DOI: 10.35191/MEDSOFT_2018_1_30_44_60

MODELOVÁNÍ EHEALTH PROCESŮ V POMOCÍ HIERARCHICKÝCH STAVOVÝCH AUTOMATŮ (STATECHARTS)

Jiří Kofránek, Jiří Berger, Jiří Polák, Adam Vojtěch

Abstrakt

Při návrhu informačních systémů pro eHealth je podstatné dobře popsat strukturu procesů navrhovaných systémů. Modely využívající hierarchické stavové automaty (statecharts) jsou efektivním nástrojem pro jejich dynamický popis a následnou simulaci, která je podkladem interdisciplinárního porozumění mezi architekty informačních systémů, lékaři a tvůrci legislativy. Dobře navržené procesy umožní vyhnout se chybám v navrhované architektuře i nezbytné legislativní podpoře, které se později těžko napravují.

1 Nerozorané brázdy mezi obory

„Po mnoho let jsme byli dr. Rosenblueth a já přesvědčeni, že nejplodnějšími obory pro rozvoj věd jsou ty, které byly zanedbávány jako země nikoho mezi různými, pevně vymezenými vědními oblastmi" – napsal Norbert Wiener v úvodu své knihy, který dala název novému vědnímu oboru – kybernetice [1]. Měl pravdu – právě v nerozoraných brázdách na styku oborů se dají, obrazně řečeno, nalézt zlaté valounky řešení složitých problémů. Jednou z oblastí, kde mezioborová spolupráce je klíčem k řešení spletitých problémů, je rozvoj elektronického zdravotnictví (eHealth). Zdravotnické informační systémy mají ulehčit práci zdravotníkům. Prvním problémem je mezioborové porozumění mezi zdravotníky a informatiky. Informační systémy proto musí vznikat v neustálém vzájemném dialogu. Bez určitého překryvu znalostí se to ale neobejde. Informatici musí pochopit potřeby zdravotníků a nabídnout efektivní informatické řešení jejich potřeb, zdravotníci musí pochopit možnosti a způsob fungování navrhovaného informatického řešení.

Nejde ale jen o spolupráci zdravotníků a informatiků.

Pro občana platí, že činnosti, které nejsou legislativně upraveny, je dovoleno vykonávat libovolným způsobem. Ve veřejné správě je to naopak. Veřejná správa může vykonávat jen to, co jí zákon umožňuje. Ve veřejné správě je nutné veškerou činnost mít ošetřenou legislativně (veřejná správa může dělat jen ty činnosti, která má popsány v zákonech a nařízeních). Realizace národní strategie elektronického zdravotnictví proto musí být legislativně ošetřena – to ovšem vyžaduje také mezioborové porozumění i mezi informatiky, zdravotníky a legislativci, a konec konců i mezi politiky.

Legislativci musí dobře pochopit strukturu a funkci navrhovaného informatického řešení a na základě tohoto pochopení legislativně dostatečně jednoznačně popsat veškeré procesy v budovaném národním zdravotnickém informačním systému. Zde jsou zatím bohužel ještě velké problémy.

Z nedostatečné vzájemné komunikace a částečného nepochopení pak vznikají problémy, které nakonec někdy vedou i ke zbytečným sporům na půdě Poslanecké sněmovny i Senátu ohledně ochrany citlivých zdravotnických dat apod.

Ke vzájemnému porozumění mohou pomoci nástroje, které dostatečně věrně a srozumitelně popisují vzájemně provázané procesy. Jedním z prostředků pro popis propojených procesů jsou hierarchické stavové automaty (statecharts).

2 Stavové automaty a stavové diagramy

Nejjednodušším stavovým automatem je konečný automat, používaný v informatice jako teoretický model pro studium formálních jazyků. Popisuje velice jednoduchý počítač, který může být v jednom z několika stavů, mezi kterými přechází na základě symbolů, které čte ze vstupu. Množina stavů je konečná (odtud název), konečný automat nemá žádnou další paměť, kromě informace o aktuálním stavu. **Konečné automaty** se používají při vyhodnocování regulárních výrazů, např. jako součást lexikálního analyzátoru v překladačích. Konečný automat se dá formálně popsat jako uspořádaná pětice (S, Σ, σ, s, A), kde:

- S je konečná neprázdná množina stavů.
- Σ je konečná neprázdná množina vstupních symbolů, nazývaná abeceda.
- σ je tzv. přechodová funkce (též přechodová tabulka), popisující pravidla přechodů mezi stavy $S \times \Sigma \rightarrow S$
- s je počáteční stav, $s \in S$.
- A je množina přijímajících stavů, $A \subseteq S$.

Odpusťme si matematickou afektovanost a demonstrujme si konečný automat graficky (viz Obr. 1), kde kolečka znázorňují jednotlivé stavy a šipky (s přidruženým vstupním symbolem) mezi těmito kolečky popisují jednotlivé přechody. V daném případě tedy máme množinu tří stavů S0, S1 a S2. Tlustým kolečkem se šipkou je označen počáteční stav (v daném případě S0). Množina vstupních symbolů se sestává z dvou symbolů: 0 a 1. Přechodové funkce jsou znázorněny šipkami, které představují přechod ze stavu do stavu po přijetí vstupního symbolu. Konečný automat přijme určitou posloupnost vstupních symbolů a jako reakci na jednotlivé symboly se přepíná mezi jednotlivými stavy. Pokud po přijetí dané sekvence vstupních symbolů se automat ocitne v jednom z tzv. přijímacích stavů (v našem případě je množina přijímacích stavů reprezentována jedním stavem – stavem S0), hovoříme, že konečný automat vstupní sekvenci přijal, a v opačném případě ji nepřijal.

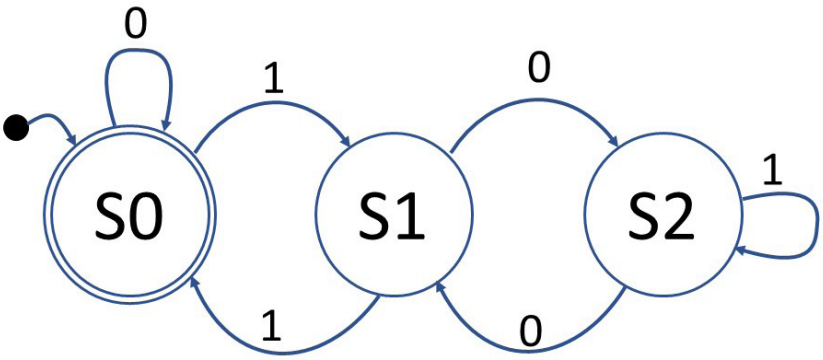
Chování konečného automatu si demonstrujme na příkladu zpracování vstupní sekvence 1011. Na počátku je automat ve stavu S0. Na vstup nejprve přijde první symbol, jednička. Z grafu vyplývá, že na příchod jedničky ve stavu S0 automat reaguje přechodem do stavu S1. Dále přichází nula, ze stavu S1 se příchodem nuly přechází do stavu S2. Poté přichází jednička, ze stavu S2 se příchodem jedničky přechází do stavu S2 (tzn. zůstává se ve stejném stavu). Nakonec přichází další jednička, takže automat opět zůstává ve stavu S2. Stav S2 ale nepatří do množiny A tudíž tento automat vstup 1011 nepřijal. Řetězec 1011 nepatří do jazyka přijímaného tímto automatem. Například sekvence 1111 (reprezentující ve dvojkové soustavě číslo 16) nebo 10110111 (ve dvojkové soustavě číslo 183) skončí ve stavu S0 a patří tedy do jazyka přijímaného tímto automatem. Automat, zobrazený na obr. 1 přijímá všechna čísla (zobrazená ve dvojkové soustavě) dělitelná třemi.

Konečný automat může také generovat výstup – výstupní symbol je generován na přechodu vždy po přijetí vstupního symbolu (tzv. Mealyho automat) nebo při vstupu do stavu (Moorův automat). Na Obr. 2 je modifikace automatu z Obr. 1. Generované výstupní symboly jsou v jednotlivých přechodech vyznačeny za lomítkem (jedná se tedy o Mealyho automat). Tento automat např. po přijetí sekvence 10110111 vygeneruje sekvenci 0100100 – tedy invertovanou sekvenci dvojkových čísel.

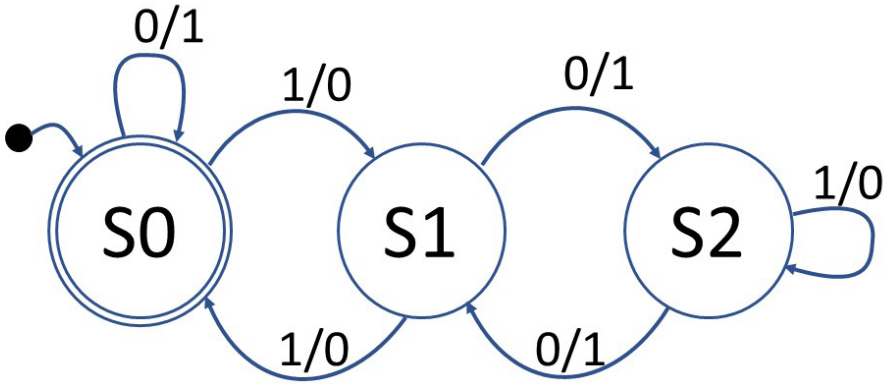
Stavové automaty se v informatice široce používají, např. jako součást lexikálního analyzátoru v překladačích.

Stavové automaty mají ale mnohem širší uplatnění. Obecně slouží k **popisu dynamiky systému**. Jejich pomocí můžeme poměrně jednoduchým a obecně srozumitelným způsobem graficky popsat dynamické procesy reálného světa tak, že si celou složitou dynamiku chování jednotlivých objektů rozdělíme na konečný počet stavů. Grafický zápis vývoje systému, který má konečný počet stavů se nazývá **stavový diagram** (anglicky State Diagram). Takovým systémem může být konečný automat (stavový automat) či další podobné systémy, které vyjadřují stavy určitého objektu a přechody (přechodovou funkci) mezi nimi. Diagram popisuje chování systému, které je vyjádřeno

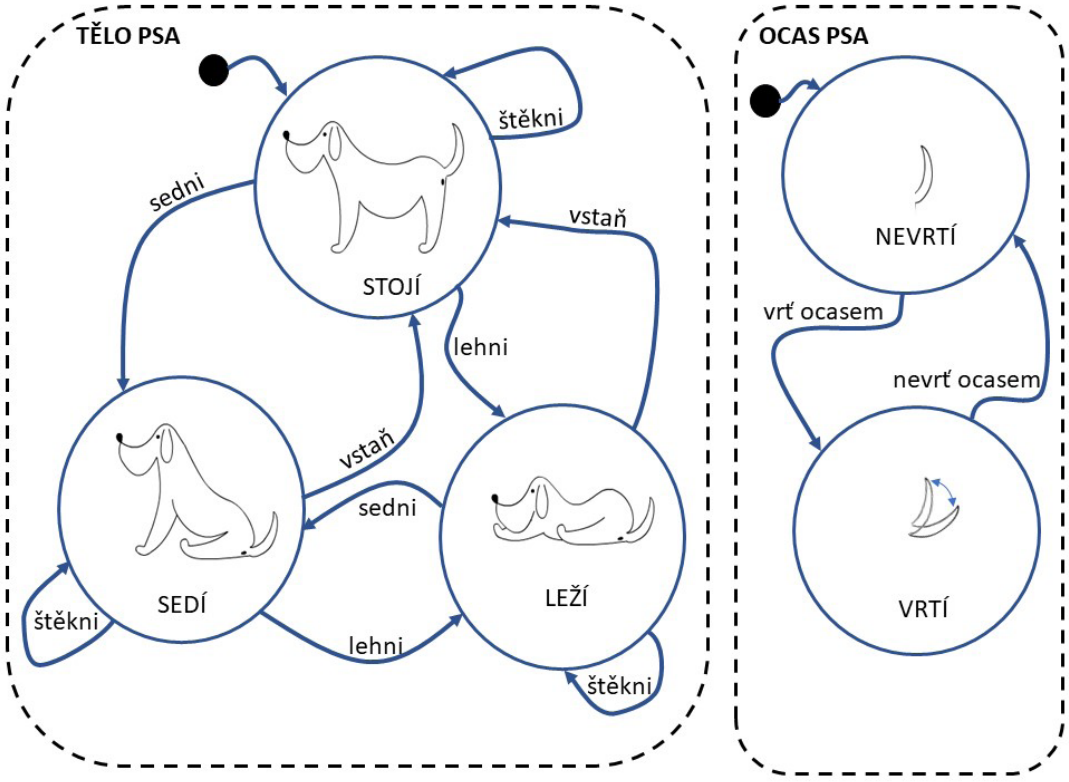
DOI: 10.35191/MEDSOFT_2018_1_30_44_60



Obrázek 1 – Konečný automat obsahující tři stavy, pracující s abecedou (množinou) symbolů {0,1}. Počáteční stav je S0. Pokud po přijetí sekvence symbolů automat skončí v přijímacím stavu (označeném dvojitým kolečkem - v daném případě je to stav S0), pak automat řetězec přijal (rozpoznal), pokud ne, tak ho nepřijal. Tak například sekvence znaků 100100010 nebyla přijata, sekvence 10010011 byla přijata. Automat rozpoznává čísla dělitelná třemi v dvojkové soustavě..



