TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Validace konfiguračních souborů Flow123d

Magisterský projekt

M14000168

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

Autor práce: **Bc. Tomáš Křížek**
Vedoucí práce: Jiří Vraný, Ph.D.

Liberec 2015

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

Validation of Flow123d configuration files

Project report

M14000168

Study programme: N2612 – Electrical Engineering and Informatics
Study branch: 1802T007 – Information Technologies

Author: **Bc. Tomáš Křížek**
Supervisor: Jiří Vraný, Ph.D.

Liberec 2015

TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky, informatiky a mezioborových studií

Ústav NTI Školní rok: 2014/15

ZADÁNÍ SEMESTRÁLNÍHO PROJEKTU

Název projektu: Validace konfiguračních souborů Flow123D

Řešitel(-é): Bc. Tomáš Křížek

Studijní obor: 1802T007 / Informační technologie

Vedoucí projektu: Mgr. Jiří Vraný Ph.D.

Zadání:

1. Seznamte se s aplikaci Flow123D, zejména s formáty konfiguračních souborů použitých v jednotlivých verzích.
2. Na základě získaných znalostí navrhněte sadu nástrojů pro validaci konfiguračních souborů, které budou dále využity jako součást připravovaného editoru v rámci aplikace GeoMop.
3. Návrh implementujte v jazyce Python a otestujte jeho funkčnost.

Doporučená literatura:

[1] BŘEZINA, Jan a kol. *Manuál pro Flow 123D* [online]. [cit. 2015-01-18]. Dostupné z: http://flow123d.github.io/

[2] PYTHON SOFTWARE FOUNDATION. Python 3.4.3 documentation [online]. 2015. [cit. 2015-01-18]. Dostupné z: https://docs.python.org/3/

Požadované náležitosti:

K obhajobě předložit zprávu o řešení projektu včetně anotace v českém a anglickém jazyce a ukázkovou aplikaci. V závěrečné zprávě i ve zdrojových kódech uveďte, který ze členů týmů je autorem konkrétní části kódu.

Termíny:

Odevzdání dokumentace a obhajoba podle harmonogramu výuky na FM.

Datum zadání: 18.1.2015 Podpis vedoucího projektu:

Prohlášení

Byl jsem seznámen s tím, že na můj magisterský projekt 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 mého magisterského projektu pro vnitřní potřebu TUL.

Užiji-li magisterský projekt nebo poskytnu-li licenci k jeho 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.

Magisterský projekt jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mého magisterského projektu a konzultantem.

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

Datum:

Podpis:

Abstrakt

Tento projekt se zabývá problematikou validace konfiguračních souborů Flow123d, jejichž struktura je dynamicky definována v závislosti na verzi Flow123d. V projektu jsou popsány specifika formátu konfiguračních souborů, jako jsou reference nebo automatické konverze. Dále je navržena datová struktura pro zpracování konfiguračních souborů a dynamicky definovaných datových typů. Součástí projektu je i dokumentace vytvořeného rozhraní.

Abstract

This project addresses issues surrounding the validation of Flow123d configuration files. Data structure of these files is dependent on the version of Flow123d. This paper describes the specifics of the configuration file format. These include references or automatic conversions. Next, a data structure for processing of configuration files and dynamycally defined data types is designed. The project also documents the created interface.

Obsah

1 Úvod

Tento projekt řeší problematiku validace konfiguračních souborů pro Flow123d [1]. Software Flow123d slouží k simulaci toku podzemních vod a transportu látek nebo tepla. Jeho konfigurační soubory mají dynamicky definovanou strukturu, která se liší podle verze softwaru. Cílem je vytvořit nástroj, který určí, zda zadaný konfigurační soubor je validní pro danou verzi.

Konfigurační soubory jsou ve formátu CON, který se podobá standardnímu formátu JSON. Od tohoto formátu se mírně odlišuje syntaxí. Konkrétní odlišnosti jsou popsány v kapitole 2. Kromě odlišné syntaxe také umožňuje vytvářet reference (viz kapitola 2.1), se kterými je potřeba náležitě pracovat při zpracování dat. Formát CON také umožňuje v některých případech použít zkrácený zápis, který se poté čtečkou formátu automaticky zkonvertuje na požadovaný tvar. Problematikou automatických konverzí se zabývá kapitola 2.2.

