Modelování, simulace a realizace softwarového procesu

Ivo Vondrák

Abstrakt

Současná etapa v oblasti tvorby softwarových produktů je charakteristická snahou zajistit jeho maximální kvalitu a spolehlivost prostřednictvím přesně definovaného softwarového procesu. Cílem tohoto příspěvku je prezentovat výsledky výzkumu zabývajícího se vývojem metod a systémů počítačové podpory právě pro oblast softwarového procesu. Tento výzkum je financován výzkumnými laboratořemi Systems and Information Science Labs firmy Texas Instruments, Inc. a řešen na katedře informatiky FEI VŠB - Technická univerzita Ostrava.

1. Úvod

Dlouhodobý výzkum financovaný americkým ministerstvem obrany a prováděný institutem pro oblast softwarové inženýrství (SEI - Software Engineering Institute) při Carnegie-Mellon University prokázal, že kromě použitých lidských zdrojů a technologií vývoje softwaru je určující pro jeho kvalitu právě softwarový proces. Pokusme se nejprve o jeho definici a vymezení základních pojmů, které nás budou v dalším provázet:

"Softwarový proces je množina po částech uspořádaných procesních kroků svázaných s množinou artifaktů, lidskými a počítačovými zdroji, organizačními strukturami a podmínkami, které směřují ke splnění cíle - vytváření a udržování softwarových produktů."

Pojem *procesní krok* použitý ve výše uvedené definici může reprezentovat elementární procesní krok - *aktivitu*, která nemá svou viditelnou vnitřní strukturu, nebo se může jednat opět o *subproces* se všemi výše definovanými atributy určený k plnění některých ze subcílů výsledného cíle. Nástroji realizace, či provedení těchto procesních kroků jsou pak *lidské a počítačové zdroje* tvořící součást procesu. *Artifakty* nazýváme výsledné požadované produkty a nutné meziprodukty vytvořené nebo modifikované během procesu.

Poznamenejme, že v daný okamžik je nutné si uvědomit rozdíl mezi softwarovým procesem a projektem. Softwarový proces reprezentuje obecný předpis - abstrakci, jak řešit jednotlivé projekty, které lze v uvedeném kontextu chápat jako instance této abstrakce. Nabízí se zde analogie s objektově orientovaným programováním, kde abstraktní datový typ (třída) slouží jako předpis pro vytváření jednotlivých objektů (instance třídy). V našem případě lze zjednodušeně říci, že proces je třídou a jednotlivé projekty řešené podle daného procesu pak reprezentují instance této třídy.

Úroveň definice a využití softwarového procesu je dnes hodnocena podle stupnice SEI 1 ÷ 5 vyjadřující vyspělost firmy či organizace z daného hlediska. Tento model hodnocení vyspělosti a schopnosti dodavatele softwarového produktu se nazývá Capability Maturity Model (CMM) a jeho jednotlivé úrovně lze stručně charakterizovat asi takto:

Úroveň SEI 1: Počáteční (Initial) - firma nemá definován softwarový proces a každý projekt je řešen případ od případu (ad hoc).

Úroveň SEI 2: Opakovatelná (Repeatable) - firma identifikovala v jednotlivých projektech opakovatelné postupy a tyto je schopna reprodukovat v každém novém projektu.

Úroveň SEI 3: Definovaná (Defined) - softwarový proces je definován (a dokumentován) na základě integrace dříve identifikovaných opakovatelných kroků.

Úroveň SEI 4: Řízená (Managed) - na základě definovaného softwarového procesu je firma schopna jeho řízení a monitorování.

Úroveň SEI 5: Optimalizovaná (Optimized) - zpětnovazebná informace získaná dlouhodobým procesem monitorování softwarového procesu je využita ve prospěch jeho optimalizace.

Vývoj a následná implementace formalních metod, které dokáží podchytit celý proces identifikace a použití softwarového procesu zřejmě může výrazně urychlit přechod mezi jednotlivými úrovněmi a výrazně tak zvýšit kvalitu výsledného produktu.

2. Formální model softwarového procesu

Modelování softwarového procesu vyžaduje adekvátní formalismus pro své vyjádření. Vzhledem k tomu, že neoddělitelnou součástí jakéhokoliv procesu, tedy i softwarového, je výskyt souběžně prováděných procesních kroků, byl za základ pro modelování softwarového procesu vybrán aparát Petriho sítí. Jedná se tedy o orientovaný graf tvořený dvěmi typy uzlů - *místy* (places), na kterých může spočívat jedno nebo více značení (tokens) a *přechody*, které mohou být aktivovány (fired) a přenést značení ze vstupních míst na výstupní místa aktuálního přechodu. Petriho sítě tak umožňují modelovat události, podmínky a jejich vzájemné vztahy. Podmínky jsou vyjádřeny pomocí značení, která jsou přítomna na jednotlivých místech. Zatímco aktivace přechodu může být chápána jako vznik události. Formálně lze Petriho síť vyjádřit následujícím způsobem:

$$(P, T, I, O), \tag{1}$$

kde

- P - množina míst
- T - množina přechodů
- I - vstupní funkce
- O - výstupní funkce.

