České vysoké učení technické v Praze
Fakulta elektrotechnická

Tématické okruhy ke státní závěrečné zkoušce pro magisterský studijní program Otevřená Informatika (OI)

http://www.fel.cvut.cz/cz/education/master/topicsOI.html

OI Mgr - SI

Vygenerováno: 14. května 2015 15:28

Obsah

1 Sémantika: operační sémantika, denotační sémantika, pevný bod funkce, vázání jmen, stav programu a data.

Sémantika programovacích jazyků je v teorii programovacích jazyků obor zabývající se důsledným matematickým popisem významu programovacího jazyka [1].
3 hlavní charakteristiky jazyka jsou:

- **syntaxe** se zabývá formou (znaky a jejich vztahy),
- **sémantika** významem znaků,
- **pragmatika** závislostí konstrukcí na nositeli - implementace.

Operační sémantika je přístup, který definuje sémantiku programovacího jazyka tak, že určí jak je libovolný program vykonán na počítači, jehož činnost je známa. Může to být nějaký abstraktní stroj, který je dostatečně jednoduchý pro snadné pochopení jeho činnosti (například Turingův stroj). Operační sémantika potom specifikuje, jak tento stroj zpracovává program v definovaném jazyku.

Operační exekuce

- Program + vstupy → Vstupní funkce
- Iniciální konfigurace
- (Mezikonfigurace ...)
- Finalní konfigurace
- Výstupní funkce → Odpověď

1.1 Sémantika malého kroku (SOS)

- Definujeme přepisovací relaci $\Rightarrow \in Expr \times Expr$.
- Výraz $e \Rightarrow e'$ znamená, že přepis e na e' je vykonán v jednom kroku.

Formální definice $S = \langle CF, \Rightarrow, FC, IF, OF \rangle$

- CF – doména konfigurací (obor hodnot)
- $\Rightarrow$ – přepisovací relace (transformuje konfigurace) ($\Rightarrow \subseteq CF \times CF$)
- FC – množina finálních konfigurací (nezjednodušitelné konfigurace) ($FC \subseteq CF$)
- IF – vstupní funkce (Prog × Inputs) → CF.
- OF – výstupní funkce FC → Answer

Pravidla $e, e_1, e_2, e' \in Expr \quad n, n' \in Num$

$$\frac{}{\triangle n \Rightarrow -n}$$

$$\frac{}{n \odot n' \Rightarrow n + n'}$$

$$\frac{e \Rightarrow e'}{e\triangle \Rightarrow \triangle e'}$$

$$\frac{e_1 \Rightarrow e'}{e_1 \odot e_2 \Rightarrow e' \odot e_2}$$

$$\frac{e_2 \Rightarrow e'}{e_1 \odot e_2 \Rightarrow e_1 \odot e'}$$

1.2 Sémantika velkého kroku (BOS)

Odlišný přístup než SOS. Program je vyhodnocen v jednom kroku. Zde je definována přepisovací relace

$$\Longrightarrow \in Expr \times Num$$

Pravidla $e, e_1, e_2 \in Expr \quad n, n_1, n_2 \in Num$

$$\frac{}{n \Longrightarrow n}$$

$$\frac{e \Longrightarrow n}{\triangle e \Longrightarrow -n}$$

$$\frac{e_1 \Longrightarrow n_1 \quad e_2 \Longrightarrow n_2}{e_1 \odot e_2 \Longrightarrow n_1 + n_2}$$

► **Příklad SOS (jedno z možných vyhodnocení):**

$$\frac{\dfrac{\dfrac{}{\triangle 15 \Rightarrow -15}}{(\triangle 15) \odot (\triangle 24) \Rightarrow -15 \odot (\triangle 24)}}{\triangle((\triangle 15) \odot (\triangle 24)) \Rightarrow \triangle(-15 \odot (\triangle 24))}$$

► **Příklad BOS:**

$$\frac{\dfrac{\dfrac{\dfrac{}{15 \Longrightarrow 15 \quad 24 \Longrightarrow 24}}{\triangle 15 \Longrightarrow -15 \quad \triangle 24 \Longrightarrow -24}}{(\triangle 15) \odot (\triangle 24) \Longrightarrow -15 + -24}}{\triangle((\triangle 15) \odot (\triangle 24)) \Longrightarrow -(-15 + -24)}$$

1.3 Denotační sémantika

Denotační sémantika používá k popisu sémantiky programovacího jazyka funkce. Tyto funkce přiřazují sémantické hodnoty správným syntaktickým zápisům. Příkladem může být funkce Val, která přiřadí výrazu jeho hodnotu:

$$\text{Val} : \text{Výraz} \rightarrow \text{Integer}$$

Doménou syntaktických funkcí se nazývá syntaktická doména - u funkce Val je to množina všech správných aritmetických výrazů. Obor hodnot je sémantická doména - zde množina celých čísel. Denotační definice jazyka má tedy 3 části - syntaktickou doménu, sémantickou doménu a definici jednotlivých funkcí.

