Jak spíchnout SCORM

Příručka pro vývojáře výukového obsahu standardu SCORM 1.2

Verze 1.2 – koncept 0.8

Claude Ostyn
Stratég výukových standardů
Click2Learn, Inc.

Překlad (bez záruky): Jan Škorvánek

Obsah

Jak spíchnout SCORM 1

Obsah 2

Úvod 3
Přehled 3
Komu je dokument určen 4
A co SCORM 1.3? 4
Nástroje tvorby(Authoring tools) 4
Příklady 5

Kapitola 1 – náhled na SCORM 6
Co je SCORM? 6
Co znamená SCORM konformní výukový obsah (SCORM-compliant content)? 6
Druhy výukových objektů standardu SCORM 7
Online nebo Offline výukové objekty 7
Organizace a řazení výukových objektů 8

Organizace výukových objektů a objektů SCO 8
Organizace položek 8
Podobnost s ostatními modely organizačního uspořádání výukového obsahu 8
Řazení 9
Zvláštní pravidla pro některé položky 9

Implementace SCORM konformních výukových objektů a objektů SCO 10
Runtimová služba 10
Primitivní SCO 10
Datové SCO (Data enabled SCO) 11
Nalezení adaptéru API 12
SCORM komunikační relace 12

Kapitola 3 – Vytvoření jednoduchého SCO 14
Vytvoření primitivního SCO 14
Obecný script pro jednoduché objekty SCO 15

Kapitola 4 – Vaše SCO jako webová stránka 16
Okno „scéna“ 16
Management okna 16
Rozšíření 17

Kapitola 5 – přeměna obsahu zobrazitelného v prohlížeči na SCO 17

Kapitola 6 – Tvorba vícestránkového SCO 20
Tři metody k uchování stavu ve vícestránkových SCO 20
Použití framesetu pro vícestránkové SCO 21
Složitější příklad 21

Kapitola 7 – řízení stavu SCO a komunikace 23
Složitější opětovně použitelný SCO script..... 23
Složitější SCO, které při uvolňování z paměti nahlašuje data o průchodu kurzem 28
Složitější SCO, které hlásí data o průchodu kurzem v okamžiku, kdy se objeví ... 30
Nejasnosti statusu..... 31
Jak a kdy nastavovat informace o statusu 31

Kapitola 8 – vrstvená architektura objektů SCO 32

Kapitola 9 – pozastavení a obnovení..... 33
Vícestránkové SCO, které může být pozastaveno na kterékoli stránce..... 33
Opětovné použití obecné logiky 35

Kapitola 10 – sledování cílů v rámci SCO..... 38

Kapitola 11 – Zobrazení objektu SCO ve fullscreenovém módu..... 43

Kapitola 12 – Balení SCORM konformního obsahu..... 48
Organizační model IMS & SCORM obsahu 48
Vytvoření balíčku SCORM 49
Cesty k obsahu a adresářům 50
Uspořádání objektů SCO 50

Dodatek: Různorodé zdroje a poznámky k implementaci..... 51
Sekvenční diagram 51
Ukázka manifestu balíčku SCORMu 1.2..... 52

O autorovi..... 56

Úvod

Přehled

Referenční model sdíleného obsahu (SCORM – Shareable Content Object Reference Model), uveřejněný v projektu ADL (Advanced Distributed Learning), je de facto standardem e-learningového obsahu. Tento dokument popisuje, jak vytvořit webový obsah, který splňuje specifikace SCORMu 1.2. Stejně jako specifikace SCORMu je tento dokument technickým dokumentem, nikoliv instruktáží tvorby výukového obsahu. Stejně jako SCORM se nezabývá vzdělávací kvalitou výukového obsahu, ale prvky, které jsou nezbytné k tomu, aby byl tento obsah přenosný a splňoval požadavky standardu SCORM. Dokument obsahuje funkční příklady k ilustraci různých aspektů specifikací SCORMu.

Tento dokument je rozdělen na několik kapitol. Po rychlém náhledu na SCORM je každá kapitola založena na základním modelu objektu SCO, aby bylo možné postupně vystavět způsobilejší a složitější výukové objekty odpovídající standardu SCORM. Závěrečná kapitola shrnuje základy „balení“ – jak shromáždit výukové objekty do SCORM konformního balíčku, který může být přemisťován a rozvinut v kterémkoliv SCORM kompatibilním systému pro distribuci výukového obsahu studentům.

Komu je dokument určen

Pokud již používáte vývojový nástroj nebo vývojové prostředí, které odpovídá standardu SCORM, jako je ToolBook nebo Aspen 2.1 LCMS, pravděpodobně nebudete tento dokument potřebovat, ačkoli jeho úvod může přispět k porozumění základní technologii a pravidlům.

V závislosti na důvodu a zájmu můžete číst tento dokument různými způsoby:

- Jako příručku ke specifikaci SCORMu, abyste získali lepší představu o tom, co ve specifikaci je a co tam není a prohlédli si příklady, dokumentující použití ve výukovém obsahu.
- K porozumění různým technickým aspektům specifikace SCORMu v příkladech.
- Jako technickou příručku se sadou příkladů, které mohou pomoci uspořit čas při reálné implementaci výukového obsahu.

Tento dokument je zamýšlen pro vývojáře výukového obsahu na různých stupních technické odbornosti. Může být rovněž užitečný pro projektanty vzdělávání, pracovníky zajišťující jeho kvalitu a pro programové manažery, kteří chtějí lépe porozumět tomu, co SCORM 1.2 umožňuje a co jej činí obtížným. Primárně je tento dokument nicméně určen technické části publika s alespoň určitými znalostmi webových konceptů a zvládající základy HTML a JavaScriptu.

A co SCORM 1.3?

Výukový obsah vytvořený ve shodě s návody v tomto dokumentu se nestane automaticky zastaralým s příchodem SCORMu 1.3 nebo dalšími budoucími specifikacemi, protože:

- SCORM 1.2 je o mnoho jednodušší než SCORM 1.3 a je snadnější ověřit splnění podmínek standardu. Vzhledem k tomu, že existují celé kategorie výukového obsahu, které nevyžadují nové prvky obsažené ve SCORMu 1.3, očekává se, že podpora SCORMu 1.2 pro běžnou distribuci obsahu ke studentům prostřednictvím LMS, jeho migraci a archivaci, bude pokračovat ještě dlouhou dobu.
- Masivní implementace podpory SCORMu 1.3 v komerčních produktech může nastat v průběhu 3 měsíců až 2 let, pokud lze soudit z historie SCORMu 1.2.
- Očekává se, že systémy s podporou obsahu ve SCORM 1.3 budou rovněž podporovat obsah ve SCORMu 1.2, ať už přímo nebo s pomocí obecného adaptéru.
- Tento dokument aplikuje vrstvený přístup, který umožní hromadné aktualizace obsahových objektů SCORMu z verze 1.2 na verzi 1.3 jednoduchou aktualizací jednoho nebo dvou obecných souborů se scripty.

Po konečné specifikaci standardu SCORM 1.3 na konci roku 2003 je plánována aktualizace tohoto dokumentu. Plánuje se zahrnutí sekce, ve které bude vysvětleno, jak aktualizovat obsah vytvořený dle tohoto dokumentu tak, aby odpovídal standardu SCORM 1.3.

Nástroje tvorby(Authoring tools)

Vývojovým nástrojem používaným při tvorbě příkladů v tomto dokumentu je obyčejný textový editor. V těchto příkladech nicméně nenajdete žádný vydatný obsah a interakce, jejichž tvorbu by skutečné vývojové nástroje usnadňovaly. Webově orientovaný LCMS nástroj jako Aspen 2.1 společnosti Click2learn nebo nástroje

tvorby pracovní plochy jako je ToolBook 8.6 zvládají mnoho věcí, popsaných v tomto dokumentu, zcela automaticky: Jednoduše specifikujete, že chcete vyexportovat obsah jako SCORM balíček a software to pro vás zařídí. Tyto nástroje jsou navrženy speciálně tak, aby Vás ušetřily technických detailů popsaných ve specifikaci SCORMu a v tomto dokumentu, díky čemuž se můžete soustředit na to, co má obsah vašeho kurzu předvést, říci a naučit.

Tak například: LCMS (Learning Content Management Server) Aspen je komplexní, dostupná a 100% webově orientovaná aplikace, která umožňuje rychlou tvorbu, distribuci a management výukového obsahu na vysoké úrovni pro veškeré Vaše projekty. Aspen LCMS je navržen tak, aby usnadnil oddělení prezentace, logiky a chování výukových objektů a poskytl tak maximální flexibilitu a možnost jejich opětovného použití. Týmově orientované prostředí pro tvorbu umožňuje společný vývoj, dovoluje vzdělávacím designérům, oborovým expertům, projektovým manažerům a kontrolorům pracovat společně tak, aby bylo možné rychle vytvářet vysoce kvalitní kurzy. V této souvislosti hraje SCORM konformita sice malou, ale kritickou úlohu.

Užijete-li k tvorbě výukového obsahu ToolBook, Aspen LCMS nebo jiný SCORM konformní nástroj tvorby, vlastní kód a struktura webového dokumentu, které tyto nástroje vygenerují, budou poměrně odlišné od struktur popsaných v tomto dokumentu. Existuje mnoho způsobů, jak dosáhnout SCORM konformity, což je v pořádku, protože je tak zde ponechán prostor pro kreativitu.

Příklady

Příklady v tomto dokumentu ilustrují, jak *může* být SCORM konformní výukový obsah implementován. Je možné použít kterýkoliv z kódů obsažených v ukázkách tohoto dokumentu, ale pokud tak učiníte, měli byste řádně odkázat na Click2learn Inc. Pokud se rozhodnete použít kterýkoli ukázkový kód, činíte tak zcela na vlastní riziko. Ukázky a vzorky kódu v přiložených souborech fungujících příkladů nebyly zamýšleny jako reálný produkční kód a nepředstavují kód, který Click2learn implementuje do svých produktů. Některé kódy a prvky, které by mohly být užitečné v reálných produktech, nebyly po úvaze použity, protože představují vlastnictví a obchodní tajemství společnosti Click2learn, Inc. Obzvláště popis obsluhy chyb je minimální, aby seznam vzorků kódu zůstal stručný. Vyčerpávající testování, které by v případě použití v reálném produkčním kódu probíhalo za použití všech běžných prohlížečů, nebylo u většiny uveřejněných vzorků provedeno. Byly testovány v Microsoft Internet Exploreru 6.0 a SCORM 1.2.3 Test Suite, a měly by fungovat i v ostatních prohlížečích a jejich různých verzích, výsledky však mohou být proměnlivé. Při práci s příklady doporučujeme mít při ruce k nahlédnutí specifikaci SCORMu 1.2.

Vaše vlastní implementace, styl programování a metody se mohou zcela lišit od těch, které jsou zde nastíněny. Nicméně výsledný produkt by měl zůstat SCORM konformní a předvídatelně kooperovat v rámci ostatních systémů podporujících SCORM. V tom je kouzlo standardu. Interoperabilita je to, oč kráčí. Přejděme nyní k implementaci.

Aktuální příklady, stejně jako nejnovější verzi tohoto dokumentu naleznete na http://home.Click2learn.com/standardswork. (nenaleznete, pozn. překl.)

Kapitola 1 – náhled na SCORM

Co je SCORM?

SCORM je sada specifikací, které popisují:

- jak vytvářet webově orientovaný výukový obsah, který může být distribuován ke studentům prostřednictvím různých LMS podporujících SCORM, a u kterého je možné zaznamenávat akce prováděné s tímto obsahem.
- Co musí LMS se SCORM podporou zvládat, aby správně interpretoval a zaznamenával akce prováděné se SCORM konformním učebním obsahem.

Specifikace SCORMu jsou založeny na různých dalších odvětvových standardech a specifikacích. Současná oficiální verze je 1.2.

Specifikace SCORMu neobsahuje všechny aspekty činností spojených s výukou; například nespecifikuje, jak ukládat informace o průchodu kurzem a jaké zprávy se budou generovat, jaké pedagogické nebo výukové modely by měly být použity, nebo jak shromažďovat informace o studentech.

SCORM 1.2 dále nespecifikuje, jak je výukový obsah řazen runtimovou službou. Nejběžnějším předpokladem je, že uživatel si může vybrat kteroukoliv část výukového obsahu. Budoucí specifikace SCORMu budou definovat, jak se má výukový obsah chovat při řazení. Zatím SCORM 1.2 poskytuje plnou specifikaci toho, jak má být výukový obsah zabalen a přemisťován, instalován v LMS nebo archivován způsobem plug-and-play.

Pracuje se na verzi 1.3, která dále rozšiřuje SCORM 1.2 tím, že specifikuje způsob, jakým mají být doplněny předpisy pro řazení. Tato nová verze pravděpodobně nebude dokončena dříve než v roce 2003 a výukový obsah, který splňuje podmínky standardu SCORM 1.2 by měl být funkční i při implementaci SCORMu 1.3.

Co znamená SCORM konformní výukový obsah (SCORM-compliant content)?

Ve specifikaci standardu SCORM 1.2 je SCORM konformní výukový obsah buď – v terminologii SCORMu – *balíček agregovaného obsahu* (Content Aggregation Package) nebo *zdrojový balíček* (Resource package). Zdrojový balíček je souborem výukových aktiv, která nejsou určena k distribuci jako takové, tedy například k archivaci nebo přemisťování souboru takových aktiv. V tomto dokumentu nenaleznete popis toho, jak vytvářet nebo používat zdrojové balíčky, soustředí se na distribuovatelný výukový obsah. Balíček agregovaného obsahu je:

- určen k distribuci výukového obsahu ke studentovi prostřednictvím webového prohlížeče
- popsán prostřednictvím metadat
- organizován jako strukturovaný soubor jednoho nebo více výukových objektů zvaných sdílené výukové objekty (Shareable Content Objects, obvykle se používá zkratka SCO)
- zabalený takovým způsobem, aby mohl být importován do LMS s podporou SCORM nebo uložen v archivu takového systému

SCORM konformní výukový obsah je tvořen několika objekty SCO agregovanými do balíčku výukového obsahu. SCO jsou specializované typy výukových objektů. Každé

SCO je jednotkou výukového obsahu, která může být použita ve výuce v LMS podporujícím SCORM za účelem vytvoření užitečného výukového dojmu.

Objekty SCO použité v balíčku standardu SCORM mohou být plně zahrnuty v balíčku, nebo použity v odkazu. Například za určitých podmínek může výuková sekvence obsahovat výukové objekty, které jsou na jiném serveru. Bezpečnostní omezení implementovaná ve webových prohlížečích, která mají zabránit nebezpečným zneužitím serverů, činí využití výukových objektů, které jsou uloženy na jiném serveru, obtížnějším. Jak vyřešit tento problém – to je úkol pro LMS a jejich dodavatele. Se samotným výukovým obsahem nelze udělat nic, co by pomohlo obejít tuto bezpečnostní bariéru.

