MIND$_A$NAO

—

ETL proces řízený meta daty

Ing. Petr Aubrecht

Centrum aplikované kybernetiky
Elektrotechnická fakulta
České vysoké učení technické

Abstrakt Oblast datových skladů se v několika posledních letech velmi úspěšně rozvíjí a využívá. Prostředí pro grafický návrh datového skladu, snadné dotazování a zároveň rychlé vyhodnocování a další části datových skladů jsou dnes na velice dobré úrovni.
Trochu stranou hlavního zájmu stojí skupina nástrojů, souhrnně nazývaná ETL — Extraction, Transformation, and Load. Jejich úkolem je zpracovávat data, přicházející často z velmi různorodých zdrojů, provádět převody mezi formáty, případně náročnější výpočty, a nakonec upravená data uložit do vnitřní databáze datového skladu.
Dále popsaný systém se zabývá zjednodušením a zefektivnění celého procesu ETL za pomoci meta dat. Meta data jsou použita jednak k popisu struktury dat, jednak k popisu vlastní transformace. Knihovna pro jejich spravování je součástí systému MIND$_A$NAO a obsahuje i metody pro přístup k popsaným datům. Meta data pak určují způsob zacházení s těmito daty.
Pro interpretaci meta dat, pro přístup a zpracování dat byl vytvořen speciální programovací jazyk — Sumatra, podobný jazykům Java a C. V tomto jazyku je možné přistupovat k datovým objektům, k meta datům, případně jsou zabudovány některé příkazy pro práci přímo se soubory.
Dále systém pracuje na principu „šablon“ (templates), který umožňuje jednou navrhnout typ transformace s jeho následným mnohonásobným využitím. Šablony určují kostru transformace, doplňovanou parametry z meta dat.
Celý systém MIND$_A$NAO je striktně ovládán meta daty. Ta jsou zatím upravována přímým zápisem do textových souborů, pracuje se na grafickém prostředím. To by mělo zastřešit z hlediska uživatele celý systém a skrýt implementační detaily.

Obsah

1 Úvod

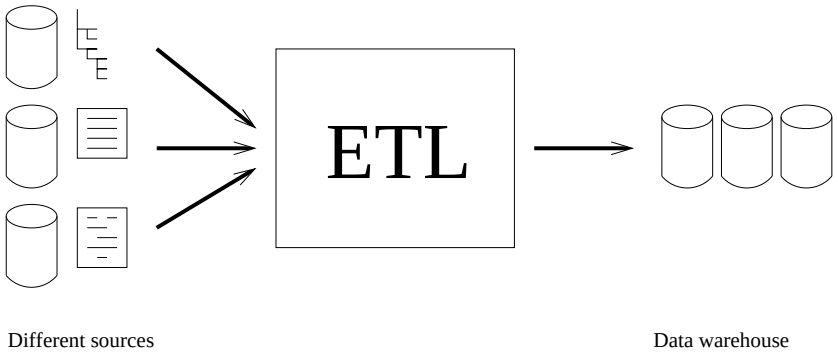
Nástroje ETL jsou určeny pro zpracování velkých objemů dat z různých zdrojů a jejich uložení do datového skladu. Obecná charakteristika takovýchto systémů je:

- schopnost zpracovávat různorodá data
- flexibilita návrhu transformací, tedy přenosu dat mezi datovými formáty

Tyto zdroje jsou obvykle umístěny fyzicky na různých místech. Prvním úkolem je tedy přenos dat na jedno místo.

Dalším krokem je sjednocení přístupu ke všem datům, protože ta přicházejí z různých systémů, které data pořizují. Kromě několika standardních formátů (DBF, comma delimited text, XML) využívají většinou vlastní formát dat (často záznamy pevné délky).

ETL nástroj musí být schopen zvládnout i různé zápisy jednotlivých polí. Příkladem může být zápis data a času, který se liší podle jednotlivých zemí a např. ODBC ani BDE ovladače nejsou schopny se s tímto úkolem spolehlivě vyrovnat.



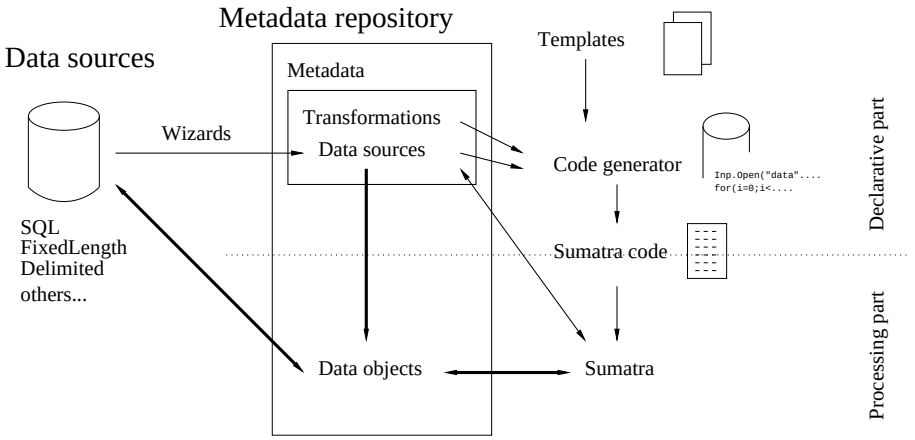
Obrázek 1. Obecné schéma ETL

2 Struktura řešení MIND$_A$NAO

Základní struktura systému MIND$_A$NAO je na obrázku 2.