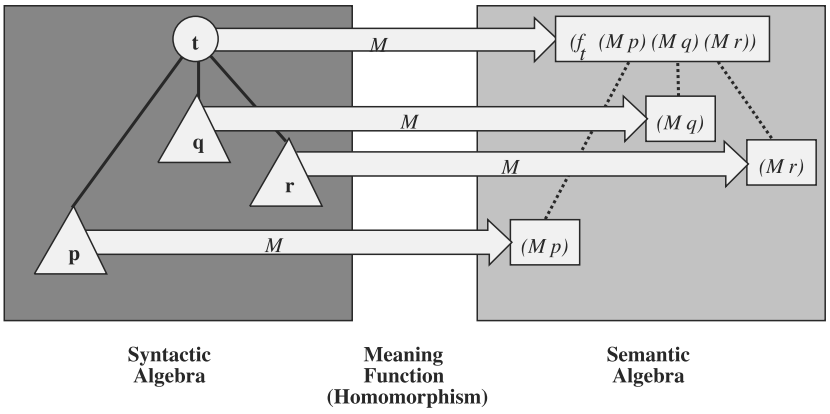
- **Syntaktická algebra** - popisuje abstraktní syntaxy jazyka, může být specifikována gramatikou.
- **Sémantická algebra** - modeluje význam frází, skládá se z kolekce sémantických domén.
- **Významová funkce** - mapuje elementy syntaktické algebry k jejich významu v sémantické algebře. Funkce musí být homomorfismus mezi syntaktickou a sémantickou algebrou.

1.3.1 Významová funkce (Meaning function)

Uvažujte M jako významovou funkci a t je prvek v abstraktním syntaktickém stromu s potomky $t_1, \ldots, t_k$. Pak

$$(Mt) = (f_t(Mt_1) \ldots (Mt_k))$$

kde f_t je funkce určená syntaktickou třídou.



Denotační sémantika $Expr ::= Num \quad | \quad \triangle\, Expr \quad | \quad Expr \odot Expr$
Sémantická doména N

$$[\![n]\!] = n$$

$$[\![\triangle e]\!] = [\![\triangle]\!]([\![e]\!])$$

$$[\![\triangle]\!] = \lambda x. -x \quad \text{(např. unární mínus)}$$

$$[\![e_1 \odot e_2]\!] = [\![\odot]\!]([\![e_1]\!], [\![e_2]\!])$$

$$[\![\odot]\!] = \lambda x, y. x + y \quad \text{(např. plus)}$$

1.4 Pevný bod funkce

Jako pevný bod označujeme bod, který se v daném zobrazení zobrazí sám na sebe. Označuje se také jako samodružný bod. Například pevnými body funkce $f(x) = x^2 - 4x + 6$, jsou čísla 2 a 3.

- bod, ve kterém platí $f(x) = x$
- využívá se pro rekurzivní funkce
- Y kombinátor v lambda kalkulu: `Y = λy(λx.y(xx))(λx.y(xx))`

Př: | Generující funkce faktoriálu fact = `λF.λX.if x==0 then 1 else F(decrement(X))`. Generující funkci dám do Y kombinátoru:

```
Y fact =
       = λf (λx.y(xx))(λx.y(xx)) fact =
       = (λx.fact(xx))(λx.fact(xx)) =
       = fact( (λx.fact(xx) (λx.fact(xx) ) =
       = fact(Y fact)
```

`Y fact` je pevný bod funkce, která počítá faktoriál.

1.5 Vázání jmen

- Vázání jmen se vyskytuje v lambda kalkulu.
- Funkce, co mají jen vázané proměnné, vrátí při každém zavolání stejný výsledek. Funkce s volnými proměnnými jsou závislé na globálním kontextu.
- Funkce, co mají jen vázané proměnné, se v λ-kalkulu nazývají kombinátory.

Ve funkci λx.x řekneme, že x je *vázaná* proměnná, protože její výskyt v těle předcházel λ.x. Proměnná, které nepředchází λ je nazývána *volná*

$$(\lambda x.xy)$$

$$(\lambda x.x)(\lambda y.yx)$$

1.6 Stav programu

Stav = proměnné v prostředí, proměnné mají typ, prostředí = množina všech proměnných, kontext (v handoutech Γ (Gamma)) Čistě funkční jazyky (a matematika) jsou bezestavové, stavové výpočty mohou být reprezentovány jako iterace skrz stavy.

Čistě funkční jazyky (a matematika) jsou bezestavové, stav může být modelován jako iterace skrz stavy.

► **Funkce na nalezení maxima z pole:**

$$max : N* \to N$$

$$max(\langle a_1, \ldots, a_n \rangle) = loop(\langle a_1, \ldots, a_n \rangle, 1, 0)$$

$$loop : N* \times N \times N \to N$$

$$loop(\langle a_1, \ldots, a_n \rangle, c, m) = m \qquad \text{if} \quad c > n$$

$$loop(\langle a_1, \ldots, a_n \rangle, c, m) = loop(\langle a_1, \ldots, a_n \rangle, c+1, m) \qquad \text{if} \quad c \leq n \wedge a_c \leq m$$

$$loop(\langle a_1, \ldots, a_n \rangle, c, m) = loop(\langle a_1, \ldots, a_n \rangle, c+1, a_c) \qquad \text{otherwise}$$

Monády - struktury (typy), co reprezentují výpočet jako sekvenci kroků.

1.7 Data programu

Dělí se na **součiny**, **sumy** a **sumy součinů**.

- Součiny
  - Positional data = N-tice, každý prvek může mít jiný typ
  - Sequence, List = pole
  - Named = třída
  - Nonstrict, stream = data, která se získají/vypočítají v okamžiku, kdy je potřeba (např. InputStream, odněkud se to vezme)
- Sumy - Union v C, nadtypy v Javě.
- Sumy součinů - Binární a ternální operátory, double dispatch.

2 Statická sémantika: typy, polymorfní typy, typy vyššího řádu, rekonstrukce (inference) typů, abstraktní typy.

Statická sémantika je řešena při překladu programu, zde jsou definovány a deklarovány jednotlivá pravidla a prvky programovacího jazyka. V těchto prvcích je zahrnuta jazyková konstrukce, její typy parametrů, význam příkazů a další prvky. Statická sémantika dále kontroluje statické typy a práci s tabulkou definovaných programových symbolů.