Druhy výukových objektů standardu SCORM

Webově orientované výukové objekty, které mohou být jako individuální aktivita zahrnuty do balíčků určených pro distribuci v LMS podporujících SCORM, jsou nazývány SCO. V praxi se setkáváme se dvěma druhy:

- *minimální SCO*. Představuje ho HTML výukový obsah nebo služba, která může být spuštěna v okně prohlížeče a využívá aplikační programový interface (Application Program Interface, API) pro minimální komunikaci s LMS. LMS může zaznamenávat čas mezi spuštěním a standardním ukončením takového objektu. Obvykle může být výukový obsah, který je možné spustit v okně prohlížeče, a který neobsahuje odkazy na další výukový objekt, přeměněn na SCO tak, že je „obalen“ („wrapped“) jako SCO. Může to být HTML stránka, soubor Adobe Acrobatu nebo textový soubor. Rovněž je akceptována sada HTML stránek, pokud odkazují jen jedna na druhou v rámci této sady a nikoli na jiné výukové objekty.
- *datové SCO*. Je stejné jako základní SCO, ale mimo to využívá SCORM API k odesílání a získávání dat do a z LMS. Tato data mohou zahrnovat informace o průchodu výukovým obsahem, informace o studentovi, atd. - jak je definováno ve specifikaci SCORM 1.2.

Online nebo Offline výukové objekty

Nehledě na to, kolik prvků SCORMu používají, SCORM konformní výukové objekty nekomunikují přes web se vzdáleným serverem. Komunikují pouze s ostatními objekty v rámci stejného prostředí prohlížeče na straně klienta. Takový objekt poskytne realizátor distribučního prostředí. Existuje jasně definovaný způsob, který umožní výukovému obsahu nalézt takový objekt a ke komunikaci s ním užít jednoduchého JavaScript kódu nebo jeho ekvivalentu. Má to několik výhod:

- objekty SCO mohou být velmi snadno implementovány, protože není nutné zahrnout komplexní komunikační protokol potřebný pro odesílání a přijímání dat přes web.
- objekty SCO mohou být spouštěny v offline prostředí, aniž by vyžadovaly lokální webový server nebo proxy server, protože komunikují s ostatními objekty lokálního prohlížeče namísto s objekty na serveru.

Instance implementace SCORM API je na straně klienta vytvořena runtimovou službou ještě před tím, než je spuštěno SCO. Tato implementace může být u různých dodavatelů různá. API může být implementováno například v HTML framesetu, který obsahuje frame „scéna“, v rámci kterého jsou spouštěny výukové objekty.

Organizace a řazení výukových objektů

Osoba nebo entita, která tvoří balíček výukových objektů, rozhoduje o tom, jak budou tyto objekty seřazeny. Nicméně, jelikož SCORM 1.2 nedefinuje žádné informace o řazení, student si bude moci vybrat, který výukový objekt použije a v jakém pořadí.

Budoucí verze SCORMu, počínajíc SCORMem 1.3, by měly obsahovat pokročilejší řadící model, aby umožnily implementaci vydatnějších pedagogických nebo výukových modelů.

SCORM 1.2 používá jako základ pro balení a organizaci výukových objektů specifikaci IMS balení. Balíček může obsahovat více než jedno uspořádání stejného výukového objektu. Například je možné definovat dvě nebo více cest různými předlohami na rozdílných úrovních hloubky nebo pro odlišné skupiny studentů. LMS z toho může těžit umožňujíc volbu vhodnější organizace řazení.

SCORM 1.2 specifikuje, jak vytvořit balíček, ale nespecifikuje, jak LSM použije některé volitelné prvky balíčku jako je několikeré řazení výukového obsahu. Přestože výukový manažer nebo LMS vybere řazení v rámci balíčku, SCORM 1.2 specifikuje, že studentovi musí být umožněno spustit každý výukový objekt (SCO) definovaný v organizaci výukového obsahu.

Organizace výukových objektů a objektů SCO

Organizace položek

Organizace SCORM výukových objektů v balíčku je popsána v hierarchické stromové struktuře podobné struktuře výukového obsahu kurzu. SCORM nespecifikuje přesnou výšku stromu. Také nespecifikuje žádnou zvláštní terminologii k pojmenování úrovní stromu jako třeba „kurz, lekce, téma“ nebo „jednotka, modul, lekce“. Je možné použít jakoukoliv terminologii, případně nepoužít žádnou. Délka větví stromu může být různá.

Každá položka stromu může ukazovat na výukový objekt, nebo může obsahovat další položky. Každá položka musí mít název, který runtimové prostředí zobrazí studentovi.

Jakákoliv položka stromu může mít potomky a také může ukazovat na výukové objekty. Například pokud hierarchie Vašeho výukového obsahu představuje sekce, kapitoly a stránky, nadpisy kapitol mohou mít svou vlastní „titulní stránku“ (cover page). Každopádně ve SCORMu 1.3 bude možné spouštět a procházet jako objekty SCO pouze výukové objekty asociované s koncovými uzly listů.

V rámci stejného organizačního uspořádání výukového obsahu lze míchat a spojovat objekty SCO na jakýchkoliv úrovních technické kompatibility. Například je možné agregovat jednoduché, jednostránkové objekty SCO vytvořené v Notepadu s komplexními SCO vytvořenými v pokročilých vývojových nástrojích jako ToolBook.

Podobnost s ostatními modely organizačního uspořádání výukového obsahu

Starší organizační modely, jako je AICC bloková/AU struktura, mohou být zakresleny přímo do tohoto organizačního modelu.

Modely organizačního uspořádání výukového obsahu, které používají orientované grafy, nemohou být zastoupeny přímo ve SCORM hierarchické stromové struktuře. Nicméně mnoho orientovaných grafů může být nahrazeno vytvořením několikerých položek ukazujících na další organizační uspořádání v balíčku namísto odkazování přímo na SCO. Pro více informací, jak toho dosáhnout s pomocí sub-manifestu viz popis manifestu balíčku a specifikace IMS balíčku výukového obsahu.

Řazení

SCORM 1.2 nedefinuje, jak řadit jednotlivá SCO. Předpokládá se řazení dle uživatelské volby.

Uživatelské řazení

V uživatelském řazení umožňuje runtimová služba uživateli vybrat si z mnoha položek celého uspořádání výukových objektů. V závislosti na implementaci může být řazení uskutečněno prostřednictvím viditelného stromu, menu nebo sady vnořených menu. SCORM 1.2 nespecifikuje, jak má uživatelské rozhraní k výběru položky vypadat.

Například typická implementace obsahu kurzu může pracovat asi takto: Když student vybere položku v obsahu kurzu, položka je graficky zvýrazněna a v okně „scéna“ se spustí odpovídající výukový objekt. Pokud má položka potomky, ale žádné vlastní výukové objekty, bude zvýrazněn první potomek obsahující výukový objekt a tento objekt bude také spuštěn.

Zvláštní pravidla pro některé položky

Mastery score

Aktivitě, která užívá SCO je možné přiřadit mastery score. Pokud je score hlášeno objektu SCO, runtimová služba porovná score s mastery score, aby nastavila status aktivity na „splněno“ nebo „nesplněno“. To přepíše jakýkoli status, který mohl být hlášen objektem SCO.

Timeout a akce timeoutu

Pro detaily viz specifikace SCORMu. Runtimová služba může ukončit SCO v okamžiku, kdy povolený čas vypršel; nicméně toto chování není příliš dobře popsáno a nemusí být dostupné ve všech implementacích, protože požaduje složitější runtimové prostředí na straně klienta (uživatele).

Nezbytné předpoklady

SCORM 1.2 definuje jen skutečně základní podobu předpokladů nezbytných pro splnění cílů stanovených tvůrcem výukového obsahu. Předpoklad odkazuje na další části v organizačním stromu, které musí být dokončeny nebo zvládnuty. Nicméně vzhledem k tomu, že specifikace SCORMu nedefinují chování spojené s nezbytnými předpoklady příliš jednoznačně a definují pouze způsob, jakým je specifikovat, je jejich použití na vlastní riziko. SCORM 1.3 bude obsahovat obsáhlejší pravidla pro řazení, která se nebudou spoléhat na spletité odkazy mezi položkami a namísto toho dovolí adaptivní rozhodování o řazení založené na statusu výukových cílů.

Implementace SCORM konformních výukových objektů a objektů SCO

Runtimová služba

Aby bylo možné implementovat SCORM výukové objekty a objekty SCO, bude užitečné částečně porozumět runtimovému prostředí, ve kterém budou používány. Abychom odlišili tuto runtimovou službu od zbytku LMS a jeho přihlašovacích, ověřovacích, archivačních a ohlašovacích služeb, budeme označovat runtimové prostředí a procesy, které řídí jako „runtimovou službu“.

Odlišení LMS a runtimové služby může být prospěšné, protože LMS a uživatel mohou mít například rozdílné požadavky na dobu odezvy.

Runtimová služba může být částečně implementována na straně klienta (tj. například na studentově počítači) a částečně na straně serveru (tj. někde na serveru). Ve stavu offline je strana serverového prostředí emulována aplikací, která běží na studentově počítači.

Strana klienta runtimové služby:

- je poskytována LMS systémem
- je implementována jako webová stránka nebo jako frameset v okně prohlížeče
- poskytuje studentovi některé nezbytné komponenty runtimového uživatelského rozhraní, takové jako obsah kurzu nebo navigační tlačítka
- spouští výukové objekty ve okně „scéna“, které je buď částí framesetu, nebo separátním oknem, které je vytvářeno dle potřeby
- zahrnuje objektovou instanci aplikačního programového prostředí pojmenovanou „API“, která může být vyhledána a volána JavaScriptem nebo ECMAScriptem
- je generována podle potřeby na straně serveru

Strana serveru runtimové služby a zbytek LMS jsou ve SCORM výukových objektech pro skripty zcela neviditelné. Uživatelské rozhraní komponent runtimové služby, takové jako obsah kurzu nebo navigační tlačítka, jsou SCORM výukovým objektům rovněž zcela skryté. To, jak je implementována runtimová služba na straně klienta a jak komunikuje se serverem, je pro obsah zcela neviditelné. Jedinými částmi runtimové služby, které SCORM výukové objekty mohou detekovat, a se kterými mohou komunikovat, jsou:

- okno „scéna“
- API adaptér
- funkce poskytované API adaptérem

Výukovým objektům je zakázáno odkazovat na jiné výukové objekty v rámci okna „scéna“. Další výukové objekty může do okna „scéna“ nahrávat pouze runtimová služba. Runtimová služba využívá uspořádání obsahu definované v balíčku obsahu (content package) jako vodítko k řízení navigace mezi výukovými objekty.

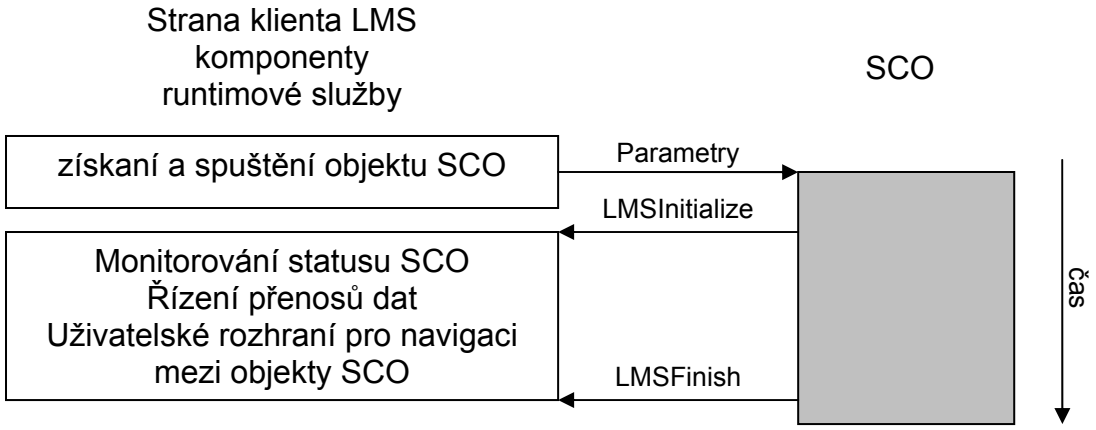
Primitivní SCO

SCO je v podstatě jakýsi webový obsah, který komunikuje s runtimovou službou poté, co byl spuštěn a znovu v okamžiku, kdy je ukončen.

Ve své nejjednodušší formě by takovým SCO objektem mohla být webová stránka s krátkým scriptem. Může také sestávat z několika HTML stránek. Vzorky v příloze tohoto dokumentu ukazují, jak může být jednostránkové nebo vícestránkové SCO jednoduché. Obzvláště použijeme-li při inkluzi předem připravené obecné skripty.

Když se SCO nahrává, script nachází API adaptér a volá funkci adaptéru *LMSInitialize*.

Když je SCO ukončováno, nebo kdykoli před tím, pokud SCO udává, že je ukončeno, volá skript funkci API adaptéru *LMSFinish*.



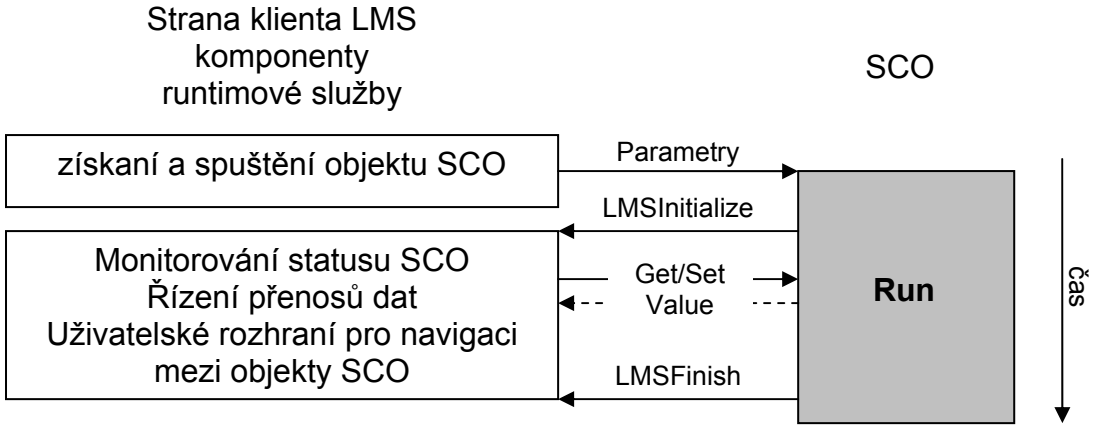
Obr. 2.1 – Jednoduché SCO

Chování a úkoly jednoduchého SCO

- Po spuštění najít API adaptér, hledání probíhá specifickým způsobem
- Po jeho nalezení zavolat *LMSInitialize* k zahájení komunikační relace
- Po jeho skončení zavolat *LMSFinish* k ukončení komunikační relace

Datové SCO (Data enabled SCO)

Datové SCO volá *LMSInitialize* a *LMSFinish* stejně jako jednoduché SCO, ale kromě toho si v době mezi těmito dvěma voláními vyměňuje data s runtimovou službou. SCORM 1.2 využívá datový model zvaný CMI, založený na dřívější specifikaci vyvinuté komisí Aviation Industry Computed-Based Training Commitee (AICC). Datový model je jasně definovaný katalog datových elementů. Jak runtimová služba, tak objekt SCO rozumí datovému modelu CMI. Například, pokud SCO je test (quiz), pak score, které obdrží student, je hodnotou, jež může být uložena jako element datového modelu. SCO pak může posílat tento datový element runtimové službě a runtimová služba už ví, kde a jak ukládat score jako informaci o průchodu kurzem.



