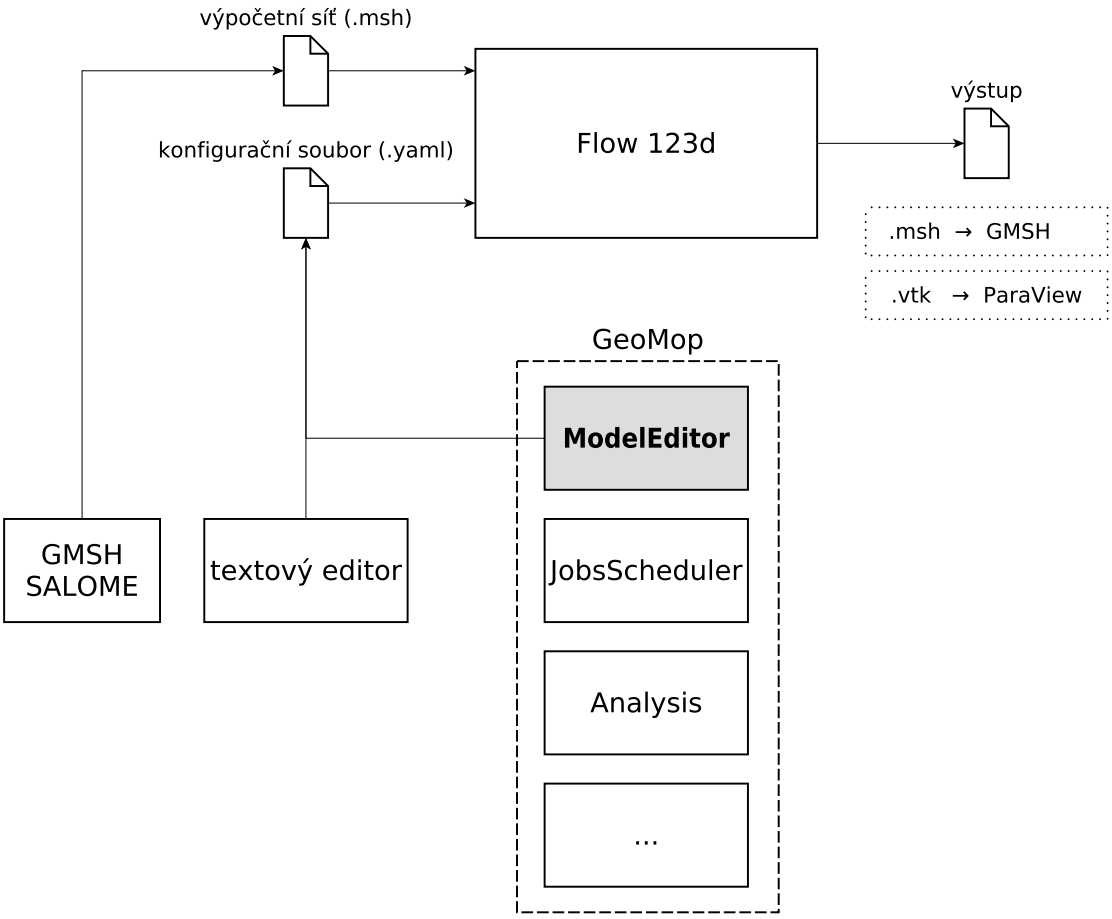
Na obrázku 1 jsou znázorněny vstupy a výstupy simulátoru Flow123d spolu s pomocnými aplikacemi, které uživatelé často používají. Vstupem do simulátoru Flow123d jsou dva soubory. První z těchto souborů popisuje výpočetní síť pomocí seznamu uzlů a elementů. Jedná se o textový soubor ve formátu `.msh`. 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 dokumentace nebo automatické doplňování textu. Vytváření a editace konfiguračních souborů se tak uživateli značně zjednoduší. *ModelEditor* je jednou ze součástí aplikace GeoMop, která obsahuje i další komponenty, které ovšem nejsou předmětem této práce.

GeoMop má sloužit jako nástroj, který usnadní práci se simulátorem Flow123d. Jeho další komponenty mají za úkol např. zajišťovat vzdálené spouštění Flow123d na výpočetních clusterech. To bude zajišťovat modul JobsScheduler, který sjednotí rozhraní a postup spouštění Flow123d na různých výpočetních clusterech. Další součástí aplikace GeoMop bude modul Analysis, který umožní úlohy parametrizovat a dále je potom provádět pro různé sady hodnot. GeoMop je aktuálně ve vývoji a je

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

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




Obrázek 1: Simulátor Flow123d a pomocný software

možné, že se bude rozšiřovat o další funkce.

Výstupem ze softwaru 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.

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

Vzhledem k tomu, že úlohy zadávají lidé, je potřeba určit nějaké společné rozhraní, 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. Navíc díky tomu, že textový soubor umožňuje kromě přenosu samotných dat i tato data nějakým způsobem popsat, potom 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.

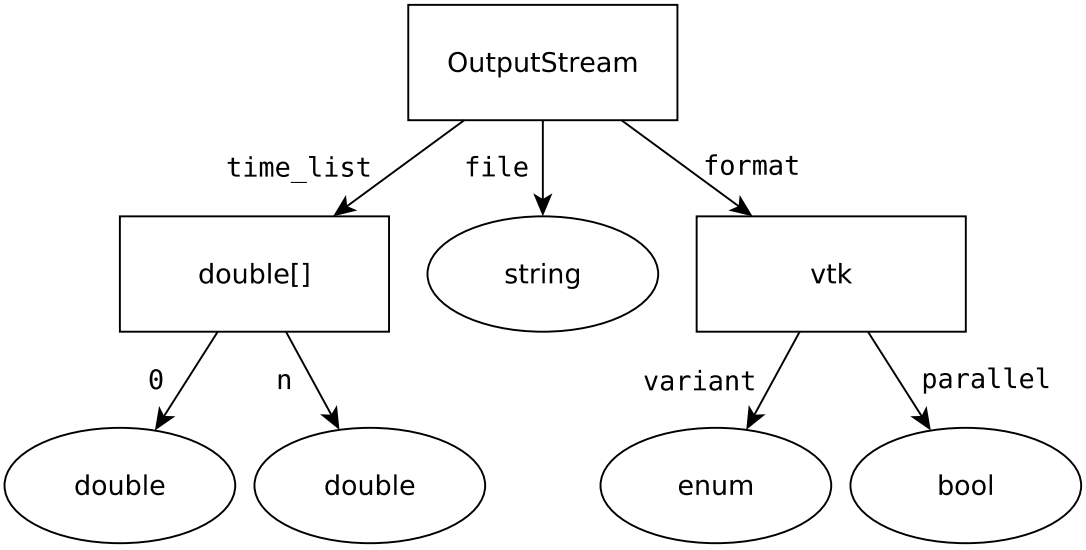
V konfiguračních souborech je tedy potřeba ukládat dvojice klíč a hodnota. Hodnota může být buď primitivního nebo složeného datového typu. Reprezentace primitivních datových typů je většinou triviální a spočívá pouze v převodu hodnoty na textový řetězec, pokud jím není. Podporované primitivní datové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

- booleovské hodnoty,
- celá čísla,
- desetinná čísla,

- hodnoty výčtového typu (tzv. enum),
- řetězce[4].