Ústřední částí uvedeného řešení je tzv. „Metadata repository.“ V repository je uložen popis jak zdrojových, tak cílových dat a popis transformací.

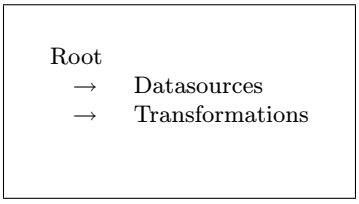
Repository také obsahuje objekty pro přístup k datům, která popisuje.



Obrázek 2. Celkový přehled systému MIND$_{\text{A}}$NAO

3 Metadata repository

Repository má stromovou strukturu,[1] jejíž základní struktura je přehledně ukázán na obrázku 3.



Obrázek 3. Základní struktura repository

3.1 Meta data

Hlavním úkolem repository je ukládání meta dat — tedy „dat o datech."

Meta data jsou v tomto systému dvojího druhu. Jednak jsou to deklarativní meta data, popisující strukturu a typ datových zdrojů, jednak procedurální, které popisují transformace datových zdrojů mezi sebou.

Popis datových zdrojů V tomto případě se ukládají informace o datových zdrojích, tedy o jejich typu a způsobu přístupu k nim, místu uložení, o atributech (fields) a jejich typu a formátu.

[1] stromovou strukturu mají například adresáře

```
Datasources
  →  VOSS
       →  type        "DelimitedText"
       →  delimiter   ";"
       →  filename    "AM9903.TXT"
       →  Fields
            →         Time
                        →  type      "DateTime"
                        →  format    "MM.DD.YYYY HH:MM:SS"
            →         MichalQ1
                        →  type      "Number"
```

Obrázek 4. Příklad deklarativních meta dat

Datové zdroje jsou tabulky, které mají množinu atributů (fields). Jejich formát je dán parametrem `type`, který určuje, jaký typ přístupu k datům se použije. `DelimitedText` znamená, že jde o textový soubor, pro který je v Metadata repository připraven odpovídající objekt pro přístup k datům.

`Fields` určuje, jaké atributy jsou v daném datovém zdroji k dispozici. V příkladu na obrázku 4 jsou atributy `Time` a `MichalQ1`.

Popis transformací Popis transformací se skládá z popisu šablony, která se použije a atributů, které chování šablony upravují.

```
Transformations
  →  Testing
       →  Desc        "Testing transformation for VOSS data"
       →  Template    "TransformTable.sum"
       →  Params
            →         From        "VOSS"
            →         To          "VOSSout"
            →         CodeFile    "testvoss.sum"
```

Obrázek 5. Příklad procedurálních meta dat

Transformace na obrázku 5 je příklad transformace, kde se vzor (šablona, viz dále) čte ze souboru `Transformtable.sum` a má parametry `From`, `To` a `CodeFile`.

Vlastní transformace proběhne tak, že se v šabloně nahradí některé části podle parametrů, čímž vznikne kód v jazyku Sumatra. Ten se posléze spustí.

3.2 Datové zdroje

Metadata repository obsahuje kromě nástrojů pro práci s meta daty také objekty pro ovládání datových zdrojů. Platí, že pro každý typ (je uveden v parametru `type` u každého datového zdroje) existuje právě jeden objekt, který je připraven pro práci s tímto typem.

Nejčastějším a proto nejdůležitějším typem je `DelimitedText`. Jde o ASCII[2] textový soubor, kde jednotlivé záznamy (records) jsou uvedeny po řádkách a atributy (fields) definovaným znakem. Tento znak (obvykle čárka, středník nebo tabulátor) musí být uveden v parametru `delimiter`.

Druhým základním typem zdroje je `SQL`, který čte data z databází disponujícími tímto jazykem. Tento datový zdroj je silně závislý na platformě. V současnosti je připravován pro rozhraní ODBC a BDE pod Windows. Součástí definice datového zdroje je jednak alias — rozlišení databáze a jednak samotný SQL příkaz.

V současné době je vyvíjen vzdálený typ datového zdroje, založený na protokolu JDBC, databázovém připojení pro jazyk Java. Tento datový zdroj vyžaduje na vzdáleném počítači spuštěný server (součástí MIND$_{\text{A}}$NAO), ke kterému se připojuje přes TCP/IP.

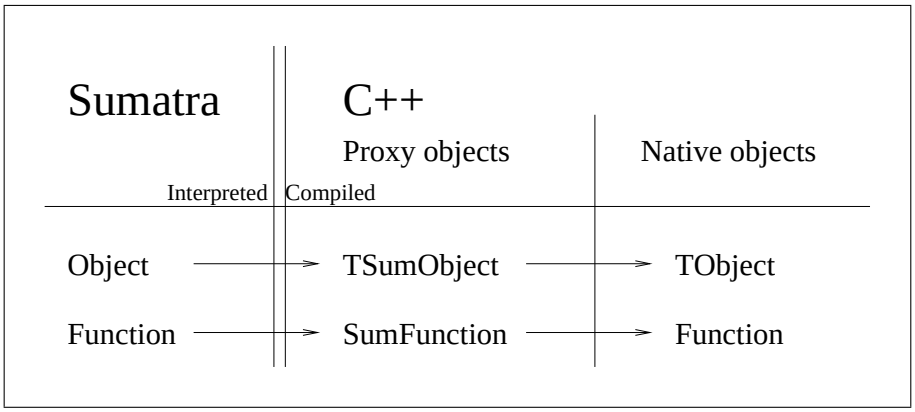
Systém MIND$_{\text{A}}$NAO disponuje průvodci (Wizards, viz přehledový obrázek), kteří automaticky rozpoznají tvar dat a sami uloží příslušnou informaci.