Vstupní funkce I resp. výstupní funkce O definuje pro každý přechod t_j množinu vstupních resp. výstupních míst $I(t_j)$ resp. $O(t_j)$. V případě, že místo může obsahovat více než jedno značení můžeme stav Petriho sítě vyjádřit pomocí tzv. vektoru značení:

$$\mu = (\mu_1, \mu_2, \ldots, \mu_i, \ldots, \mu_n), \qquad (2)$$

kde

μ_i - počet značení nesených místem $p_i \in P$

n - počet míst v Petriho síti.

Aktivace přechodu vede k odstranění značení ze všech jemu předcházejících míst a k následné distribuci značení na všechna místa následující za aktuálním přechodem. Vektor značení se tak dynamicky mění podle toho, které přechody se stávají aktivními.

Provádění (reprezentované aktivací jednotlivých přechodů) Petri sítě je nedeterministické, což znamená, že v případě možnosti aktivace více přechodů v daném časovém okamžiku nelze predikovat, který z nich bude aktivován jako první.

Ukončeme nyní tento úvod do oblasti teorie Petri sítí doporučením, že v případě zájmu čtenáře o hlubší a mnohem podrobnější informace na téma Petriho sítí lze tyto nalézt např. v odborné literatuře [1].

Modelování reálných procesů vyžaduje další rozšíření paradigma Petriho sítí. Prvním z těchto rozšíření odráží nutnost zavedení doby trvání provedení přechodu a tento asociovat s další nezbytnou informací. Přechod tak bude reprezentovat procesní krok, který dále budeme nazývat *transakcí*. Další informace asociovaná s transakcí bude následujících dvou typů:

- *Řídicí informace* daná množinou vstupních podmínek C a výstupních podmínek E definovaných tímto způsobem:

$$C = \{c_1, c_2, \ldots, c_i, \ldots, c_k\} \quad \text{a} \quad E = \{e_1, e_2, \ldots, e_i, \ldots, e_l\}. \qquad (3)$$

Prvky množin c_i resp. e_i jsou reprezentovavány uspořádanými dvojicemi

$$c_i = (n_i, v_i) \quad \text{resp.} \quad e_i = (n_i, v_i),$$

kde

n_i - jméno proměnné

v_i - hodnota proměnné; $v_i \in \{true, false, undefined\}$.

- *Datový tok* daný množinou vstupujících dat transakce In a množinou vystupujících dat transakce Out definovaných obdobně jako v předchozím případě:

$$In = \{i_1, i_2, \ldots, i_j, \ldots, i_k\} \quad \text{a} \quad Out = \{o_1, o_2, \ldots, o_i, \ldots, o_l\}. \qquad (4)$$

Prvky těchto množin i_j resp. o_i jsou opět reprezentovanými uspořádanými dvojicemi

$$i_j = (n_j, v_j) \quad \text{resp.} \quad o_i = (n_i, v_i)$$

kde

n_j - jméno proměnné

v_i - hodnota proměnné reprezentovaná textovou informací.

Důvod zavedení řídicí informace spočívá ve snaze odstranit již výše zmíněný problém nedeterminismu Petriho sítí. V tomto novém přístupu může být transakce provedena pouze v tom případě, že všechny její předcházející místa obsahují značení a současně jsou uspokojeny všechny její vstupní podmínky. Po provedení transakce tato nastaví hodnoty výstupních podmínek a vytvoří značení, která jsou distribuována na všechna její následující místa. Vstupní a výstupní data slouží k modelování artifaktů, které jsou transakcí vytvářeny, modifikovány či zužitkovány.

Místa mají v tomto formálním popisu stejnou úlohu jako v klasických Petriho sítích. Důležitým rozšířením je však následující princip, kde jednotlivá značení neslouží pouze k synchronizačním účelům, ale stávají se nosiči řídicí informace a datových toků mezi jednotlivými transakcemi. V podstatě se jedná o objekty, jejichž stav je reprezentován aktuální řídicí informací a definovanými artifakty. Formálně můžeme tato značení definovat jako uspořádanou trojici následujícím způsobem:

$$\left(t_r, E_{t_r}, Out_{t_r}\right), \tag{5}$$

kde

t_r - *tvůrce* (transakce, která vytvořila značení)

E_{t_r} - řídicí informace (výstupní podmínky definované tvůrcem)

Out_{t_r} - artifakta (výstupní data definovaná tvůrcem).