- Staticky typované jazyky požadují uvedení datového typu u každé deklarace. Zde nelze deklarovat proměnnou, či funkci nebo objekt bez zadání datového typu.
- Všechny typové kontroly jsou prováděny staticky při překladu. Už při překladu má být každé proměnné přiřazen datový typ.
- Je možné daný datový typ přímo přetypovat. Přetypování především slouží k obcházení typových kontrol.
- **Výhodou statického typování je lepší možnost odhalení typových chyb.**
- Hlavní nevýhodou této metody je větší složitost programových konstrukcí, délka zdrojového kódu a tím i menší pružnost programovacího jazyka.
- K nejčastěji vyskytovaným běhovým chybám patří přetečení datového typu.
- Mezi neznámější zástupce staticky typovaných jazyků patří Java, Ada a jazyk C.

2.1 Typy

$$\Gamma \vdash e : A \qquad \text{„}e \text{ je well-formed term typu } A \text{ v prostředí } \Gamma\text{“}$$

- Typování
  - **statické** - formálně specifikováno typovým systémem (judgements, typová pravidla, prostředí). Typová pravidla rozhodují platnost rozhodnutí (judgements) na základě jiných rozhodnutí o kterých je známo, že jsou platné.
  - **dynamické**
- Typ je množina přípustných hodnot a operací s nimi.
- **TopType**
- **Type preservation** - zachování typu během přepisovací relace $\forall e, e' \in Expr : (\vdash e : t) \wedge e \rightsquigarrow e' \Rightarrow (\vdash e' : t)$
- **Progress** - když mám nějaký term v konfiguraci, tak pak to je buď finální konfigurace, nebo lze ještě nejméně jednou přepsat. Tzn. dobře otypovaný term neskončí ve *stuck* stavu (stav, pro který není definován výstup). (důkaz lze udělat analýzou pravidel). $\forall e \in Expr : (\vdash e : t) \Rightarrow e \in (Num \cup Bool) \vee \exists e' \in Expr : e \rightsquigarrow e'$.
- Type preservation + progress = **Soundness**. An argument is sound if and only if: *The argument is valid* and *All of its premises are true.* (i.e. All men are mortal. Socrates is a man. Therefore, Socrates is mortal.)

- **Terminace** - důkaz např pomocí *energie*, nadefinuji „energii“, nadefinuji, že při každém přepsání se musí snížit.
- **Determinismus** - programovací jazyk je deterministický právě tehdy když existuje právě jeden výstup pro každý pár *programu* a *vstupů*. $\forall v, v' \in (Num \cup Bool) : \forall e \in Expr : e \rightsquigarrow^* v \wedge e \rightsquigarrow^* v' \Rightarrow v = v'$. Platí to, protože platí konfluence relace.
- **Konfluence** - „strom vyhodnocování relace se rozdělí a pak se zase spojí“.

2.2 Polymorfní typy

Polymorfní typ je typ, jehož operace mohou být aplikovány na hodnoty jiného typu.

- **Rekurzivní** - používají se na zakodování seznamů a stromů
- **Univerzální**
- **parametrický polymorfismus** - pomocí něj lze zapsat funkce jedním stylem a zároveň zachovat bezpečné typování. Tzn. funkce je zapsána genericky a umí zpracovávat vstupy bez ohledu na jejich typ. Typický příklad je třeba funkce append která se může zavolat s jakýmkoliv typem a správně se vyhodnotí. Není potřeba deklarovat `append :Integer` nebo `append:Bool`.
- **ad-hoc polymorfismus** - jedna funkce může mít mnoho implementací na základě zpracování jednotlivých typů - přetěžování funkcí v Javě.

2.3 Typy vyššího řádu

- Product Type
- Union Type
- Function Type
- Record Type

Jde o takzvané monády a debuggable functions (tohle je spíš možné využití).

Monády - umožňují řetězit procedury za pomocí čistě funkcionálního programování. Skládají se ze dvou operací a to bind a return(unit).

„In functional programming, a monad is a structure that represents computations defined as sequences of steps: a type with a monad structure defines what it means to chain operations, or nest functions of that type together. This allows the programmer to build pipelines that process data in steps, in which each action is decorated with additional processing rules provided by the monad.“

2.4 Rekonstrukce typů

Schopnost z proměnné vyvodit její typ. Je to vlastnost některých silně staticky typovaných jazyků.

„Type inference refers to the automatic deduction of the data type of an expression in a programming language. If some, but not all, type annotations are already present it is referred to as type reconstruction.“

Například v csharpu, stačí místo typu psát klíčové slovo `var`.

```
var results = new Car();
// results je typu Car
```

2.5 Abstraktní typy

Příkladem jsou Abstraktní třídy v Javě. Vyskytují se hlavně ve staticky typovaných jazycích.

Fronta, HashMapa, Množina (Set), Seznam (List), Zásobník (Stack), ...

3 Teorie HCI, kognitivní aspekty, způsoby interakce, speciální uživatelská rozhraní.

3.1 HCI - Human-Computer Interaction

Human-Computer Interaction (česky interakce člověk - počítač) je průnikový **obor, který se zabývá fenoménem tvorby UI**. Zabývá se analýzou návrhu, vyhodnocování a zavádění interaktivních výpočetních systémů používaných lidmi a jevů, které interakci doprovázejí. Skládá se ze tří částí: **jedinec, počítač a způsob, jakým dohromady spolupracují**.

Cílem je návrh a vývoj prostředků a systémů, které jsou použitelné, efektivní, bezpečné a intuitivní. Dále se snaží přizpůsobit výměnu dat mezi lidmi a stroji tak, aby byla méně stresující a náchylná k nedorozumění.