Obr. 2.2 – SCO může volat API adaptér aby získal nebo nastavil hodnotu

Chování a úkoly datového SCO

- Po spuštění hledá API adaptér, hledání probíhá specifickým způsobem
- Po jeho nalezení volá *LMSInitialize* k zahájení komunikační relace
- Během realizace: volá *LMSGetValue*, *LMSSetValue*, *LMSCommit*, *LMSGetLastError* a *LMSGetErrorString* dle potřeby během provádění SCO
- Po skončení komunikační relace volá *LMSFinish* k jejímu ukončení
- při neočekávaném uvolnění z paměti se pokouší volat *LMSSetValue* aby poslal neuložená data LMS, pak volá *LMSFinish*. Tento úklid by měl být zahrnut v handleru události *onUnload*, která je aktivována objektem SCO uvolňovaným z paměti. Vzhledem k tomu, že chování prohlížečů během vykonávání *onUnload* může být nespolehlivé, není zde garance, že LMS data efektivně přijme a uloží. Mějte na paměti, že pokud bylo *LMSFinish* voláno už dříve, vše co bylo zmíněno, bude runtimovou službou ignorováno.

Nalezení adaptéru API

SCO musí najít API adaptér, aby mohlo komunikovat s LMS. Protože SCO může být spuštěno ve framu v rámci framesetu nebo v popup okně, existuje zvláštní vyhledávací pořadí. Hledání končí, jakmile je API adaptér nalezen: Nejdříve, pokud existuje rodičovské okno SCO, dívá se tam a následně v řetězci rodičovských oken, dokud nenalezne nejvyšší okno. Pokud touto metodou nedojde k nalezení adaptéru, hledá v okně openeru, pokud existuje, a dále v řetězci rodičů takového okna, dokud nenalezne nejvyšší okno.
Toto striktní vyhledávací pořadí je navrženo tak, aby umožnilo budoucí nebo pokročilou implementaci, ve které by bylo možné spustit více než jedno SCO současně, a aby bylo zajištěno, že každé SCO najde jen tu instanci API adaptéru, která byla určena ke komunikaci s tímto SCO.

SCORM komunikační relace

Závazné události při spouštění SCO

Jediné dvě události specifikované ve SCORMu k řízení komunikační relace mezi SCO a runtimovým prostředím jsou *LMSInitialize* a *LMSFinish*. *LMSFinish* je „závěrečná událost“ v tom smyslu, že jakákoliv komunikace ze strany SCO bude runtimovou službou ignorována poté, co bylo úspěšně zavoláno *LMSFinish*.

Mějte na paměti, že volání *LMSFinish* znamená pouze to, že SCO už nadále nepotřebuje komunikovat s runtimovou službou. Runtimová služba by neměla interpretovat volání *LMSFinish* jako signál k pokračování dalším objektem SCO. Budoucí verze SCORMu mohou poskytovat další indikátor ke specifikaci takového automatického chování, ale ve SCORMu 1.2 takový indikátor není.

Jakmile bylo jednou *LMSFinish* voláno, SCO se nesmí znovu pokoušet obnovit komunikaci voláním *LMSInitialize*. *LMSInitialize* může SCO volat jen v případě, že dojde k jeho uvolnění z paměti a následnému opětovnému spuštění.

Neočekáváné uvolnění objektu SCO z paměti

SCO může být uvolněno z paměti vždy, když uživatel zvolí jiné SCO. Také prohlížeče záměrně nebrání uživateli zavřít okno „scéna“, ve kterém je SCO spuštěné. Aby byl minimalizován dopad takového neočekávaného uvolnění z paměti, měli byste zahrnout handler události „*onUnload*“ v nejvyšší úrovni objektu SCO, kterou volá *LMSFinish*.

Prohlížeče se mohou při provádění události uvolňování z paměti chovat poněkud svérázně. Zdá se, že to platí pro případy, kdy skripty spuštěné událostí unload neukončí provádění, obzvláště když obsahují volání serveru s dlouhou prodlevou, protože souvislosti nahrávání nečekají, až skončí souvislosti skriptů.

Protože uvolňování z paměti může být nespolehlivé, je vhodnější před započetím procesu uvolnění z paměti zavolat handler pro *onbeforeunload*. Naneštěstí je *onbeforeunload* implementováno pouze v Microsoft Internet Exploreru a není zahrnuto v HTML standardu. Rovněž užití *href* k volání skriptu, což bývá běžným trikem na mnoha webových stránkách, zapříčiní volání *onbeforeunload*.

Nejbezpečnější metodou je volat *LMSSetValue* jakmile se objeví důležitá událost v objektu SCO, aby se uchovaly informace, které chcete zachovat a hned nato volat *LMSCommit*, takže pokud je SCO neočekávaně uvolněno z paměti, předejde se kritickým ztrátám informací.

Není problém volat *LMSFinish* více než jedenkrát, ale po prvním úspěšném *LMSFinish* runtimová služba ignoruje jakoukoliv další komunikaci objektu SCO s výjimkou volání funkcí k získání chybových informací.

Kdy volat LMSFinish

Abyste se ujistili, že LMSFinish bude volána i když je SCO uvolněno z paměti neočekávaně, řádně inicializované SCO by mělo být naprogramováno tak, aby volalo *LMSFinish* ve všech následujících případech a za následujících podmínek:

- Když už není třeba komunikovat s runtimovou službou. Například když uživatel kliká na tlačítko „odeslat“ na konci testu a nemá mu již být umožněno změnit odpovědi, SCO může nahrát výsledky testu pomocí *LMSSetValue* a poté zavolat *LMSFinish*.
- Pokud *LMSFinish* nebylo ještě úspěšně voláno, když je zpracovávána událost prohlížeče *onbeforeunload* (typicky se tak děje pouze v případě, že prohlížečem je Internet Explorer)
- Pokud *LMSFinish* nebylo ještě úspěšně voláno, když je zpracovávána událost prohlížeče *onUnload*

Poznámka: Volání *LMSFinish* je úspěšné, pokud volání vrací „*true*“. Volání může vrátit *false*, pokud runtimová služba stanoví, že nastala koncová chyba, což jí zamezuje v normálním ukončení komunikační relace. Ve specifikaci SCORMu není definováno, jak zvládat tuto situaci, ale jedním ze způsobů může být pokusit se nastavit timer tak, aby po několika sekundách zavolal LMSFinish znovu.

Kapitola 3 – Vytvoření jednoduchého SCO

Vytvoření primitivního SCO

Toto SCO implementuje pouze povinné prvky. Při nahrávání volá *LMSInitialize* a při uvolňování z paměti volá *LMSFinish*.

```
<html>
<head>
<script type="text/javascript" language="JavaScript">
  var nFindAPITries = 0;
  var API = null;
  function FindAPI(win) {
    while ((win.API == null) && (win.parent != null) && (win.parent != win)) {
      nFindAPITries ++;
      if (nFindAPITries > 500) {
        alert("Error in finding API -- too deeply nested.");
        return null
      }
      win = win.parent
    }
    return win.API
  }
  function init() {
    if ((window.parent) && (window.parent != window)){
      API = FindAPI(window.parent)
    }
    if ((API == null) && (window.opener != null)){
      API = FindAPI(window.opener)
    }
    if (API == null) {
      alert("No API adapter found")
    } else {
      API.LMSInitialize("")
    }
  }
  function finish() {
    if (API != null) {
      API.LMSFinish("")
    }
  }
</script>

<title></title>
</head>
<body onload="init()" onunload="finish()">
<h1>Hello world</h1>
</body>
</html>
```

Ukázka 3.1 – Primitivní SCO

komentáře:

- většina scriptu je věnována nalezení API adaptéru, který poskytuje LMS v prostředí DOM ještě před tím, než s ním může SCO komunikovat. Hledání probíhá ve specifickém pořadí.
- je zde volitelný limit, který určuje do jaké hloubky budou vnořená okna prohledávána

- testování *(window.parent != window)* je nezbytné kvůli odlišnostem v Internet Exploreru

Obecný script pro jednoduché objekty SCO

Analyzujete-li script předcházejícího primitivního SCO, nejspíše zaregistrujete, že velká část tohoto scriptu by byla stejná bez ohledu na vlastní obsah objektu SCO. Toho může být využito k vytvoření opětovně použitelného scriptu. Ten by mohl vypadat asi takto:

```
// Saved as file "SCORMGeneric.js"
// Generic Simple SCO Script -
// Concocted by CO 2002-10-14
var nFindAPITries = 0;
var objAPI = null;
var bFinishDone = false;
function FindAPI(win) {
  while ((win.API == null) && (win.parent != null) && (win.parent != win)) {
    nFindAPITries ++;
    if (nFindAPITries > 500) {
      alert("Error in finding LMS API -- too deeply nested.");
      return null
    }
    win = win.parent
  }
  return win.API
}
function APIOK() {
  return ((typeof(objAPI)!= "undefined") && (objAPI != null))
}
function SCOInitialize() {
  if ((window.parent) && (window.parent != window)){
    objAPI = FindAPI(window.parent)
  }
  if ((objAPI == null) && (window.opener != null)){
    objAPI = FindAPI(window.opener)
  }
  if (!APIOK()) {
    alert("Learning Management System interface not found.");
    return "false"
  } else {
    return objAPI.LMSInitialize("")
  }
}
function SCOFinish() {
  if ((APIOK()) && (bFinishDone == false)) {
    bFinishDone = (objAPI.LMSFinish("") == "true")
  }
  return (bFinishDone.toString())
}
```

Ukázka 3.2 – Obecný, opakovaně použitelný script pro jednoduché objekty SCO

komentáře:

Tento skript se ujišťuje, že metoda API *LMSFinish* nebude volána znovu v případě, že už byla úspěšně zavolána. Jednostránkové SCO, které užívá tento jednoduchý script by mohlo vypadat asi takto:

```
<html>
<head>
<script src="SCORMGeneric.js" type="text/javascript" language="JavaScript">
</script>
<title>A very simple SCO</title>
</head>
<body onload="SCOInitialize()"
      onunload="SCOFinish()"
      onbeforeunload="SCOFinish()">
<h1>Hello world</h1>
</body>
</html>
```

Ukázka 3.3 – primitivní SCO opakovaně s užitím opakovaně použitelného obecného scriptu

komentáře:
Všimněte si, že tag <body> těchto html stránek zahrnuje handlery událostí, které volají funkce v obecném skriptu poté, co je stránka nahrána nebo když je uvolňována z paměti. Handler události pro *onbeforeunload* je volán také těsně před tím, než je stránka uvolněna z paměti. Tuto událost podporuje pouze Microsoft Internet Explorer, a proto musí být pro ostatní prohlížeče nadále specifikováno *onUnload* jako aktivátor konečné události. Tam, kde je to možné, preferuje se aktivace události *onbeforeunload*, protože zpracování této události nekoliduje s nahráváním další stránky, které už může probíhat, zatímco probíhá zpracování *onUnload*. Obecný script zajistí, že LMSFinish bude voláno pouze jedenkrát, ať už zpracování ukončování aktivuje onbeforeunload nebo onunload.

Kapitola 4 – Vaše SCO jako webová stránka

Jako člověk, která vytváří výukový obsah, nemáte kontrolu nad tím, jak bude Vaše SCO spouštěno a obzvláště nad rozměry okna „scéna“, ve kterém bude SCO spuštěno runtimovou službou. Vaše SCO bude vždy spuštěno jako webová stránka. To je jediná věc, na kterou se můžete spolehnout.

Okno „scéna“

SCO neovládá typy, výchozí velikost, nebo okrasné prvky okna „scéna“ a nelze spoléhat na to, že kolem něj, tedy v oblastech obrazovky, které okno „scéna“ obklopují, bude viditelný obsah, speciální barevné schéma nebo obsah kurzu.

Například LMS Aspen společnosti Click2learn spouští objekty SCO ve framu. Rozměry tohoto okna „scéna“ jsou tvořeny zbytkem aktuálního okna Aspenu po zobrazení okna pro obsah kurzu vlevo (200 pixelů) a pruhem pro management Aspenu nahoře (50 pixelů). Pokud Váš balíček obsahuje pouze jedno SCO, nezobrazí se okno s obsahem kurzu a okno „scéna“ pro toto SCO bude stejně široké jako okno Aspenu. Aby byl váš výukový obsah kompatibilní s Aspenem, Click2learn doporučuje, aby byl pro použití testován v okně 600x550 pixelů. Ostatní LMS implementace mají co do velikosti oken podobná omezení.

Budoucí verze SCORMu mohou zahrnovat specifikace technických metadat, které umožní vývojářům přesněji specifikovat požadovanou a preferovanou velikost a typ okna „scéna“. Nicméně zatím práce na definici takových metadat teprve probíhají.

Management okna

SCO nesmí předpokládat, že poběží v okně nejvyšší úrovně, nebo se pokoušet vynutit si spuštění v tomto okně. Musí být navrženo tak, aby bylo kompatibilní pro spuštění ve framu, vnořeném na jakékoliv úrovni.

S výjimkou hledání a komunikace s objekty API nesmí SCO komunikovat nebo manipulovat s ostatními okny, která existují v prostředí poskytovaném learning management systémem.

SCO smí otvírat závislá popup okna, ale nepokládá se to za vhodnou praktiku. V každém případě je SCO zodpovědné za zavírání takových oken před tím, než je uvolněno z paměti. Po uvolnění z paměti po sobě objekt SCO nesmí zanechat žádnou stopu v uživatelském prostředí. Zkušenost ukázala, že spolehlivé použití popup oken vyžaduje velmi obezřetný návrh a rozsáhlé testování nečekaných situací a chování, jako např.: uživatel oživuje nesprávné okno, nečekané zavírání okna openeru, systémové konfigurace s více monitory, utility blokující popup okna a podobně. Na druhou stranu jsou závislá popup okna jediným způsobem, jak získat SCO jako fullscreenové okno, protože SCO neovládá okno „scéna“, v jehož rámci je distribuováno.

Rozšíření

Objekty SCO mohou disponovat rozšířenými způsoby chování, které ve SCORMu nejsou specifikovány, ale které jsou odsouhlaseny mezi obchodními partnery. SCO je konformní pouze tehdy, pokud může být distribuováno do LMS, který podmínky standardu bezvýhradně splňuje, a který neimplementuje rozšířená chování. Jinými slovy, pokud je Vaše SCO závislé na specifických nástrojích určitého LMS, nebo pokud se pokouší pozměnit normální chování běžného LMS, pak není konformní.