Jelikož se struktura konfiguračních souborů se liší podle verze Flow123d, validace nespočívá pouze v ověření dat podle pevně daných pravidel. Pro validaci je nutné dynamicky načíst datové typy, které lze v rámci datové struktury použít. Sada těchto datových typů je dodána ve formátu JSON. Popis struktury konfiguračních souborů a možných datových typů je rozebrán v kapitole 3.

V kapitole 4 se nachází dokumentace k vytvořenému rozhraní. Jsou zde popsány jednotlivé moduly, jejich funkčnost a použití. Také se zde nachází popis datových struktur pro reprezentaci obsahu konfiguračních souborů a jejich struktury.

2 Formát konfiguračních souborů

Konfigurační soubory, které je potřeba zvalidovat, jsou často psány ručně a nejsou strojově generovány. Formát CON, ve kterém jsou napsány, byl tomuto přizpůsoben. Syntaxe CON je podobná standardnímu formátu JSON, ale pro snažší zápis se mírně liší. Rozdíly v syntaxi oproti formátu JSON jsou následující.

- Klíče neobsahující mezeru mohou být napsány bez uvozovek.
- Klíč může být od hodnoty oddělen znakem „=“ místo „:“.
- V souboru se mohou vyskytovat komentáře. Jsou povoleny víceřádkové komentáře uzavřené mezi sekvencemi `/*` a `*/`, nebo řádkové komentáře, které jsou uvozeny sekvencí `//`.

Formát CON byl navržen pro inicializaci datových typů v C++. Kromě syntaktických rozdílů tedy předpokládá následující sémantická pravidla.

- Povolené datové typy jsou skalární hodnoty, pole nebo záznamy.
- Pole jsou vždy tvořeny prvky stejného typu.
- Záznam má určitý typ, který definuje seznam povolených klíčů.
- Každý klíč má daný typ, jméno a implicitní hodnotu.
- Klíče napsané velkými písmeny mají speciální význam a jsou zpracovány čtečkou CON formátu. Názvy ostatních klíčů jsou malými písmeny.
- U záznamů lze použít polymorfismus, kde klíč `TYPE` udává typ záznamu.
- Lze použít reference na data v rámci CON souboru pomocí speciálního klíče `REF`.

Bližší specifikace formátu CON lze nalézt v dokumentaci Flow123d [2].

2.1 Reference

Formát CON umožňuje místo libovolné hodnoty, záznamu nebo pole uvést referenci na jiná data. Po načtení souboru je nutné tyto reference zpracovat tak, aby jejich chování odpovídalo referencím a nikoli pouhým kopiím. Pokud se tedy změní původní data, změní se i reference a naopak. Zpracování referencí by mělo proběhnout ihned po načtení souboru, tedy před automatickými konverzemi.

Reference jsou v konfiguračních souborech zapsány jako záznam, který obsahuje jediný klíč – `REF`. V tomto klíčí je uvedena cesta, která určuje, na která data reference ukazuje. Cesta může být absolutní i relativní a ukazuje na libovolný uzel ve stromu. Pokud je součástí cesty záznam, v cestě je uveden klíč, který se vybere. Pro položky, které jsou součástí pole, se uvede jejich index (číslováno od 0). Pro oddělení úrovní ve stromu je použit znak `/`.

Reference mohou vypadat například následovně.

- `{REF = "/system/output_streams/0/format/TYPE"}`
- `{REF = "../../primary_equation/solver"}`

2.2 Automatické konverze

Pro usnadnění psaní konfiguračních souborů existují určitá pravidla, která umožňují zkrátit zápis a vynechat některé údaje, které jsou poté doplněny automaticky. Existují dva typy automatických konverzí.

První konverze proběhne, pokud se na vstupu očekává pole a místo něj se na vstupu nachází skalární hodnota nebo záznam. V tomto případě dojde k tomu, že se vytvoří pole s očekávanou dimenzí a do něj se vloží jediná položka – hodnota, která je na vstupu.