4 Sumatra

Základy jazyka Sumatra jsou popsány v Sumatra Basics [1]. Jde o skriptovací jazyk vycházející syntaxí z jazyků C++ a Java, jehož možnosti lze rozšiřovat pomocí objektů a funkcí definovaných v C++ a registrovaných v Sumatře. Vztahy mezi objekty v C++ a v Sumatře ukazuje obrázek 6.

Levá část obrázku odpovídá skriptu v jazyku Sumatra. Objekt, pojmenovaný `Object` odpovídá v interních strukturách Sumatry objektu `TSumObject`, který ovládá objekt aplikace `TObject`. V případě Metadata repository lze v kódu Sumatry použít `DataSource`, ovládající (na pravé straně) objekt `TDataSource`. Takto lze pracovat ze skriptu v Sumatře i s dalšími objekty repository.

[2] ASCII — American Standard Code for Information Interchange, standardní kódování znaků



Obrázek 6. Struktura jazyka Sumatra

5 Šablony

5.1 Princip

Při vytváření transformací se uživatel brzy dostane do situace, kdy pracuje na stále se opakujících podobných úkolech. Omezení pouze grafického rozhraní je v tom, že sice lze snadno navrhnout transformaci, ale už ji nelze použít jako šablonu pro další. Kopírování celého původního projektu je akceptovatelné, dokud se nevyskytne potřeba upravit něco ve všech doposud vytvořených. Příkladem může být sada transformací zpracovávajících data z několika stejných zdrojů. Pokud se změní formát, je třeba bezchybně(!) a naprosto stejně opravit všechny transformace.

Abychom takové problémy obešli, vyvinuli jsme princip šablon. Jsou to programy v jazyku Sumatra, které navíc obsahují speciálně definované značky. Tyto značky se před transformací (tedy použitím šablony) nahradí podle parametrů transformace.

Tak je možno vytvářet knihovny obecných transformací, které budou upřesněny až parametry podle konkrétních požadavků.

Při vytváření šablon lze použít kromě značek pro čtení parametrů také značky, které automaticky generují větší kód. Příkladem může být značka pro vytvoření proměnných podle definice datového zdroje. Na místě této značky se automaticky vygeneruje seznam definic proměnných. Jinou, podobnou značkou lze těmto proměnným přiřadit aktuální hodnoty načtené z datového zdroje.

Příklad jednoduché šablony na kopírování celé tabulky je na obrázku 7. Skript, vytvořený z této šablony, otevře vstupní a výstupní datový zdroj (vstupní pouze pro čtení), bude postupně procházet vstupní zdroj a všechna data, jejichž jména si odpovídají, budou kopírována.

Používání značek v šablonách poskytlo další možnosti, které nejsou jinak běžně dostupné. Největší význam mají značky, pomocí kterých lze zpracovávat

```
##DSopen(##param(From),RO)          // open source data
##DSopen(##param(To))               // open destination data
##DSvariables(##param(To))          // create destination variables
if(##param(From)
         .FirstRecord())
{
 do
 {
  ##DSvariables(##param(From))      // create source data vars
  ##DSvariablesLoad(##param(From)) // load source vars
  ##DScopyMatchingFieldsEx()        // copy all matching fields
  ##param(To).InsertRecord();}
  ##DSvariablesSave(##param(To))    // save data into DS
  ##param(To).PostRecord();}
 }
 while(##param(From)                //process all source data
                .NextRecord());
}
##DSclose(##param(To))              // close datas
##DSclose(##param(From))
```

Obrázek 7. Příklad šablony kopírující tabulku

historii — několika značkami lze vytvořit situaci, kdy k aktuální pozici v datovém zdroji je k dispozici také historie, tedy několik (pevně daných) pozic zpět.

Šablony vytváří (respektive vůbec s nimi přijde do styku) pouze administrátor, který bude celý systém spravovat. Běžní uživatelé budou v systém vytvářet transformace a ze šablon budou jen vybírat.

Současně je několik základních šablon součástí systému a pokud se nevyskytne speciální požadavek, není potřeba je měnit.

5.2 Zpracování šablony

Šablona poskytuje pouze základní informaci, kostru nějaké transformace. Teprve po dodání parametrů je možné ji zpracovat a vytvořit tak konečný skript v jazyku Sumatra.

Pokud chce uživatel vytvořit novou transformaci, vybere si šablonu a podle dokumentace k dané šabloně doplní parametry. Všechny tyto informace jsou udržovány v metadata repository ve větvi Transformations.

Tím je transformace dostatečně specifikovaná. Dál systém spustí „generátor kódu," který vygeneruje na základě šablony a parametrů instanci — skript transformace v jazyku Sumatra. Tento skript lze uložit a později opakovaně spouštět již bez závislosti na změnách šablony nebo parametrů.

6 Příprava transformace

V této části se budeme věnovat celému procesu přípravy transformací.

Prvním krokem je příprava definice dat. K tomu účelu byly vyvinuty pomocné programy (na obrázku 2 jsou označené jako Wizards).