Obrázek 2 – Mealyho konečný automat – modifikovaný automat z Obr. 1. Při přechodech automat generuje znaky z výstupní abecedy (množiny). Výstupní abeceda je v daném případě {0,1}. Při přijetí sekvence 10010011 se generuje zároveň invertovaná sekvence 01101100.



Obrázek 3 – Stavový diagram zadávající chování animovaného grafického prvku.

průchodem stavy a je řízeno vnějším vstupem (vnějšími událostmi).

Ve stavovém diagramu si graficky nejdříve vyznačíme jednotlivé stavy, které mezi sebou propojíme přechody. Přechod z jednoho stavu do druhého je vyvolán nějakou vnější událostí. Přechodem může být i přechod do stejného stavu. Při definici stavového automatu je vždy jeden ze stavů označen jako výchozí (počáteční), v němž se objekt nachází na počátku. Pokud chceme stavovým automatem popsat také ukončitelnou činnost přidáme též i koncový stav.

Popis chování stavovým diagramem si ukažme na jednoduchém příkladu. Na Obr. 3 je zadaný jednoduchý scénář chování grafického objektu – animované figurky psa, která reaguje na vnější události (příkazy) „sedni", „lehni", „vstaň", „štěkni", „vrť ocasem" a „nevrť ocasem". Stavy jsou zobrazeny kružnicemi, přechody šipkami. Vstupní bod diagramu je označen tlustým kolečkem se šipkou vedoucí ke vstupnímu stavu. Grafické objekty se mohou sestávat z několika komponent – např. v daném případě se figurka psa sestává z těla a ocasu. Chování jednotlivých komponent je popsáno dvěma stavovými automaty („TĚLO PSA" a „OCAS PSA"). Chování těla je popsáno třemi stavy „STOJÍ", LEŽÍ",„SEDÍ" a chování ocasu popisují dva stavy „VRTÍ SE" a „NEVRTÍ SE".

Přechody mezi jednotlivými stavy jsou řízeny událostmi, a proto pomocí událostí bychom mohli figurku ovládat jako grafickou loutku. Figurku bychom třeba chtěli ovládat pomocí uživatelského rozhraní reagujícího na počítačovou myš. Tento požadavek můžeme také zapsat pomocí další sady stavových automatů (viz Obr. 4).

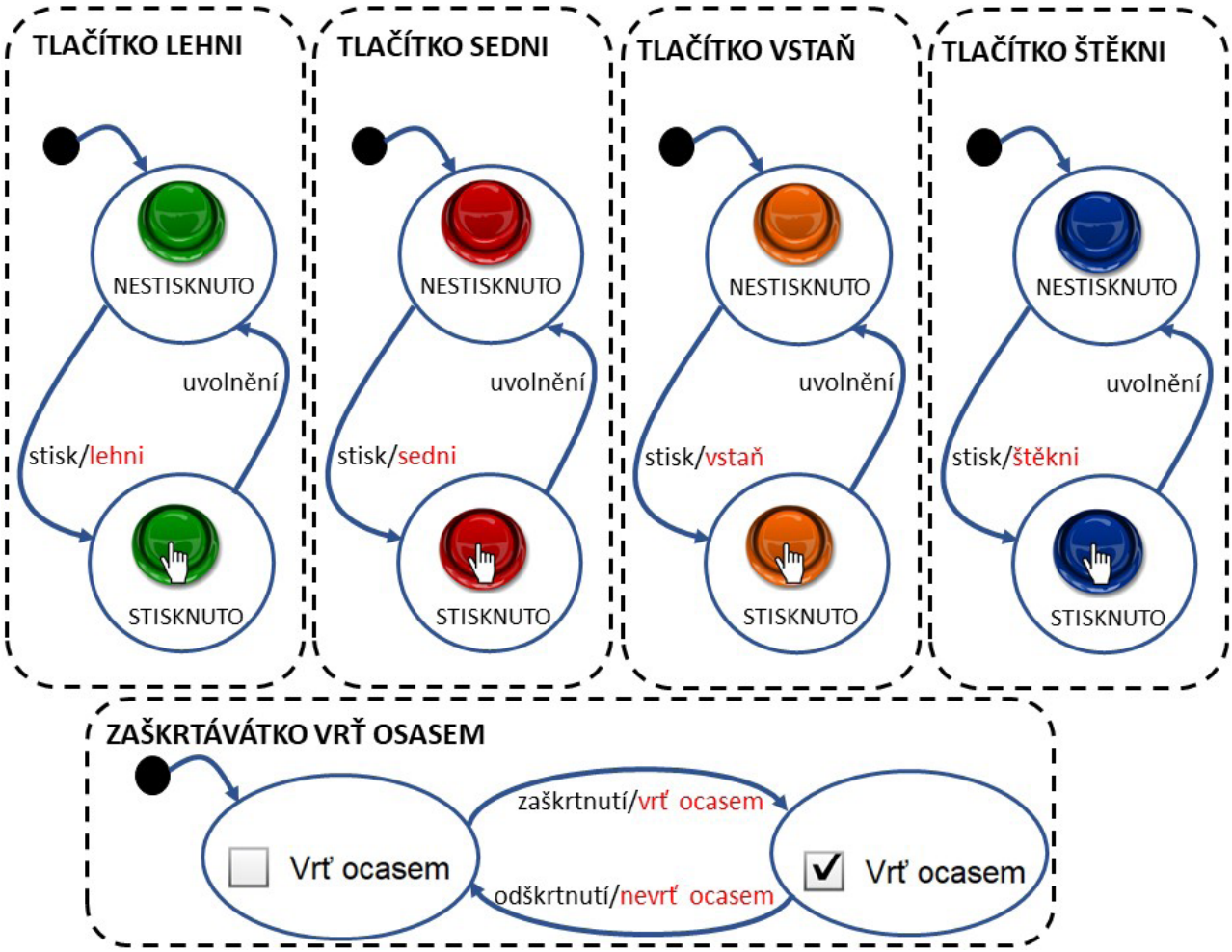
Stisk tlačítka nebo zaškrtávacího políčka (check box) vyvolá příslušné události a následné přechody ovládacích prvků. Při těchto přechodech se generují další události (obdobně jako u Mealyho automatu). Na tyto události pak reaguje stavový automat „TĚLO PSA" nebo „OCAS PSA". Tak např. událost „stisk", generovaná stiskem tlačítka ve stavovém automatu „TLAČÍTKO LEHNI" vyvolá další událost „lehni" na níž reaguje stavový automat „TĚLO PSA" příslušným přechodem do stavu „LEŽÍ". Vyvolanou událost zapisujeme po lomítku za původní událostí – v daném případě „stisk/lehni" – viz Obr. 4.

Pomocí stavových diagramů můžeme tímto způsobem zapsat požadované chování, kterými nutíme figurku psa sednout, vstát nebo lehnout a přitom vrtět či nevrtět ocasem.

Propojenými stavovými automaty tak můžeme zapsat přesný scénář požadovaného chování vyvíjené interaktivní grafické aplikace.

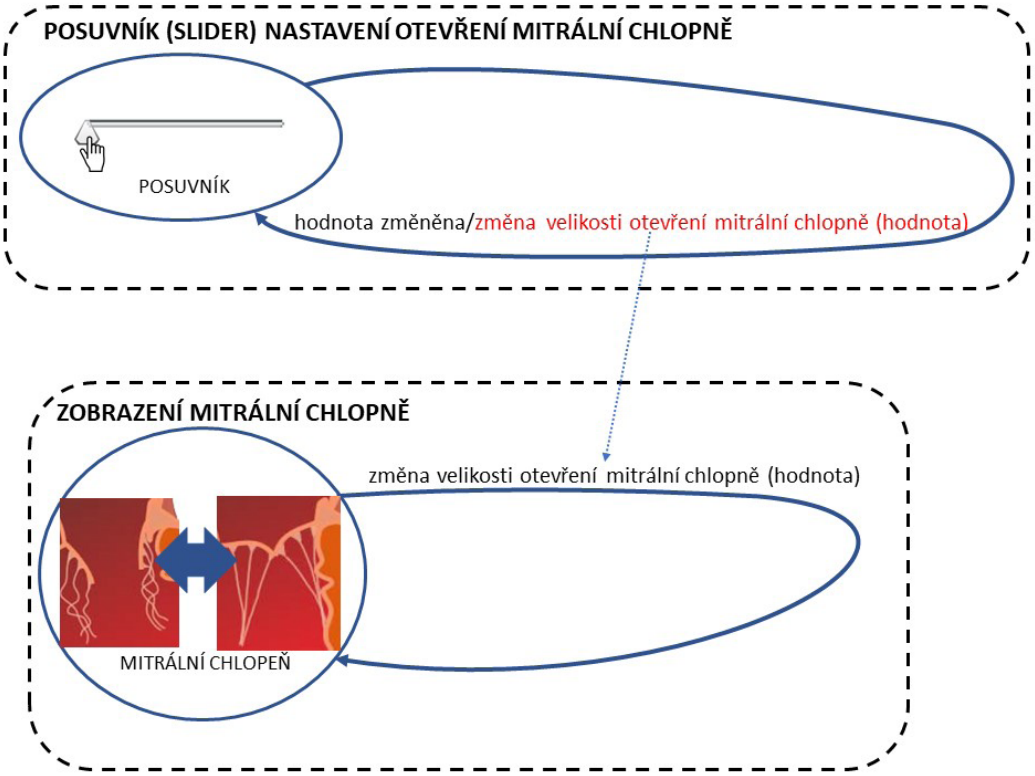
Popis chování grafických objektů pomocí stavových diagramů se nám velmi osvědčil při vytváření interaktivních výukových simulátorů. Tvorba simulátorů vyžaduje multioborovou spolupráci s výtvarníky – výtvarníci podle stavových diagramů vytvářejí grafické „loutky", které reagují na příslušné uživatelské vstupy nebo výstupy z modelu. Stavovými diagramy můžeme přesně popsat, co od výtvarníka chceme, jak se interaktivní grafický objekt bude chovat, a jaké bude mít návaznosti na uživatelské rozhraní a na model. Stavové diagramy, díky své srozumitelnosti a jednoznačné vypovídací schopnosti, jsou vynikající nástrojem pro mezioborovou spolupráci.