Druhá konverze proběhne, pokud je na vstupu pole nebo skalární hodnota místo záznamu. Tento záznam ale musí automatickou konverzi podporovat a to tím, že jeho definice obsahuje klíč `reducible_to_key`, který určuje, do kterého klíče v záznamu se má zapsat hodnota na vstupu. Ostatní hodnoty klíčů nabývají výchozích hodnot. Tato konverze funguje obdobně i pro abstraktní záznamy, které mají definované výchozího potomka.

3 Struktura konfiguračních souborů

Jak již bylo zmíněno v kapitole 2, data v konfiguračním soubor podléhají určitým sémantickým pravidlům. Uvedená pravidla jsou pouze obecná a jejich konkrétní podoba je závislá na verzi Flow123d. Požadavky na strukturu jsou popsány v dokumentaci Flow123d a pro potřeby aplikace je jejich specifikace dodána ve strojově čitelném formátu JSON.

V této kapitole je popsán formát souboru specifikací. Nachází se v něm seznam datových typů, jejich popis, vlastnosti a struktura. Z jednotlivých položek lze vytvořit sada povolených datových typů, které lze použít v konfiguračním souboru. Každý datový typ má specifikované `id`, které ho jednoznačně identifikuje, a `input_type`, který určuje základní datový typ, od kterého je daný typ odvozen. Základními datovými typy jsou skalár (viz tabulka 1), pole (*Array*), záznam (*Record*) a abstraktní záznam (*AbstractRecord*).

Data v konfiguračním souboru tvoří stromovou strukturu. Pro jejich ověření se postupuje od kořene stromu k listům. Popis datového typu kořenu stromu je vždy uveden v seznamu jako první. Aby konfigurační soubor odpovídal dané specifikaci, musí být všechny uzly stromu validní. Pokud validní nejsou, zaznamená se, k jaké chybě došlo a kde.

Tabulka 1: Datové typy skalárních hodnot

Základní datový typ	**Popis**	**Pravidlo pro validaci**
Integer	celé číslo	hodnota musí být v rozmezí `min` a `max`
Double	desetinné číslo	hodnota musí být v rozmezí `min` a `max`
Bool	přepínač	–
String	řetězec	–
Selection	výběr z množiny	musí obsahovat hodnotu z množiny povolených hodnot
FileName	cesta k souboru	–

3.1 Skalární hodnoty

U skalárních typů spočívá validace v ověření správného datového typu a ověření validačního pravidla, pokud nějaké existuje. Pro číselné hodnoty se kontroluje rozmezí, které je určeno hodnotami `min` a `max` u specifikace datového typu. Speciální validaci vyžaduje také typ *Selection*. U tohoto datového typu je dána množina povolených hodnot (`values`), kde každá hodnota má jméno (`name`), které ji jednoznačně identifikuje. Zadaná hodnota v *Selection* je validní, pokud se shoduje se jménem některé z povolených hodnot.

3.2 Pole

Typ *Array* má podobně jako číselné typy specifikované hodnoty `min` a `max`. U pole tyto hodnoty určují minimální, resp. maximální počet položek. Pravidlo pro validaci pole tedy spočívá v ověření počtu položek pole. Dále je nutné ověřit, zda jsou validní jednotlivé položky v poli. Pole jsou vždy homogenní, tedy všechny jeho položky jsou stejného typu, který je určen pomocí `subtype`.

3.3 Záznam

Typ *Record* má unikátní jméno (`type_name`) a obsahuje libovolný počet klíčů. Pokud je potomkem nějakých abstraktních záznamů, tak je jejich seznam uveden v položce `implements`. Položka `keys` definuje seznam povolených klíčů. V ní jsou dále uvedeny tyto položky:

- `key`: definuje název klíče,
- `description`: popis klíče,
- `type`: typ klíče,
- `default`: obsahuje dále `type`, který určuje typ výchozí hodnoty a `value` implicitní hodnotu.