Kapitola 5 – přeměna obsahu zobrazitelného v prohlížeči na SCO

Prohlížeče mohou zobrazovat různé druhy zdrojů, jako jsou dokumenty Adobe Acrobat, videa Macromedia Flash, textové nebo obrázkové soubory. Tyto zdroje nemohou komunikovat jako SCO. K „obalení“ (wrap) téměř jakéhokoliv obsahu zobrazitelného v prohlížeči do podoby SCO mohou být použity dvě techniky. První z nich je vytvoření HTML stránky s objektovým elementem, který čerpá ze zdroje. Jiná technika – mnohem jednodušší – je použít obecný frameset, který může být využit k zobrazení téměř jakéhokoliv obsahu zobrazitelného v prohlížeči. Frameset sám je oním SCO.

SCO implementované jako stránka obalující webový zdroj

Obr. 5.1 – Užití HTML stránky k přeměně webově orientovaných položek na SCO

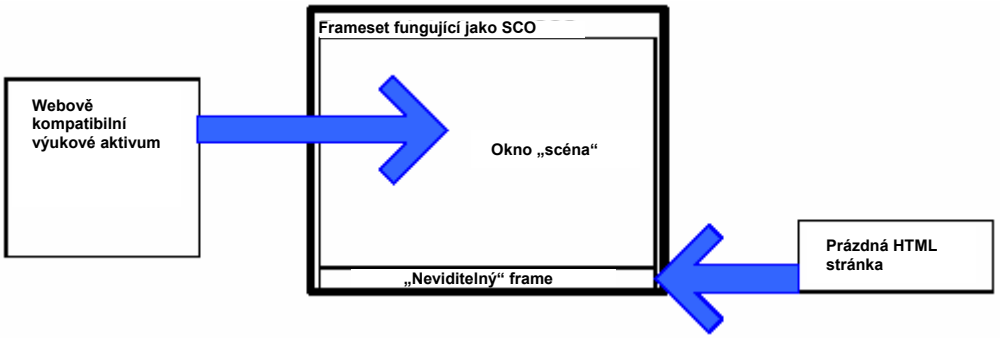
Tento wrapper využívá jednoduché HTML stránky. Stránka opakovaně používá jednoduchý obecný script z ukázky 3.2 k přeměně obrázku na SCO.

```
<html>
<head>
<script src="SCORMGeneric.js" type="text/javascript" language="JavaScript">
</script>
<title>Some picture</title>
</head>
<body bgcolor="#FFFFFF" onload="SCOInitialize()" onunload="SCOFinish()"
onbeforeunload="SCOFinish()">
<img src="myimage.jpg" alt="Caption for my image" width="600" height="400" />
<body>
</html>
```

Ukázka 5.1 – Primitivní wrapper k přeměně jakéhokoliv prvku na SCO

Poznámka: Může se zdát, že užitečnost takového postupu je omezená, ale namísto obrázku je samozřejmě možno zahrnout objekty jako jsou např. videa užitím patřičného tagu <object> a jeho parametrů.

SCO implementované jako frameset obalující webově orientovaný aktivní prvek



Obr. 5.2 – Použití framesetu k přeměně jakéhokoliv webově orientovaného prvku na SCO

Tento obal používá frameset s fingovanou stránkou, která z praktických důvodů zůstane neviditelná. Frameset užívá jednoduchý obecný script z ukázky 3.2 k přeměně dokumentu Adobe Acrobatu na SCO.

```
<html>
<head>
<script src="SCORMGeneric.js" type="text/javascript" language="JavaScript">
</script>
<title>Some PDF document</title>
</head>
<frameset border="0" rows="100%,*" onload="SCOInitialize()" onunload="SCOFinish()"
onbeforeunload="SCOFinish()">
<frame src="somedoc.pdf" scrolling="AUTO" />
<frame src="dummypage.htm" />
</frameset>
</html>
```

Ukázka 5.2 – Jednoduchý framesetový wrapper k přeměně jakéhokoliv prvku na SCO

Tato technika je také velmi vhodná, pokud chcete zobrazit externí webovou stránku. Buďte jen opatrní, aby se takový typ externího zdroje nepokoušel prosadit do horního framu, protože by to v ovládaném prostředí narušilo distribuci objektu SCO.

Poznámka: Fingovaná stránka může být jakýkoliv platný HTML dokument. Může se stát, že starší verze Netscapu nezobrazí frameset, dokud nespecifikujete hodnotu „100%,1“ pro *rows* namísto „100%,*“. Výsledkem bude jednopixelový okraj framu, který se v podstatě spojí se spodkem okna framesetu.

Použití SCO wrapperu ke sledování použití zdroje

Můžete přidat jednoduchý skript, který dovolí objektu SCO hlásit kompletní status od chvíle, kdy byl spuštěn, což umožní sledování toho, zda byl spuštěn zdroj.

```
<html>
<head>
<script src="SCORMGeneric.js" type="text/javascript" language="JavaScript">
</script>
<script type="text/javascript" language="JavaScript">
function init() {
  if (SCOInitialize() == "true") {
    objAPI.LMSSetValue("cmi.core.lesson_status","completed")
  }
}
</script>
<title>Some picture</title>
</head>
<body bgcolor="#FFFFFF" onload="init()" onunload="SCOFinish()"
onbeforeunload="SCOFinish()">
<img src="myimage.jpg" alt="Caption for my image" width="600" height="400" />
<body>
</html>
```

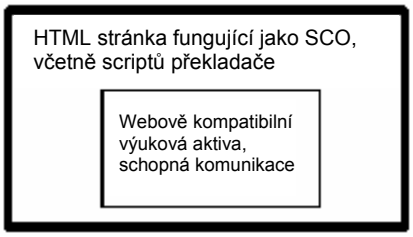
Ukázka 5.3 – Jednoduchý wrapper k přeměně jakéhokoliv prvku na SCO s hlášením o užití

Komentáře: Tato stránka poté, co je nahrána, volá vlastní funkci *init.* Funkce *init* pak volá funkci *SCOInitialize* z obecného scriptu. Proběhne-li tato funkce úspěšně, tzn. *SCOInitialize* bylo schopno nalézt objekt API adaptéru a úspěšně zavolat *LMSInitialize* tohoto objektu, *init* volá funkci *LMSSetValue* objektu API, aby nastavilo status na „*completed“.*

Pokročilé užití SCO wrapperů

Vrstva překladače pro nekompatibilní výukové objekty

Tato wrappovací technika může být rovněž užita k „obalení“ dalších scriptovatelných webových zdrojů, které nemohou komunikovat přímo s objektem SCO. Například video Macromedia Flash s vestavěnými způsoby chování v podobě scriptů může být vnořeno do HTML stránky se scripty. Flash video může využívat FSCommands ke komunikaci se skripty na stránce. Naopak skripty na stránce mohou dostávat data skrz rozhraní SCORMu a použít je k ovládání Flash videa. Typickým příkladem by mohl být bookmarking, kde SCO wrapper umožňuje Flash videu dostat se na stejný frame, ve kterém bylo SCO zobrazeno, když opouštělo předchozí relace.

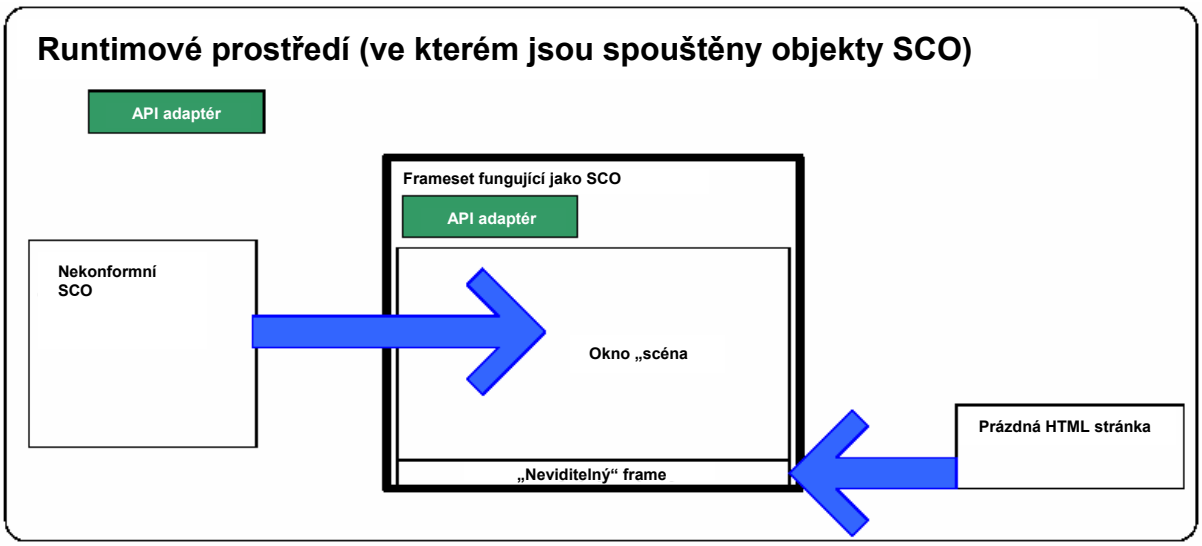


Obr. 5.3 – SCO wrapper pro nekompatibilní výukové objekty

Vrstva překladače pro nekonformní objekty SCO

Výše znázorněná wrappovací technika může být také užitečná ke spouštění objektů SCO v jiných objektech SCO. Ačkoli se tento druh aplikace může zdát bizarní, je možné ho využít například při spouštění nekompletních a nesprávně kódovaných odkazů objektů SCO v jakémkoliv SCORM konformním prostředí bez nutnosti jejich překódování. V tomto případě wrapper objektu SCO vytváří instanci svého vlastního API objektu, který klientské SCO najde jako první. Zatímco klientské SCO

komunikuje s API objektem wrapperu, wrapper může falšovat data tak, jak je potřeba ještě před tím, než přepošle volání ostatním API adaptérům, které nalezne v prostředí, do nějž distribuuje.



Obr. 5.4 – SCO v roli vrstvy překladače pro jiné SCO

Kapitola 6 – Tvorba vícestránkového SCO

Žádná ze specifikací standardu SCORM Vám nezakazuje používat více než jeden HTML soubor v rámci SCO a navigovat ze stránky na stránku uvnitř SCO. A vskutku, mnohdy jediná stránka nemůže zaručit adekvátní nebo zamýšlený výukový efekt.

Nicméně při využívání vícestránkových HTML k utváření objektu SCO existuje jistý problém. Zatímco přecházíte ze stránky na stránku, neuchovává se žádná informace o stavu. Například pokud stránka 1.html zavolá *LMSInitialize,* přejdete-li na stráku 2.html, nikde se neuchová informace o tom, že *LMSInitialize* už bylo voláno. Jelikož SCO si musí pamatovat alespoň stav komunikační relace s API, je nutné navrhnout metodu k uchování takové informace. Pokud taková metoda existuje, umožňuje rovněž objektu SCO uchovávat záznam o dalších důležitých informacích, jako je například status cílů výuky nebo jaké by mělo být pořadí stránek.

Nebudeme se zde zabývat případem, kdy byste chtěli změnit prvky zobrazené v rámci zavedené HTML stránky. To nepředstavuje problém, protože stránka samotná zůstává nahraná.

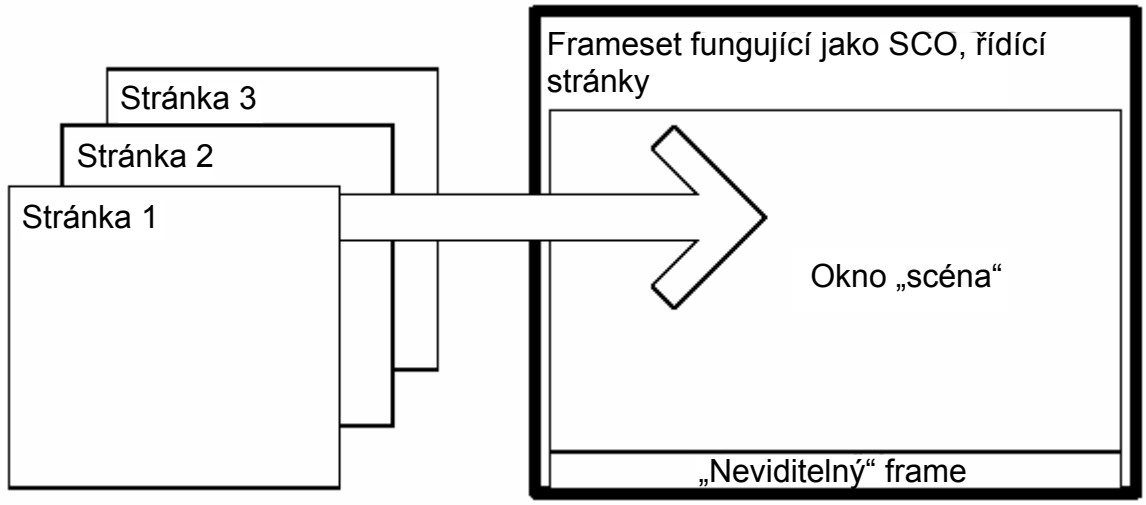
Tři metody k uchování stavu ve vícestránkových SCO

Protože od runtimové služby nelze očekávat žádnou pomoc při uchovávání informací o stavu, musí SCO využít perzistentní metody, která může být implementována v prohlížeči na straně klienta. Byly navrženy tři postupy:

- s použitím framesetu, ve kterém mohou být zobrazeny samostatné stránky, aniž by byl frameset uvolněn z paměti.
- s použitím samostatného závislého okna, které může zůstat otevřené i když se v okně „scéna“ mění stránky. Typicky by takové okno bylo „skryto“ za hlavním oknem nebo odsunuto z viditelné oblasti obrazovky.
- užitím relačních cookies

Použití framesetu pro vícestránkové SCO

Zdaleka nejsilnější metodou se zdá být užití framesetu. Ve skutečnosti je jako „scéna“ pro stránky použit pouze jeden frame. Druhý frame, který je některými prohlížeči vyžadován, zobrazuje pouze fingovanou stránku a je smrsknut do nejmenší možné velikosti, takže je virtuálně neviditelný.



Obr. 6.1 – Vícestránkové SCO implementované jako frameset

```
<frameset rows="100%,*">
<frame name="myStage" title="Learning Object display window" src="page1.htm" />
<frame src="dummypage.htm" title="Empty Placeholder"/>
</frameset>
</html>
```

Ukázka 6.1 – Fragment HTML: Struktura jednoduchého vícestránkového SCO framesetu

Složitější příklad

Tento příklad třístránkového SCO opětovně používá obecný script pro primitivní SCO popsaný v ukázce 3.2. Implementuje následující prvky:

- Je možné navigovat tam a zpět mezi SCO stránkami, aniž byste opustili SCO
- Stránky neobsahují složité scripty
- SCO může být násilně uvolněno z paměti na kterékoliv ze stránek
- Příklad ukazuje, jak může stránka nastavit status SCO na „*completed*“, když je tohoto statusu dosaženo.