3.2 Kognitivní aspekty

- **kognitivní psychologie** zkoumá proces myšlení, učení a rozhodování
- mentální model:
  - kognitivní struktura
  - vnitřní reprezentace okolního světa, kterou si vytváříme v hlavě
  - jak objekty určité třídy reagují s objekty jiné třídy, jak objekty v průběhu interakce mění své vlastnosti
  - založeny na zkušenosti, mohou být nepřesné, neodpovídat zákonům fyziky
  - lze je použít k predikci (kam dopadne hozený míč)
- kognitivní model uživatele - model, jak uživatel pracuje, na jehož základě se předpoví jeho chování (interakce s UI), výhody: nemusí se vytvářet prototypy, není nutné testování se skutečnými uživateli, vědecký základ pro návrh
- estetika a efektivita kognitivních funkcí - důležitost vizuální podoby, atraktivní věci jsou použitelnější

Kognitivní teorie v HCI Modelování úkolů - metody KLM, GOMS

- GOMS (Goals, Operators, Methods, Selectors) - popis struktury úloh, task rozpadlý do menších subtasků v hierarchicky přehledné síti,expert provádí UI operace
- KLM (Keystroke-Level Model) - low-level verze GOMS, uživatelské operátory (K-keystroke, P-point, D-drawing, M-mental think), tabulkové časy pro každý operátor, každé operaci se přiřadí čas vykonání
- Hick's Law - čas potřebný k rozhodnutí se, n stejně pravděpodobných možností, průměrný čas výběru jedné z nich: $T = b \log_2(n+1)$
- Fitt's Law - předpovídá jak dlouho trvá uživateli vybrat cíl, vyhodnocení vstupních zařízení, pohyb k cíli o velikosti S ve vzdálenosti D: $T = a + b \log(\frac{D}{S} + 1)$, a, b - konstanty závislé na zařízení

- Model Human Processor / Human Information Processor Model - model lidského poznání vytvořený za použití teorií uvedených výše, modeluje, jak uživatel zachází s informacemi, perceptuální, kognitivní a motorický subsystém

3.3 Způsoby interakce

- přímá manipulace - hry (značné implementační požadavky)
- navigace (menu, odkazy) - web - není třeba si pamatovat příkazy jako CL, ale zabírá příliš místa na obrazovce
- formuláře - web
- příkazy - unix, první metoda komunikace člověk - PC, BNF, konečný automat
- přirozený jazyk
- nové typy interakce - mluva, gesta, eye-tracking, haptické (hmatová odezva) displeje

3.4 Speciální UI

- mluvící systémy AI (Eliza, Cortana, Siri atd.)
- multimodální - kombinování více vstupů
- pokročilý vstup z klávesnice - pubtran se „učí“
- magnetický prsten - klikání, scrollování
- haptická odezva
- papírový mobil - ohýbací gesta
- spolupráce telefonů např při sharování
- multi-touch

Architektura UI Cíl: oddělení UI a aplikace, výběr možností prezentace informace uživateli, koordinace interakce, modifikovatelnost a přenositelnost. Interaktivní systém poskytuje tři funkce (vrstvy):

- prezentační (UI)
- dialogovou (komunikace s uživatelem)
- aplikační (vlastní účel SW systému)

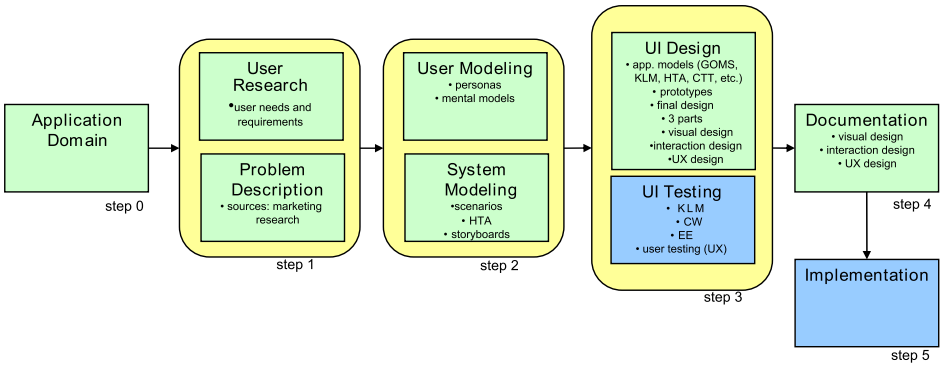
Monolická architektura = všechny části v jednom

Seeheim model URL sémantická vazba je často pomalejší, přímá vazba mezi aplikační a prezentační vrstvou regulovaná dialogem umožní okamžitou odezvu (switch).

Výhody:oddělená prezentační vrstva podporuje přenositelnost a modifikovatelnost, oddělená aplikační vrstva dovoluje modifikace aplikace beze změny UI, oddělená dialogová část umožňuje změnit uživatelskou interakci beze změny prezentační části. Nevýhody: řada modifikací se promítá do všech částí, komplikované sémantické vazby.

lexical	–	*syntactic*	–	*semantic*
Prezentace	–	Dialog	–	Aplikace

4 Metody návrhu, uživatelské a konceptuální modely



Cyklus tvorby UI

- Návrh - porozumění uživateli (cílová skupina) a jeho potřebám
  - **analýza úlohy** - výkonnost software, hardware, uživatele při provádění úlohy, co uživatelé dělají, co k tomu potřebují za nástroje, co potřebují vědět, metoda: HTA
  - popis průběhu dialogu, slouží k následující implementaci UI
- Implementace - prototypování
- Vyhodnocení - hodnocení prototypu ve spolupráci s uživateli (kvalitativní & kvantitativní)
- další iterace...