Pro požadovanou transformaci nalezneme vhodnou šablonu. Nejjednodušší cesta je najít takovou šablonu, která dělá co nejvíce z požadované funkčnosti automaticky.

Dále specifikujeme její parametry. Přitom některé parametry mohou být i části kódu. Pro tyto parametry je lepší ukládat její obsah do souboru a ne přímo do repository alespoň po dobu ladění kvůli snazší editaci.

Systém vygeneruje cílový skript v Sumatře (viz kapitola 5.2). Tento skript je již připravená celá transformace a lze ji opakovaně spouštět.

7 Grafické prostředí

Dosud popsaný systém je sice dostatečně obecný a po získání zkušeností i dostatečně jednoduchý, ale je těžké v něm navrhovat systém transformací od začátku.

Pro tento účel je vyvíjeno grafické prostředí, pomocí něhož je možné nadefinovat datové zdroje, připravit transformace a nakonec je i spustit. Prostředí je vytvářeno v jazyku Java, takže ho lze spustit v libovolném systému (disponujícím grafikou).

Jedním z cílů tohoto nástroje je příprava jednoduchých transformací, kde jeden záznam ve vstupním souboru odpovídá jednomu záznamu ve výstupním, prostým tažením čar mezi odpovídajícími si datovými zdroji, následně i mezi atributy. Výsledný software by měl svými grafickými návrhovými schopnostmi odpovídat například minimálně transformačnímu návrháři (DTD) z programu Microsoft SQL Server.

8 Příklady použití

V této části budou popsány příklady použití celého systému. Budeme postupovat od nejjednodušších případů ke složitějším, abychom ukázali snadnost návrhu a zároveň sílu celého řešení. Všude, kde to bude možné, budeme používat již hotové šablony, které jsou součástí systému.

Jednotlivé kapitoly postupují od jednodušších případů ke složitějším. Přitom platí, že každá kapitola je zobecněním té předcházející.

Data, která budeme zpracovávat, pocházejí z projektu INCO–COPERNICUS 977091 GOAL (Geographical Information On-Line Analysis), aplikace A2 ([3]). V rámci této aplikace jsou k dispozici data z měření průtoků vodojemy. Celá síť je monitorována několika měřícími stanicemi. Každých deset minut se provede měření na všech stanicích. Každému bloku měření odpovídá ve výsledném textovém souboru jeden řádek, který má následující formát:

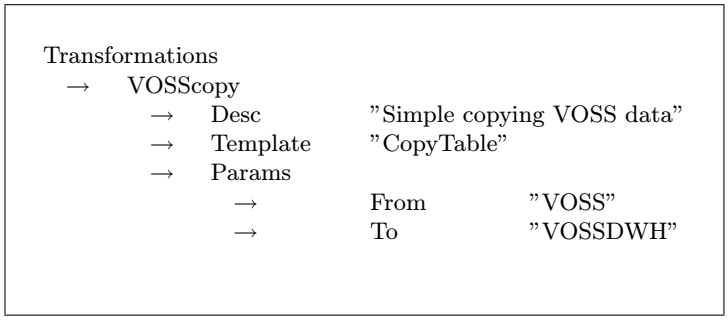
Datum a čas	měření 1	měření 2	...
01.03.1999 00:00:06	53.9	0	...
...	...	...	...

8.1 Kopírování dat

Prvním úkolem je přesunout data z jednoho systému (v našem případě textového souboru) do jiného (SQL serveru). Předpokládejme, že cílový datový zdroj má stejný formát a jde tedy opravdu o prosté kopírování obsahu.

Pro tento účel použijeme šablonu `CopyTable` a jako parametry udáme datové zdroje, mezi kterými se bude kopírovat (z `VOSS` do `VOSSDWH`). Výsledek ukazuje obrázek 8.

Transformaci, kterou jsme nyní vytvořili, stačí vygenerovat a spustit. Bude kopírovat obsah z datového zdroje `VOSS` do datového zdroje `VOSSDWH` všechna data beze změny.



Obrázek 8. Příklad kopírování dat

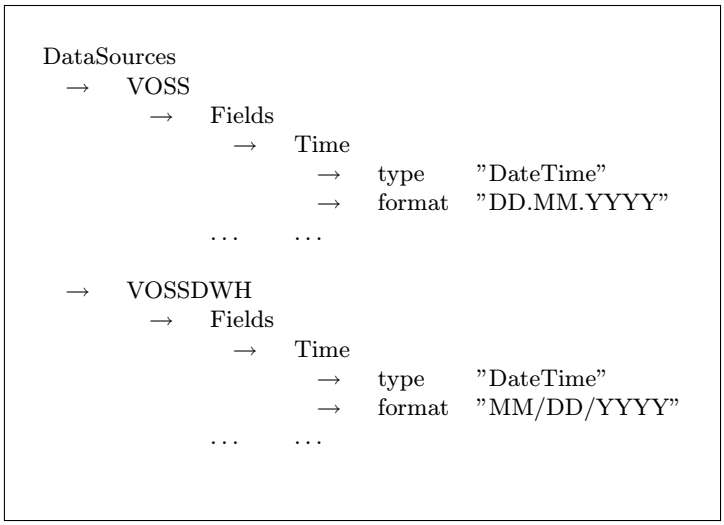
Aby nebyl první příklad *tak* jednoduchý, ukážeme si jeden typický problém, který se musí návrháři datových transformací řešit velmi často, zvlášť tuzemští. Tím problémem je formát data.