Jako příklad uvádíme vytvoření pohyblivé „loutky" srdce, ovládané modelem.

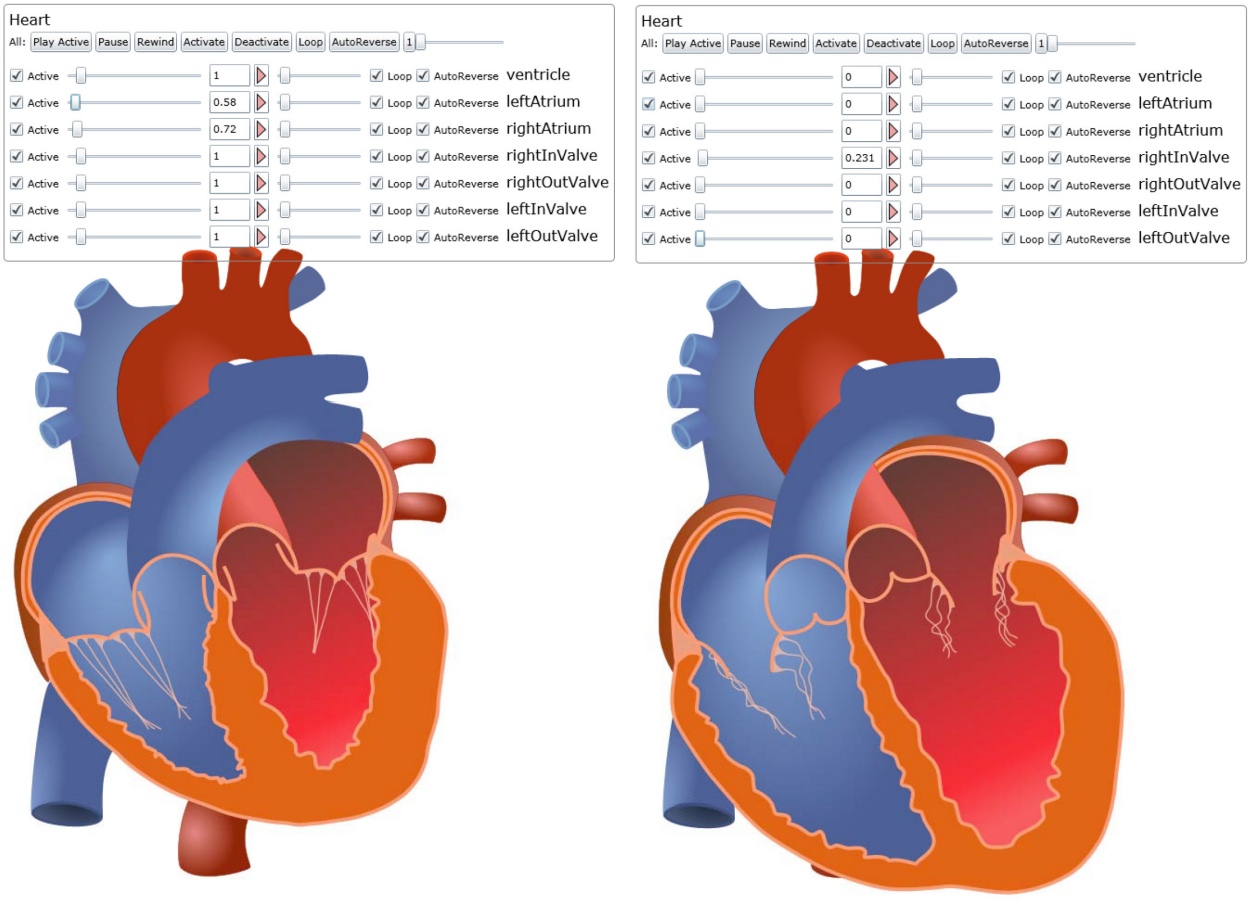


Obrázek 4 – Stavový diagram popisující reakci na uživatelské zásahy – ve spojení se stavovým diagramem z předchozího obrázku diagram popisuje ovládání animované postavičky psa.

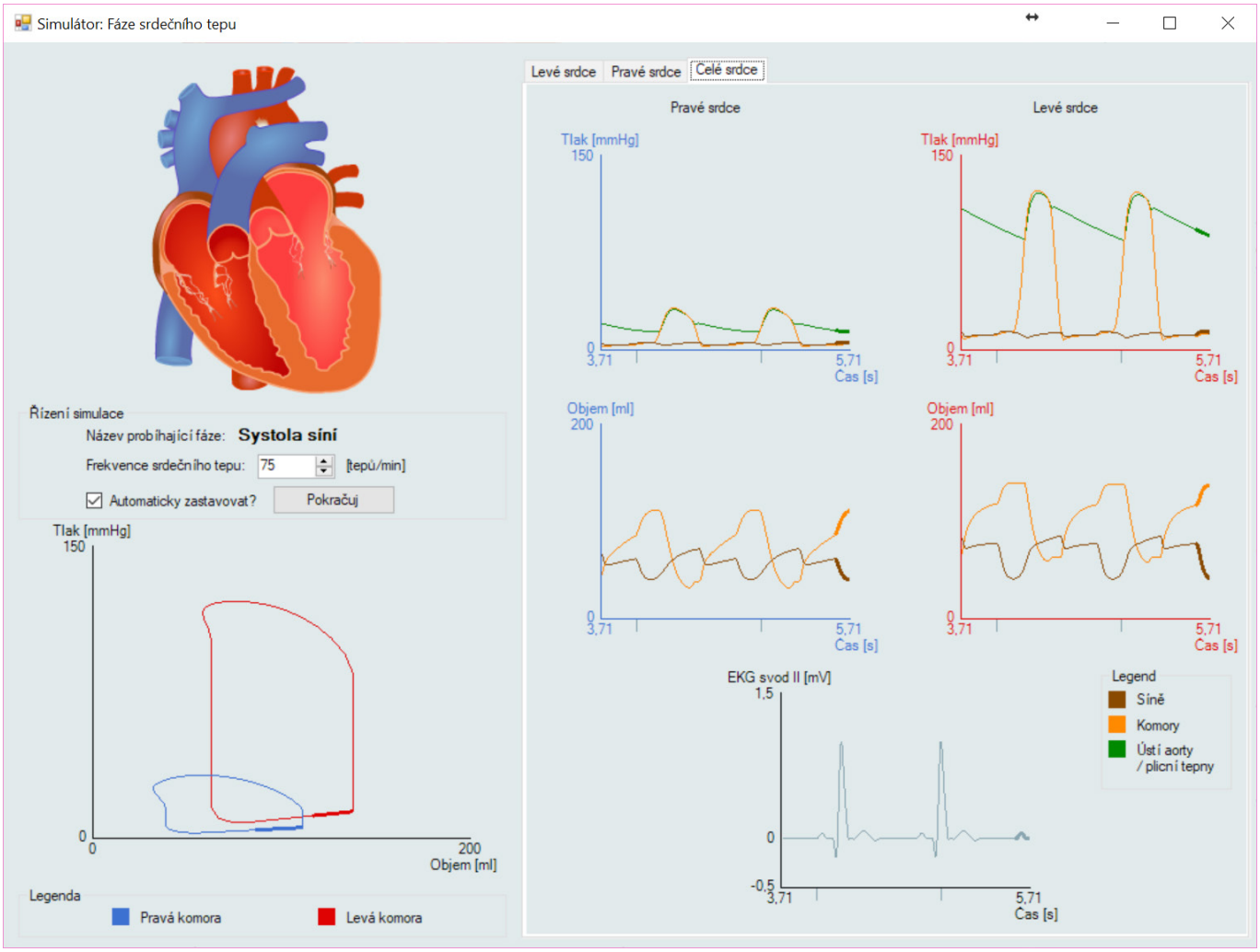
DOI: 10.35191/MEDSOFT_2018_1_30_44_60



Obrázek 5 – Stavový diagram popisující požadované ovládání grafické komponenty zobrazující mitrální chlopeň pomocí posuvníku. Při změně polohy posuvníku se generuje událost nesoucí hodnotu, podle které se nastaví otevření mitrální chlopně na hodnotu danou posuvníkem.



Obrázek 6 – Pomocný nástroj ANIMTESTER, který pomáhá grafikům vytvářet komplexní animované obrázky, jejichž tvar je ovládán „jako grafická loutka" nastavovanými hodnotami příslušných posuvníků..



Obrázek 7 – Výsledná výuková aplikace, v níž byl použit animovaný obrázek vytvořený výtvarníky s využitím nástroje ANIMTESTER. Tvar srdce je ovládán modelem na pozadí.

Nejprve pomocí stavových diagramů definujeme zadání pro grafika. Požadujeme, aby jednotlivé části srdce – (náplň komor a síní, otevírání a zavírání chlopní,) se řídilo posuvníky (slidery). Stavový diagram, zobrazující požadované ovládání jedné z komponent – otevírání a zavírání mitrální chlopně zobrazuje Obr. 5.

Výslednou aplikaci, vytvořenou podle tohoto zadání, zobrazuje Obr. 6. Grafik vytvořil pohyblivý obrázek srdce, jehož momentální tvar se „ručně" ovládá posuvníky jako „loutka". Abychom grafikům vytváření obdobně ovládaných interaktivních obrázků ulehčili, vytvořili jsme softwarový nástroj „Animtester", který jejich tvorbu usnadňuje [2].

Ve výukovém simulátoru se pak vstup hodnot měnících tvar srdce přepojil z posuvníků na výstupy ze simulačního modelu, a tvar srdce se už neovládal „ručně", ale byl řízen modelem na pozadí. Zobrazované křivky hodnot tlaků a objemů v síních a komorách pak byly provázeny pohyblivým obrázkem srdce s tepajícími síněmi a komorami a s otevíranými a uzavíranými chlopněmi (Obr. 7). Výukový simulátor je dostupný na adrese *http://physiome.cz/atlas/sim/SimulatorSrdceFaze*. Simulátor je součástí námi vytvářeného Atlasu fyziologie a patofyziologie [3] – *http://physiome.cz/atlas*.

3 Statecharts – vizuální formalizace chování komplexních systémů

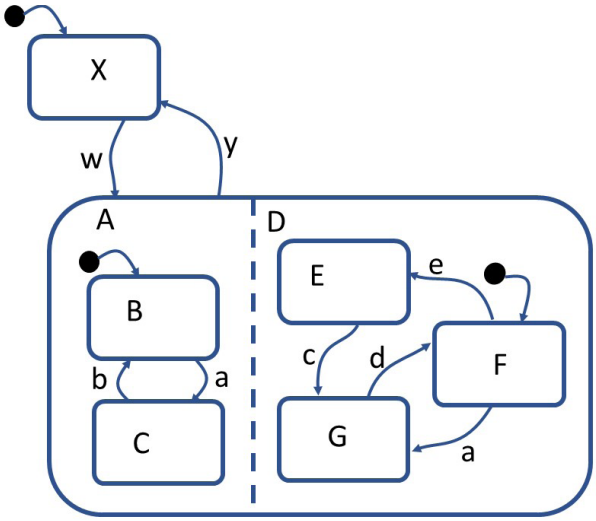
Pro popis chování složitých komplexních systémů klasické jednoúrovňové stavové automaty nestačí a jejich sémantiku bylo nutné doplnit. Zasloužil se o tom především David Harel. Popudem pro hledání nových vizuálních výrazových prostředků byla jeho účast na projektu vytváření softwaru pro avioniku vyvíjeného letadla pro Izraelské letectvo [4]. Komplexnost úlohy vedla k nově navržené technologii formalizace chování komplexních systémů, s využitím hierachických stavových automatů, kde stav v sobě může obsahovat další stavový automat, nebo několik stavových automatů, které běží paralelně (V Harelově notaci se paralelně běžící stavové automaty uvnitř daného stavu oddělují čárkami viz Obr. 8).

