Podklady k diskusi na zadavani experimentu

Václav Šmídl

28. května 2008

1 Cíl: uživatelsky přítulné a efektivní zadávání experimentů

Mělo by být snadné zadat a použít následující typy experimentů a jejich kombinací:

1.1 Trivial: odhadnutí ARX modelu z dat

Uživatel zadá

- datový soubor,
- strukturu regresoru
- způsob jakým se dozví výsledky.

Mělo by být na úrovni předpřipraveného úkolu.

1.2 Iterators: cyklus přes několik možností

Uživatel zadá

- datový soubor,
- výčet možných struktur regresorů (pole)
- způsob vyhodnocení (pole?)

Pro nestandardní typy výstupů může být třeba něco doprogramovat.

1.3 Shared parts: Agenti

Uživatel zadá

- vzorový popis agenta
- pro každého dalšího agenta se načte ten vzorový a provedou se změny

1.4 Inheritance: výběr potomka

Uživatel má (např. v dialogu) na výběr z několika různých variant algoritmu. V případě, že se liší parametrem se problém redukuje na triviální.

Pokud varianty algoritmů jsou implementovány jako různé objekty pak by měl dostat na výběr nejnižšího přípustného, kterého je možné specializovat na jednotlivé potomky.

Při změně potomka je třeba změnit zadání parametrů. Uchovávání společných nastavení by bylo hezké, ale není nutné.

2 Úrovně řešení

Problém je rozvrstven do několika úrovní, přičemž na každé z nich lze provádět různě složité úkoly. Přehled:

GUI	Uživatelsky příjemná "klikací varianta"
UserInfo	Naklikané výsledky jsou uloženy ve formě textových souborů, které se dají editovat. (Textový editor může být efektivnější než klikátka, např. global Replace)
Experiment	Zadané UserInfo se otestuje na platnost a vyrobí se spustitelný kód. Na této úrovni je třeba ošetřit iterátory atp.
Library	Sada funkcí a objektů které jsou experimentem volány.

Obecně tento přístup máme naimplementován ve dvou variantách:

2.1 Jobcontrol

Je výsledkem vývoje "zdola-nahoru" proto napíšu obráceně:

Library knihovna mixtools

Experiment z experimentů ve formě m-filů, které prováděl Ivan postupně vykrystalizovala obecná šablona, ve které se pouze zaškrtávalo provést/neprovést.

UserInfo projekt jobcontrol měl za úkol usnadnit zadávání této šablony tím, že se napíšou pouze relevantní parametry a automaticky se sestaví vzorový m-file.

GUI po ustabilizování struktury UserInfa se napsalo GUI, které je šité výsledkům na míru.

2.2 Mix3k

Systém byl od začátku navrhován tak, aby umožňoval automatické zadávání experimentu. Klíčovou roli hrají objekty:

Library knihovna se skládá z několika "výpočetních" objektů. Například objekt ARENA má atributy:

environment: model prostředí (které může například přehrávat uložená data), a

participants: seznam aktivních objektů, které se umí v prostředí pohybovat.

Kromě toho, má ARENA ma metodu run, která cyklicky volá metody environment.step() a participant(i).run().

Experiment může být samostaný m-file vytvořený z prvků knihovny, nebo může být vygenerován z UserInfa. V extrémním případě může být

```
load UserInfo
ARENA=build(UserInfo)
run(ARENA)
```

UserInfo je klíčovou částí systému, je jednoznačnou parametrizací jednoho výpočetního objektu tj, má jeho parametry jako atributy a má metody:

build která generuje výpočetní objekt

askuser která je zodpovědná za interakci s uživatelem,

Kličové je, že **atributem userinfa můžou být jiná UserInfa**.

GUI je automaticky generováno voláním metody askuser. V základní verzi se vytvoří prázdné UIExperiment a na něj se zavolá metoda askuser().

3 Řešení základních scénářů v M3k

3.1 Trivial:

Je definován ARXparticipant, který umí odhadovat ARX model a interagovat s prostředím pomocí DataSource.

Je definováno UIarxparticipant, který se umí zeptat na parametry ARX modelu.

Tj. při volání standardního Experimentu se vloží

environment typu čtení dat ze souboru

participant 1x UIarxparticipant.

Po stisknutí tlačítka RUN se vygeneruje odpovídající ARENA.

3.2 Iterators

Je podporováno pouze na úrovni Experiment. Tj. uživatel nakliká první experiment a vyrobí m-file:

```
load UserInfo
for i=1:5
  ARENA.UIparticipants{i}.str = 1:i
end
ARENA=build(UserInfo)
run(ARENA)
```

Tj. přímé vkládání v GUI není podporováno. Informace o tom co a kdy změnit je třeba psát přímo do skriptu.

Je možné navrhnout kombinaci, která vezme jiné userinfo a zpracuje ho:

```
load UserInfo
load Structures
for i=1:5
  ARENA.UIparticipants{i}.str = Structures{i}
end
ARENA=build(UserInfo)
run(ARENA)
```

3.3 Shared parts

Je možné dosáhnout použitím stejného přístupu jako v předchozím případě. V UserInfu by byl jen jeden agent a ve for cyklu by se měnil.

Další přístup, ketrý je udělán na úrovni GUI je save/load mechanismus, UserInfo vytvořené pro jednoho agenta se uloží na disk a pak se načte jako jiný agent.

Nevýhoda: Ztrácí se spojení s původním objektem. Pokud se ten změní tak se to nedozvíme...

3.4 Inheritance

Je řešeno manuálně v UI objektu, který dané objekty používá. Tj. v UIExperiment.ask() jsou vyjmenovány všechny UI enviromentů a participantů, které známe.

Nevýhoda: Pracné a neefektivná.

4 Nové XML rozhraní

Co od toho očekávám:

- Popis UserInfa v XML (nebo snad XSD?),
- vylepšení možností scénářů iterators, inheritance a shared parts.
- GUI editovatelnost ala KXMLeditor `http://kxmleditor.sourceforge.net/screenshots/screenshots.html`, který se ale už nevyvíjí...

Čeho se bojím

- Automatického GUI, kromě výše zmíněného KXMLeditoru jsem nic podobného nenašel...
- Definice UserInfa přes XSD není zrovna jednoduché
- Původní metoda build by se dala udělat přes metodu *post_hello()*, ale co a askuser? Dá se to zakomponovat do XSD?