Řekněme, že struktura dat dvou datových zdrojů je sice stejná, ale formát data se liší. V rámci projektu GOAL bylo třeba při prvních pokusech napsat speciální program, protože ani ovladače ODBC od Microsoftu, ani ovladače BDE od Borlandu pro textové soubory nebyly schopny přečíst datum ve formátu používaném v České republice, aniž by bylo třeba měnit systémové nastavení. A ani potom nefungovaly spolehlivě.

Ukažme si, jak takovou situaci zvládnout v systému MIND$_{\text{A}}$NAO. Formát data, který se chceme číst, je `den.měsíc.rok`, zapisovat se bude ve formátu `měsíc/den/rok`.

Jednoduše. Stačí použít výše zmíněnou transformaci beze změny, pouze patřičně nastavit formát data u obou datových zdrojů (pokud nebyly nastavené

automaticky průvodcem), jak je uvedeno na obrázku 9. Transformaci mezi formáty provede systém sám.

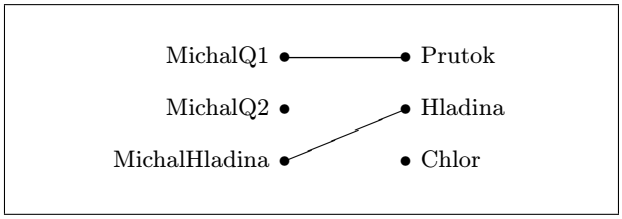


Obrázek 9. Různé formáty data pro transformaci

8.2 Mapování atributů

Prosté kopírování dat nyní rozšíříme o možnost definovat vazby mezi atributy ve vstupním a výstupním zdroji. Takovou vazbu budeme nazývat *mapování*.

Příklad mapování je na obrázku 10.



Obrázek 10. Mapování atributů

Z obrázku je jasně patrné, které atributy budou kopírovány do kterých. Tento typ transformace můžete v grafickém prostředí navrhnout tažením čar mezi odpovídajícími si atributy.

Grafické prostředí pro tento typ použije šablonu `TransformTable`, která má oproti `CopyTable` navíc parametr `Code` určující, co se bude s atributy dělat. Pro případ na obrázku 10 vygeneruje následující kód:

```
VOSSDWH_Prutok = VOSS_MichalQ1;
VOSSDWH_Hladina = VOSS_MichalHladina;
```

Tady se poprvé setkáváme s kódem, který se bude v nezměněné podobě vkládat do výsledného skriptu. Pro účely tohoto příkladu postačí vědět, že atributy datového zdroje se označují ve tvaru `DatovýZdroj_Atribut`.

Mapování se tedy provádí prostým přiřazením. Je pouze třeba dbát na typy při přiřazování. Sumatra obsahuje implicitní přetypování mezi číselnými typy, pro převod na řetězec je třeba použít vestavěné funkce (viz Sumatra Basics [1]).

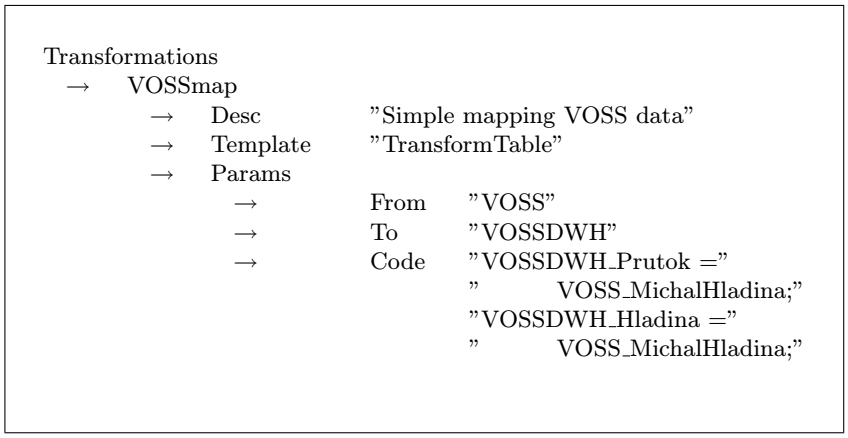
Výsledný popis transformace je na obr. 11.

8.3 Jednoduchá transformace dat

Jednoduchou transformací dat budeme rozumět transformaci, která transformuje jeden řádek ze vstupu na jeden řádek na výstupu.

Narozdíl od prostého kopírování, nyní bude potřeba vytvořit kód ručně. Až bude hotové grafické rozhraní, bude takový návrh možné vytvořit tažením čar mezi odpovídajícími si atributy.

V případu dat z VOSS se nyní budeme zabývat pouze jedním vodojemem (Michal), ze kterého jsou k dispozici měření průtoku MichalQ1 a MichalQ2. Protože však měření uvádí aktuální průtok a na výstupu je požadován objem za posledních deset minut, musíme hodnotu matematicky upravit.