Ukažme si to na příkladu.

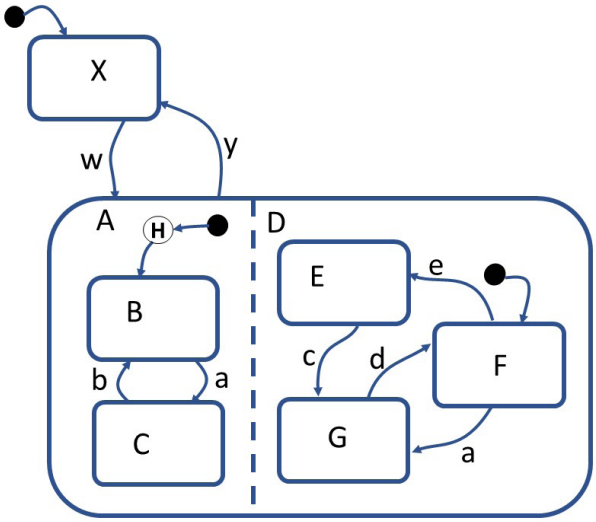
Automat zobrazený na Obr. 8 po iniciaci má aktivní stav X. Pokud obdrží událost "w", opustí se stav X a aktivuje se složený stav AD, resp. rozběhnou se v něm paralelně automaty A a D. Iniciálně se iniciuje stav A.B a stav D.F (tečkovou notací značím, že stav B je uvnitř automatu A a stav F je uvnitř automatu D). Pokud zvnějšku přijde událost "a", Automat A přejde se ze stavu A.B do stavu A.C a paralelní automat D přejde ze stavu D.F do stavu D.G. Když nyní přijde událost "y" opustí se složený stav AD a opět se aktivuje stav X. Vnitřní stavy A.C a D.G také přestanou být aktivní.

Když nyní přijde událost "w", opět se aktivuje složený stav AD. Přesto, že před předchozím opuštěním byl aktivní stav A.C a D.G, aktivují se stavy opět podle počátku, tj. aktivuje se stav A.B a A.F.

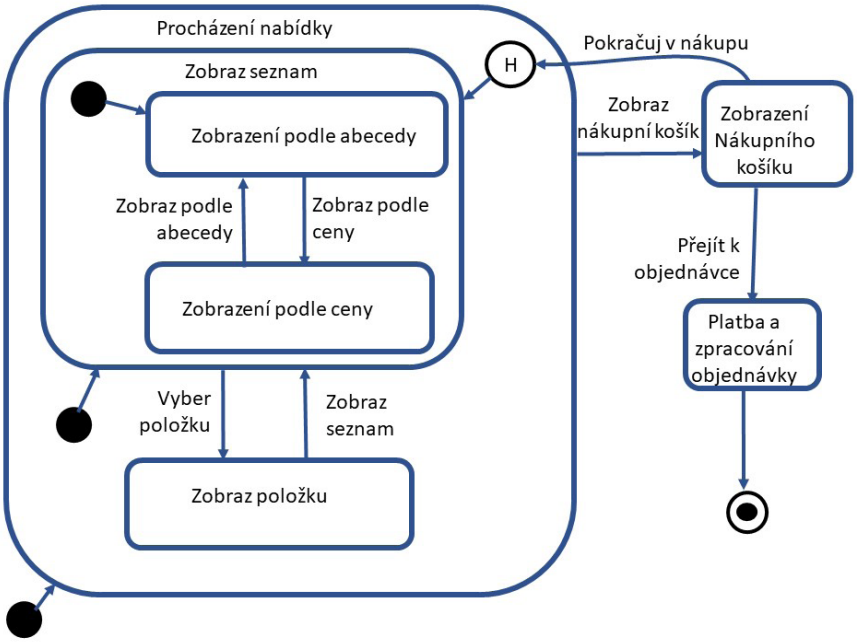
Pokud bychom chtěli, aby se po opětovné aktivaci složeného



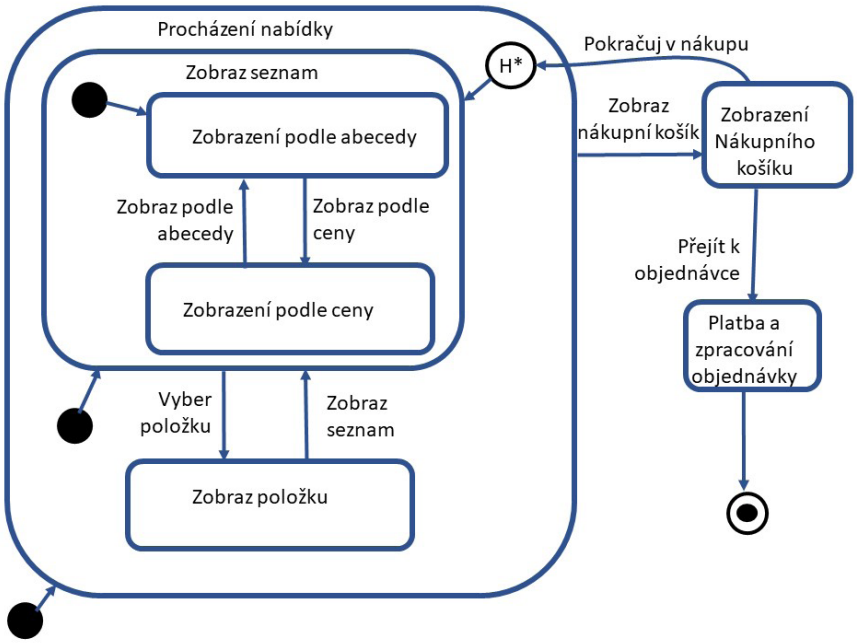
Obrázek 8 – Hierarchický stavový automat. Kompozitní stav AD obsahuje paralelně běžící stavové automaty.



Obrázek 9 – Hierarchický stavový automat s historií..



Obrázek 10 – Příklad uplatnění "mělké historie".



Obrázek 11 – Příklad uplatnění "hluboké historie"

stavu pokračovalo tam, kde se přestalo při předchozím opuštění, zavedl Harel speciální paměťovou buňku "historie". V Harelově notaci se Historie znázorňuje kroužkem s písmenem H.

Funkci si ukažme na příkladu (Obr. 9).

Po inicializaci přijde událost "w", automat přejde ze stavu X do kompozitního stavu AD. U automatu A inicializační šipka směřuje k symbolu "Historie", to znamená, že pokud byla nějaká předchozí historie, spustí se stav, který byl naposledy aktivní. Pokud ne, spustí se stav, na který ukazuje šipka od "Historie", tedy v našem případě se aktivuje stav A.B. (Protože žádná předchozí historie nebyla). U automatu D se aktivuje počáteční stav D.F. Po události "a" se aktivují stavy A.C a D.G. Pokud pak přijde událost "y", opustí se složený stav AD a opět se aktivuje stav X. Pokud se událostí "w" aktivuje znovu přechod do kompozitního stavu AD, v automatu A se aktivuje poslední stav, který byl uložen v "historii", tj. stav A.C, v automatu D (která nemá "historii") se aktivuje stav "F".

Harel také zavedl pojem "hluboká historie" (značí ji ve své notaci písmenem H s hvězdičkou). Hluboká historie znamená, že se do ní uloží historie všech podstavů.

Rozdíl mezi mělkou a hlubokou historií si ukažme na příkladu z reálného života. Na Obr. 10 je zobrazen stavový diagram chování výběru zboží v internetovém obchodě. Pokud vejdeme do obchodu, zobrazí se nám nabídky zboží podle abecedy. Můžeme si nabídku přepnout i na zobrazení podle ceny. Když si vybereme položku, můžeme si ji zobrazit – zmizí seznam a objeví se vybraná položka. Když si zobrazíme nákupní košík a vrátíme se zpět, historie si pamatuje, že jsme se dívali na položku. Pokud bychom se zpátky vrátili do zobrazení seznamu, seznam se zobrazí podle abecedy. Přepneme ho do zobrazení podle ceny a pak přeskočíme se podívat na nákupní košík. Po návratu se nám ale seznam opět zobrazí seřazený podle abecedy. Mělká historie si pamatuje pouze to, zda jsem se naposledy dívali na "Zobraz seznam" nebo na "Zobraz položku". Nepamatuje si poslední stav uvnitř stavu "Zobraz seznam", proto se "vnitřek" stavu "Zobraz seznam" inicializuje opět na položku "Zobrazení podle abecedy".

Na Obr. 11 je stejné schéma, ale s "hlubokou historií" (zobrazenou písmenem H s hvězdičkou). Pokud se z prohlížení seznamu seřazeného podle ceny "odskočíme" podívat do nákupního košíku, tak se při návratu dostaneme opět do seznamu položek zobrazených podle ceny (hluboká historie si pamatuje, kde jsme byli naposled i hluboko uvnitř složených stavů).

Harelův formalismus zápisu chování hierarchických stavových diagramů (statechart) má velkou vypovídací sílu. Podstatně rozšířil možnosti využití stavových diagramů pro vizualizaci chování i velmi komplexních systémů. Jedním z příkladů poměrně komplikovaného chování, je popis ovládání digitálních hodinek pomocí čtyř tlačítek. Stiskem těchto tlačítek se vyvolávají události a, b, c, d – reakci na tyto události popisuje hierarchický stavový automat, který Harel ve svém článku z roku 1987 [5] využil pro popis svého nového formalismu (Obr. 12).

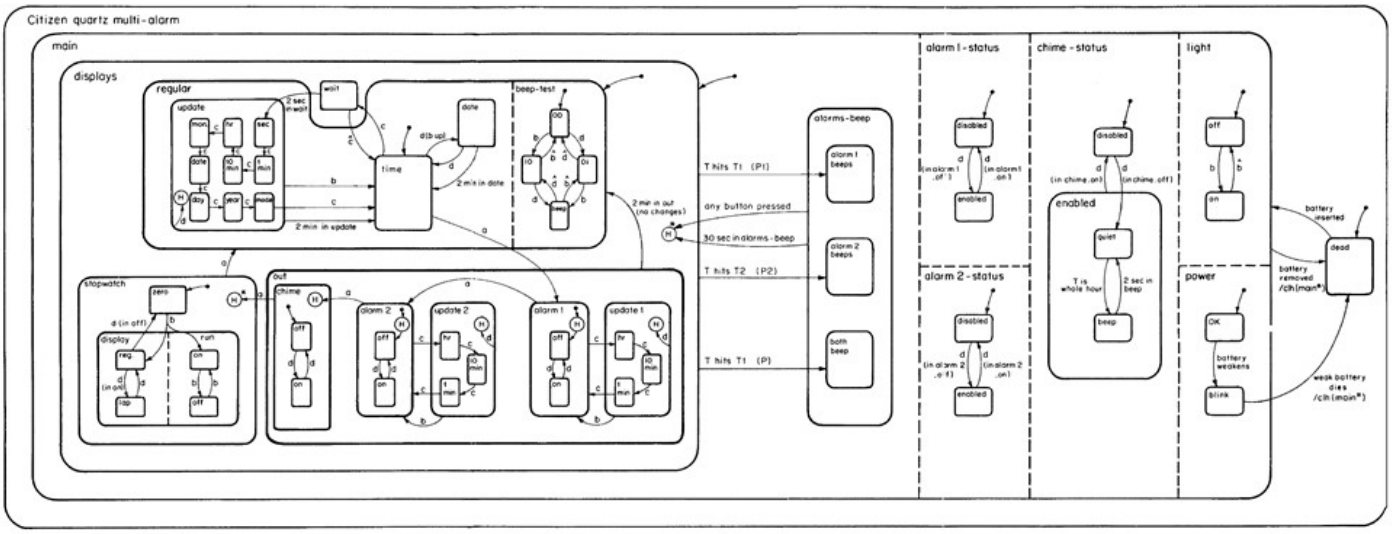
Harelův článek, nalezl velkou odezvu v odborné veřejnosti a Harelovu notaci (v různých úpravách) převzaly mnohé metodologie i softwarové nástroje pro modelování procesů. Základní myšlenky, které Harel navrhl, byly převzaty i do normy UML. Harelovy diagramy (s určitou úpravou) např. převzala do svých modelovacích nástrojů firma Mathworks – vytvořila nástroj "Stateflow", kterým je možno vytvářet hierarchické stavové automaty a propojovat je se Simulinkem a Matlabem. Hierarchické stavové automaty jsou i součástí standardní knihovny jazyka Modelica (viz Obr. 13).