Typ výchozí hodnoty může nabývat hodnot *obligatory*, *value at declaration*, *value at read time* a *optional*. Pro potřeby validace je důležitá pouze hodnota *obligatory*, která specifikuje, že klíč musí být v záznamu uveden. Všechny ostatní typy

klíčů nemusí být v záznamu uvedeny. Validace záznamů spočívá v ověření, zda jsou přítomny všechny klíče, které jsou povinné (*obligatory*), a dále v ověření typů všech klíčů, které jsou v záznamu uvedené.

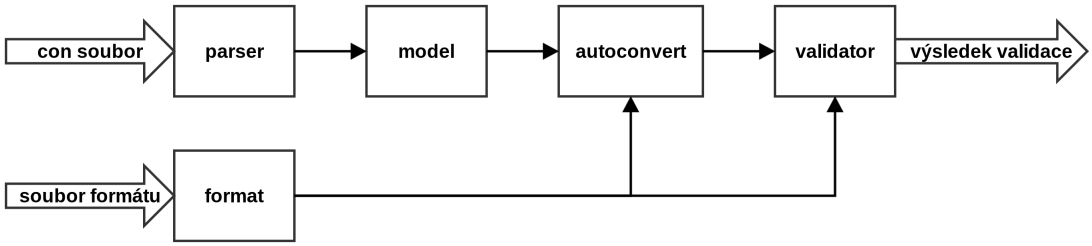
3.4 Abstraktní záznam

Speciální typ záznamu, který umožňuje polymorfismus, je *AbstractRecord*. Obsahuje položku `implementations`, ve které jsou uvedeny všechny zázamy, které ho implementují. Další důležitou položkou je `default_descendant`. Pokud je uvedena, tak definuje výchozí typ záznamu.

Při validaci záznamu je nutné nejprve určit jeho typ. K tomu slouží klíč `TYPE`, který je přímo uveden u daného záznamu v konfiguračním souboru. Pokud tento klíč není uveden, použije se typ, který je určen pomocí `default_descendant`. Pokud ani takto nelze určit typ záznamu, jedná se o chybový stav. Po určení typu záznamu se provede validace podle pravidel, které definuje přímo konkrétní datový typ.

4 Implementace

Aplikace pro validaci byla vytvořena v jazyce Python verze 3. Použití a detaily tohoto jazyka jsou popsány na webových stránkách jeho dokumentace [3]. Vytvořená aplikace se skládá z několika modulů, jejichž propojení je znázorněno na obrázku 1.



Obrázek 1: Propojení modulů

Vstupem do aplikace jsou dva soubory. Jedním z nich je samotný konfigurační soubor, na obrázku uveden jako *con soubor*. Druhým z nich je soubor, který obsahuje kompletní popis konkrétní verze formátu, tedy výčet všech možných datových typů, jejich strukturu a uspořádání, na obrázku je uveden jako *soubor formátu*. Výstupem z aplikace je informace, zda konfigurační soubor odpovídá danému formátu, tedy jestli je validní, případně informace o nalezených odlišnostech.

Přečtení souboru a vytvoření jeho datové reprezentace zajišťuje modul `parser`. Vstupem do tohoto modulu je cesta ke konfiguračnímu souboru. Modul tento soubor načte, interpretuje a vytvoří jeho datovou reprezentaci s pomocí modulu `model`.

Modul `format` zajišťuje načtení souboru formátu, který obsahuje popis struktury použitelných datových typů. Výstupem z tohoto modulu je specifikace formátu `InputTypeSpec`, která reprezentuje datový typ kořenového prvku ve stromové struktuře, kterou data tvoří.

Modul `autoconvert` provede v modelu automatické konverze. Aby bylo možné konverze provést, je potřeba kromě samotného modelu dodat tomuto modulu i očekávaný formát souboru.

Modul `validator` očekává na vstupu model, ve kterém již proběhly automatické konverze, a formát souboru. Modul ověří, zda dodaný model je v očekávaném formátu.

Následuje popis jednotlivých modulů a jejich veřejných funkcí, tříd a metod.