Obrázek 11. Příklad kopírování dat

Atributy `date_key` a `time_key` jsou číselné hodnoty odvozené z data a času. Sumatra pro potřeby konverze data a času na čísla disponuje vestavěnou funkcí `DateTime2Int`. Pro výpočet hodnoty `time_key` použijeme z výkonnostních důvodů raději lokální proměnnou.

```
VOSSDWH_date_key = DateTime2Int(VOSS_Time);
datetime time = VOSS_Time;
VOSSDWH_time_key =
     (((time.hour*60)+time.min)*60)+time.sec;
VOSSDWH_reservoir = 1; // identifikace vodojemu
VOSSDWH_tank = 1;
VOSSDWH_inflow = VOSS_MichalQ1*600;
VOSSDWH_outflow = VOSS_MichalQ2*600;
VOSSDWH_chlorine = MichalChlor;
```

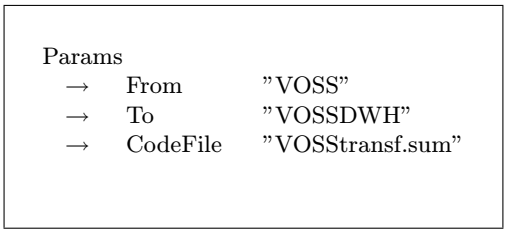
Celou výslednou transformaci ukazovat nebudeme, protože je identická (až na `Code` uvedený v minulém odstavci) s předcházejícím případem.

Protože tento typ transformace již může být značně rozsáhlý, je možno ukládat skript do souboru s tím, že při generování se z něj přečte.

Princip uložení parametru do souboru je obecný. Je-li potřeba umístit nějaký parametr mimo metadata repository (nejčastěji části kódu), lze místo jména parametru uvést jméno rozšířené o `File` a uvést jméno souboru.

V našem příkladu uložíme kód do souboru `VOSStransf.sum`. Parametry po úpravě jsou uvedeny na obr. 12.

Uložení parametru do souboru lze s výhodou používat především v případě většího kódu, protože pak lze využít libovolný editor.



Obrázek 12. Parametry s uložením v souboru

8.4 Více záznamů z jednoho vstupního

V této kapitole se zase posuneme o kousek blíže řešení celého problému. Tak, jak byl problém řešen dosud, bychom museli pro každý vodojem vytvořit zvláštní transformaci, a pak je spouštět všechny. To však vede, zvláště v případě složitějších pomocných výpočtů, ke zpomalení. Ještě více je nepříjemná nutnost vytvářet několik téměř shodných skriptů.

Budeme se tedy věnovat generování více výstupních řádek z jedné vstupní. V našem případě to znamená, že v jedné řádce na vstupu jsou informace o několika měřeních.

Pro tyto účely je k dispozici šablona `TransformTable1toN`. Je velmi podobná šabloně `TransformTable`, jen část věnovaná ukládání je umístěna do funkce `SaveOutputValues` a je třeba ji volat explicitně. Schéma použití je následující (parametr Code):

```
... transformace 1 ...
SaveOutputValues();
... transformace 2 ...
SaveOutputValues();
... transformace 3 ...
SaveOutputValues();
```

Pokud toto schéma aplikujeme na původní zadání, můžeme vytvořit následující kód, který je schopen zpracovat všechny vodojemy (z prostorových důvodů zkrácené).

```
// Společná část
int date_key = DateTime2Int(VOSS_time);
datetime time = VOSS_Time;
int time_key =
      (((time.hour*60)+time.min)*60)+time.sec;
VOSSDWH_time_key = time_key;
VOSSDWH_date_key = date_key;

//Michal 1
```

```
VOSSDWH_reservoir = 1; VOSSDWH_tank = 1;
VOSSDWH_inflow = VOSS_MichalQ1*600;
VOSSDWH_outflow = VOSS_MichalQ2*600;
VOSSDWH_chlorine = MichalChlor;
SaveOutputValues(); // Save
...
//Josefov 1
VOSSDWH_reservoir = 4; VOSSDWH_tank = 1;
VOSSDWH_inflow = (VOSS_JosefovQ1+VOSS_JosefovQ11)*600;
VOSSDWH_outflow = VOSS_JosefovQ2*600;
VOSSDWH_consumption = (+VOSS_JosefovQ2+VOSS_JosefovQ22
    -VOSS_HornickaQ1-VOSS_HornickaQ10-VOSS_MichalQ1)
    *600;
SaveOutputValues(); // Save
...
```

8.5 Zpracování historie

V minulé kapitole jsme zadání téměř vyřešili. Jediným problémem, který zůstal, je nutnost některé hodnoty počítat jako rozdíl hodnot mezi měřeními. To by šlo řešit tak, že definujeme proměnné, které budou zobrazovat předešlý stav, pomocí kterých pak budeme počítat rozdíly.

Systém MIND$_{\text{A}}$NAO má však mnohem elegantnější řešení. Pro potřeby zpracování historie se známou hloubkou byly značky pro zpracování datových zdrojů rozšířeny o určení hloubky historie. Implicitně je nastavena na 0, a proto se nezpracovává. Pokud ji použijeme, budeme mít k dispozici tuto historii ve formě proměnných stejně, jako máme aktuální data. Značky zabezpečují i posouvání historie při procházení datovým zdrojem.