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

[4]Konkrétní implementace typu řetězec je typicky pole znaků, tedy složený datový typ. Z pohledu konfiguračních souborů se však jedná o jednoduchý datový typ, protože je dále nedělitelný.

## 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. Uživatele, 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 vyskytnou 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 všimnou 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.[5]

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.[6] Podpora exportu do formátu CON se neplánuje.

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

[6] 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 dokumentace 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.

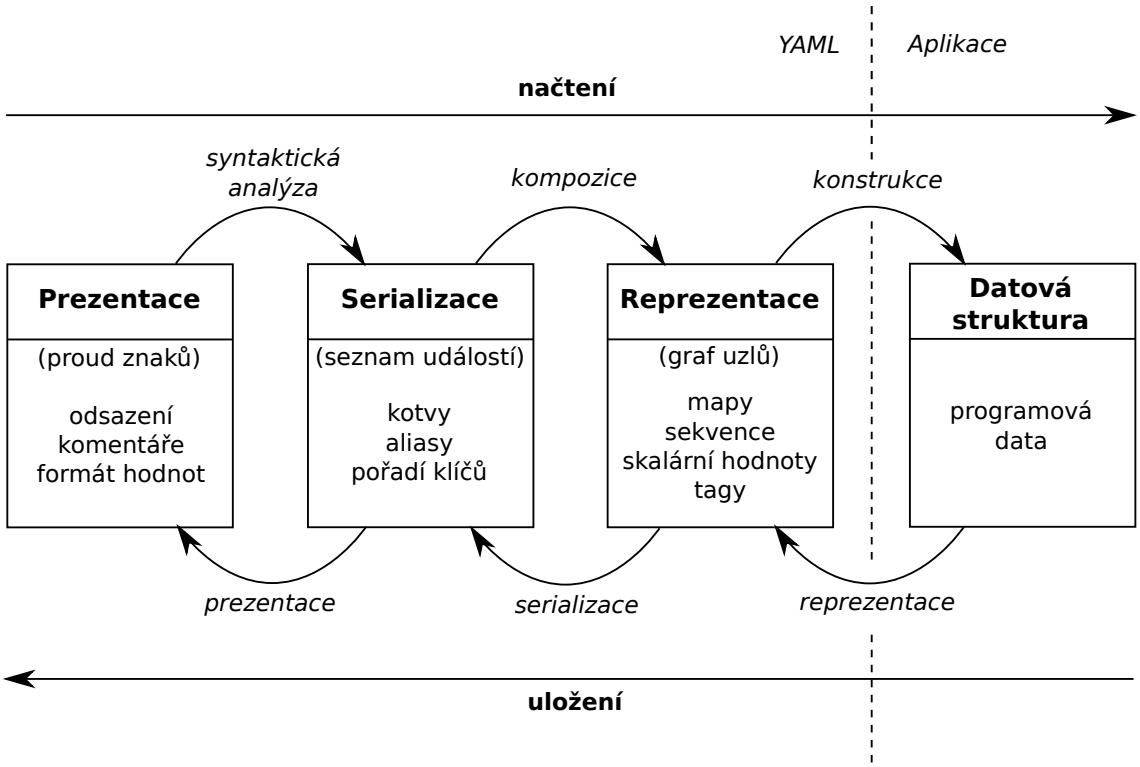
Mezi cíle formátu YAML patří mimo jiné jednoduchá čitelnost a přenositelnost, což jsou pro konfigurační soubory ideální vlastnosti. Dalším z cílů je čtení pomocí jediného průchodu, což s sebou oproti formátu CON přináší určité komplikace a změny, které jsou dále rozvedeny v kapitole **??**.

### 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 syntaktická analýza, která převede vstupní soubor na posloupnost událostí.




Obrázek 4: Zpracování formátu YAML

Syntaktická 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 **??**.

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-li 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 **??**.

### 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, kontextová dokumentace nebo automatické doplňování textu.

### 2.2.1 Základní datové typy

Jak již bylo zmíněno v kapitole **??**, 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 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 zkomplikuje 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 dokumentace 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

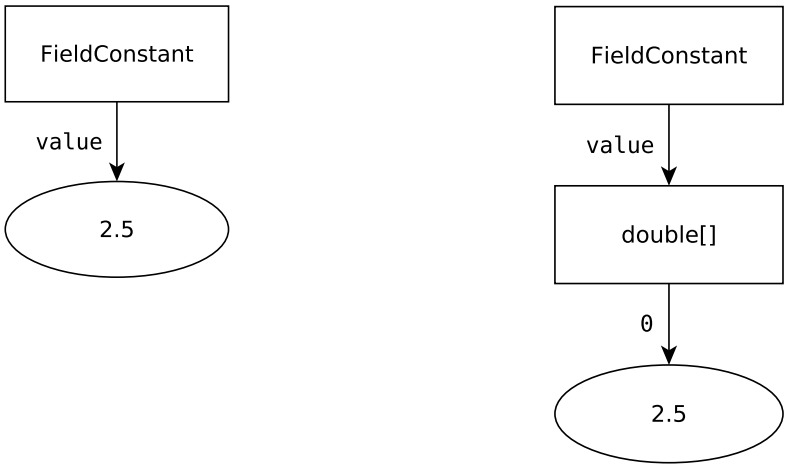
Autokonverze na pole je konverze, která se použije, pokud se dle specifikace formátu na dané pozici v datové struktuře očekává pole (typ *Array*) a místo něj je nalezen libovolný jiný datový typ – ať už se jedná o primitivní datový typ nebo o záznam.