V ukázce 6.2 je hlavní HTML soubor pro frameset.
Ukázka 6.3 představuje fingovanou stránku, kterou by uživatel normálně neviděl a další tři ukázky zachycují stránky 1 až 3 objektu SCO. Vyjma obsahu těchto stránek a navigačních objektů jsou stránky 1, 2 a 3 identické. Skripty obsahují paranoidní resynchronizační prvek pro případ navigační události jako je třeba kliknutí na tlačítko Zpět v paletě nástrojů prohlížeče, která se projeví mimo kontrolu scriptu framesetu: Každá stránka potvrdí své číslo poté, co je nahrána, takže navigační script ve framesetu nemůže nedopatřením překročit hranice pole stránky.

```
<html>
<head>
<script src="SCORMGeneric.js" type="text/javascript" language="JavaScript">
</script>

<script type="text/javascript" language="JavaScript">
  // Manage page navigation
  var znPages=3;
  var znThisPage=1;
  function NextPage() {
    if (znThisPage < znPages){
      znThisPage++;
      myStage.location.href = "page" + znThisPage + ".htm"
    }
  }
  function PreviousPage() {
    if (znThisPage > 1){
      znThisPage--;
        myStage.location.href = "page" + znThisPage + ".htm"
    }
  }
  function SetThisPage(n) {
    znThisPage=n
  }
</script>

<title>Multiple page SCO sample</title>
</head>
<frameset rows="100%,*" onload="SCOInitialize()" onunload="SCOFinish()"
onbeforeunload="SCOFinish()">
<frame name="myStage" title="Learning Object display window" src="page1.htm" />
<frame src="dummypage.htm" />
</frameset>
</html>
<html>
<head></head>
<body> </body>
</html>
```

Ukázka 6.2 – HTML zdroj: Frameset jednoduchého vícestránkového SCO

```
<html>
<head><title>empty</title></head>
<body> </body>
</html>
```

Ukázka 6.3 – HTML zdroj: Frameset fingované stránky ve vícestránkovém SCO

```
<html>
<head><title>Page 1</title></head>
<body onload="window.parent.SetThisPage(1)">
<p>This is page 1</p>
<p align="right">
<a href="JavaScript:window.parent.NextPage()">Next</a>
</p>
</body>
</html>
```

Ukázka 6.4 – HTML zdroj: Frameset stránky 1 jednoduchého vícestránkového SCO

```
<html>
<head><title>Page 2</title></head>
<body onload="window.parent.SetThisPage(2)">
<p>This is page 2</p>
<p align="right">
<a href="JavaScript:window.parent.PreviousPage()">Previous</a>
<a href="JavaScript:window.parent.NextPage()">Next</a>
</p>
</body>
</html>
```

Ukázka 6.5 – HTML zdroj: Frameset stránky 2 jednoduchého vícestránkového SCO

```
<html>
<head><title>Page 3</title></head>
<body onload="window.parent.SetThisPage(3)">
<p>This is page 3 (the last page)</p>
<p align="right">
<a href="JavaScript:window.parent.PreviousPage()">Previous</a>
</p>
</body>
</html>
```

Ukázka 6.6 – HTML Zdroj: Frameset poslední stránky jednoduchého vícestránkového SCO

Problematika funkčnosti framesetů

Srozumitelnost je důležitý předpoklad. Ačkoli frame „scéna“ a frame okno ve kterém je přehráváno SCO budou vizuálně nerozeznatelné, je důležité pojmenovat každý frame tak, aby uživatelé, kteří spoléhají na vstřícné technologie, mohli identifikovat hlavní „scénu“ pro Vaše SCO. Hlavička kterékoliv HTML stránky, kterou zobrazíte v těchto framech musí mít také „title“.

DHTML vs. frameset

Samozřejmě, že namísto framesetů může být pro vícestránkové objekty SCO použito DHTML, ale takové implementace jsou spíše složitější, protože různé prohlížeče a různé jejich verze mají tendence vyžadovat rozdílné, charakteristické implementace DHTML obsahu. Příklad s framesetem, který jsme zde nastínili, byl úspěšně testován v řadě současných prohlížečů. Protože DHTML je obtížné programovat a odlazovat, pokročilé nástroje tvorby jako je ToolBook, které generují DHTML kód, zatímco Vám dovolují pracovat v mnohem pohostinnějším prostředí, jsou vesměs nejvhodnější cestou k implementaci objektů SCO založených na DHTML.

Kapitola 7 – řízení stavu SCO a komunikace

Teď, když už víme, jak implementovat a řídit základní komunikaci se SCORM API, je na čase pustit se do skutečné práce. SCO si může vyměňovat data s runtimovou službou přes API adaptér. Typicky to mohou být data o pokrocích studenta a jeho úspěších.

V následujícím příkladu vytvoříme jednoduché SCO, které přenáší informace o statusu a score do LMS.

Po důkladném přečtení specifikací SCORMu přijdeme na určitá specifika, někdy jsou to složitá pravidla a způsoby chování, které stanovují kdy a jak by měla být data přenášena. Tato pravidla jsou založena na nejlepších praktikách shromážděných během let řízení počítačově orientovaných instrukcí.

Složitější opětovně použitelný SCO script

Toto SCO bude muset kromě *LMSFinish* a *LMSInitialize* volat i další metody API adaptéru. Nesmí také obdržet nebo nastavovat data předtím, než bylo úspěšně voláno *LMSInitialize,* nebo poté, co bylo úspěšně voláno *LMSFinish*. Tento druh logiky je běžný u mnoha objektů SCO; je to užitečné pro zahrnutí do opakovaně použitelných scriptů.

SCORM 1.2 navíc specifikuje jisté vztahy a způsoby chování mezi datovými prvky. Například SCO vyžaduje od LMS *mastery score* a toto *mastery score* ovlivňuje hodnotu statusu passed/failed pro toto SCO. Případně by mělo SCO poskytovat data

o minimálním a maximálním score v momentě kdy hlásí score, aby se ujistilo, že si bude LMS score správně interpretovat.

Opakovaně použitelný script, který použijeme v tomto cvičení, implementuje následující prvky:

- řízení komunikační relace API: Hledání API adaptéru, volání *LMSInitialize* a volání *LMSFinish* se uskutečňuje automaticky, pokud do svého dokumentu vložíte příslušná volání v rámci tagů <body> nebo <frameset>.
- scripty stránky nebo framesetu nemusí vědět, kde se nachází objekt API adaptéru. Mohou volat API pomocí transferových funkcí: *SCOGetValue, SCOSetValue, SCOCommit,SCOGetLastError, SCOGetErrorString.*
- Inicializace: Jakmile je volání *LMSInitialize* úspěšné:
    - script si prověřuje mód prostřednictvím API a nastaví indikátor, pokud je režim „browse“
    - Pokud režim není „browse“, script ověřuje status prostřednictvím API. Pokud je „not attempted“, je změněn na „incomplete“
    - Script ověřuje, zda stránka nebo frameset má funkci *SCOInitData*. Pokud ji najde, zavolá ji.
    - Script zaznamenává aktuální čas, takže je později schopen hlásit uplynulý čas této relace.

- úklid: Před voláním *LMSFinish* hlásí script uplynulý čas aktuální relace. Poté ověřuje, zda stránka nebo frameset má funkci *SCOSaveData.* Pokud ano, zavolá ji.
- Hlášení času relace. Script hlásí uplynulý čas relace od okamžiku ukončení inicializace až po ukončení SCO. Specifický SCO script může také volat *SCOReportSessionTime()* kdykoliv po inicializaci a před tím, než je relace ukončena.
- logika dat score: Volání *LMSSetValue* jsou řízena tak, aby se ujistila, že minima a maxima score jsou hlášena zároveň s hrubým score. Předpokládá se, že hrubé score bude normalizováno v rozmezí 0..100. Pokud ve svém SCO použijete jiné rozmezí, přepište hodnoty *SCOScoreMin* a *SCOScoreMax* ve skriptu svých stránek. Všimněte si, že tato verze scriptu neověřuje, zda runtimová služba poskytuje jiné hodnoty pro min a max score, což by zároveň vyžadovalo komplikovanější nastavování aktuálně hlášeného rozmezí score a hrubého score.
- aktualizace statusu passed/failed. Pokud je hodnota *masteryScore* dostupná z runtimové služby, pak kdykoliv Vaše SCO hlásí score, status passed/failed je hlášen automaticky. Pokud z runtimové služby není dostupné žádné mastery score, bude použita přednastavená hodnota pseudokonstanty *SCO_MasteryScore.*
- Pokud *SCO_MasteryScore* nevyústí v číselnou hodnotu, status passed/failed nebude vůbec nastaven.
- dokončení logiky statusu. Protože v datovém modelu CMI status „failed“ nebo „passed“ znamená „completed“, script zahrnuje jistou logiku k uchování statusu „failed“ nebo „passed“, pokud je nastaven.

Ačkoli je tento script o dost složitější, než script z ukázky 3.2, může být také využit primitivním SCO. Jednoduché, jednostránkové SCO, užívající tento složitý script, je v ukázce 7.1. Jediná odlišnost od ukázky 3.3 je URL zahrnutého scriptu a přidání funkce k nastavení statusu na „completed“:

```
<html>
<head>
<script src="SCORMGenericLogic.js" type="text/javascript" language="JavaScript">
</script>
<title>A very simple SCO</title>
<script type="text/javascript" language="JavaScript">
function SCOSaveData(){
  // this function will be called when this SCO is unloaded
  SCOSetStatusCompleted()
}
</script>
</head>
<body onload="SCOInitialize()"
      onunload="SCOFinish()"
      onbeforeunload="SCOFinish()">
<h1>Hello world</h1><p>My status will be set to completed when I am unloaded.</p>
</body>
</html>
```

Ukázka 7.1 – Primitivní SCO s opakovaně použitelným složitějším skriptem doplněným o hlášení o dokončení

Toto SCO za běhu automaticky aktualizuje informace o statusu a času relace.

Dokončení nahrávání se složitějším opakovaně použitelným skriptem

Obecný, opakovaně použitelný script by neměl automaticky nastavovat *cmi.core.lesson_status* na *completed*, protože se nedá odhadnout záměr tvůrců různých SCO.

Jednoduché SCO, popsané výše, se dá považovat za dokončené, pokud ho uživatel viděl. Abychom to zaznamenali, můžeme pro tento účel jednoduše přidat funkci *SCOSaveData.* V okamžiku, kdy se volá *SCOFinish* a SCO je uvolňováno z paměti, zavolá obecný script *SCOSaveData* ještě před tím, než se volá *LMSFinish.* SCO v ukázce 7.1 zahrnuje funkci *SCOSaveData*, která postupně volá funkci obecného scriptu k nastavení statusu na „completed“. Volá raději tuto funkci, než aby nastavovala status přímo, protože se vyžaduje určitá doplňková obecná ověřovací logika.

Obrázek 7.2 ukazuje opakovaně použitelný script, který udělá veškerou tuto práci. V dodatku k další logice zahrnuje lepší obsluhu chyb než jak je tomu v ukázce 3.2.

```
// Saved as file "SCORMGenericLogic.js"
// SCORM 1.2 SCO Logic management script sample
// Copyright 2001,2002,2003 Click2learn, Inc.
// This version concocted by Claude Ostyn 2003-02-22
//
// This script implements various aspects of
// common logic behavior of a SCO.
// The SCO can be a HTML document or a frameset.
//
// Change these preset values to suit your taste and requirements.
var g bShowApiErrors = true;
var g_strAPINotFound = "Management system interface not found.";
var g_strAPITooDeep = "Cannot find API - too deeply nested.";
var g strAPIInitFailed = "Found API but LMSInitialize failed.";
var g strAPISetError = "Trying to set value but API not available.";

var g_nSCO_ScoreMin = 0;      // must be a number
var g nSCO ScoreMax = 100;    // must be a number > nSCO Score Min
var g_SCO_MasteryScore = 100; // value by default; may be set to null
var g_bMasteryScoreInitialized = false;

////////// API INTERFACE INITIALIZATION AND CATCHER FUNCTIONS ////////
var g_nFindAPITries = 0;
var g objAPI = null;
var g bInitDone = false;
var g bFinishDone = false;
var g_bSCOBrowse = false;
var g dtmInitialized = new Date(); // will be adjusted after initialize

function AlertUserOfAPIError(strText) {
  if (g_bShowApiErrors) alert(strText)
}

function FindAPI(win) {
  while ((win.API == null) && (win.parent != null) && (win.parent != win)) {
    g nFindAPITries ++;
    if (g_nFindAPITries > 500) {
      AlertUserOfAPIError(g strAPITooDeep);
      return null;
    }
    win = win.parent;
  }
  return win.API;
}

function APIOK() {
  return ((typeof(g_objAPI)!= "undefined") && (g_objAPI != null))
}

function SCOInitialize() {
  var err = true;
  if (!g_bInitDone) {
    if ((window.parent) && (window.parent != window)){
      g_objAPI = FindAPI(window.parent)
    }
    if ((g objAPI == null) && (window.opener != null))   {
      g objAPI = FindAPI(window.opener)
    }
    if (!APIOK()) {
      AlertUserOfAPIError(g strAPINotFound);
      err = false
    } else {
      err =  g objAPI.LMSInitialize("");
      if (err == "true") {
        g_bSCOBrowse = (g_objAPI.LMSGetValue("cmi.core.lesson_mode") == "browse");
        if (!g bSCOBrowse) {
          if (g objAPI.LMSGetValue("cmi.core.lesson status") == "not attempted") {
              err =  g_objAPI.LMSSetValue("cmi.core.lesson_status","incomplete")
          }
```

```
        }
      } else {
        AlertUserOfAPIError(g_strAPIInitFailed)
      }
    }
  }
  if (typeof(SCOInitData) != "undefined") {
    // The SCOInitData function can be defined in another script of the SCO
    SCOInitData()
  }
  g_dtmInitialized = new Date();
  return (err + "") // Force type to string
}

function SCOFinish() {
  if ((APIOK()) && (g bFinishDone == false)) {
    if (typeof(SCOSaveData) != "undefined"){
      SCOReportSessionTime()
      // The SCOSaveData function can be defined in another script of the SCO
      SCOSaveData();
    }
    g bFinishDone = (g objAPI.LMSFinish("") == "true");
  }
  return (g_bFinishDone + "" ) // Force type to string
}

// Call these catcher functions rather than trying to call the API adapter directly
function SCOGetValue(nam) {return ((APIOK())?g_objAPI.LMSGetValue(nam.toString()):"")}
function SCOCommit(parm)  {return ((APIOK())?g objAPI.LMSCommit(""):"false")}
function SCOGetLastError(parm){return ((APIOK())?g objAPI.LMSGetLastError(""):"-1")}
function SCOGetErrorString(n){return ((APIOK())?g_objAPI.LMSGetErrorString(n):"No API")}
function SCOGetDiagnostic(p){return ((APIOK())?g objAPI.LMSGetDiagnostic(p):"No API")}

//LMSSetValue is implemented with more complex data
//management logic through the SCOSetValue function below

var g bMinScoreAcquired = false;
var g_bMaxScoreAcquired = false;

function SCOSetValue(nam,val){
// Special logic to manage some special values
  var err = "";
  if (!APIOK()){
      AlertUserOfAPIError(g_strAPISetError + "\n" + nam + "\n" + val);
      err = "false"
  } else if (nam == "cmi.core.score.raw") err = privReportRawScore(val)
  if (err == ""){
      err =  g_objAPI.LMSSetValue(nam,val.toString() + "");
      if (err != "true") return err
  }
  if (nam == "cmi.core.score.min"){
    g_bMinScoreAcquired = true;
    g nSCO ScoreMin = val
  }
  else if (nam == "cmi.core.score.max"){
    g_bMaxScoreAcquired = true;
    g nSCO ScoreMax = val
  }
  return err
}
function privReportRawScore(nRaw) { // called only by SCOSetValue
  if (isNaN(nRaw)) return "false";
  if (!g_bMinScoreAcquired){
```

```
    if (g objAPI.LMSSetValue("cmi.core.score.min",g nSCO ScoreMin+"")!="true"){
      return "false"
    }
  }
  if (!g bMaxScoreAcquired){
    if (g_objAPI.LMSSetValue("cmi.core.score.max",g_nSCO_ScoreMax+"")!="true"){
      return "false"
    }
  }
  if (g_objAPI.LMSSetValue("cmi.core.score.raw", nRaw)!= "true") return "false";
  g_bMinScoreAcquired = false;
  g bMaxScoreAcquired = false;
  if (!g_bMasteryScoreInitialized){
    var nMasteryScore = parseInt(SCOGetValue("cmi.student_data.mastery_score"),10);
    if (!isNaN(nMasteryScore)) g SCO MasteryScore = nMasteryScore
  }
  if (isNaN(g_SCO_MasteryScore)) return "false";
  var stat = (nRaw >= g SCO MasteryScore? "passed" : "failed");
  if (SCOSetValue("cmi.core.lesson status",stat) != "true") return "false";
  return "true"
}

function MillisecondsToCMIDuration(n) {
//Convert duration from milliseconds to 0000:00:00.00 format
  var hms = "";
  var dtm = new Date();  dtm.setTime(n);
  var h = "000" + Math.floor(n / 3600000);
  var m = "0" + dtm.getMinutes();
  var s = "0" + dtm.getSeconds();
  var cs = "0" + Math.round(dtm.getMilliseconds() / 10);
  hms = h.substr(h.length-4)+":"+m.substr(m.length-2)+":";
  hms += s.substr(s.length-2)+"."+cs.substr(cs.length-2);
  return hms
}

function SCOReportSessionTime() {
// SCOReportSessionTime is called automatically by this script,
// but you may also call it at any time also from another SCO script
  var dtm = new Date();
  var n = dtm.getTime() - g dtmInitialized.getTime();
  return SCOSetValue("cmi.core.session_time",MillisecondsToCMIDuration(n))
}

function SCOSetStatusCompleted(){
// Since only the designer of a SCO knows what completed means, another
// script of the SCO may call SCOSetStatusComplete to set completed status.
// The function checks that the SCO was not launched in browse mode, and
// avoids clobbering a "passed" or "failed" status since those imply "completed".
  var stat = SCOGetValue("cmi.core.lesson_status");
  if (!g bSCOBrowse)
    if ((stat!="completed") && (stat != "passed") && (stat != "failed")){
      return SCOSetValue("cmi.core.lesson_status","completed")
    }
  } else return "false"
}
```