Analýza

- specifikace aktivit, které systém bude dělat
- specifikace uživatelů - ti, kteří aktivity budou dělat
- volba formy řešení - forma UI, SW podporující UI, OS, systémové požadavky, HW

Uživatel

- Uživatelské požadavky - Obecné požadavky: fyzické, kognitivní, sociální. Mohou být také specifické požadavky, které se vztahují přímo k problémmu.
- Modely uživatele - KLM (Keystroke-level model), persony

Principy použitelného návrhu (usable design)

- jednoduché a přirozené dialogy v jazyku uživatele
- konzistentnost akcí, příkazu, layoutu, terminologie
- minimalizovat paměťovou zátěž uživatele - rozpoznávání je snazší než vzpomínání
- zpětná vazba
- kontrola vstupu
- snadné vrácení akcí - podpoří chuť experimentovat
- výrazně značená ukončení - uživatel se nesmí ocitnout v pasti
- zkratky - rychlé provedení časté akce pro zkušené uživatele
- robustní systém poskytující snadno ovladatelné prostředky k nápravě chyb
- užitečná nápověda a dobrá dokumentace (hledána v kritických situacích)

Terminologie

- Goal - to čeho chceme dosáhnout
- Task - posloupnost aktivit, která dá *goal*
- Action - krok nebo akce - část *tasku*

4.1 Specifikace požadavků

- HTA (Hierarchical task analysis) - dekompoziční strom, kde je *goal* zakreslen do postupně se rozpadajících menších *tasků*. (CTT[1] - strom s operátory a symboly)
- Storyboard - série snímků a skeč§ („komiks“ popisující *goal*)
- Scénáře - jednoduché výpravné příběhy průběhu úkoli *„Uživatel napíše všechny účastníky akce, poté systém zkontroluje zda je vše vyplněno v pořádku a vytvoří událost...“.*
- Případy užití - popis interakce člověka se systémem.

4.2 Uživatelský průzkum

Získávání informací o budoucích uživatelech systému, jejich **potřeby, zvyky , zkušenosti a dovednosti**.

- **kvalitativní** - menší vzorek lidí, více informací, rozhovor, etnografické (pozorování chování) pozorování
- **kvantitativní** - více lidí, méně informací, průzkumy, testy, pozorování
- kombinovaný

[1] Concurrent Task Tree

Persona Detailně popsaný hypotetický uživatel reprezentující nějakou uživatelskou skupinu. Je založen na nasbíraných informacích. Nevýhody mohou být: nekonzistentní uživatel, představování si sama sebe jako uživatele.

Metody sběru dat

- pozorování - introspekce (sebepozorování) - zahrnuje kognitivní průchod, extrospekce
- rozhovor - strukturovaný vs. volný, být neutrální a zvědavý (30-90 min), přímé otázky, žádná anonimita, vliv tazatele
- dotazování (průzkum) - jednoduché otázky, používat rozsahy, neopakovat otázky, lidé lžou (chtějí vše, levně a ihned)
- experiment

4.3 Modely pro návrh UI

Konceptuální model (design model) znamená to, jak to je navrženo. **Uživatelský** model (user model) značí co uživatel očekává. Nesoulad modelů vede k pomalému provádění úloh, chybám a frustraci

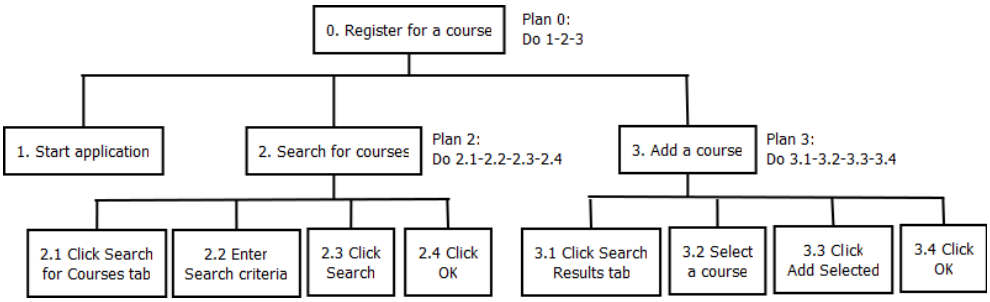
Mentální model Uživatelovo porozumění jak se objekty chovají a jak akce prováděné přes UI ovlivňují jejich chování získané na základě zkušeností. Očekávané struktury a chování (menu, ukládání souborů, zpětná vazba, interpretace akcí). Vědomé i podvědomé procesy, které obsahují aktivaci obrazů a analogií. Hluboké a mělké modely (řízení auta vs. fungování auta).

UI musí prezentovat model vizuálně, mapování reálných prvků na rozhraní. Dobrý konceptuální model zahrnuje:

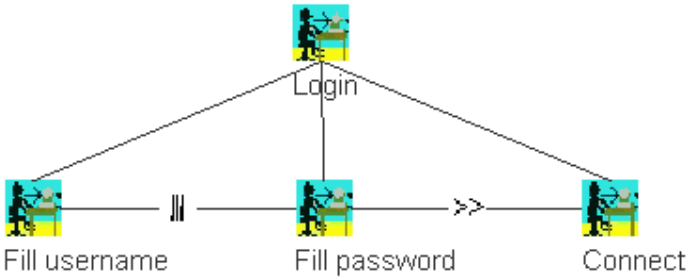
- dostupnost funkcí (affordances)
- návaznost (kauzalita)
- omezení (constraints)
- mapování jednotlivých kroků na akce
- vzory chování cílových uživatelů

5 Formální popis uživatelských rozhraní

HTA (Hierarchical task analysis) Dekompoziční strom, kde je nějaký cíl (úloha) zakreslen do postupně se rozpadajících menších podúloh. Používá se ve fázi návrhu UI pro popis vzájemného uspořádání podúloh.