Značení jsou uchovávána na místech ve frontě. K aktivaci transakce dochází v tom případě, že všechna vstupní místa obsahují značení, jejichž řídicí informace splňují vstupní podmínky transakce. Formálně vyjádřeno:

$$C_j \subseteq \bigcup_{p \in I(t_j)} E_{tok_p}, \tag{6}$$

kde

C_j - množina vstupních podmínek transakce t_j

tok_p - první značení ve frontě na místě p

$I(t_j)$ - množina vstupních míst transakce t_j

E_{tok_p} - řídicí informace svázaná se značením tok_p.

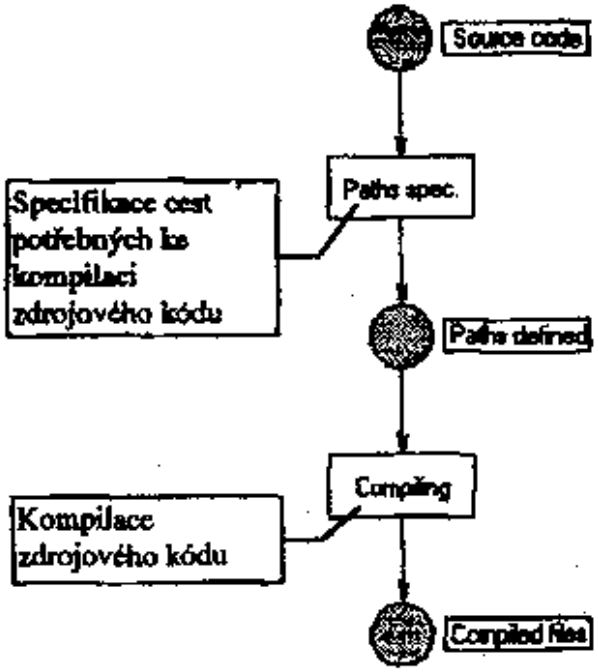
Slovně interpretováno, každá vstupní podmínka je porovnána se jménem a hodnotou proměnné obsažených ve sjednocení veškeré řídicí informace dané prvními značeními fronty všech předcházejících míst dané transakce. V případě úspěšného porovnání všech vstupních podmínek (hodnota *undefined* vždy vyhovuje) dojde k aktivaci transakce, v jejímž důsledku jsou všechna tato značení z předcházejících míst odebrána.

3. Hierarchická struktura procesu

Už z definice procesu vyplývá, že představený formalismus nutně vyžaduje aparát umožňující modelovat hierarchické vlastnosti softwarového procesu. V našem případě stačí, aby transakce byla atomickou (aktivita) nebo opět naší modifikovanou Petriho sítí reprezentující vnořený subproces. V tomto druhém případě je nutné specifikovat ve vnořeném procesu propojovací místa společná pro tento proces a proces rodičovský, vlastnící.

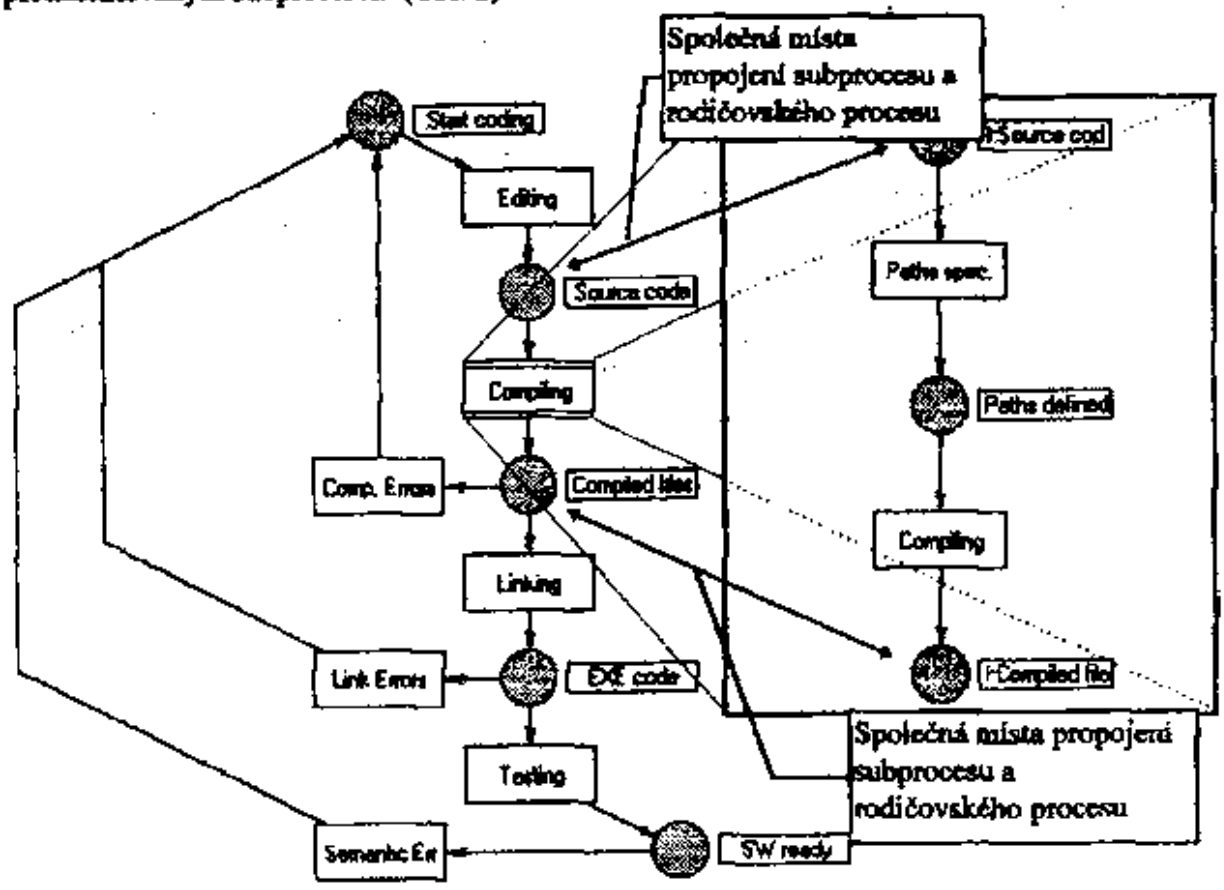
Pokusme se výše uvedené demonstrovat na jednoduchém případě kódování programu. Předpokládejme, že v knihovně předdefinovaných procesů se nachází proces popisující kompilaci zdrojového kódu (Obr. 1). Kružnice vyjadřují místa, prosté obdélníky aktivity a