Ukázka 7.2 – Složitější script k řízení stavu SCO

V následujících dvou příkladech si ukážeme, jak může být složitý script použit ke značnému zjednodušení tvorby, testování a obsluhy jakéhokoli SCO, které hlásí systému LMS jiné druhy dat o průchodu kurzem.

Složitější SCO, které při uvolňování z paměti nahlašuje data o průchodu kurzem

SCO v ukázce 7.3 obsahuje jednoduché hodnocení ve formě různě obodovaných odpovědí na otázku typu multichoice. V tomto příkladu může uživatel změnit odpověď kolikrát chce. Status a score jsou hlášeny v reálném čase a mohou být hlášeny více než jednou s jinou hodnotou.

Toto je samozřejmě pouze strohý příklad, který můžete implementovat mnoha různými způsoby. Ukazuje nicméně, jak mohou opakovaně použitelný script a jednoduchá pravidla odvrátit složitost řízení komunikačních stavů objektu SCO a sledování výsledkových dat a dat o statusu.

Jak uživatel odpovídá, výsledky jsou zaznamenávány objektem SCO v proměnné. Výsledek je hlášen pouze v případě, že bylo SCO uvolněno z paměti, ale ještě před tím, než obecný script zavolá *LMSFinish.* Stane se tak, pokud obecný script hledá a nalezne ve scriptu objektu SCO funkci *SCOSaveData.*

Ve výsledku, pokud SCO použije *LMSGetValue* k obdržení statusu od runtimové služby, dostane pouze poslední hlášenou hodnotu, která nemusí být synchronizovaná s tím, co SCO chápe jako aktuální score. Jak tento problém vyřešit? Odpověď je v následujícím příkladu.

```
<html>
<head>
<script src="SCORMGenericLogic.js" type="text/javascript" language="JavaScript">
</script>
<script type="text/javascript" language="JavaScript">
var znScore = 0;
function EvaluateMultipleChoiceItem(obj) {
  znScore = obj.value
}
function SCOSaveData() {
  // this function is called by the generic script before calling LMSFinish
  if (!isNaN(znScore)) {
    SCOSetValue("cmi.core.score.raw", znScore); // this will set pass/fail status
    SCOCommit() // to make sure that, no matter what happens, this has been recorded
  }
}
function ShowSCORMStatus(){
  alert('Current status is ' + SCOGetValue('cmi.core.lesson_status'))
}
</script>
<title>One question test</title>
</head>
<body onload="SCOInitialize()"
      onunload="SCOFinish()"
      onbeforeunload="SCOFinish()">
<form>Which of these is the correct answer?<br />
<input type="radio" name="q1" value="60"
onclick="JavaScript:EvaluateMultipleChoiceItem(this)" />This, kind of.<br />
<input type="radio" name="q1" value="100"
onclick="JavaScript:EvaluateMultipleChoiceItem(this)" />This, absolutely.<br />
<input type="radio" name="q1" value="0"
onclick="JavaScript:EvaluateMultipleChoiceItem(this)" />Definitely not this.<br />
</form>
<p>
<form>
<input type="button" onclick="JavaScript:ShowSCORMStatus()" value="Check Status" />
</form>
</p>
</body>
</html>
```

Ukázka 7.3 – Jednostránkové SCO, které nastavuje SCORM data při uvolňování SCO z paměti

Komentář k obrázku:
Všimněte si, že zde není žádná potvrzovací akce, protože by to způsobilo volání *onbeforeunload* v Internet Exploreru, který postupně spouští předčasné volání *LMSFinish.* Použití konceptu jako je <a href="javascript:xxx()"> bylo také zamítnuto, jelikož při kliknutí na odkaz rovněž způsobuje volání *onbeforeunload.* Tento problém se vyskytne pouze v případě, že HTML dokument, kde se tyto koncepty objevují, je zároveň samotným SCO. Samozřejmě, že by stačilo jednoduše nepoužít *onbeforeunload* v tagu <body>, ale s rizikem toho, že SCO bude o něco méně spolehlivé.

Když podstupujete tuto volbu, mějte na paměti, že je dobrým zvykem, aby Vaše stránky byly přístupné. Pomocné prvky v prohlížeči nemusí spustit scriptové události onclick asociované s textovými fragmenty; textové fragmenty se také nezobrazí automaticky jako hyperlinky.

Optimálnost:
Obecně řečeno není přístup ke sledování ve SCORMu uvedený v ukázce 7.3, kde jsou data hlášena jen když je SCO uvolňováno z paměti, optimální, a to z několika důvodů:

- hlášení velkého množství dat při události unload v roli spouštěcí události by mohlo být náročné. Pokusy ukázaly, že při některých implementacích může zpoždění na druhém konci databáze překročit prohlížečem povolený čas k ukončení výkonu scriptů stránek uvolněných z paměti.
- Pokud dojde k osudné události, všechna data zaznamenaná objektem SCO v průběhu relace budou ztracena
- Informace zaznamenané runtimovou službou a informace ve SCO se mohou desynchronizovat. Vesměs to nepředstavuje vážný problém, ale příklad z ukázky 7.3 ukazuje, jak to může omezit některé aplikace, protože informace přístupné skrz API jsou zastaralé.

Naštěstí existuje důkladnější přístup, který hlásí důležitá data kdykoliv se objeví; takový postup popisuje následující příklad.

Složitější SCO, které hlásí data o průchodu kurzem v okamžiku, kdy se objeví

Stejně jako SCO v ukázce 7.3 obsahuje ukázka 7.4 jednoduché hodnocení ve formě různě obodovaných odpovědí na otázku typu multichoice. Rozdíl je v tom, že jak uživatel odpovídá, score není zaznamenáváno pouze interně, ale je také ihned nahlášeno prostřednictvím SCORM API. Takže na rozdíl od předchozího příkladu dotaz na status vrátí poslední hlášenou hodnotu, která bude synchronizována s tím, co SCO chápe jako aktuální score.

```
<html>
<head>
<script src="SCORMGenericLogic.js" type="text/javascript" language="JavaScript">
</script>
<script type="text/javascript" language="JavaScript">
function EvalMultipleChoiceItem(obj) {
  var v = obj.value;
  if (!isNaN(v)) {
    SCOSetValue("cmi.core.score.raw",v); // this will also set pass/fail status
    SCOCommit() // to make sure that, no matter what happens, this has been recorded
  }
}
function ShowSCORMStatus(){
  alert('Current status is ' + SCOGetValue("cmi.core.lesson status"))
}
</script>
<title>One question test</title>
</head>
<body onload="SCOInitialize()"
      onunload="SCOFinish()"
      onbeforeunload="SCOFinish()">
<form>Which of these is the correct answer?<br />
<input type="radio" name="q1" value="60"
onclick="JavaScript:EvalMultipleChoiceItem(this)" />This, kind of.<br />
<input type="radio" name="q1" value="100"
onclick="JavaScript:EvalMultipleChoiceItem(this)" />This, absolutely.<br />
<input type="radio" name="q1" value="0"
onclick="JavaScript:EvalMultipleChoiceItem(this)" />Definitely not this.<br />
</form>
<p>
<form>
<INPUT TYPE="BUTTON" onClick="JavaScript:ShowSCORMStatus()" NAME="CheckStatus"
VALUE="Check Status">
</form>
</body>
</html>
```

Ukázka 7.4 – Jednostránkové SCO, které nastavuje SCORM data v reálném čase

Nejasnosti statusu

Specifikace SCORMu 1.2 převzala mnoho z datového modelu užívaného ke komunikaci mezi obsahem a runtimovou službou ze starších směrnic a doporučení AICC Computer Managed Instruction (CMI). Jména elementů toto dědictví odrážejí. Například „*cmi.core.lesson_status“* znamená „element lesson_status definovaný v datovém modelu core datového modelu CMI.“

Zvláštním aspektem této specifikace je, že status SCO („*cmi.core.lesson_status“*) může nést pouze jednu hodnotu, ale tato hodnota musí reprezentovat různé druhy informací o statusu. Přípustné hodnoty, které reprezentují mastery jsou „*passed*“ nebo „*failed*“. Nelze například vyjádřit „incomplete and passed“, protože by to vyžadovalo dvě hodnoty a dovolena je pouze jedna.

Budoucí verze specifikace SCORMu napraví tento problém, ale zatím je třeba předpokládat, že „*passed*“ nebo „*failed*“ znamená zároveň „*completed*“, alespoň tam, kde se jedná o jednoduché SCO.

Jak a kdy nastavovat informace o statusu

Když je SCO spuštěno poprvé, jeho status je „*not attempted“.* SCORM specifikuje, že toto je hodnota, která by v tomto případě měla být objektu SCO předána. Jakmile vstoupí uživatel do interakce se SCO, status tohoto SCO se změní na „*incomplete*“. Pokud uživatel splnil vše, co po něm tvůrce objektu SCO zamýšlel požadovat, status se může změnit na „*completed*“. Pokud je zahrnuta míra splnění, která může být

shrnuta jako hodota *pass/fail*, status se mění buď na „*passed*“, nebo na „*failed*“, což zároveň znamená „*completed*“.

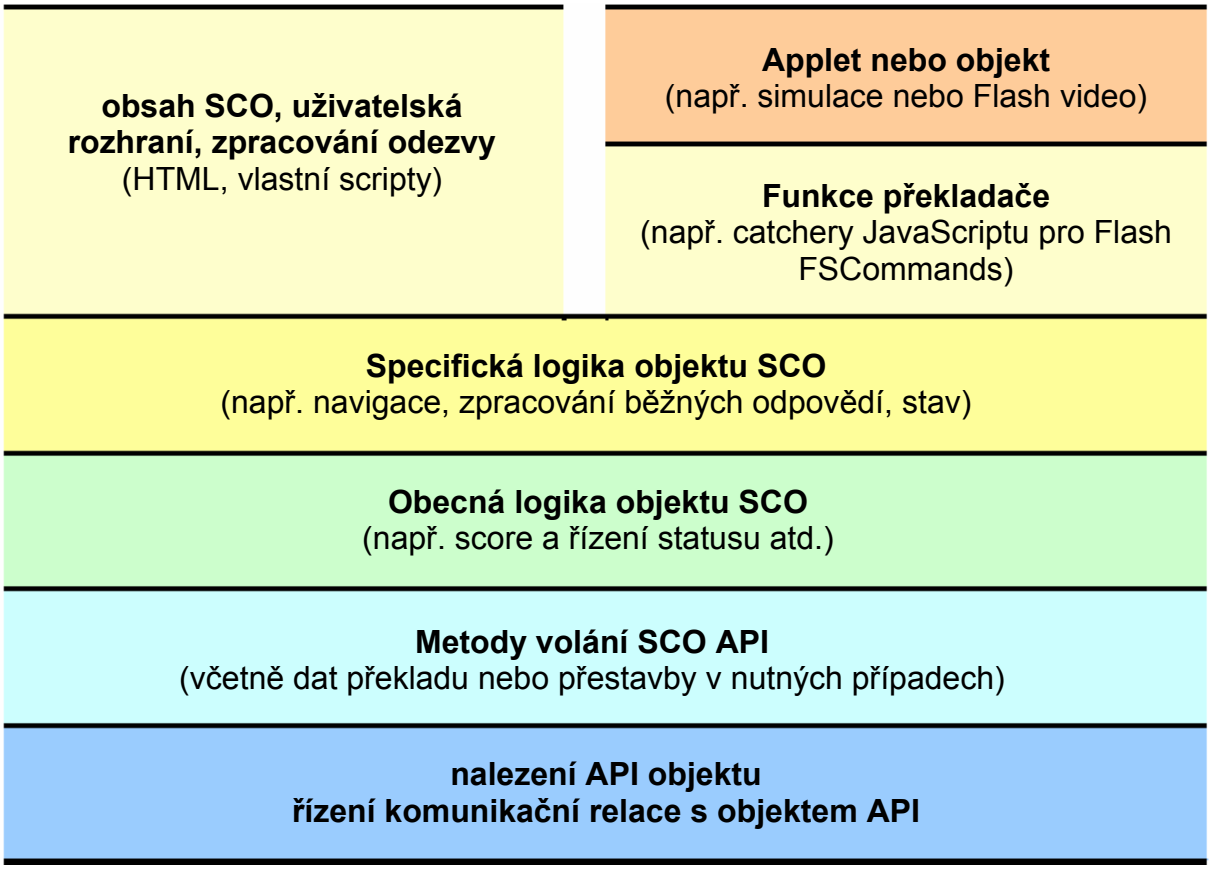
Všimněte si, že existuje nejméně jeden případ, kdy se status může změnit mimo kontrolu SCO: Pokud SCO hlásí score a runtimová služba ví o mastery score, status je runtimovou službou změněn na „*passed*“ nebo „*failed*“ porovnáním hlášeného score s mastery score. SCO může získat hodnotu tohoto mastery score a také ji použít k určení statusu *pass/fail*, ale runtimová služba bude mít vždy poslední slovo.