Existuje i řada placených i Open Source nástrojů umožňujících vytvářet stavové diagramy, vizuálně testovat jejich chování a podle nich pak generovat i zdrojový kód v různých programovacích jazycích – patří k nim např. nástroj "YAKINDU Statechart Tools", který je možné pro nekomerční použití stáhnout z adresy *https://www.itemis.com/en/yakindu/state-machine*.

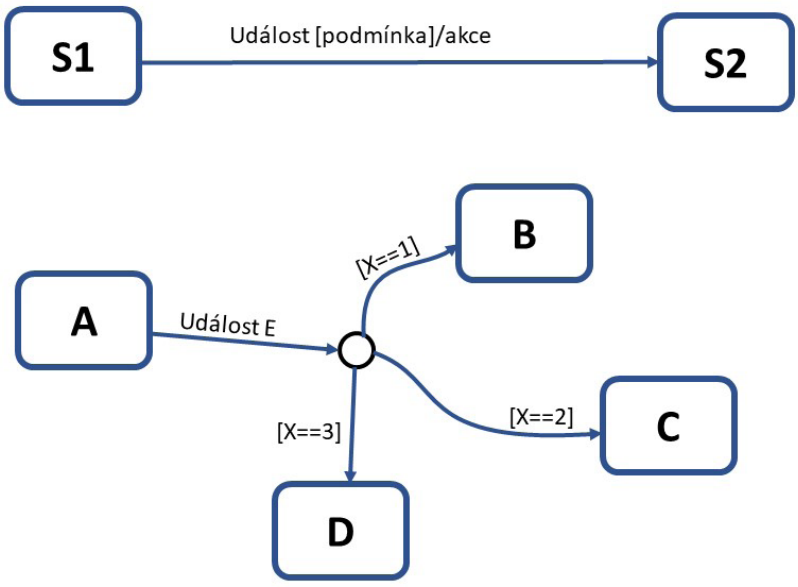
4 Nástroje a metody modelování procesů

Hierarchické stavové automaty jsou samy o sobě dobrým formalizmem pro vizualizaci chování složitých systémů. Předpokládá se ale, že modelovaný systém, jehož chování simulujeme pomocí stavových diagramů, již dobře známe. To je časté u technických systémů a zařízení (viz Harelův příklad popisu funkce hodinek). Pokud ale systém nemáme přesně specifikovaný, ke stavovým diagramům, popisujícím jeho chování, je poměrně dlouhá cesta.

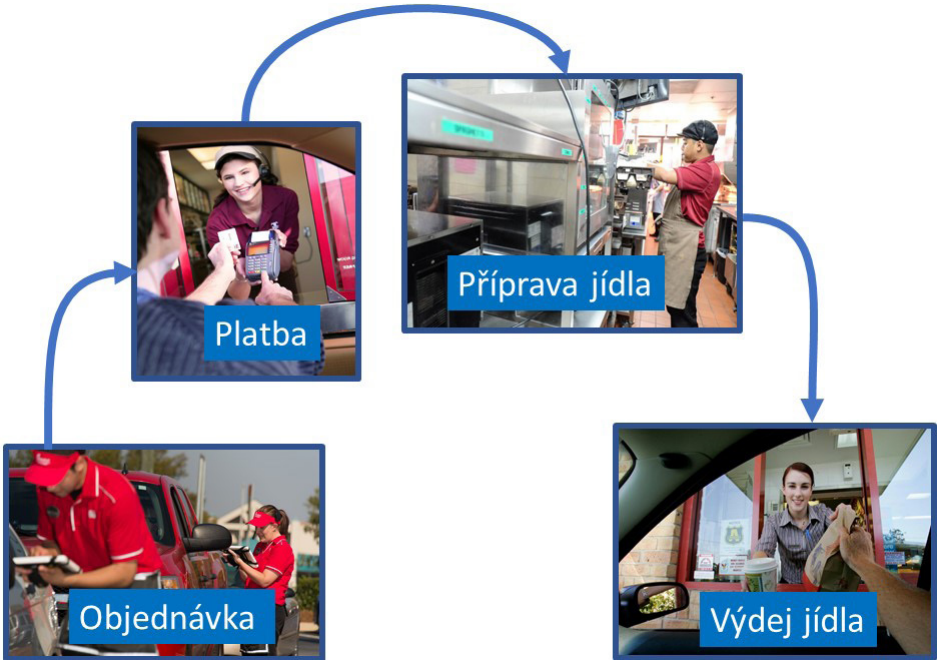
Uveďme si nějaký jednoduchý příklad – třeba objednávání jídla z auta (drive through) v systémech rychlého občerstvení. Řidič přijíždí autem k okénku objednávek – tam už na něj (v moderních systémech) čeká obsluha s tabletem, zaplatí jídlo, v ku-



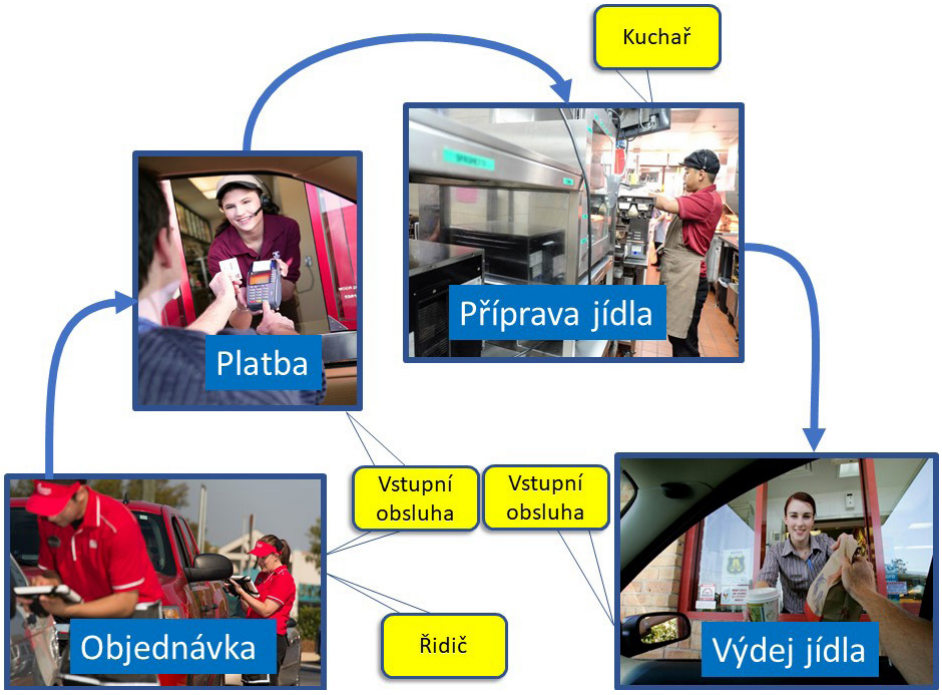
Obrázek 12 – Stavový diagram (statechart) popisující činnost hodinek Citizen Quartz Multi-Alarm III z původní Harelovy práce [5].



Obrázek 13 – Příklad formalismu hierarchických stavových automatů v nástroji Stateflow od firmy Mathworks. Podmínky přechodu, testování podmínky, za které je přechod možný, i následná generovaná akce se zapisují ve formě Událost [podmínka]/akce. Na obrázku je také zobrazeno větvení – když zvnějšku zazní "Událost E", testují se podmínky (podle směru hodinových ručiček). Pokud je nějaká z nich splněna, přechod se uskuteční a stav se změní. Pokud ne, přechod se neuskuteční.



Obrázek 14 – Na první pohled jsou procesy objednávání jídla z auta (drive-through) v systémech rychlého občerstvení jednoduše popsatelné stavovými diagramy.



Obrázek 15 – Ve skutečnosti je systém složitější. Nejprve je zapotřebí určit participanty těchto procesů a určit jejich role a návaznosti..

chyni ho zatím připraví. Řidič pak jede k výdejovému okénku a tam přebere hotové jídlo (Obr. 14). Na první pohled se tento příklad zdá jako velmi jednoduchý systém a rovnou bychom chtěli kreslit stavový diagram, jako bychom tam vlastně měli jen čtyři sekvenční stavy – Objednávka – Platba – Příprava jídla – Výdej jídla.

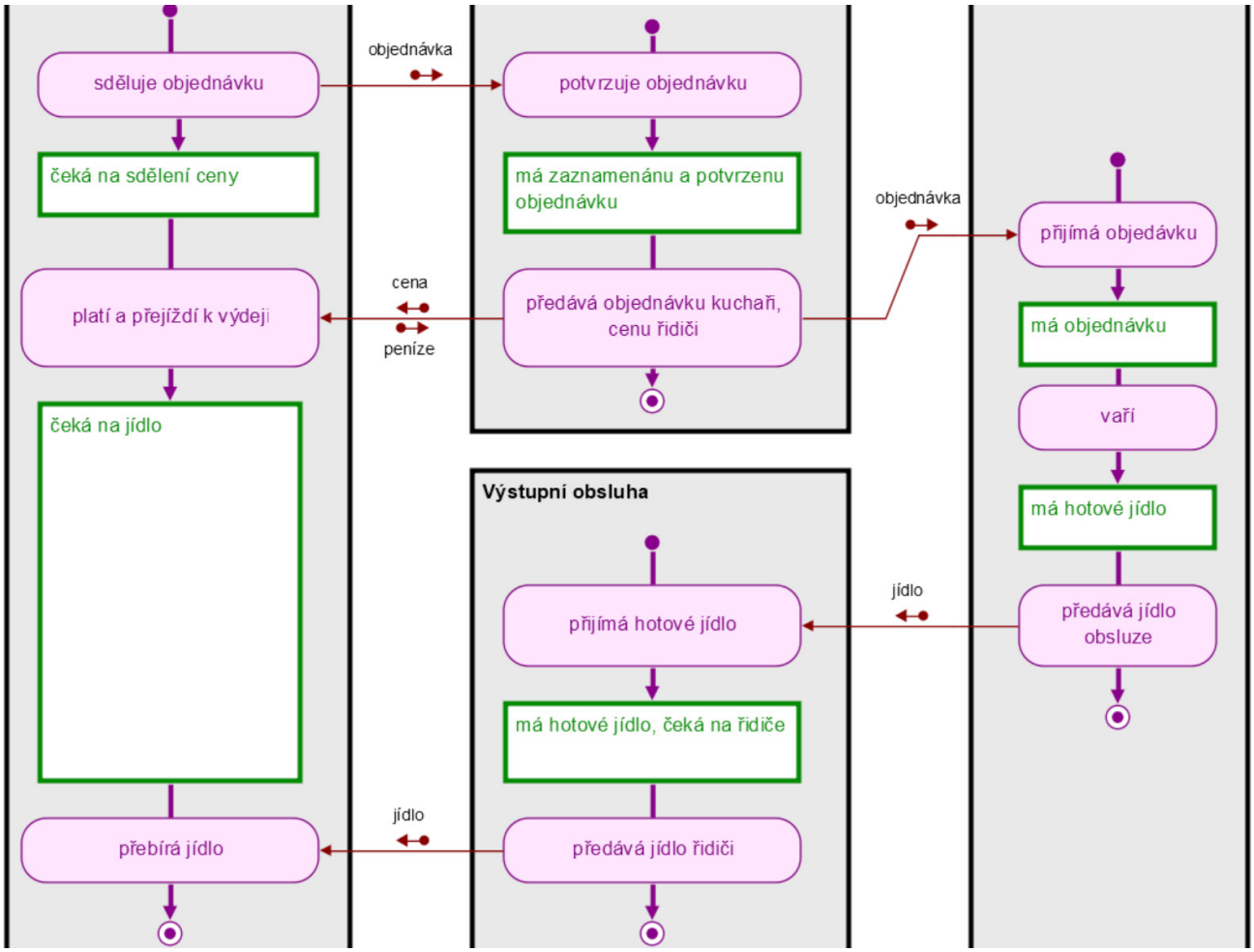
Ale pozor, systém je složitější. Nejedná se o jeden sekvenční proces. Řidičů – zákazníků je více, po vyřízení objednávky jednoho řidiče se vyřizuje objednávka dalšího řidiče v řade, platby se také přijímají postupně od různých řidičů a kuchař také nevaří jen pro jednoho zákazníka.