obdélníky se zdvojenou horní a spodní hranou pak reprezentují vnořený subproces. Šipky určují směr předávání značení a tedy i pořadí aktivace jednotlivých transakcí.



Obr.1 Proces kompilace

Vlastní proces kódování se skládá z řady kroků jako je editace, kompilace, spojování a samozřejmě testování. Také proces identifikace chyb je součástí procesu kódování programu. V našem případě bude procesní krok kompilace reprezentován již dříve zmíněným předdefinovaným subprocesem (Obr. 2).



Obr. 2 Hierarchický proces kódování

Vlastní proces propojení je realizován prostřednictvím substituce původních míst v procesu kompilace (*SourceCode* a *CompiledFiles*) odpovídajícími vstupními a výstupními místy z rodičovského procesu (identifikace propojovacích míst je zajištěna uvozujícím symbolem I: - Interface).

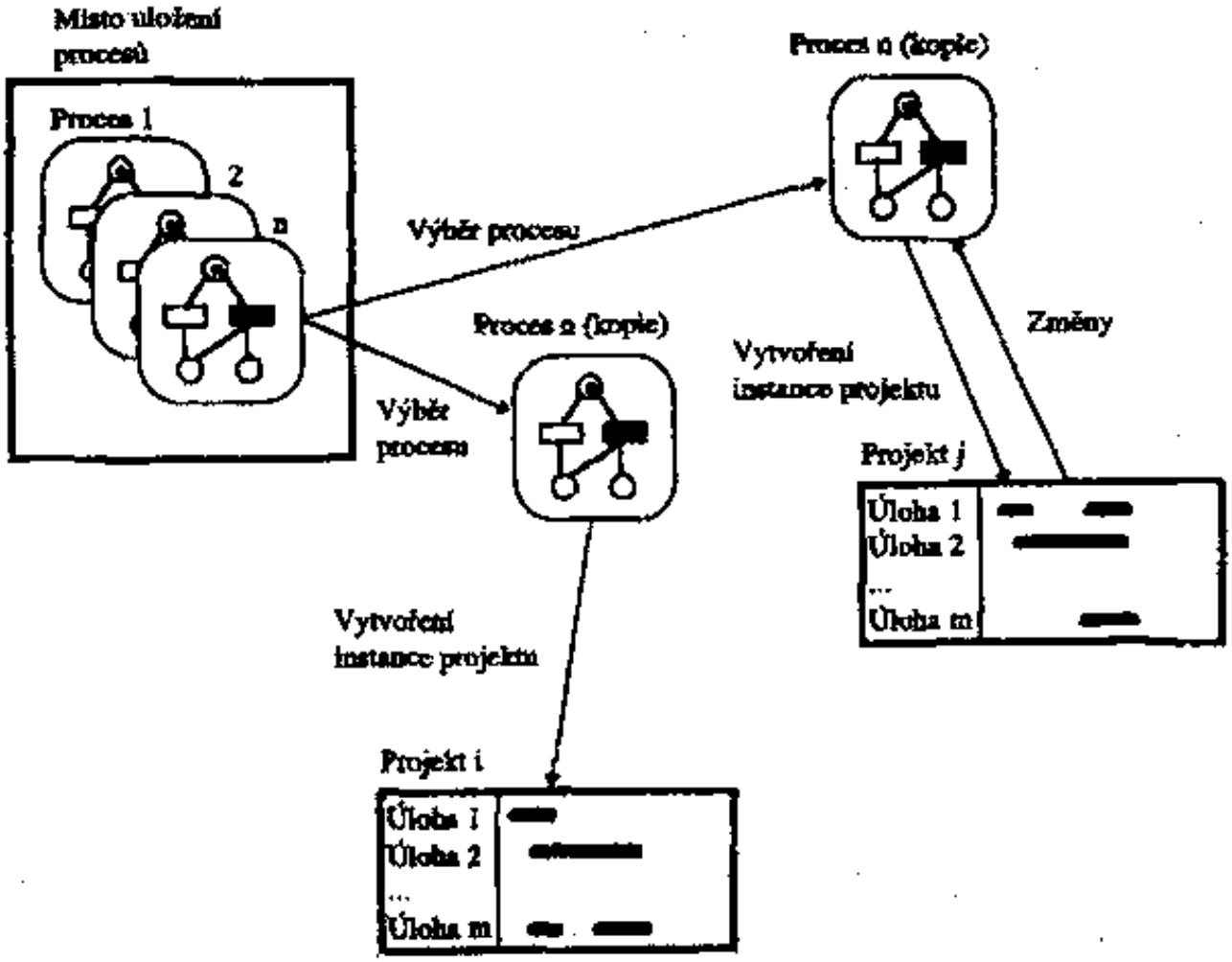
Díky možnosti předávat informaci mezi jednotlivými transakcemi pomocí objektu značení, lze definovaný proces modifikovat i během jeho provádění v konkrétním projektu. Pouze výraz (6) musí platit pro všechny transakce naší modifikované Petriho sítě.