Opakovaně použitelný script v ukázce 7.2 řídí informace o statusu ve shodě s těmito jednoduchými pravidly, proto mohou být scripty pro samotné objekty SCO tak jednoduché.

Kapitola 8 – vrstvená architektura objektů SCO

Možná jste začali vnímat trend v dosud uvedených ukázkách: Scripty na stránkách SCO zastávají pouze lehkou práci; tu těžkou vykonají opakovaně použitelné scripty. Také samotné opakovaně použitelné scripty směřují k izolaci detailů komunikace s API od logiky toho, co a kdy přenášet.

Tento vrstvený přístup je shrnut na obrázku 8.1



Obr. 8.1 – přístup skriptovacích vrstev objektu SCO

Existují tři důvody, pro které je v uvedených ukázkách použit tento přístup:

- Jednodušší přizpůsobení objektů SCO uvedených v příkladech. Jakmile byla jedna z nižších vrstev odlazena, může být používána znovu a znovu a šetří tak čas potřebný při koncentraci na vyšší vrstvy.

- Jednoduchost údržby. Oddělením funkcí do separátních vrstev mohou být tyto vrstvy rovněž odděleně spravovány. Například přidání logiky překladu dat do jedné vrstvy neovlivní ostatní vrstvy.
- Budoucí ochrana. Budoucí verze SCORMU pravděpodobně uvedou více či méně patrné změny v API, API protokolu, komunikačním datovém modelu nebo v chybových kódech. Revize pouze nižších, opakovaně použitelných vrstev, může z velké části, ne-li zcela, odstranit důsledky spojené s těmito změnami. Například pokud budoucí verze SCORMu bude začínat použitím objektu SOAP namísto objektu DOM rozhraní API, vyřeší to výměna pouze spodních dvou vrstev na obr. 8.1, což ochrání investice do nákladnějších vlastních vrstev obsahu.

Následující příklady budou za účelem vyšší funkčnosti stavět na tomto vrstveném přístupu

Kapitola 9 – pozastavení a obnovení

Pokus o dokončení SCO může vyžadovat více než jednu relaci. SCORM 1.2 umožňuje pozastavení distribuce objektu SCO a jeho pokračování v pozdější relaci.

Předtím než je relace objektu SCO pozastavena:
- musí SCO hlásit, že jeho výstupní mód je „*suspend*“.
- SCO musí požádat API adaptér buď o uložení pozastavených dat nebo o ustanovení nějaké lokace. Obsah a forma pozastavených dat nebo lokace takových dat jsou definovány objektem SCO. LMS nebo runtimová služba se nepokouší interpretovat tyto informace; musí je pouze uložit a zpřístupnit objektu SCO při dalším spuštění v pozdější relaci. Velikost pozastavených dat je limitována 4096 znaky.

Informace o pozastavení vyplývá ze souvislostí. Pokud je SCO spuštěno v jiné souvislosti - tzn. například pro jiného uživatele, v novém pokusu, nebo z důvodu, že stejné SCO je zrovna užíváno v jiné výukové aktivitě - pozastavená informace se na tuto souvislost nepřenese.

Pokud je SCO spuštěno ve stejné souvislosti:
- SCO se zeptá API adaptéru, zda existuje předchozí pozastavení relace
- pokud odpověď je „*resume*“, pak SCO požádá API adaptér o pozastavená data nebo lokaci takových dat, informace uložené během předchozích relací.

Všimněte si, že objektu SCO nic nebrání v tom, aby okamžitě deklaroval, že chce být pozastaven kdykoliv je uvolňován z paměti, a aby ukládal pozastavená data nebo lokaci takových dat v každém kritickém okamžiku během distribuce objektu SCO. Zvláště v nestálých prostředích pro distribuci to umožňuje objektu SCO pokračovat od takového kritického okamžiku, byla-li distribuce z nějakého důvodu nestandardně přerušena bez možnosti provést řádné ukončení.

Vícestránkové SCO, které může být pozastaveno na kterékoli stránce

Následující příklad představuje vícestránkové SCO, které může být pozastaveno kdykoliv a obnoveno ze stránky, na které bylo pozastaveno. V zájmu stručnosti bude v příkladu použit pouze prvek lokace dat a velmi jednoduchá pozastavená data, ale můžete jej snadno rozšířit tak, aby bylo možno uložit další smysluplná pozastavená data podle potřeby.

Tento příklad obsahuje mnohem méně komponent a způsobů chování než předcházející příklady, poskytneme proto před uvedení příkladu poněkud zřejmější vysvětlení.

Struktura

Toto SCO je vytvořeno jako frameset podle modelu z předcházející kapitoly. Struktura stránky je nicméně poněkud odlišná. Struktura stránky zahrnuje nahrávací stránku, na kterou se uživatel z navigace nedostane a koncovou stránku, ze které se uživatel nedostane na další stránku.

Stránka 1 by mohla být obsahem kurzu s odkazy na ostatní stránky v rámci SCO, nebo může být první v přímočaré sekvenci stránek.

Obr. 9.1 – šablona struktury násobných stránek

Jak to funguje

Frameset je SCO a scripty určené ke komunikaci s API jsou řízeny framesetem. SCO volá API pouze v případě, že je zcela nahráno. Proto nemůže zjistit, zda je obnovováno, nebo startováno *ab initio*, dokud není stránka zcela zobrazena. Pokud je obnovováno, může být donuceno ukázat jinou stránku než stránku 1. Proto, když je SCO spuštěno, vždy začíná nahrávací stránkou, která není částí normální sekvence stránek. Tato stránka je na obr. 9.1 zobrazena jako „stránka 0“. Normální navigace stránkami objektu SCO nedovolí dostat se na stránku 0. Zatímco probíhá inicializace objektu SCO, tato stránka je určena výhradně k zobrazení jiných věcí, než stránka 1. Použití nahrávací stránky také řeší problém s časováním scriptů spouštěných nahrávacími událostmi, protože když prohlížeč inicializuje frameset, spouští nahrávací událost stránek nahrávaných do framů ještě před spuštěním nahrávací události pro samotný frameset. Pokud se tato první nahraná stránka pokusí při svém nahrávání zavolat API prostřednictvím svého rodičovského framesetu, neuspěje.

Co se bude dít dál, to záleží na datech, která SCO dostává prostřednictvím API adaptéru. Pokud se SCO neobnovuje, nebo pokud během předešlé relace nebyla uložena žádná lokace, pak okamžitě přechází na stránku 1. Pokud se obnovuje a z předchozí relace je dostupná platná lokace, okamžitě přechází na tuto lokaci, aniž by kdy ukázalo stránku 1.

Obrázek 9.1 také ukazuje „koncovou stránku“. Takovouto stránku můžete, nebo nemusíte použít ve vlastní aplikaci. Pokud uživatel vykoná cokoliv co považujete jako „koncové“, navigace by měla přejít na tuto stránku a uživatel by neměl být schopen navigovat zpět na jinou stránku. Například, pokud sada vašich stránek je sérií otázek v testu (quiz), mohla by koncová stránka zobrazovat konečné score poté, co uživatel klikne tlačítko „hotovo“ umístěné na další stránce. Uvolnění objektu SCO z paměti v okamžiku, kdy je zobrazena koncová stránka, nenastaví výstupní hodnotu na „*suspend“*, ale nastaví ji na “” (prázdný řetězec) a hlásí score a eventuálně status „*passed*“ nebo „*failed*“. Na druhou stranu uvolnění objektu SCO z paměti v okamžiku,

kdy je zobrazena kterákoliv jiná stránka nastaví výstupní mód na „*suspend*“, ale zároveň ponechá hodnotu statusu na „*incomplete*“.
V tomto příkladu nastavuje SCO určité pozastavené informace pokaždé, když je zobrazena stránka, na kterou je možné navigovat. Pokud je SCO uvolněno z paměti v okamžiku, kdy se nachází na stránce, na kterou je možné navigovat, a později je znovu spuštěno ve stejné souvislosti, bude tato pozastavená informace poskytnuta runtimové službě a použita k návratu na stejnou stránku. Koncová stránka se používá jako místo, kam se uživatel dostane po kliknutí na „Vše hotovo“ na poslední stránce. Pokud uživatel vybere „Vše hotovo“, zobrazí se koncová stránka. Je li SCO uvolněno z paměti v okamžiku, kdy se nachází na koncové stránce, nastaví výstupní hodnotu na ““ (prázdný řetězec). Při příštím spuštění objektu SCO to bude vypadat, jako když začíná znovu od začátku.

Opětovné použití obecné logiky

Tento příklad využívá opětovně použitelný script z ukázky 7.2. Přidává do něj vlastní logiku za účelem řízení funkčnosti pozastavení a obnovení. V zájmu stručnosti tento příklad nezahrnuje záležitosti jako sledování a nastavování score a nastavuje pouze informaci o dokončení poté, co byly uživateli zobrazeny všechny stránky. Informace o tom, které stránky byly navštíveny se ukládají od relace k relaci do pozastavených dat. Nejaktuálnější stránka se ukládá od relace k relaci jako hodnota pro lokaci dat.

Tento frameset je téměř úplně stejný jako ukázka 6.2, je do něj pouze přidána logika k uchování výsledku sledování toho, která stránka už byla zhlédnuta, nastavení informace o dokončení a způsoby chování při pozastavení a obnovení.

```
<html>
<head>
<script src="SCORMGenericLogic.js" type="text/javascript"
   language="JavaScript">
</script>

<script type="text/javascript" language="JavaScript">
// Manage page navigation
var znNavigablePages=3;
var znThisPage=0;
var zaVisitedPages = new Array(znNavigablePages)
function NextPage() {
  if (znThisPage < znNavigablePages){
    znThisPage++;
    myStage.location.href = "page" + znThisPage + ".htm"
  }
}
```

```
function PreviousPage() {
  if (znThisPage > 1){
    znThisPage--;
    myStage.location.href = "page" + znThisPage + ".htm"
  }
}
function GoToPage(n) {
  if ((!isNaN(n)) && (n >= 1) && (n <= znNavigablePages)){
    myStage.location.href = "page" + n + ".htm";
  }
}
function SetThisPage(n) { // called by each navigable page when displayed
  var i = 0; var nCnt = 0
  znThisPage = n;
  zaVisitedPages[n-1] = true;
  for (i = 0 ; i < znNavigablePages; i++){
    if (zaVisitedPages[i]) { nCnt++ }
  }
  if (nCnt == znNavigablePages) {
    SCOSetStatusCompleted()
  }
  // Make sure this is recorded, just in case
  SCOSetValue("cmi.core.exit","suspend");
  SCOSetValue("cmi.core.lesson_location", znThisPage);
  SCOSetValue("cmi.suspend_data",zaVisitedPages.join(","));
  SCOCommit()
}
function AllDone() {
  znThisPage = znNavigablePages + 1;
  myStage.location.href = "endpage.htm"
}
function SCOInitData() {
 var loc = 1;
  if (SCOGetValue("cmi.core.entry") == "resume"){
    var SuspendData = SCOGetValue("cmi.suspend_data")
    if (SuspendData.length > 0) {
      zaVisitedPages = SuspendData.split(",")
    }
    var loc =(parseInt(SCOGetValue("cmi.core.lesson_location")));
   if (isNaN(loc)) loc = 1;
  }
 GoToPage(loc)
}
function SCOSaveData() {
  if (znThisPage > znNavigablePages) {
    SCOSetValue("cmi.core.exit","") // this is the endpage, no need to resume
  }
}
</script>

<title>Sample Multiple Page SCO with Suspend and Resume</title>
</head>
<frameset rows="100%,*"
      onload="SCOInitialize()"
      onunload="SCOFinish()"
      onbeforeunload="SCOFinish()">
>
<frame name="myStage" title="Learning Object display window" src="page0.htm" />
<frame src="dummypage.htm" />
</frameset>
</html>
```

Ukázka 9.1 – HTML zdroj: obnovitelný frameset vícestránkového SCO

```
<html><!-- saved as file "dummypage.htm" -->
<head><title>empty</title></head>
<body> </body>
</html>
```

Ukázka 9.2 – HTM zdroj: Fingovaná stránka ve framesetu vícestránkového obnovitelného SCO

```
<html><!-- saved as file "page0.htm" -->
<head><title>---</title></head>
<body>One moment, please...</body>
</html>
```

Ukázka 9.3 – HTML zdroj: Stránka 0 framesetu vícestránkového obnovitelného SCO

```
<html><!-- saved as file "page1.htm" -->
<head><title>Page 1</title></head>
<body onload="window.parent.SetThisPage(1)">
<p>This is page 1</p>
<p align="right">
<a href="JavaScript:window.parent.NextPage()">Next</a>
</p>
</body>
</html>
```

Ukázka 9.4 – HTML zdroj: Stránka 1 framesetu vícestránkového obnovitelného SCO

```
<html><!-- saved as file "page2.htm" -->
<head><title>Page 2</title></head>
<body onload="window.parent.SetThisPage(2)">
<p>This is page 2</p>
<p align="right">
<a href="JavaScript:window.parent.PreviousPage()">Previous</a>
<a href="JavaScript:window.parent.NextPage()">Next</a>
</p>
</body>
</html>
```