CTT (Concurrent Task Tree) Podobný strom jako HTA, ale s operátory a symboly. Úloha přihlášení: vyplním uživatelské jméno a zároveň heslo, pak je mi umožněno se připojit.



- Enabling T1 >> T2
- Disabling T1 [> T2
- Interruption T1 |> T2
- Choice T1 [] T2
- Iteration T1* nebor $T1_n$
- Concurrency T1 ||| T2
- Optionality [T]

Storyboard Série snímků a skečů („komiks“ popisující nějakou úlohu).

Scénáře Jednoduché výpravné příběhy průběhu úkolu

„Uživatel napíše všechny účastníky akce, vyplní místo a datum konání. Systém poté zkontroluje zda je vše vyplněno v pořádku a vytvoří událost.“.

Případy užití Popis interakce člověka se systémem..

Keystroke-Level Model (KLM) Cílem je vypočítat čas potřebný pro provedení úlohy Operátory:

- stisk klávesy (**K**eystroke) - určený rychlostí psaní
- ukázat na cíl na displeji (**P**ointing) - určeno pomocí Fitt's Law
- položit ruku na vstupní zařízení (**H**oming) - odhad měřením
- mentální příprava akce (**M**ental preparation) - odhad měřením, heuristika pro předřazení
- čas reakce systému (**R**eaction)

Jsou časové odhady (tabulkové) pro každý operátor. Předpokládá provádění úloh bez chyby, předpovídá jen efektivitu, ignoruje paralelní zpracování, prokládání úloh, mentální zátěž, plánování a řešení úlohy („přemýšlecí" čas, uvažovány jsou jen holé akce)

Goals, Operators, Methods, Selection Rules (GOMS) Nejznámější používaná metoda. Složky:

- **Goals** - cíle z hlediska úmyslů koncového uživatele
- **Operators** - elementární perceptuální, kognitivní a motorické akce s fixním časem bez ohledu na kontext
- **Methods** - posloupnost operátorů a podcílů
- **Selection rules** - if-then pravidla určující, kterou metodu použít

Předpokládá provádění úloh bez chyby, úlohy musí mít přesně definovaný cíl, nemodeluje proces řešení problému, chování uživatele

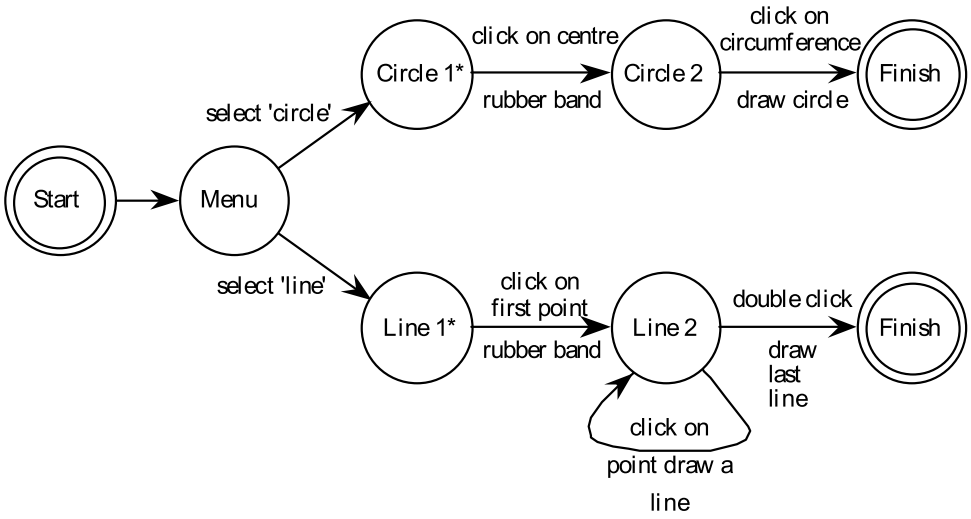
Dialog modeling Z HTA máme představu o posloupnosti kroků, potřebujeme popsat, jak při provádění kroků spolu budou komunikovat uživatel a počítač - jak bude probíhat dialog

- textové (gramatiky, produkční pravidla, událostní algebry)
- diagramy - (STN, PN, flowcharts, JSD)

State Transition Networks (STN) Varianta konečných automatů, konečný počet stavů a přechodů mezi nimi, automat se nachází v pravě jednom stavu (stavy jsou disjunktní). Reakcí na každý uživatelský vstup je přechod z daného stavu do nového stavu. Stav má přiřazenou akci, musí být odlišitelný od jiných stavů, charakterizován vstupy, které k němu vedou. Přechod mezi stavy může být vázán podmínkou, lze k nim přiřazovat popis akcí.

+ model UI, se kterým lze experimentovat
+ možnost automatického nebo poloautomatického vytváření UI
+ kontrola vlastností (úplnost, reversibilita, dostupnost, nebezpečné stavy - ukončení bez uložení)
− některá zařízení mohou mít velká množství stavů

hierarchické STN - varianta pro popis složitých dialogů, obsahuje sub-dialogy (vnořené další sítě).