Nejprve je třeba definovat kolik účastníků, nebo, přesněji řečeno, participantů (nemyslíme tím jen fyzické osoby) se účastní.

V daném případě se jedná o řidiče – ten dává objednávku, platí a nakonec dostává jídlo.

Dalším participantem je vstupní obsluha – ta přijímá objednávky a platbu od řidiče. Dále je participantem kuchař (neboli kuchyně – nemyslíme tím jednu konkrétní osobu, ale instituci). Ten přijímá objednávku, vaří jídlo a předává hotové jídlo výstupní obsluze. Výstupní obsluha předává hotové jídlo řidiči (Obr. 15).

Zdá se tedy, že součástí řešení bude ne jeden, ale čtyři



Obrázek 16 – První verze procesního diagramu objednávání jídla z auta.

procesní diagramy popisující činnost řidiče, vstupní a výstupní obsluhy a kuchaře.

Procesní diagram je zobrazen na Obr. 16.

Stavy jsou zobrazeny hranatými obdélníky a přechodové procesy mezi stavy zobrazují obdélníky s kulatými rohy.

Stavy mezi sebou komunikují a tím se spouští přechody. Nejprve řidič přijíždí a sděluje objednávku. Vstupní obsluha potvrzuje objednávku a proces vstupní obsluhy přechází do stavu "má zaznamenánu a potvrzenu objednávku". Řidič je zatím ve stavu "čeká na sdělení ceny". Přechodový proces "platí a přejíždí k výdeji" se spustí teprve až dostane zprávu od procesu "předává objednávku kuchaři, cenu řidiči" atd.

Popis procesu může být více podrobný, pokud budeme podrobněji zvažovat tok peněz, vydávání účtenky a nutnost mít ingredience pro přípravu jídla (Obr. 17). Přidáme tedy ještě dalšího účastníka – "pokladnu". Pokladna nemá stav, má jen proces "přijímá peníze, vydává drobné a účtenku". Budeme-li popisovat více podrobně i činnost kuchaře, a popisovat získávání a spotřebu nutných ingrediencí pro připravovaná jídla, přibude nám další participant – "stav zásob".

Procesní diagramy byly nakresleny pomocí nástroje CraftCASE, který není určený jen na kreslení, ale jde o softwarový nástroj primárně určený pro modelování, testování a simulaci procesů (Obr. 18).

Nástroj CraftCASE je distribuován anglickou společností The CRAFT.CASE LIMITED Company, a jak vidíte, je i v češtině. Není divu, jeho autorem je Jiří Berger, spoluautor tohoto článku. Na tento nástroj mu byl udělen US Patent číslo 7,904,431 s názvem "METHOD AND SYSTEM FOR AUTOMATED REQUEST MODELLING.

Nástroj Craft CASE je založen na softwarové podpoře metodologie BORM. BORM je zkratka pro Business Objects Relation Modelling.

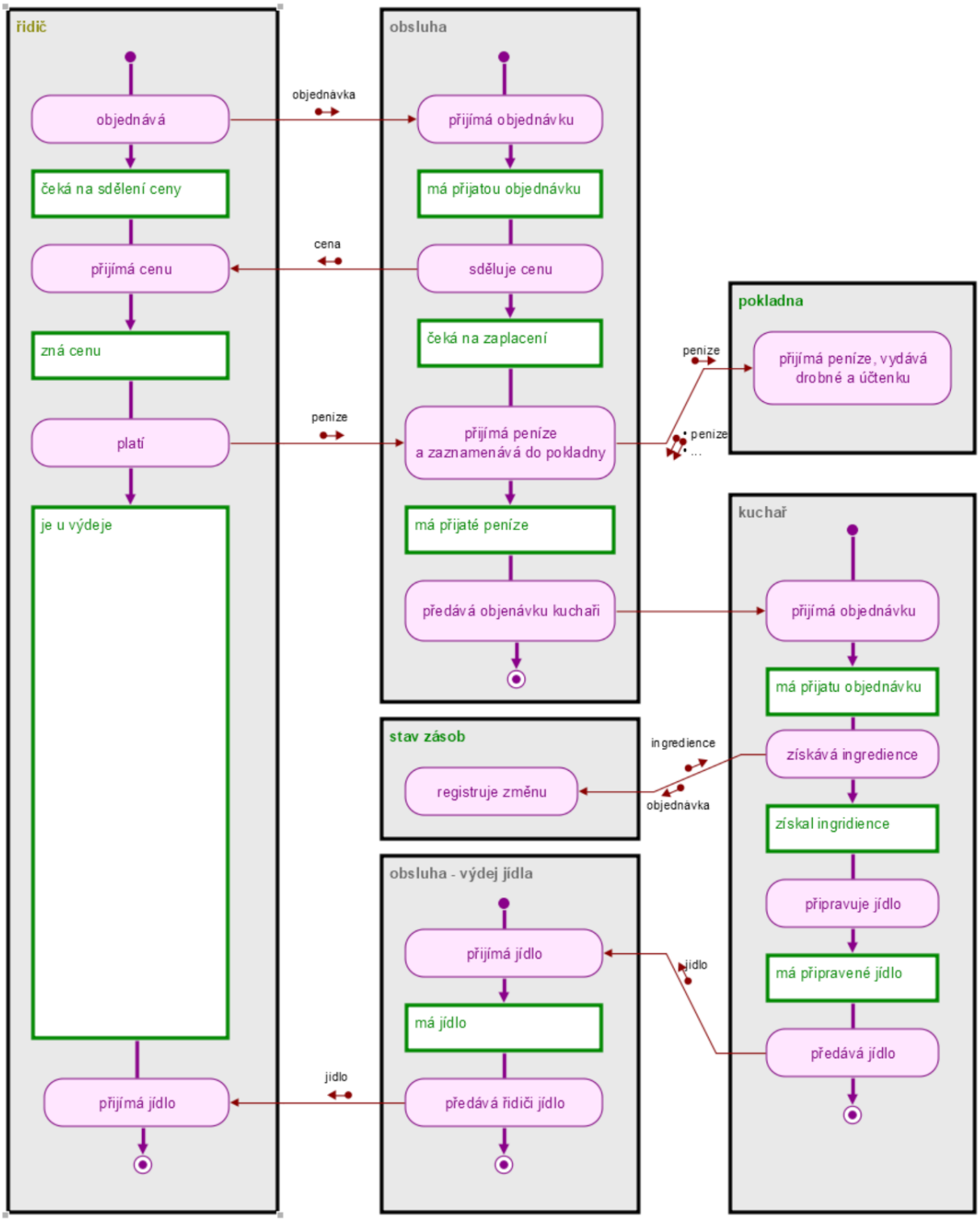
Slovo "business" má v angličtině širší význam než obchod – reprezentuje v podstatě jakoukoli činnost. A business inženýrství se věnuje analýze a pochopení procesů. Metodologie BORM [6,7] se zaměřuje se na **překonání propasti mezi business inženýrstvím a softwarovým inženýrstvím**. Metoda je vyvíjena od roku 1993. Vznikla jako projekt VAPPIENS financovaný British Council. Vývoj byl podporován poradenskou společností Deloitte od roku 1996, kdy se začala používat pro velké poradenské projekty.

Metoda byla úspěšně používána poradenskou společností Deloitte Advisory Czech Republic v rámci velkých projektů pro Českou poštu, energetiku, telekomunikační společnosti, aerolinky a další klienty. Přesnější popis metody je nad rozsah tohoto článku.

Nástroj CraftCASE byl úspěšně využit pro modelování projektu "Procesní modelování agend veřejné správy" – výsledek modelovaní procesů je zveřejněn na adrese *https://rpp-aism-pub.egon.gov.cz/AISM* (viz Obr. 19). Jednotlivé procesní diagramy je možné prohlížet a také i animovat (výstupem nástroje CraftCASE je totiž i interaktivní animace modelovaných procesů, kterou je možné spouštět ve webovém prohlížeči. Dva příklady procesních diagramů agend občanských průkazů a cestovních dokladů jsou uvedeny na Obr. 20 a 21.

5 Modelování procesů pro eHealth

Legislativa v oblasti eHealth musí přesně popisovat procesy týkající se použití zdravotnických informačních systémů včetně



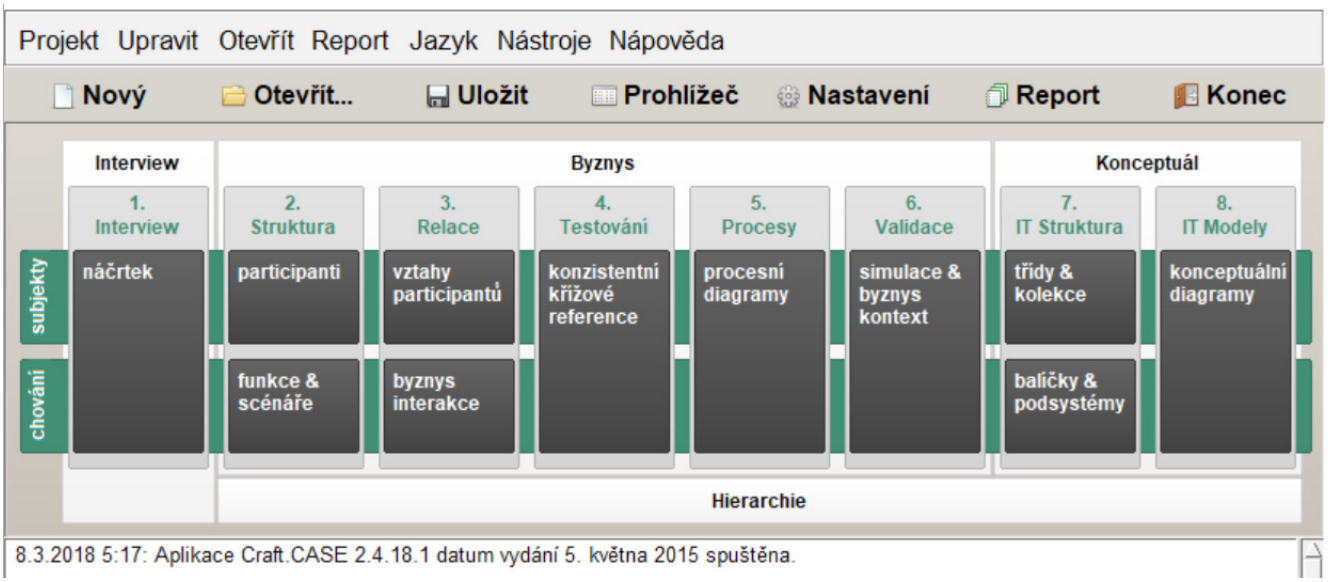
Obrázek 17 – Přesnější verze procesního diagramu objednávání jídla z auta.

všech jejich návazností. Musí odrážet představy softwarových architektů i zdravotníků.