**Zapsaná datová struktura**

```
value: 2.5
```

**Skutečná datová struktura**

```
value: [2.5]
```




Obrázek 5: Autokonverze na jednorozměrné pole

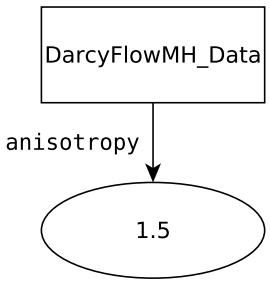
Obrázek 5 znázorňuje autokonverzi hodnoty na jednorozměrné pole. V klíči `value` v záznamu *FieldConstant* se očekává jednorozměrné pole, ale místo něj se na dané pozici vyskytla hodnota `2.5`. Provede se tedy autokonverze na pole a poté skutečná datová struktura obsahuje jednorozměrné pole o jednom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

Autokonverze na pole funguje obdobně i pro vícerozměrná pole. Příklad této autokonverze je uveden na obrázku 6. Klíč `anisotropy` v záznamu *DarcyFlowMH_Data* má být dvourozměrné pole. 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.

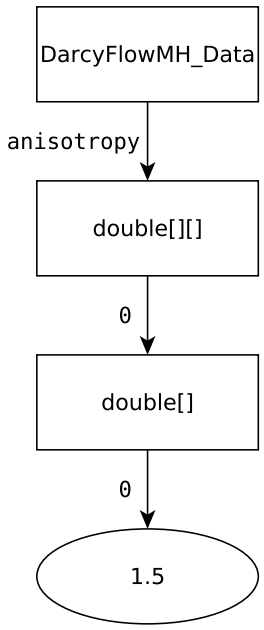
**Zapsaná datová struktura**

```
anisotropy: 1.5
```




**Skutečná datová struktura**

```
anisotropy: [[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[2] 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`.

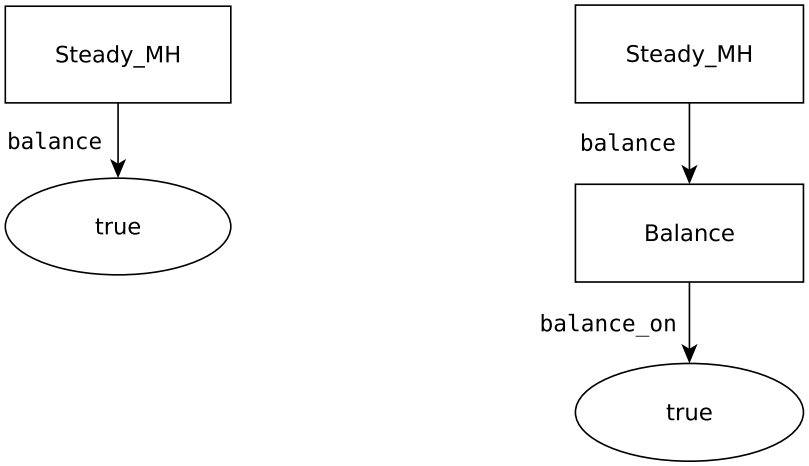
[2]Záznam je datový typ, který je odvozený ze základního typu *Record* nebo *Abstract*, viz dále.

**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 nadefinované 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 naráz 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.

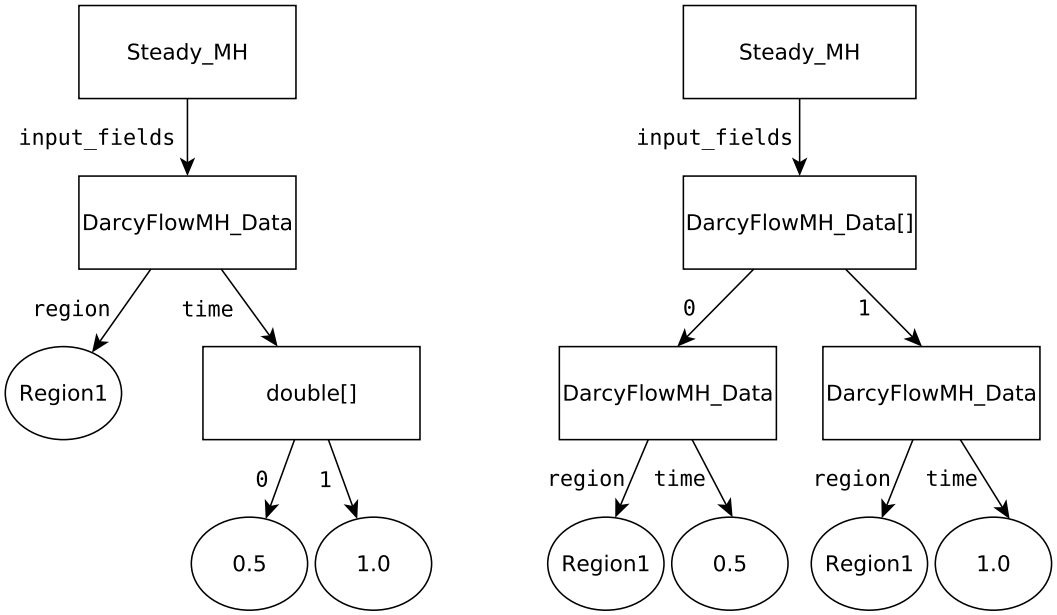
Transpozice je speciálním případem autokonverze na pole. Provádí se v případě, kdy se v datové struktuře očekává pole záznamů, ale místo něj je nalezen pouze jeden záznam. Pokud nastane tato situace, proběhnou další kontroly, zda je možné

**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
```




Obrázek 8: Autokonverze – transpozice

transpozici provést (viz kapitola **??**), 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`. bo 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 kapitole 2.1.2 byla uvedena základní syntaxe formátu YAML, která se používá pro zápis skalárních hodnot, sekvencí a map. V této kapitole je dále upřesněno použití syntaxe YAML pro zápis konfiguračních souborů včetně speciálních případů, které se používaly ve formátu CON.

Formát CON měl vyhrazené dva speciální klíče – `TYPE` a `REF`. První z nich, klíč `TYPE`, sloužil pro zadání typu konkrétního záznamu v datové struktuře abstraktního záznamu. Pomocí klíče `REF` bylo možné se odkázat na libovolnou část datové struktury.

### 3.1.1 Základní použití syntaxe

Pro reprezentace primitivních datových typů, polí a záznamů v konfiguračních souborech lze jednoduše využít výše popsanou syntaxi formátu YAML. 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.