Petriho sítě (PN) Oproti STN mají synchronizaci - pokračování při splněné podmínce

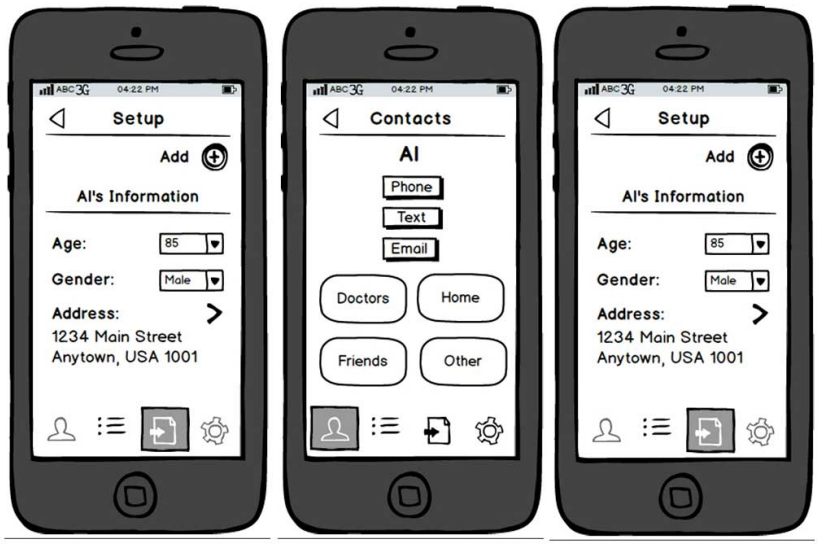
6 Prototypování uživatelských rozhraní

Prototypování se dělá v ranných fázích návrhu. Rozděluje na **Low-Fidelity** a **High-Fidelity** prototypování.

6.1 Low-Fidelity

První náčrtky rozhraní. Jsou to víceméně narychlo načmárané skicy bez detailů, spíše jde o základní rozvržení (layout) prvků, žádná interakce. Typicky skicy na papíru nebo na elektronických prototypovacích SW (není na finálním zařízení). Je to spíše o rychlém zhodnocení nápadů, které jsou převedeny na papírový prototyp.

- velké množství nápadů/alternativ
- krátká doba vývoje - hodiny/dny
- neběží na finálním zařízení
- bez interakce
- testování v laboratoři

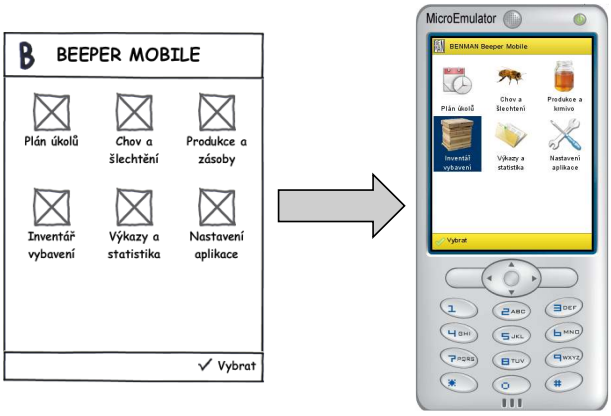


Např. prototypování mobilní app. Vytvoří se několik papírků, každý popisuje krok aplikace. Interakce v podobě fiktvního klikání na papír, dochází k výměně papírků (změna stavu) a tím dochází k fiktivní interakci.

6.2 High-Fidelity

Iluze finálního vizuálního (i interagujícího) návrhu. Vzhled by měl následovat *guidelines* cílové platformy (MS Windows, Android, iOS, ..). Prototyp již ve funkční podobě na cílovém zařízení, např. na telefonu. Interakce realizována jakoby to byla již výsledná aplikace, ovšem logika ještě nemusí být implementována (dummy data, *Wizard of Oz*, atd.,). Hlavní části

- málo alternativ (pokud vůbec nějaké jsou)
- dlouhá doba vývoje - dny/týdny
- dostupné na finálním zařízení
- podoba a interakce téměr podobná té finální
- testování v laboratoři a v „terénu“ (v reálných podmínkách využití)



Problémy s prototypováním

- U lo-fi prototypů dochází ke skipování hlubokých (detailních) uživatelských požadavků
- user confusion: hi-fi prototyp vs. finální aplikace
- drahé a žrout času (speciálně hi-fi)

7 Techniky správy a organizace rozsáhlých softwarových projektů. Nástroje pro správu verzí zdrojových kódů, sledování chyb, pro automatické generování dokumentace a podporu orientace v rozsáhlých projektech. Způsoby komunikace mezi vývojáři navzájem a i s uživateli. Systémy pro sledování a řešení chyb a uživatelskou podporu. Open-source licence a z nich vyplývající práva a licence. Postup začlenění úpravy (patche) do velkého open-source projektu (např. Linuxové jádro)

Motivace pro verzování zdrojového kódu Verzování kódu umožňuje sdílení kódu mezi vývojáři, zálohování, paralelní vývoj v několika souběžných větvích, vrácení se ke konkrétní revizi kódu, zjištění autorství nebo zobrazení statistik. Bez verzovacího systému není možná efektivní spolupráce více vývojářů na jednom projektu. Verzovací systém přináší výhody i v případě, že na projektu pracuje jediný vývojář.

7.1 Nástroje pro správu verzí kódu

GIT je distribuovaný SCM[2] od Linuse Torvaldse. Každá working copy je zároveň repozitář, nezávislé na centrálním serveru. Spoustu operací jako merge, branch,... lokálně. Commity jsou hash celé historie vedoucí ke commitu. Nevýhoda: častější konflikty Subversion (SVN) je centrální SCM, ale rozšířený = dobré nástroje, GUI atd. Míň konfliktů, ale závislost na serveru. CVS podobné SVN ale starší, nevýhody: drahé branchování, problémy s Unicode, netrackuje přejmenovávání a mazání souborů.

7.2 Nástroje pro sledování chyb (bug trackers)

Jsou nástroje (např. bugzilla), které sledují a soustřeďují nalezené chyby. Každá chyba má vypsaný svůj *ticket*, ve kterém jsou veškeré informace k chybě: popis, priorita, reportující osoba, přiřazená osoba, návrh úpravy (např. v podobě patche).