4. Architektura systému

Vytvořený systém podpory návrhu, simulace a provádění softwarového procesu (PAEF - Process Asset Engineering Facility) implementuje výše uvedený formalismus reprezentace procesu výhradně na bázi principů objektově orientovaného paradigma tak, jak je nabízeno prostředím jazyka Smalltalk. Použitý objektový model systému umožňuje reprezentovat každou část systému ve formě dočasného nebo perzistentního objektu. Právě druhý způsob umožňuje uchovávat celý nebo jednotlivé části procesu v jednoduché objektové databázi stejně jako jeho jednotlivé instance reprezentované softwarovými projekty.

Základní idea systému pak vychází z postupného sestavování procesu prostřednictví atomických aktivit a/nebo již existujících procesů uložených jako perzistentní objekty v databázi. Proces může být konkretizován do projektu cestou transformace jednotlivých transakcí na úlohy (*tasks*), které mají přiřazeny konkrétní predikované doby trvání a samozřejmě také lidské a počítačové zdroje nutné k jejich provedení. Trvalé uložení projektů ve formě perzistentních objektů umožňuje pro podobné si zakázky klonovat jim odpovídající již existující projekty a jejich opětovné použití.

V počátečním použití systému PAEF je velmi nepravděpodobné, že definované procesy budou splňovat všechny požadavky na ně kladené, avšak s postupem času, tak jak softwarový proces vyspívá, bude se zvyšovat i jejich použitelnost. V případě přijetí nové zakázky manažer tohoto projektu provede výběr nejvhodnějšího procesu z databáze procesů. Vzhledem k tomu, že si tímto vytváří svou lokální kopii procesu může tento měnit a modifikovat podle své potřeby (Obr. 3). V případě, že procesní inženýr akceptuje tyto změny jako obecně platné a užitečné, lze je odrazit i v původním procesu, nebo vytvořit v databázi nový proces pro danou třídu projektů.