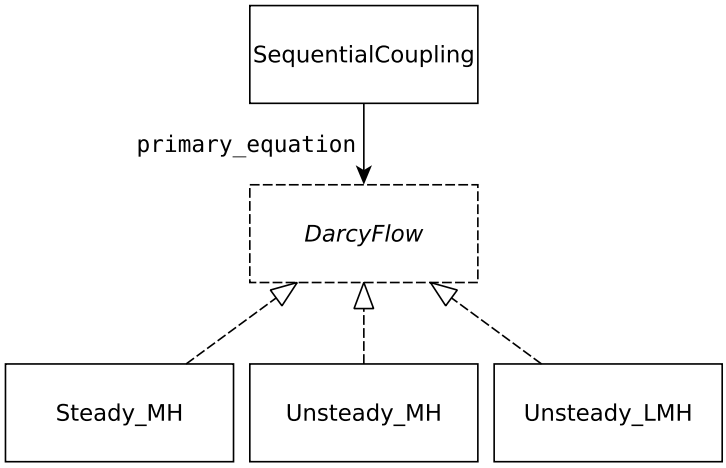
Pro reprezentaci polí se doporučuje použít syntaxe pro blokový zápis, kromě případů, kdy je zjednodušený zápis výrazně kratší bez újmy na čitelnosti (např. zápis pole celočíselných hodnot). Pro reprezentaci záznamů se doporučuje používat výhradně syntaxe pro blokový zápis. Zjednodušený zápis záznamů 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 stejné. 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 51.

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 51 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 některá data byla totožná s daty v jiné části datové struktury, stačilo použít referenci.

Reference se ve formátu CON používala 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.3 na straně 52.