7.3 Nástroje pro automatické generování dokumentace

Javadoc generuje HTML dokumentaci z komentářů v Java kódu, Doxygen je multijazykový generátor dokumentace z kódu, Enterprise Architect umí generovat UML diagramy z kódu.

7.4 Systémy pro spolupráci mezi vývojáři

GitHub je populární sociální platforma pro vývojáře na hosting a spolupráci open-source projektů založený na použití Gitu, Trac je project management systém v Pythonu, který si

[2]Source Code Management

můžete nasadit na vlastní server, obsahuje wiki, bug tracking, time management, etc. Spíše pro menší projekty. JIRA je SaaS systém podobný Tracu se spoustou pluginů, hodí se pro větší projekty.

- Google Groups a podobné mailing listy mohou také sloužit k podpoře.
- wiki stránky
- fóra

7.5 Licence

Svobodný software je software, který respektuje svobodu svých uživatelů a poskytuje jim čtyři základní svobody, které svobodný software definují (publikace FSF 1986):

1. svoboda používat program za jakýmkoliv účelem
2. svoboda zkoumat a upravovat program (předpokladem je přístup ke zdrojovému kódu)
3. svoboda šířit původní verzi programu
4. svoboda šířit upravenou verzi programu

- **Komerční software** – licence daná smluvními podmínkami jež uživatel potvrzuje při nákupu SW
- **Freeware** – zdarma, většinou bez zdrojových kódů, podmínky mohou omezovat další šíření, (komerční) použití, zkoumání
- **Shareware** – jako freeware, ale specifikuje pro které druhy použití je nutné pořídit placenou verzi
- **Permisivní** (akademické) licence (BSD, MIT) – povolují použití/integraci do komerčního SW, vyžadují jen uvádění autora/ů (to je i instituce)
- **Copyleftové** (reciproční) licence (GPL, LGPL, MPL) vyžadují zahrnutí uživatelů do okruhu oprávněných osob k právu nakládat s dílem (modifikovat ho a šířit za stejných podmínek)

Upozornění: Definice open-source nevyžaduje *copyleft*.

7.5.1 BSD (Berkeley Software Distribution)

BSD licence je licence pro svobodný software, mezi kterými je jednou z nejsvobodnějších. Umožňuje volné šíření licencovaného obsahu, přičemž vyžaduje pouze uvedení autora a informace o licenci, spolu s upozorněním na zřeknutí se odpovědnosti za dílo.

7.5.2 MIT License

Licence podobná BSD licenci umožňuje se software nakládat téměř libovolně (používat, kopírovat, modifikovat, slučovat, publikovat, distribuovat či prodávat), jedinou podmínkou je zahrnutí textu licence do všech kopií a odvozenin software.

7.5.3 Apache License

Stejné myšlenkové základy jako licence BSD a MIT. Výslovná zmínka možnosti šířit odvozená díla pod jinou kompatibilní licencí.

7.5.4 GNU GPL (GNU General Public License)

GPL je nejpopulárnějším a dobře známým příkladem silně copyleftové licence, která vyžaduje, aby byla odvozená díla dostupná pod toutéž licencí.

GNU General Public License. Software šířený pod licencí GPL je možno volně používat, modifikovat i šířit, ale za předpokladu, že tento software bude šířen bezplatně (případně za distribuční náklady) s možností získat bezplatně zdrojové kódy. Toto opatření se týká nejen samotného softwaru, ale i softwaru, který je od něj odvozen. Na produkty šířené pod GPL se nevztahuje žádná záruka. Licence je schválená sdružením OSI a plně odpovídá Debian Free Software Guidelines.

V rámci této filosofie je řečeno, že poskytuje uživatelům počítačového programu práva svobodného softwaru a používá copyleft k zajištění, aby byly tyto svobody ochráněny, i když je dílo změněno nebo k něčemu přidáno. Toto je rozdíl oproti permisivním licencím svobodného softwaru, jejímž typickým případem jsou BSD licence

7.5.5 GNU LGPL (GNU Lesser General Public License)

Lesser/Library GPL. Licence je kompatibilní s licencí GPL. Pod touto licencí se šíří zejména knihovny, protože narozdíl od licence GPL umožňuje nalinkování LGPL knihovny i do programu, který není šířen pod GPL.

7.5.6 MPL (Mozilla Public License)

Mozilla Public License. Základním elementem pokrytým licencí je každý jednotlivý zdrojový soubor. Autor takového souboru umožňuje komukoliv používat, měnit a distribuovat jeho zdrojový kód (i jako součást většího díla). Každá změna původních souborů je krytá licencí, tzn. musí se tedy zveřejnit. To samé platí pokud přenesete část původního souboru do nového souboru, tj. celý nový soubor je pak nezbytné zveřejnit. Pokud vytváříte nový produkt přidáním nových souborů, můžete pro tyto nové soubory použít libovolnou licenci. Binární verze lze licencovat libovolně, pokud to není výslovně v rozporu s MPL (zákaz distribuce zdrojů). Produkty pod touto licencí jsou distribuované jak jsou ("as is"), tj. bez záruk libovolného druhu.

7.5.7 CreativeCommons

Je license vhodná pro jakýkoli obsah, nejenom pro software. Do license si člověk může dát 1-4 z těchto atributů:

- Attribution - nutno uvést autora originálního projektu
- Noncommercial - možno upravovat jenom pro nekomerční účely

- NoDerivatives - možno pouze kopírovat dané dílo, ale nikoliv ho upravovat
- Share-alike - znamená virální copyleft

7.6 Kontribuce do projektu

TODO

Reference

[1] Přispěvovatelé wikipedie. Sémantika programovacích jazyků. Dostupné z: http://cs.wikipedia.org/wiki/Sémantika_programovacích_jazyků.