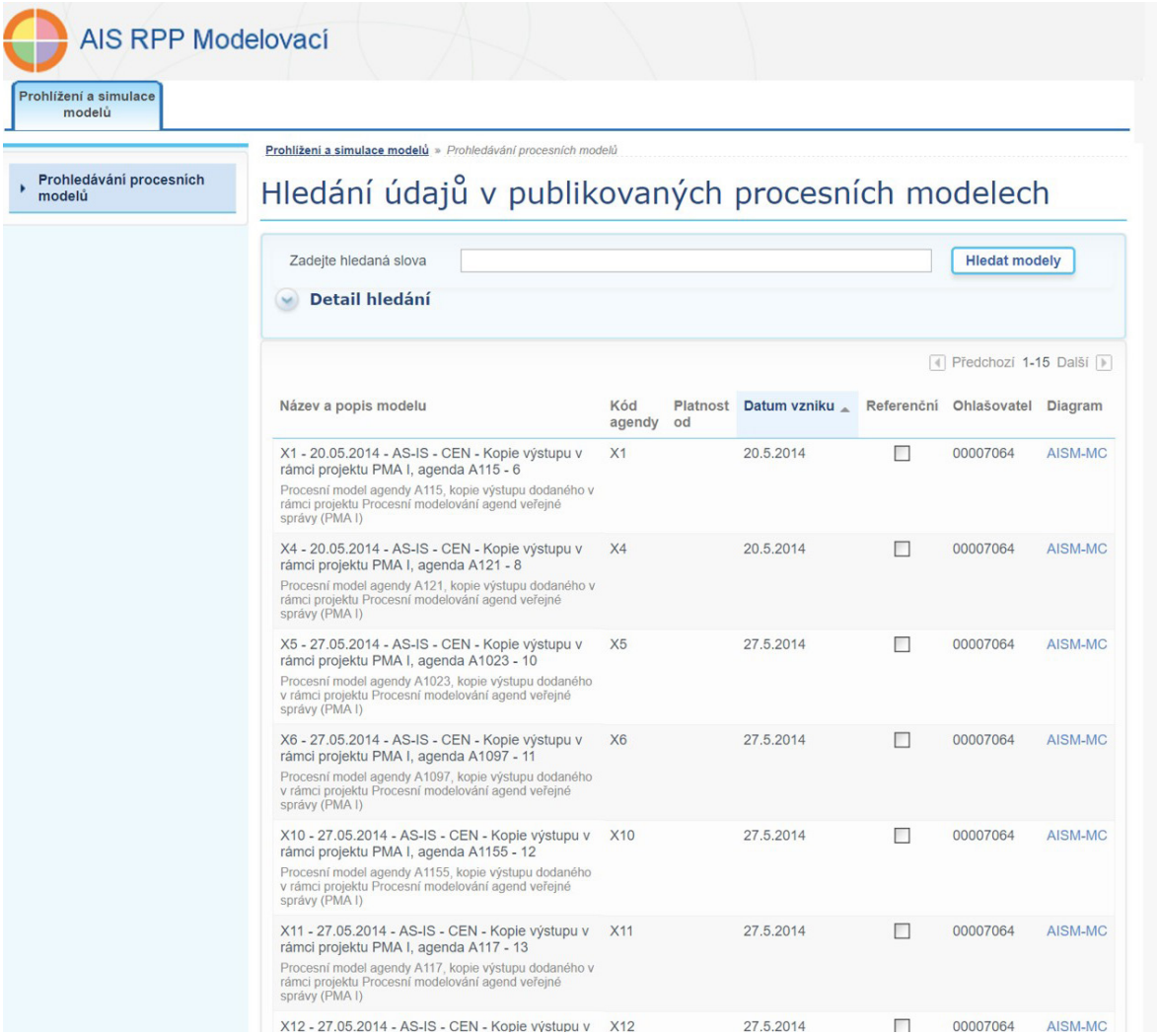
Jak již bylo řečeno v úvodu, veřejná správa může vykonávat jen to, co jí zákon umožňuje. Veřejná správa může dělat jen ty činnosti, která má popsány v zákonech a nařízeních. Znamená to v průběhu tvorby národního systému elektronického zdravotnictví mít legislativně ošetřeny veškeré výstupy. To ale předpokládá mezioborovou spolupráci a mezioborové porozumění mezi zdravotníky, informatiky i tvůrci legislativy.

Procesní modelování je nástrojem mezioborového porozumění. Krom toho, simulace procesů umožní vyhnout se procesním chybám. (Kuplříkladu kdy v nějaké konstelaci určitý stav čeká nekonečně dlouho a nelze spustit jeho další přechod do jiného stavu). Výsledkem takového omylu jsou legislativní chyby, na které se pak přijde až v průběhu realizace. Modelování procesů umožňuje tyto problémy odhalit předem.

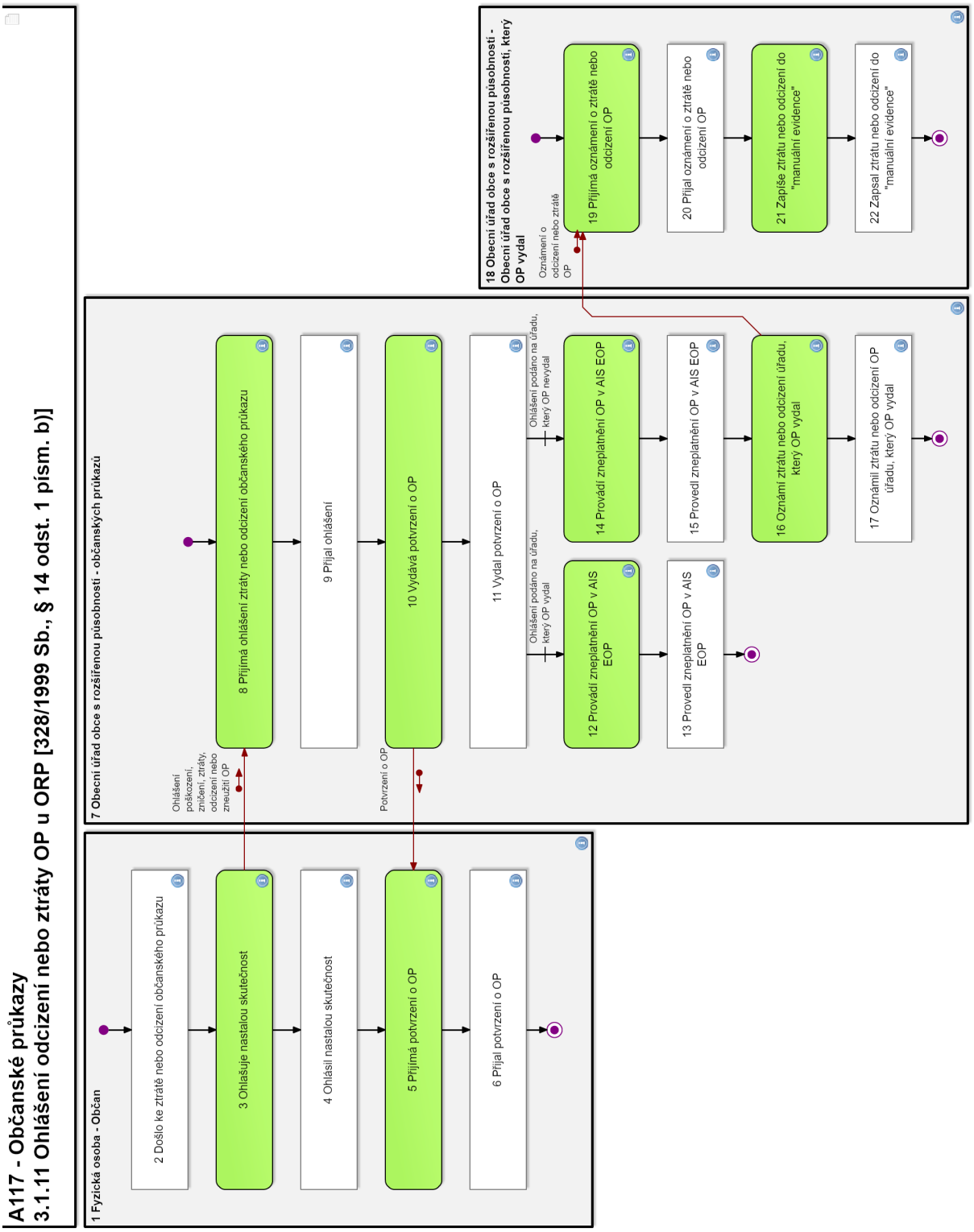
V současné době probíhá analýza a modelování procesů elektronické preskripce, jako podklad pro tvorbu příslušné legislativy (Obr. 22 a 23).



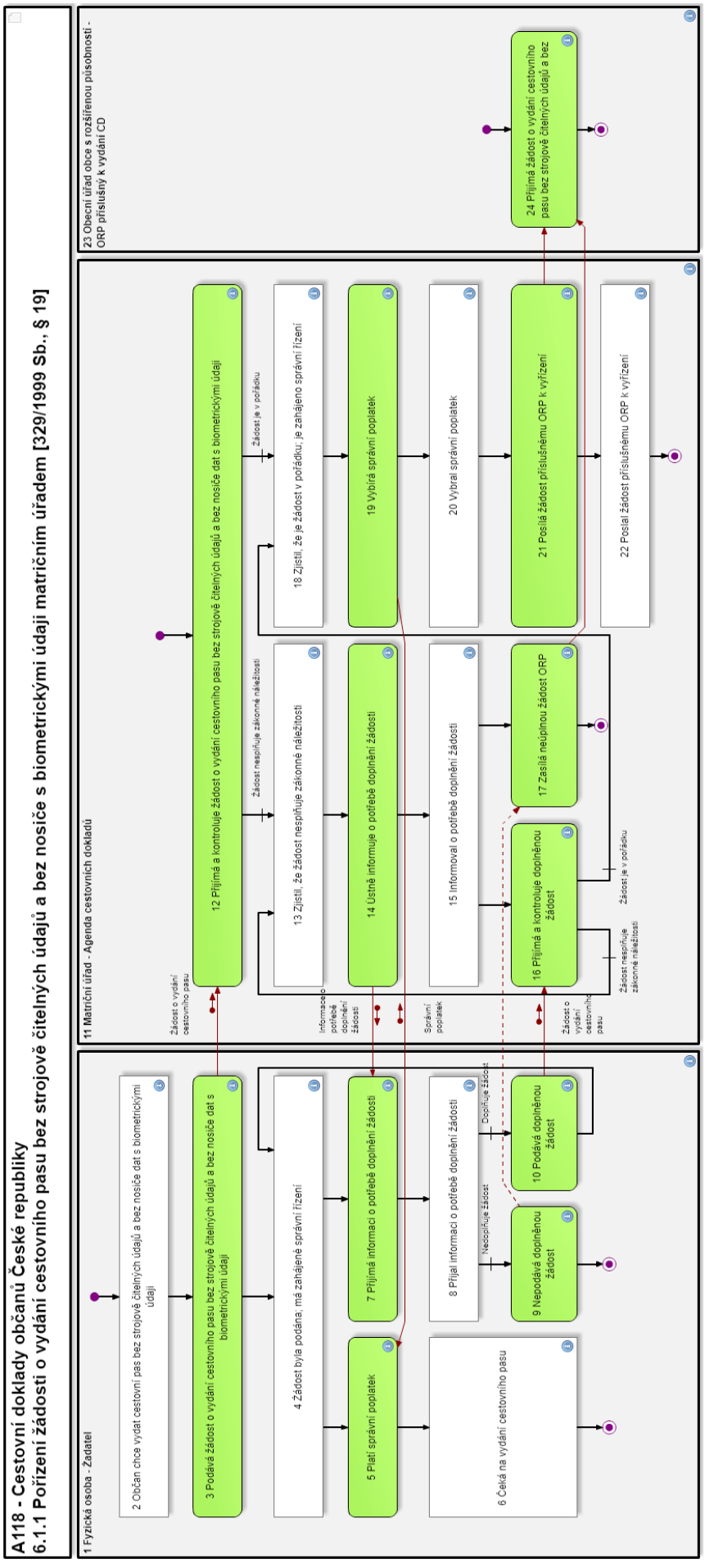
Obrázek 18 – Softwarový nástroj CraftCASE pro modelování, testování a simulaci procesů



Obrázek 19 – Výsledky projektu "Procesní modelování agend veřejné správy". Kliknutím na položky vpravo můžete zobrazit jednotlivé procesní diagramy. V procesních diagramech je možné spouštět animaci, která krok po kroku simuluje průběh a návaznosti modelovaných procesů.