4.1 Modul `model`

`DataNode`

Třída reprezentuje jednotlivé uzly, které jsou součástí datového stromu. Obvykle se pracuje s kořenem stromu, ze kterého je možné se dostat k libovolnému jinému uzlu.

Atributy

`value` dle typu uzlu obsahuje buď přímo hodnotu, nebo potomky ve formě seznamu nebo slovníku. Potomci jsou opět typu `DataNode`. Pokud je nastavena `ref`, vrací se `value` reference.

`ref` umožňuje nastavit referenci na jiný objekt typu `DataNode`. Ve výchozím stavu obsahuje hodnotu `None`.

`path` vrací absolutní cestu k tomuto uzlu ve stromu.

`its` *nepovinný*; určuje `InputTypeSpec` (viz kapitola 4.2) tohoto uzlu.

Metody

`DataNode(data, [parent, [name]])` inicializuje třídu podle argumentu `data`. Pokud se jedná o stromovou strukturu, která je tvořena slovníky a seznamy, dojde rekurzivně k vytvoření celého stromu a tato třída je kořenem celého stromu. Volitelné atributy `parent` a `name` mají využití především pro inicializaci potomků ve stromové struktuře, kdy `parent` specifikuje nadřazený uzel a `name` udává název uzlu, který se použije při generování cesty ve stromu.

`get(path)` interpretuje zadanou cestu `path` a vrátí uzel na dané cestě ve stromu, pokud existuje. V opačném případě se vyvolá výjimka *LookupError*.

4.2 Modul `format`

`InputTypeSpec`

Třída reprezentuje konkrétní datový typ, který se může vyskytnout na vstupu. Obsahuje všechna omezení a pravidla, která jsou potřeba pro validaci tohoto datového typu. Obdobně jako data tvoří stromovou strukturu a obvykle se tak pracuje s kořenovým typem, který popisuje kompletní strukturu dat.

Atributy

Existence jednotlivých atributů je závislá na základním datovém typu, z kterého je tento typ odvozen.

`id`	unikátní identifikátor datového typu
`input_type`	základní datový typ, ze kterého je tento typ odvozen (možné typy viz kapitola 3)
`name`	*nepovinný*; název datového typu pro generování dokumentace
`full_name`	*nepovinný*; úplný název datového typu pro generování dokumentace
`description`	*nepovinný*; popis datového typu pro generování dokumentace
`min`	pro *Integer*, *Double*, *Array*; udává minimální hodnotu, resp. minimální počet prvků
`max`	pro *Integer*, *Double*, *Array*; udává maximální hodnotu, resp. maximální počet prvků
`values`	pro *Selection*; slovník obsahující možné hodnoty, každá hodnota má název (`name`) a popis (`description`)
`file_mode`	pro *FileName*; určuje zda-li je soubor pro čtení/zápis
`subtype`	pro *Array*; určuje `InputTypeSpec` položek v poli
`type_name`	pro *Record*; unikátní název datového typu záznam
`type_full_name`	pro *Record*; úplný unikátní název datového typu záznam
`keys`	pro *Record*; obsahuje popis klíčů záznamu (viz kapitola 3.3)
`implements`	*nepovinný*, pro *Record*; určuje, který *AbstractRecord* tento záznam implementuje
`reducible_to_key`	*nepovinný*, pro *Record*; určuje, který klíč se má použít při automatické konverzi (viz kapitola 2.2)

`implementations` pro *AbstractRecord*; slovník obsahující dvojice `type_name` a `InputTypeSpec`, které implementují tento abstraktní typ

`default_descendant` *nepovinný*, pro *AbstractRecord*; výchozí `InputTypeSpec`, který se použije, pokud není uveden jiný datový typ

`parse_format(filename)`

Funkce přečte soubor s cestou `filename` a vrátí kořenový datový typ `InputTypeSpec`.

4.3 Modul `autoconvert`

`convert(node, [its])`

Funkce provede automatické konverze (viz kapitola 2.2) na vstupních datech `node`. Funkce rekurzivně prochází vstupní strom, proto se volá se pro kořen stromu. Pokud není uveden datový typ `its`, použije se datový typ uzlu `node.its`.