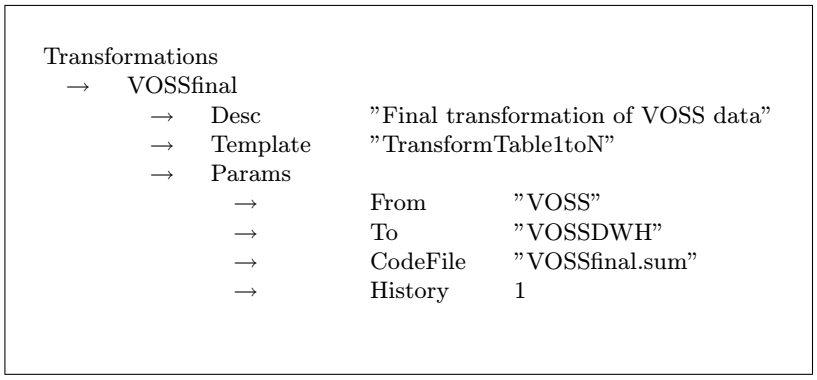
Dosud jsme k datům přistupovali pomocí proměnných zapsaných ve tvaru `DatovýZdroj_Atribut`. Proměnné pro přístup k historickým datům mají podobný tvar: `DatovýZdroj_Hloubka_Atribut`. Pokud tedy budeme potřebovat hodnotu `MichalQ1` z minulého záznamu (tedy historie -1), je přístupná pod jménem `VOSS_1_MichalQ1`. Aktuální hodnoty jsou přístupné v původním tvaru.

Nyní tedy můžeme přistoupit k řešení posledního problému. Výstupní atribut `Delta_Level` má obsahovat rozdíl úrovní hladiny mezi aktuálním a posledním měřením. Také nyní můžeme přesně specifikovat časový interval, který je mezi měřeními a nemusíme se spoléhat, že se podaří získat hodnoty včas. Rozdílem časů nahradíme dosud používanou „magickou“ konstantu 600.

Obrázek 13 ukazuje definici finální transformace. Využijeme možnosti uložit kód externě, proto zde uvedeme pouze jméno souboru. Jeho obsah uvedeme dále.

Výsledný skript v souboru `VOSSfinal.sum` je velmi podobný tomu z minulé kapitoly. Přibude pouze zpracování proměnných z historie:

```
// Společná část
int date_key = DateTime2Int(VOSS_Time);
datetime time = VOSS_Time;
```



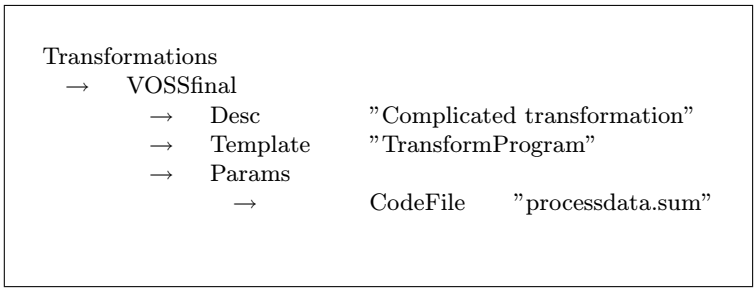
Obrázek 13. Příklad transformace s historií o hloubce 1

```
int time_key =
      (((time.hour*60)+time.min)*60)+time.sec;
VOSSDWH_time_key = time_key;
VOSSDWH_date_key = date_key;
double deltatime = VOSS_Time - VOSS_1_Time;
double delta=(((deltatime.hour*60)+deltatime.min)*60)
                  +deltatime.sec;

//Michal 1
VOSSDWH_reservoir = 1; VOSSDWH_tank = 1;
VOSSDWH_inflow = VOSS_MichalQ1*delta;
VOSSDWH_outflow = VOSS_MichalQ2*delta;
VOSSDWH_chlorine = MichalChlor;
SaveOutputValues(); // Save
...
//Josefov 1
VOSSDWH_reservoir = 4; VOSSDWH_tank = 1;
VOSSDWH_inflow = (VOSS_JosefovQ1+VOSS_JosefovQ11)*600;
VOSSDWH_outflow = VOSS_JosefovQ2*600;
VOSSDWH_consumption = (+VOSS_JosefovQ2+VOSS_JosefovQ22
    -VOSS_HornickaQ1-VOSS_HornickaQ10-VOSS_MichalQ1)
    *600;
SaveOutputValues(); // Save
...
```

8.6 Kompletní program

Pokud dosud popsané šablony nestačí na požadavky kladené na transformaci, ani nemá smysl psát šablonu kvůli jednomu případu, lze napsat celý program, viz obr. 14.



Obrázek 14. Příklad transformace s kompletním programem

Přesto je možné využít ze šablon alespoň značky. Před tím, než se skript spustí, provede se s ním stejné zpracování, jako se šablonou.[3]

Jako ukázkový příklad si uvedeme zpracování datového zdroje, který má řádky obsahující v prvním sloupci číslo, ale až v následujícím záznamu (řádce) odpovídající další data. Ukázka dat viz obr.:

Cislo	Pocet 1
155432	
	7
557915	
	5
...	...