Formát YAML poskytuje podobnou funkčnost 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
```

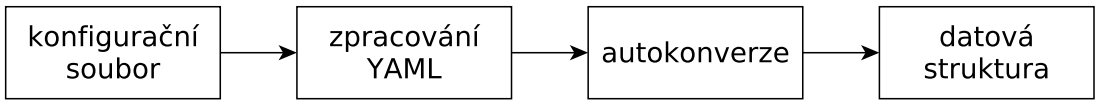
Použití kotev a aliasů bylo navrženo ve formátu YAML za stejným účelem jako reference ve formátu CON – ke zrychlení zápisu duplicitních dat. Formát YAML byl ovšem navržen tak, aby byl schopný zpracovávat i proud dat, nikoli pouze kompletní soubory. To vedlo k tomu, že kotvy je nutné vždy definovat dříve,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

Tím se použití kotev a aliasů liší od referencí ve formátu CON. V tom 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.



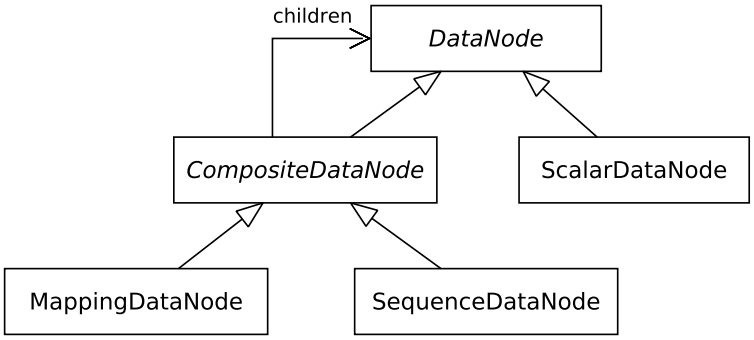

Obrázek 11: Získání datové struktury z konfiguračního souboru

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

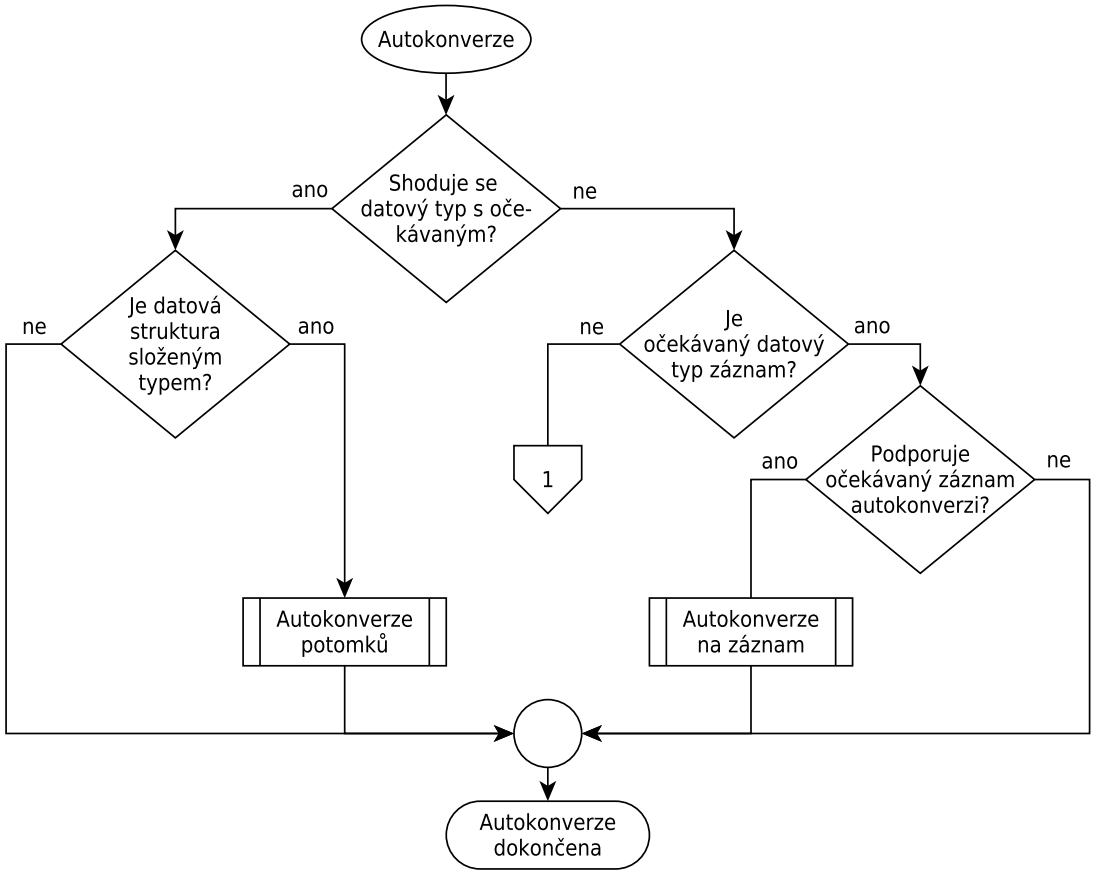
Obecná abstrakce datového uzlu je reprezentována třídou *DataNode*. Od této třídy jsou odvozeny implementace *ScalarDataNode* a *CompositeDataNode*. Třída *ScalarDataNode* reprezentuje list stromu a tím pádem obsahuje nějakou skalární hodnotu. Třída *CompositeDataNode* reprezentuje interní uzel stromu, který obsahuje další uzly *DataNode*. Existují dvě odvozené třídy – *MappingDataNode* pro záznamy a *SequenceDataNode* pro pole.

V aplikaci se pracuje s abstrakcí 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.

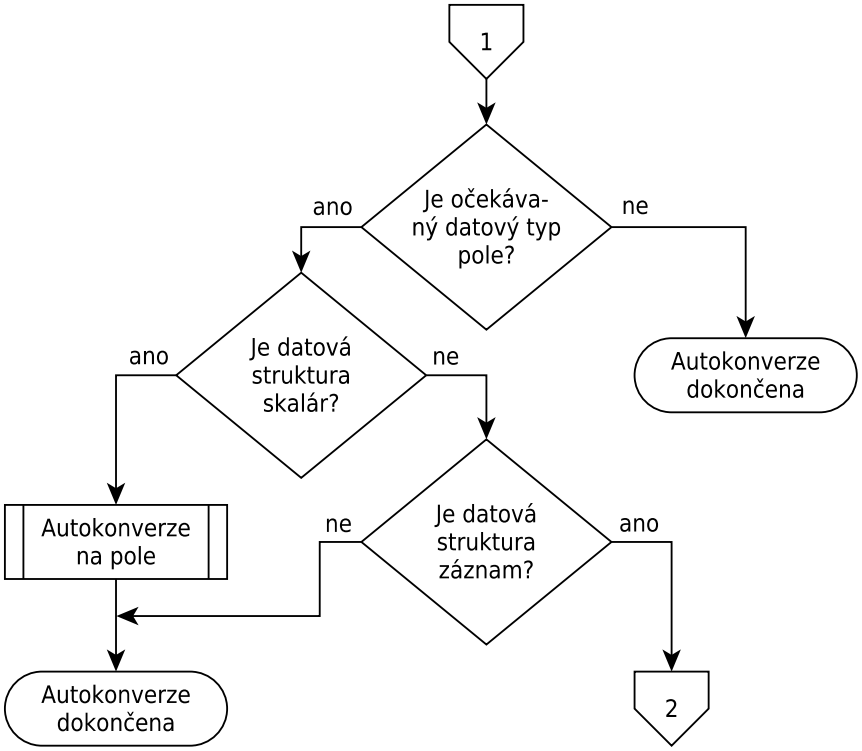



Obrázek 13: Proces autokonverze I.

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 kaptiole 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.

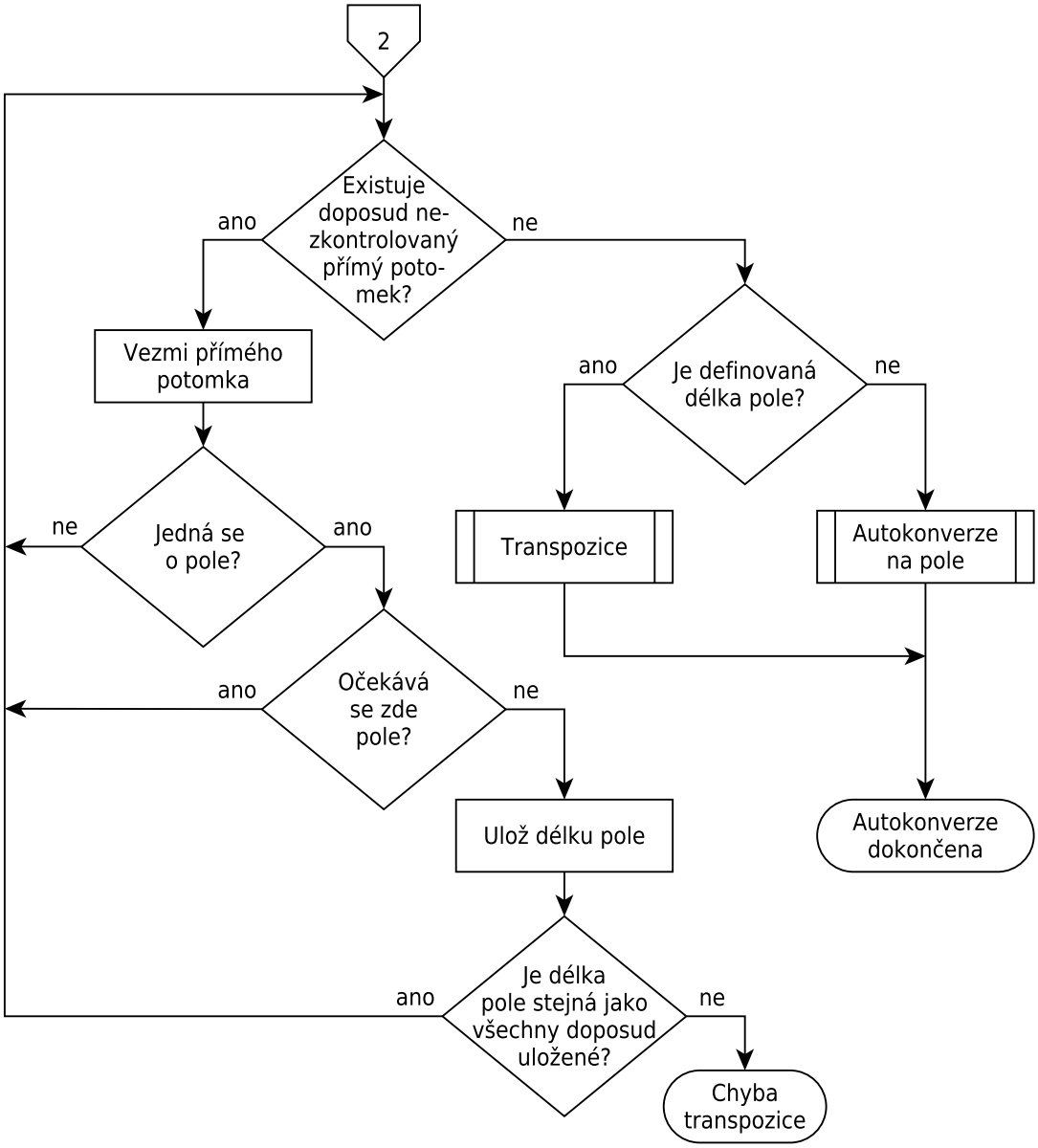
Pokud se očekává pole a místo toho je uzel typu záznam, tak situace je komplikovanější, protože je potřeba určit, zda stačí provést autokonverzi na pole, nebo zda-li se jedná o transpozici. Tento proces popisuje poslední obrázek týkající se procesu autokonverzí, obrázek 15.

Detekce transpozice spočívá v následujících krocích. Za prvé se již počítá s faktem, že v této fázi musí dojít buď k transpozici nebo autokonverzi na pole (pokud nedojde k chybě transpozice díky špatným vstupním datům). Díky tomu je znám očekávaný datový typ 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.

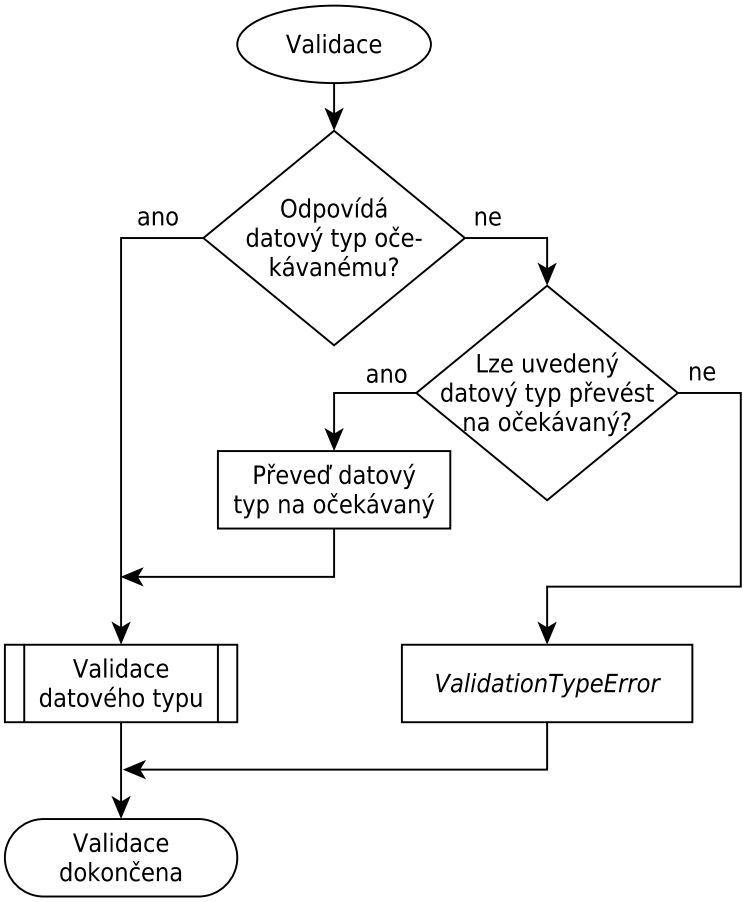



Obrázek 15: Proces autokonverze III.