4.4 Modul `validator`

`Validator`

Třída slouží pro validaci vstupních dat podle zadaného formátu. Kromě určení, zda jsou data validní, umožňuje identifikovat detekované odlišnosti od požadovaného formátu.

Atributy

`errors` seznam obsahující uzly (`DataNode`), ve kterých došlo k chybě, a příslušné výjimky (`Exception`), které nastaly

`console_log` vrací řetězec, který obsahuje zformátovaný text pro výpis chyb do konzole

Metody

`validate(node, [its])` Metoda určí, zda uzel `node` odpovídá formátu `its`. Vrací *True* nebo *False*, podle toho, zda uzel odpovídá danému formátu. Pokud není uveden formát `its`, použije se `node.its`. Po zavolání této metody je možné získat výjimky,

ke kterým došlo při validaci, pomocí atributu `errors`. Proces validace probíhá tak, jak byl popsán v kapitole 3.

4.5 Modul `parser`

`parse_con(filename)`

Funkce přečte soubor s cestou `filename` a vrátí datovou reprezentaci vstupního CON souboru. Při načítání souboru je využit balík demjson [4]. Funkce zároveň vytvoří příslušné reference, které byly zadány pomocí klíče REF, tak jak bylo popsáno v kapitole 2.1.

Výstupem z funkce je kořen (`DataNode`) načtené datové struktury. Jedná se pouze o reprezentaci dat z konfiguračního souboru, která jsou nezávislá na očekávaném formátu. Oproti tomu data, která projdou automatickou konverzí, jsou již závislá na konkrétním formátu.

5 Závěr

V rámci projektu bylo vytvořeno funkční rozhraní pro ověření konfiguračních souborů pro program Flow123d, které mají v různých verzích odlišnou strukturu. Aplikace podporuje speciální vlastnosti formátu CON, jako jsou reference nebo automatické konverze.

Pro potřeby aplikace byla vytvořena datová struktura pro reprezentaci obsahu konfiguračních souborů. Některé její vlastnosti byly navrženy s ohledem na budoucí použití, které je popsáno dále.

Součástí aplikace je i datová struktura pro popis struktury formátu, který je určen dynamicky podle verze Flow123d. Vytvořená datová struktura umožňuje načíst libovolnou verzi formátu a přizpůsobit jí proces validace. Datová struktura pro popis formátu bude dále použita generování nápovědy a dokumentace v aplikaci GeoMop.

Vytvořené rozhraní má být jednou ze součástí nově vznikající aplikace GeoMop, která bude sloužit uživatelům aplikace Flow123d. Načtení a validace datové struktury bude součástí kontextu Model, který bude umožňovat práci s různými verzemi konfiguračních souborů.

Navržený validátor umožní ověřit data, která uživatel zadá v grafickém rozhraní. Jeho hlavní funkcí a přínosem je však určení verze formátu konfiguračního souboru. Na základě ověření, zda data odpovídají odlišným verzím formátu, bude možné určit správnou verzi formátu souboru a případně tento soubor zkonvertovat na formát jiné verze.

Použitá Literatura

[1] Flow123d [online]. 2015. [cit. 2015-05-11]. Dostupné z: `http://flow123d.github.io/`

[2] BŘEZINA, Jan, Jan STEBEL, David FLANDERKA, Pavel EXNER a Jiří HNÍDEK. 2015. *Flow123d* [online]. Liberec [cit. 2015-05-11]. Dostupné z: `http://bacula.nti.tul.cz/~jan.brezina/flow123d_packages/1.8.2_release/flow123d_1.8.2_doc.pdf`. Documentation of file formats and brief user manual. Technical university of Liberec.

[3] PYTHON SOFTWARE FOUNDATION. *Python 3.4.3 documentation* [online]. 2015 [cit. 2015-05-14]. Dostupné z: `https://docs.python.org/3/`

[4] MERANDA, Deron. Demjson and jsonlint. *Python: module demjson* [online]. 2014 [cit. 2015-05-13]. Dostupné z: `http://deron.meranda.us/python/demjson/`