Řekněme, že transformace má vytvořit v cílovém datovém zdroji tolik záznamů, kolik je v atributu Pocet s odpovídajícím číslem z předešlého záznamu.

```
// Open From read-only
##DSopen("Data1",RO)
// Open To
##DSopen("Data2")
//Prepare variables for VOSSout fields
##DSvariables(##param(To))
if(##param(From).FirstRecord())
{
  do
  {
    // first line
    int Cislo; int i;
    ##ASTDatasourceVariablesInput(##param(From));
```

[3] Ve skutečnosti je `TransformProgram` také šablona, ale je triviální a obsahuje pouze odkaz na parametr `Code`.

```
    Cislo = Data1_Cislo;
    // second line
    ##param(From).NextRecord();
    ##DSvariables(##param(From));
    ##DSvariablesLoad(##param(From));
    for(i=0;i<Data1_Pocet;i++)
    {
      // fill the output record
      Data2_Cislo = Cislo;
      // insert data to new record
      ##param(To).InsertRecord();
      ##DSvariablesSave(##param(To))
      ##param(To).PostRecord();
    }
  }
  while(##param(From).NextRecord());
}
// Closing and deleting from memory
##DSclose(##param(To))
##DSclose(##param(From))
```

Jak je vidět z kódu, není psaní celého programu příliš přehledná záležitost. Možná by bylo lepší vytvořit šablonu a některé části nahradit parametry. Nicméně i za těchto podmínek je psaní programu podstatně jednodušší, než kdyby uživatel musel psát text programu pomocí čistých funkcí. Značky zkracují text programu podobně jako makra v jazyku C.

9 Výhody

Výhody doposud popsaného řešení:

Jednoduchost: **Grafické uživatelské rozhraní** nabízí návrh jednoduché datové transformace metodou drag&drop, čímž je návrh záležitostí kreslení myší po obrazovce

Komplexnost: **Napsání celého program v jazyku Sumatra** je druhým extrémem oproti návrhu v grafickém prostředí. Takovýto přístup je pracný, ale dovoluje využít všech možností repository, jejích datových zdrojů i možnosti jazyka Sumatra

Šablony transformací: **Hotové šablony lze využít mnohokrát.** Pro opakovaný typ transformace lze navrhnout šablony obsahující značky, které jsou nahrazeny parametry konkrétních transformací

Rozšiřitelnost I: **Typy datových zdrojů** je velice snadné rozšířit. Pro každý nový požadovaný typ stačí napsat novou třídu, která bude z datového zdroje číst a psát

Rozšiřitelnost II: **Jazyk Sumatra** umožňuje uživateli definovat nové objekty a funkce. Tím je možné rozšířit možnosti Sumatry libovolným směrem; implementace funkcí je v C++

Uniformní přístup k datům: **Jednotný přístup** k datovým zdrojům dovoluje psát transformace bez ohledu na původ dat

Flexibilní formátování dat: **Parametry atributů** určují tvar čtených i zapisovaných dat — formátování. To je definováno značně volně, takže odpadá potřeba zpracovávat vstupy programem (např. rozpoznávat složitý formát data a času, zpracovávat oddělovače tisíců u čísel). Všechny tyto a mnohé další úkoly lze vyřešit nastavením formátu

10 Použití

Prezentovaný systém je používán v projektu GOAL.

Pro přípravu struktury dat byl použit průvodce pro textový soubor, který automaticky rozpoznal formát a datové typy. Stačilo pouze upravit jeden sloupec (nebyl rozpoznán sloupec s datem a časem) a nadefinovat názvy sloupců. Podobně byla vytvořena struktura dat na straně SQL serveru (průvodce pro připojení SQL databází).

Transformace byla v prvním kroku vytvořena ručně, tedy napsáním celého programu. Sloužila především k otestování funkčnosti Sumatry, repository a datových zdrojů.

Ve druhém kroku byla transformace přepsána pomocí šablon, takže se výrazně zjednodušila a zpřehlednila.

Testování výkonnosti zatím nebylo provedeno, protože doba potřebná pro poslání dat SQL serveru vysoko převyšovala jakoukoliv jinou část a měření tak nebylo rozumně realizovatelné.

11 Závěr

Celý systém MIND$_{\text{A}}$NAO se při testovacím použití osvědčil a nyní se připravuje jeho využití a ověření v jiných projektech.

Využívání meta dat v celém systému usnadňuje návrh datových struktur i transformací. Jednou definovaná meta data jsou používána několikrát, takže např. zásah do struktury dat se projeví na všech relevantních místech.

Šablony transformací výrazným způsobem zjednodušují a urychlují vytváření transformací. Autor pouze vyplňuje důležité části, kostra je již dána. Navíc lze použít značky, které automaticky generují větší části kódu, které by se jinak musely definovat ručně.

Pro snadnější orientaci bylo vytvořeno grafické rozhraní, které dovoluje uživatelům pohodlnější práci. Většina potřebných operací je zde prováděna pomocí myši a pouze ve složitějších částech je nutné psát kód.

V průběhu dalšího vývoje dojde jednak k doplňování různých typů datových zdrojů, jednak k vylepšování grafického prostředí, kde by mělo být možné automaticky spouštět celé skupiny transformací tak, aby, kdykoliv to bude možné, byly úlohy spouštěny paralelně.

Poděkování

Vývoj systému MIND$_{\text{A}}$NAO je podporován Ministerstvem školství, mládeže a tělovýchovy v rámci projektu LN00B096.

Reference

1. Petr Aubrecht. Sumatra Basics. Technical report GL–121/00 1, Czech Technical University, Department of Cybernetics, Technická 2, 166 27 Prague 6, December 2000.
2. Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion*. Tools and Techniques for Computer Typesetting. Addison-Wesley, Reading, MA, USA, second edition, 1994.
3. P. Mikšovský and Z. Kouba. Application A2 Specification. Technical report TR11, INCO–COPERNICUS 977091 GOAL, Czech Technical University, Department of Cybernetics, Technická 2, 166 27 Prague 6, 1999.