## 3.4 Validace

Jednou z hlavních funkcí a přínosů editoru je funkce 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 detekované problémy či chyby se postupně ukládají. Po dokončení procesu validace je uživatel upozorněn na detekované chyby pomocí grafického uživatelského rozhraní aplikace.

Validace ke své funkci potřebuje obdobně jako autokonverze specifikaci formátu, na základě které ověřuje, zda se vstupní data mají 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 správného 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 pouze provede kontrola datového typu a případná konverze, jak bylo popsáno výše.

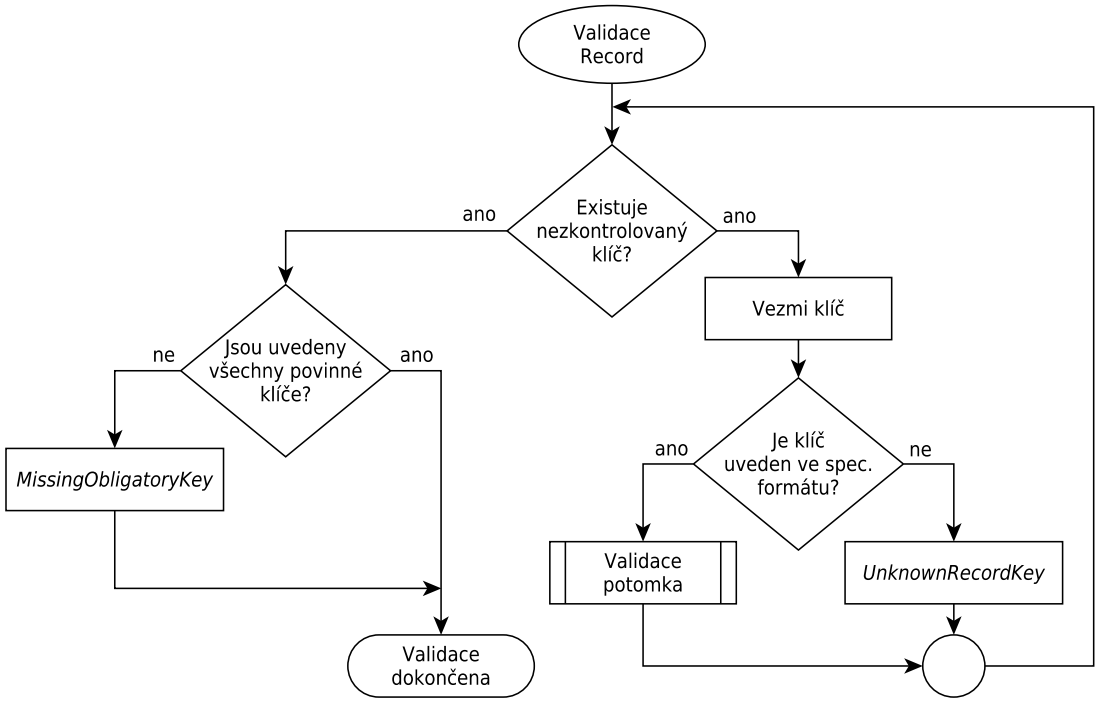
Číselné datové typy pro reprezentaci celých a desetinných čísel (*Integer* a *Double*) mohou mít ve specifikaci formátu zadaný interval platných hodnot, který je shora i zdola uzavřený. Provádí se tedy kontrola, zda se zadané číslo nachází v tomto intervalu. Pokud je hodnota menší než požadované minimum, dojde k chybě *ValueTooSmall*. V opačném případě, kdy je hodnota větší než požadované maximum, dojdě k chybě *ValueTooBig*. Validaci číselných datových typů znázorňuje obrázek B.1 na straně 53.

Posledním primitivním datovým typem, který má speciální validaci, je výčtový datový typ (*Selection*). U tohoto datového typu se oveří, 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 B.2 na straně 54.

### 3.4.2 Validace pole

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 B.3 na straně 54.

### 3.4.3 Validace záznamu




Obrázek 17: Validace záznamu – *Record*

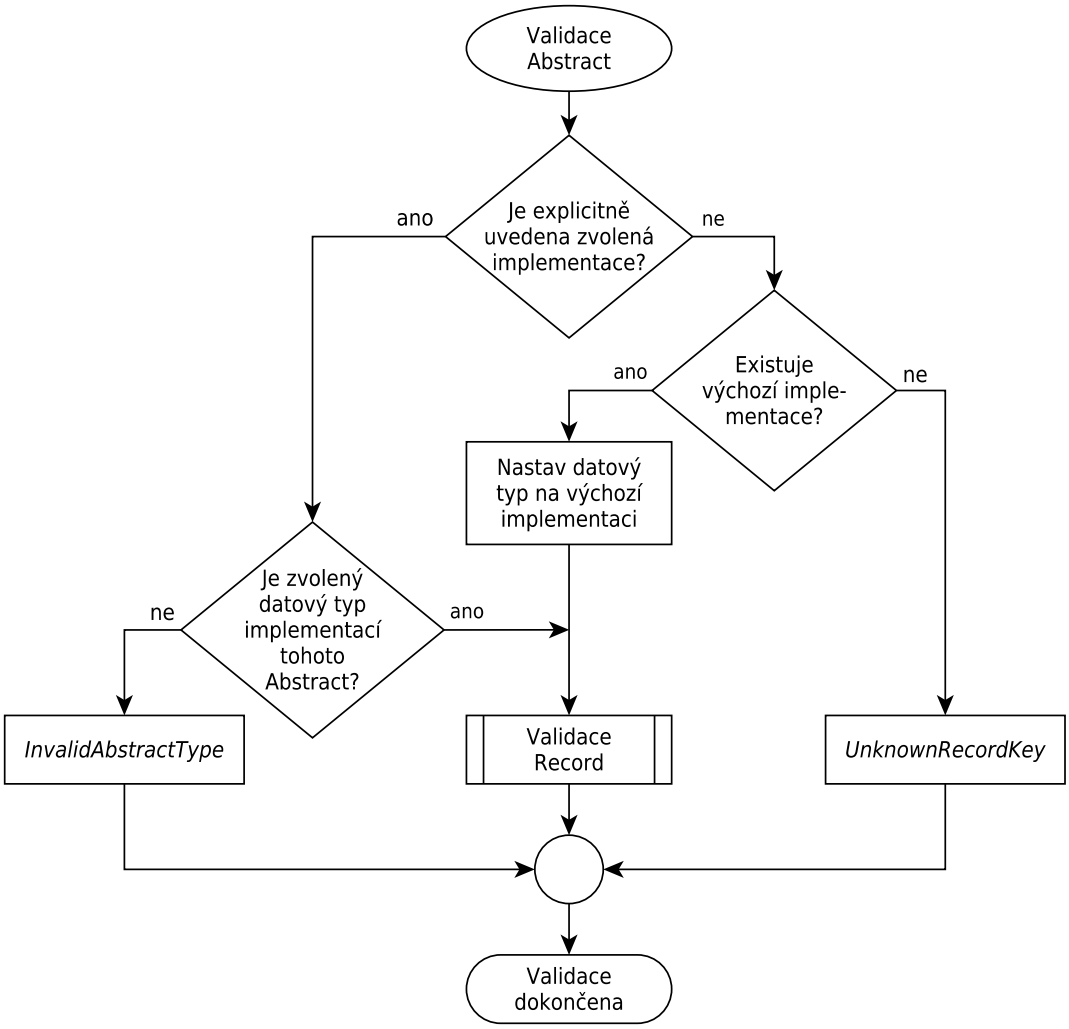
Záznamy jsou opět složeným typem, takže součástí procesu je opět validace potomků. Průběh validace záznamu je znázorněn na obrázku 17. Během tohoto procesu se postupně ověřují všechny klíče záznamu, které obsahují jeho potomky. Pro každý z těchto klíčů se najde jeho odpovídající datový typ ve specifikaci 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, tak je uživatel na chybějící klíče upozorněn pomocí chyby *MissingObligatoryKey*.

Validace umožňuje různé úrovně chybových hlášení – informativní zprávy, upozornění, chyby a fatální chyby. Ve výše popsaném příkladu může kromě chyb vzniknout i upozornění *UnknownRecordKey*. V této situaci se nejedná o chybu, protože pokud klíč ve specifikaci formátu není uveden, neznamená to, že ho konfigurační soubor nemůže obsahovat. 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 18. 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 18: Validace abstraktního záznamu – *Abstract*