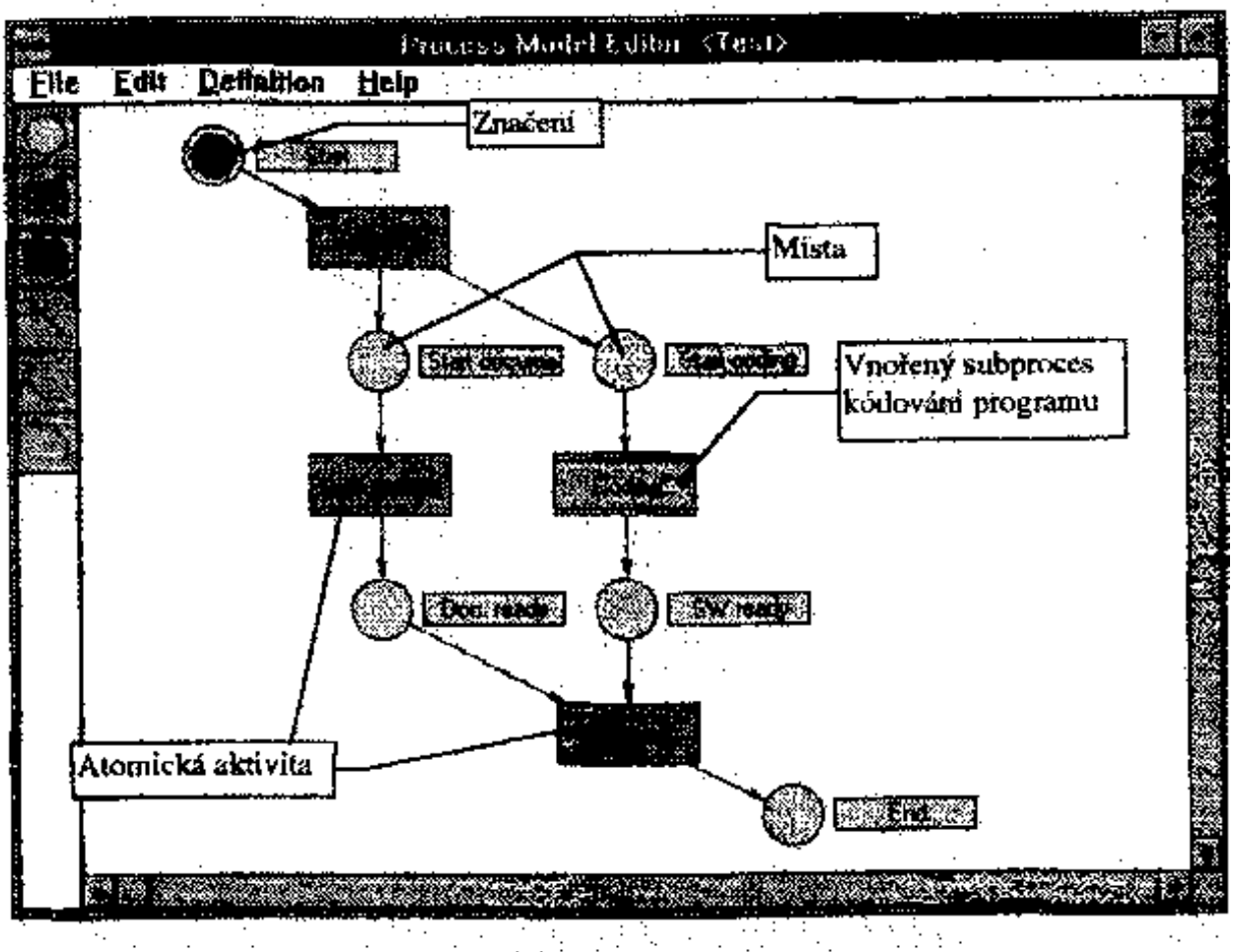


Obr. 3 Relace mezi procesy a projekty

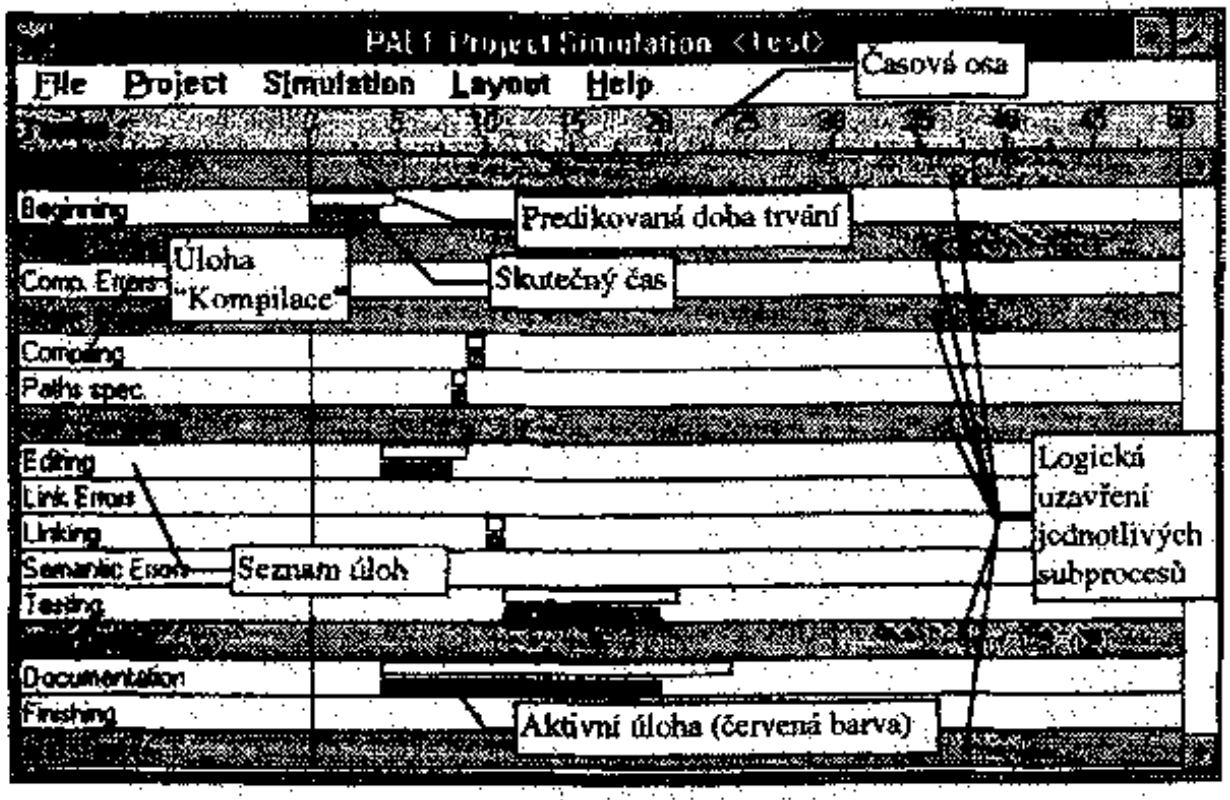
V současné době je systém implementován v prostředí MS-Windows a Windows NT a je tvořen následujícími subsystémy:

1. Editor modelu procesu
2. Simulace projektu
3. Manažer projektu
4. Člen řešitelského týmu projektu
5. Kalendář řešitele

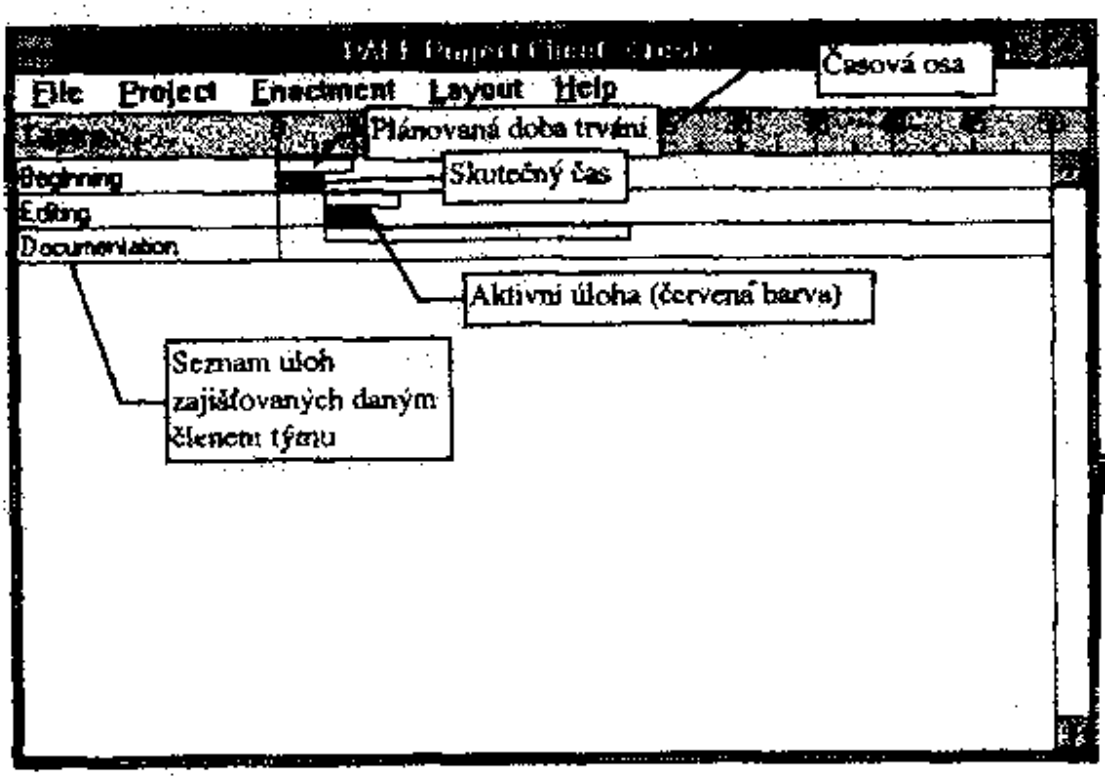
Prvé dva subsystémy mohou fungovat jako nezávislé aplikace zajišťující návrh a editaci procesu (Obr. 4) s jeho následnou konkretizací do projektu, jehož průběh může být simulací ověřen a optimalizován (Obr. 5). Následující dva subsystémy kooperují na principech architektury client/server v prostředí počítačové sítě. Instiliže aplikace *Manažer projektu* (server) zodpovídá za automatizované rozvržení jednotlivých úloh reprezentující jednotlivé procesní kroky (zobrazení je obdobné jako v případě simulace projektu Obr. 5), pak aplikace *Člen týmu* (client) aktualizuje, na základě informací dodávaných manažerem, seznam úloh zajišťovaných daným konkrétním členem řešitelského týmu (Obr. 6). Dále také poskytuje zpětnovazebnou informaci zpětky k manažeru projektu o stavu řešení těchto úloh. Poslední aplikací je *Kalendář řešitele*, který umožňuje plánovat a zaznamenávat jednotlivé úkoly řešitele včetně konkrétních úloh jim řešených v rámci daného projektu (Obr. 7).