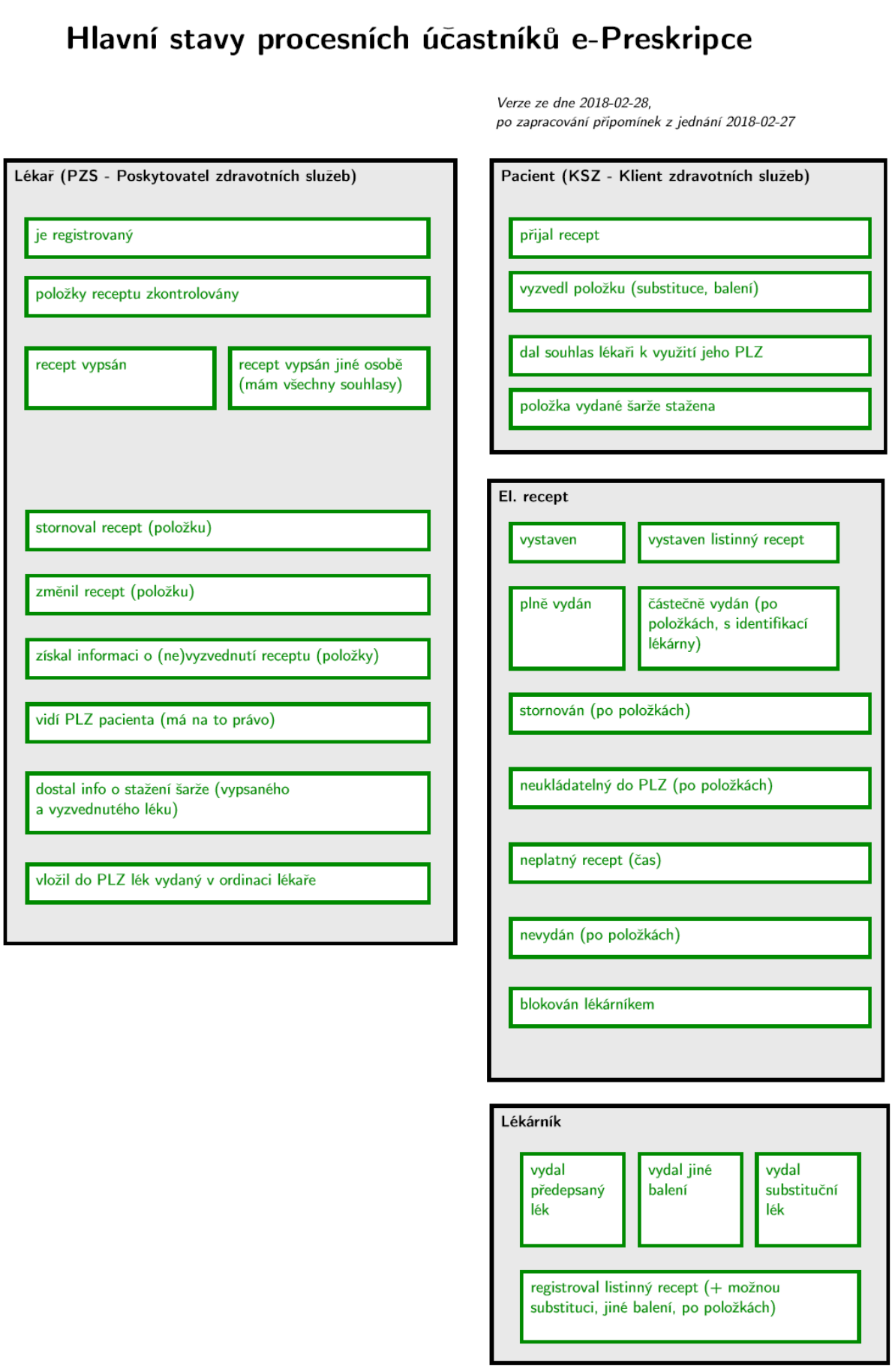


Obrázek 20 – Procesní diagram ohlášení odcizení nebo ztráty občanského průkazu

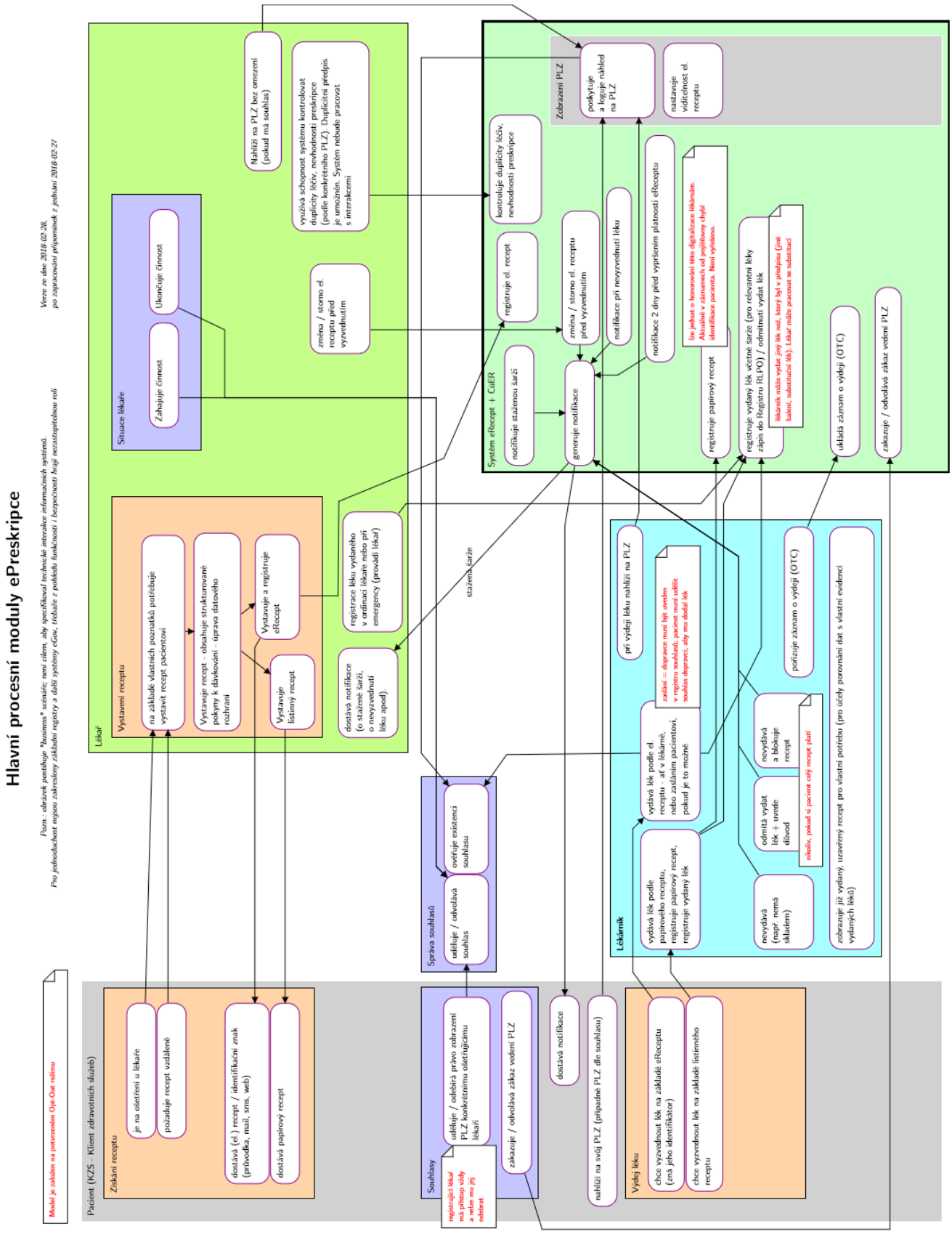


Obrázek 21 – Procesní diagram z agendy cestovních dokladů občanů České republiky

DOI: 10.35191/MEDSOFT_2018_1_30_44_60



Obrázek 22 – Jeden z průběžných výsledků modelování e-Preskripce. Výsledek byl podkladem pro diskusi v pracovní skupině a dnes je již změněn. Uvádíme jej zde jen pro ilustraci podkladů, které pracovní skupina používá. Podklady jsou průběžně generované nástrojem CraftCASE.



Obrázek 23 – Další z průběžných výsledků modelování e-Preskripce. Výsledek byl podkladem pro diskusi v pracovní skupině a dnes je již změněn. Uvádíme jej zde jen pro ilustraci podkladů, které pracovní skupina používá. Podklady jsou průběžně generované nástrojem CraftCASE.

Literatura

[1.] Wiener N. Cybernetics: Control and communication in the animal and the machine. Wiley New York; 1948.

[2.] Kofránek J, Mateják M, Privitzer P. Web simulator creation technology. MEFANET report. 2010;3: 32–97.

[3.] Kofránek J, Matoušek S, Rusz J, Stodulka P, Privitzer P, Mateják M, et al. The Atlas of Physiology and Pathophysiology: Web-based multimedia enabled interactive simulations. Comput Methods Programs Biomed. 2011;104: 143–153.

[4.] Harel D. Statecharts in the Making: A Personal Account. Commun ACM. New York, NY, USA: ACM; 2009;52: 67–75.

[5.] Harel D. Statecharts: a visual formalism for complex systems. Science of Computer Programming. Elsevier; 1987;8: 231–274.

[6.] Knott R, Merunka V, Polak J. The BORM methodology: a third-generation fully object-oriented methodology. Knowledge-Based Systems. Elsevier; 2003;16: 77–89.

[7.] Knott RP, Merunka V, Polak J. Process modeling for object oriented analysis using BORM Object Behavioral Analysis. Proceedings Fourth International Conference on Requirements Engineering ICRE 2000 (Cat No98TB100219). ieeexplore.ieee.org; 2000. pp. 7–16.

Kontakt

Jiří Kofránek

Oddělení biokybernetiky a počítačové podpory výuky
ÚPF 1. LF UK
U Nemocnice 5 128 53,
128 53 Praha 2
e-mail: kofranek@gmail.com

Jiří Berger

e-FRACTAL, s.r.o.
Vinohradská 174
130 00 Praha 3
e-mail: jiri.berger@e-fractal.cz

Jiří Polák

CACIO, z.s.
Na Cihlářce 30
150 00 Praha 5
e-mail: polakjiri@email.cz

Adam Vojtěch

Ministerstvo zdravotnictví ČR
Palackého nám. 4
128 01 Praha 2
e-mail: adam.vojtech@mzcr.cz