## 3.5 Vizualizace datové struktury

## 3.6 Kontextová dokumentace

## 3.7 Automatické doplňování textu

# 4 Implementace a testování

## 4.1 Požadavky

## 4.2 Funkce editoru

## 4.3 GUI

## 4.4 Testování

# Závěr

V rámci této diplomové práce byl vytvořen editor konfiguračních souborů pro Flow123d. Jedná se o samostatně funkční aplikaci, která je ovšem navržena s ohledem na její použití jako součást 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 případně uživatele upozornit na detekované chyby. Tato funkce uživateli přináší časovou úsporu a uživatelsky příjemnější rozhraní při odhalování chyb.

Editor dále uživatelům poskytuje kontextovou dokumentaci a našeptávač. Obě tyto funkce přizpůsobují svůj obsah na základě pozice kurzoru v textu, tedy oblasti, kterou uživatel zrovna upravuje. Pro uživatele to představuje značné zjednodušení, jelikož může využít tyto funkce místo prohledávání rozsáhlé 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 editor poskytuje 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. S ohledem na požadavek multiplatformní aplikace byl 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 se by se mohlo přizpůsobit přímo formátu Flow123d. Další možné vylepšení spočívá v rozšíření funkcionality komponenty pro vizualizaci datové struktury. Ta by mohla v budoucnu podporovat kromě zobrazení i editaci dat nebo vylepšené zobrazení speciálních datových typů.

# Literatura


[1] *Standard ECMA-404: The JSON Data Interchange Format*. [online]. Geneva: Ecma International, 2013. [cit. 2016-02-23] Dostupné z: 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 z: http://flow.nti.tul.cz/packages/1.8.2_release/flow123d_1.8.2_doc.pdf

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

[4] *Extensible Markup Language (XML) 1.0*. Fifth Edition. [online]. W3C, 2008. [cit. 2016-02-26]. Dostupné z: 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 z: http://www.yaml.org/spec/1.2/spec.html

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

# 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

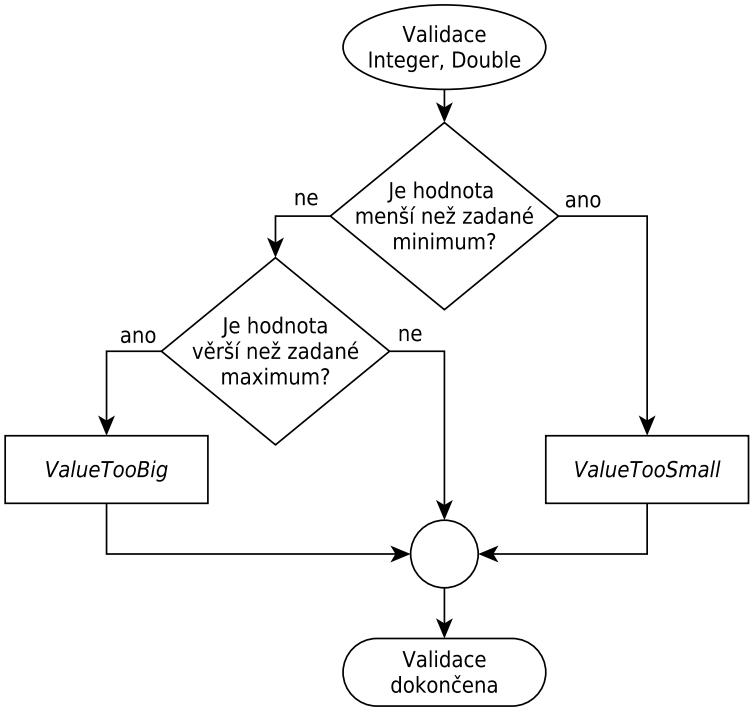
```
{
  primary_equation = {
    TYPE = "Unsteady_LMH",
    ...
    time = {
      end_time = 0.5,
      max_dt = 0.01,
      min_dt = 0.01
    }
  },