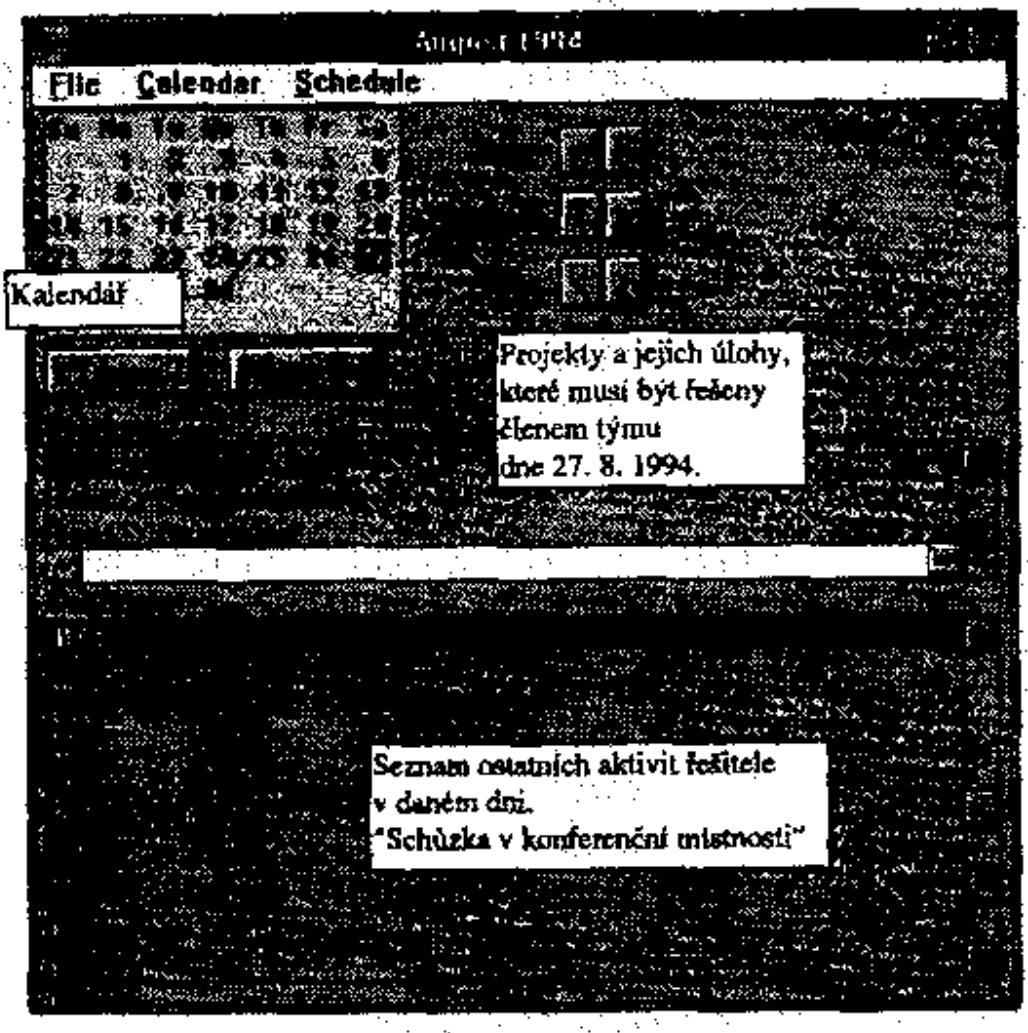
Obr. 4 Grafický editor modelu procesu: Vývoj programového vybavení



Obr. 5 Simulace projektu



Obr. 6 Seznam úloh zajišťovaných členem řešitelského týmu



Obr. 7 Kalendář řešitele

5. Několik slov závěrem

V současné době je systém PAEF využíván v testovacích úlohách jak na pracovištích řešitele tak na pracovištích zadavatele výzkumných grantů. Tyto testy by měly systém verifikovat i ověřit jeho použitelnost v praktických podmínkách. Nicméně už nyní lze konstatovat, že možnost využití formálního rámce pro definici procesu a jeho implementace prostřednictvím softwarového systému umožňuje celý postup zvyšování vyspělosti softwarového procesu urychlit a zefektivnit.

Literatura

[1] Peterson, J. L.: *Petri Nets*. ACM Comput. Surveys, Sept. 1977.

[2] Vondrák, I. - Moseley, W.: *Process Asset Engineering Facility*. Report on Research Grant. Systems and Information Science Laboratory, Texas Instruments, Inc. , Dallas, Texas, 1994

doc. Ing. Ivo Vondrák, CSc.
ved. katedry informatiky
Fakulta elektrotechniky a informatiky
VŠB - Technická univerzita Ostrava
tř. 17.listopadu
708 33 Ostrava - Poruba
tel.: (069) - 699 1272
e-mail: ivo.vondrak@vsb.cz