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

Obrázek A.3: Použití reference ve formátu CON

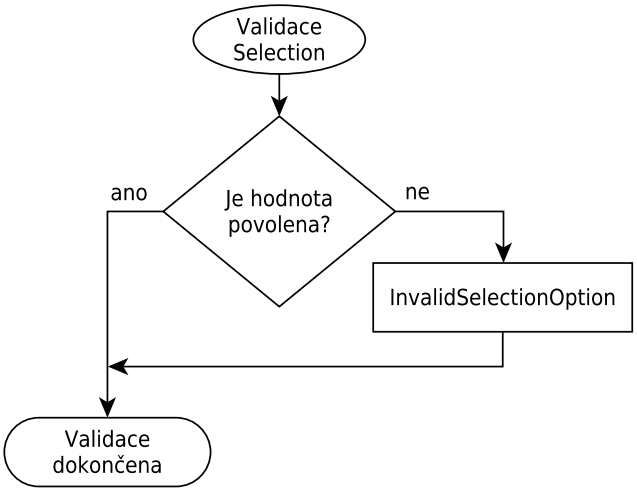
```
primary_equation: !Unsteady_LMH
  ...
  time: &time
    end_time: 0.5
    max_dt: 0.01
    min_dt: 0.01

time: *time
```

Obrázek A.4: Použití reference ve formátu YAML

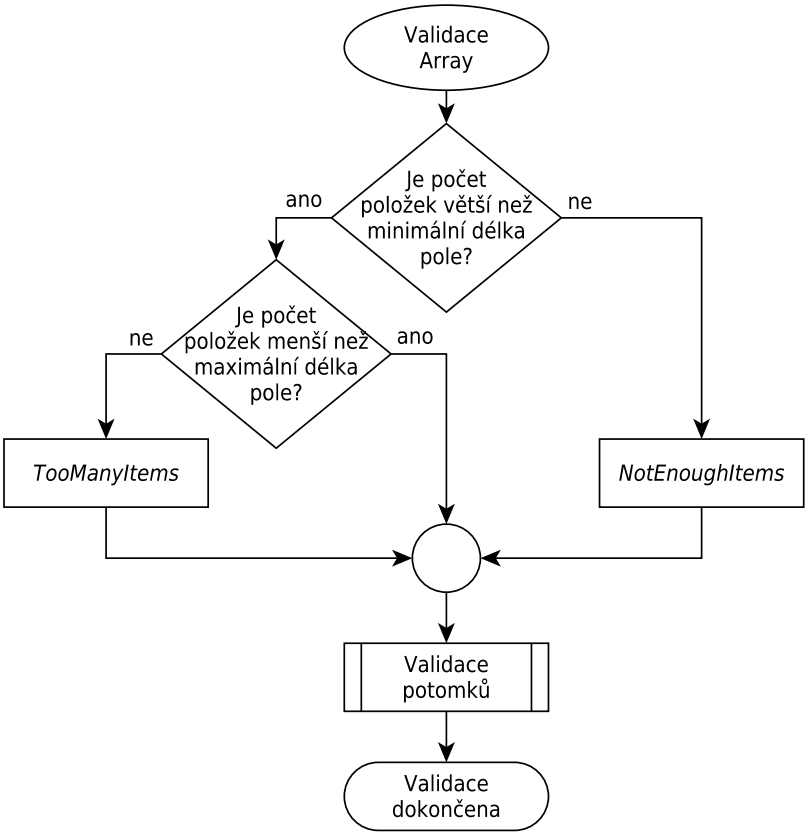
# B Dodatečné obrázky




Obrázek B.1: Validace číselných datových typů – *Integer* a *Double*




Obrázek B.2: Validace výběrového datového typu – *Selection*




Obrázek B.3: Validace pole – *Array*