Ukázka 9.5 – HTML zdroj: Stránka 2 framesetu vícestránkového obnovitelného SCO

```
<html><!-- saved as file "page3.htm" -->
<head><title>Page 3</title></head>
<body onload="window.parent.SetThisPage(3)">
<p>This is page 3 (the last navigable page)</p>
<p align="right">
<a href="JavaScript:window.parent.PreviousPage()">Previous</a>
<a href="JavaScript:window.parent.AllDone()">All done</a>
</p>
</body>
</html>
```

Ukázka 9.6 – HTML zdroj: Poslední stránka framesetu vícestránkového obnovitelného SCO, na niž je možno navigovat

```
<html><!-- saved as file "endpage.htm" -->
<head><title>---</title></head>
<body>All done!</body>
</html>
```

Ukázka 9.7 – HTML zdroj: Koncová stránka framesetu vícestránkového obnovitelného SCO

Kapitola 10 – sledování cílů v rámci SCO

SCORM 1.2 dovoluje objektům SCO sledovat plnění cílů. Několik faktů o cílech:

- Cíle objektu SCO jsou vlastně pouze malé záznamy statusu něčeho, co se nazývá cílem a co je srozumitelné pouze tomuto SCO. Cílů, které SCO sleduje, může existovat více. Například pokud Vaše SCO zahrnuje 3 aktivity, můžete definovat cíl pro každou aktivitu a užít data tohoto cíle ke sledování toho, zda už byla aktivita splněna.
- Informace o statusu, která můžete použít pro cíl, zahrnují dokončení a score. Z těchto elementů je možné použít jeden, dva, nebo žádný v závislosti na tom, co pro Vás cíle znamenají.
- Předpokládá se, že status cílů bude uchován LMS systémem alespoň od relace k relaci v rámci pokusu o dokončení SCO. Pokud jsou ale započaty nové pokusy, SCORM nespecifikuje, zda tento status přetrvá nebo ne. To záleží na implementační politice LMS.
- Specifikace SCORMu nedefinují, zda cíle SCO souvisí nebo nesouvisí s výukovými cíli, které může autor asociovat s užitím SCO.
- Neexistuje definovaná souvislost mezi statusem cíle a celkovým statusem SCO, nebo mezi statusem cíle a statusem interakcí v rámci SCO. Pokud chcete, můžete takové souvislosti vytvořit, ale jaké mají být a jaký má být jejich význam – to ve SCORMu definováno není. Můžete se například rozhodnout, že SCO bude považováno za dokončené, když všechny cíle, které pro SCO definujete, byly splněny. Nebo se můžete rozhodnout tak, že SCO bude považováno za dokončené, když bylo dosaženo 80% score pro 67% cílů. Nebo můžete použít cíle jen pro uchování sledování toho, zda určité stránky byly zhlédnuty a žádné další souvislosti se SCO jako celkem nedefinovat.
- Není definováno, zda cíle korespondují s aktuálními výukovými cíli, které mají být sledovány mimo kontext distribuce toho konkrétního SCORM balíčku. Dokážeme si představit implementaci, kde informace o statusu sledování cíle mohou být přednastaveny ještě před spuštěním obsahu a jsou založeny na existujících informacích v profilu studenta. Zatím ale neexistuje standardní postup.
- Ne každý LMS uchovává data o sledování cílů. Deklarace SCORM 1.2 konformity definují tento prvek jako volitelný, což znamená, že ho konformní LMS nemusí implementovat. Bylo ověřeno, že LMS Click2learn Aspen tento prvek zahrnuje.

Jak se liší cíle od pozastavených dat

Pozastavená data a data o lokaci jsou pro LMS skryta. LMS neočekává, že bude schopen interpretovat informaci těchto datových elementů. Na druhou stranu cíle používají jasně definovaný, standardizovaný datový model. To umožňuje LMS číst a eventuálně nastavovat data cílů.

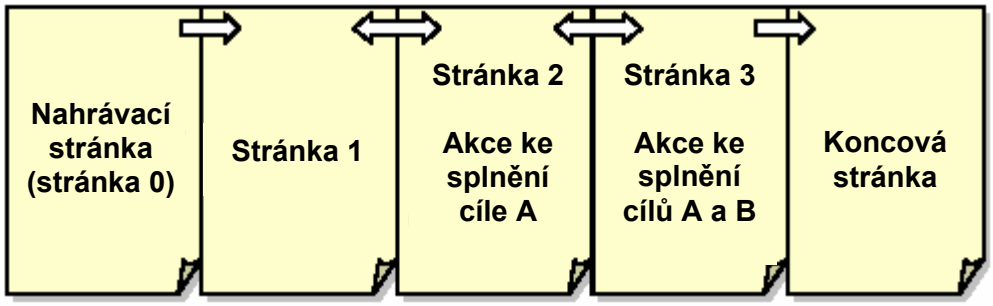
Dalším rozdílem je, že pozastavená data a lokace by neměly být uchovávány v LMS mezi jednotlivými pokusy, ale pouze mezi relacemi během stejného pokusu. Jakmile je pokus považován za dokončený, pozastavená data a lokace mohou být zahozeny. Data cílů však mohou nebo nemusí být zachována mimo životní rozsah pokusu. Tak je to uvedeno v pravidlech, která se pojí s LMS.

Praktický příklad

Příklad v této kapitole je téměř identický s předcházejícím příkladem, který používal pozastavená data a lokaci. Rozdíl je v tom, že jsou v něm užity cíle k uchování sledování toho, zda uživatel vykonal několik specifických akcí zatímco procházel objektem SCO. Se skutečným obsahem by mohla tato akce představovat dokončení praktického cvičení nebo přehrání demonstrace, případně důkaz zvládnutí konkrétního cíle. V rámci zachování jednoduchosti bude v ukázce touto akcí kliknutí na určité prvky stránek 2 a 3.

Co bude SCO vykonávat

Jak uživatel zdolává jednotlivé stránky, vzhled stránek 2 a 3 a jejich chování se mění v závislosti na statusu cílů souvisejících s těmito stránkami. Na stránce 2 je uživatel vyzván ke kliknutí na odkaz, pokud tak neučinil už dříve. Pokud uživatel přejde na stránku 2 a na odkaz již bylo kliknuto dříve, místo výzvy bude zobrazeno potvrzení o tom, že akce již byla dříve splněna. Stránka 3 pracuje stejným způsobem s tím rozdílem, že jsou na ní dvě akce sledované jako dva odlišné cíle. Stránka 3 opakovaně používá cíl ze stránky 2, takže můžete sledovat, jak může fungovat splnění stejného cíle na rozdílných stránkách, přestože data cílů jsou zaznamenávána mimo sekvenci. Uživatel může navigovat dopředu a dozadu mezi stránkami 1, 2 a 3 a pouze ze strany 3 dopředu směrem na koncovou stránku, která ukazuje konečný status cílů.



Obr. 10.1 – Stránky ve SCO uchovávající sledování individuálních cílů

Technické poznámky k příkladu

V tomto příkladu je spoléháno na LMS, že uchová sledování statusu cílů. Pokud je toto SCO spuštěno v prostředí LMS, které nepodporuje SCO cíle, nebude pracovat správně.

Tento příklad používá stejné soubory jako předcházející příklad, s výjimkou stránek 2 a 3, kam byly dodány doplňkové způsoby chování, a framesetu samotného, který zahrnuje další fragment scriptu k řízení SCO cílů. V níže uvedené ukázce bude popsán pouze ten fragment scriptu, který by měl být zahrnut ve framesetu a stránky 2 a 3 budou uvedeny v ukázce kompletní. Ostatní komponenty objektu SCO najdete v předcházejícím příkladu.

Technické poznámky k opakovaně použitelnému scriptu

Každý cíl má unikátní identifikátor, ale záznam o každém cíli je uložen v ekvivalentu typu pole. Protože SCORM nespecifikuje, zda pořadí záznamů bude v jednotlivých spuštěních stejné, měl by být v poli místo pozice použit unikátní identifikátor. Všimněte si také, že záznam o cíli může být do pole přidán pouze specifikací pozice, která je v sekvenci s existujícím záznamem o cíli. Jinými slovy není možné přidávat cíle na pozici 3 dokud jsou ještě cíle na pozicích 1 a 2.

Specifikace SCORMu 1.2 stanoví, že ID cíle je volitelné. Je to chyba, která bude ve SCORM 1.3 opravena. Bez ID cíle se musíte spolehnout na relativní pozici dat v poli cílů, abyste obdrželi data cílů, což nemusí být spolehlivé. Opakovaně použitelný script obchází tuto záludnost užíváním ID jako klíče pokaždé, když získáváte nebo nastavujete data cílů. Obdržíte-li data o cíli, script hledá záznam o cíli, který toto ID má. Nastavíte-li data pro cíl, script hledá záznam pro tento cíl. Pokud existuje, tak jej aktualizuje. Pokud neexistuje, přidá záznam pro tento cíl. Prověrkou scriptů pro stránky 2 a 3 a koncové stránky zjistíte, jak to zjednodušuje řízení cílů. Jediná věc, kterou si musíte pamatovat, je přidělit různým cílům různé ID hodnoty.

Někteří realizátoři LMS pojali model informací o cílech stylem deníku, tedy pro každou změnu cíle je vytvářen nový záznam namísto aktualizace existujícího záznamu. Script pracuje s oběma typy implementací, ale vždy se nejdříve pokouší aktualizovat existující záznam a pokud tento postup selže, pokusí se přidat záznam. V případě, že před startem SCO již v implementaci existuje instance pole cílů, hledání existujícího záznamu dat cílů mezi existujícími záznamy se provádí obráceně, aby byl vždy nalezen poslední záznam pro tento cíl pro případ, že existuje více než jeden záznam se stejnou ID hodnotou.

```
// Saved as file "SCORMObjectiveLogic.js"

function SCOSetObjectiveData(id, elem, v) {
  var result = "false";
  var i = SCOGetObjectiveIndex(id);
  if (isNaN(i)) {
    i = parseInt(SCOGetValue("cmi.objectives. count"));
    if (isNaN(i)) i = 0;
    if (SCOSetValue("cmi.objectives." + i + ".id", id) == "true"){
      result = SCOSetValue("cmi.objectives." + i + "." + elem, v)
    }
  } else {
    result = SCOSetValue("cmi.objectives." + i + "." + elem, v);
    if (result != "true") {
      // Maybe this LMS accepts only journaling entries
       i = parseInt(SCOGetValue("cmi.objectives._count"));
      if (!isNaN(i)) {
        if (SCOSetValue("cmi.objectives." + i + ".id", id) == "true"){
          result = SCOSetValue("cmi.objectives." + i + "." + elem, v)
        }
      }
    }
  }
  return result
}

function SCOGetObjectiveData(id, elem) {
  var i = SCOGetObjectiveIndex(id);
  if (!isNaN(i)) {
    return SCOGetValue("cmi.objectives." + i + "."+elem)
  }
  return ""
}

function SCOGetObjectiveIndex(id){
  var i = -1;
  var nCount = parseInt(SCOGetValue("cmi.objectives._count"));
  if (!isNaN(nCount)) {
    for (i = nCount-1; i >= 0; i--){ //walk backward in case LMS does journaling
      if (SCOGetValue("cmi.objectives." + i + ".id") == id) {
        return i
      }
    }
  }
  return NaN
}
```

Ukázka 10.1 – fragment JavaScriptu k řízení cílů SCO

```
<html>
<head>
<script src="SCORMGenericLogic.js" type="text/javascript" language="JavaScript">
</script>
<script src="SCORMObjectiveLogic.js" type="text/javascript" language="JavaScript">
</script>
...
```

Ukázka 10.2 – Fragmet framesetu modifikovaného tak, aby zahrnoval logiku cílů

```
<html><!-- saved as file "page2.htm" -->
<head><title>Page 2</title></head>
<body onload="window.parent.SetThisPage(2)">
<p>This is page 2</p>
<p align="right">
<a href="JavaScript:window.parent.PreviousPage()">Previous</a>
<a href="JavaScript:window.parent.NextPage()">Next</a>
</p>
<script type="text/javascript" language="JavaScript">
var s = "<p>";
var objectiveID = "myUniqueObjectiveName001";
function MarkADone(){
  window.parent.SCOSetObjectiveData(objectiveID, "status", "completed");
  window.parent.SCOCommit();
  window.location.href=window.location.href;
}
if (window.parent.SCOGetObjectiveData(objectiveID, "status") != "completed") {
  s = 'Please click <a href="JavaScript:MarkADone()">here</a> for objective A.'
}
else { s = 'You already completed objective A.' }
document.write('<p>' + s + '</p>')
</script>
</body>
</html>
```

Obr. 10.3 – HTML a JavaScript – Získání a nastavení dat cíle

```
<html><!-- saved as file "page3.htm" -->
<head><title>Page 3</title></head>
<body onload="window.parent.SetThisPage(3)">
<p>This is page 3 (the last navigable page)</p>
<p align="right">
<a href="JavaScript:window.parent.PreviousPage()">Previous</a>
<a href="JavaScript:window.parent.AllDone()">All done</a>
</p>
<script type="text/javascript" language="JavaScript">
function MarkObjectiveCompleted(objectiveID){
  window.parent.SCOSetObjectiveData(objectiveID, "status", "completed");
  window.parent.SCOCommit();
  window.location.href=window.location.href;
}
var s = "";
var objectiveID = "myUniqueObjectiveName001";
if (window.parent.SCOGetObjectiveData(objectiveID, "status") != "completed") {
  s = 'Please click <a href="JavaScript:MarkObjectiveCompleted(\'';
  s += objectiveID +'\')">here</a> to complete objective A.'
}
else {
  s = 'You already completed objective A.'
}
document.write(s + '</p><p>');
objectiveID = "myUniqueObjectiveName002";
if (window.parent.SCOGetObjectiveData(objectiveID, "status") != "completed") {
  s = 'Please click <a href="JavaScript:MarkObjectiveCompleted(\'';
  s += objectiveID + '\')">here</a> to complete objective B.'
}
else { s = 'You already completed objective B.' }
document.write('<p>' + s + '</p>')
</script>
</body>
</html>
```

Obr. 10.4 – HTML a JavaScript: Získání a nastavení dat pro více než jeden cíl

```
<html><!-- saved as file "endpage.htm" -->
<head><title>---</title></head>
<body>
<p>Summary of how you did on objectives:</p>
<script type="text/javascript" language="JavaScript">
var s = "";
var stat = "";
// Get the status for one objective, and turn it
// into some text to display
var objectiveID = "myUniqueObjectiveName001";
stat = window.parent.SCOGetObjectiveData(objectiveID, "status");
if (stat != "completed") stat="not completed";
s += '<p>Status for objective A: ' + stat + '</p>';
// Get the status for the other objective
// and append that to the text to display
objectiveID = "myUniqueObjectiveName002";
stat = window.parent.SCOGetObjectiveData(objectiveID, "status");
if (stat != "completed") stat="not completed";
s += '<p>Status for objective B: ' + stat + '</p>';
document.write(s)
</script>
</body>
</html>
```

Obr. 10.5 – HTML a JavaScript: Získání a zobrazení dat pro více než jeden cíl

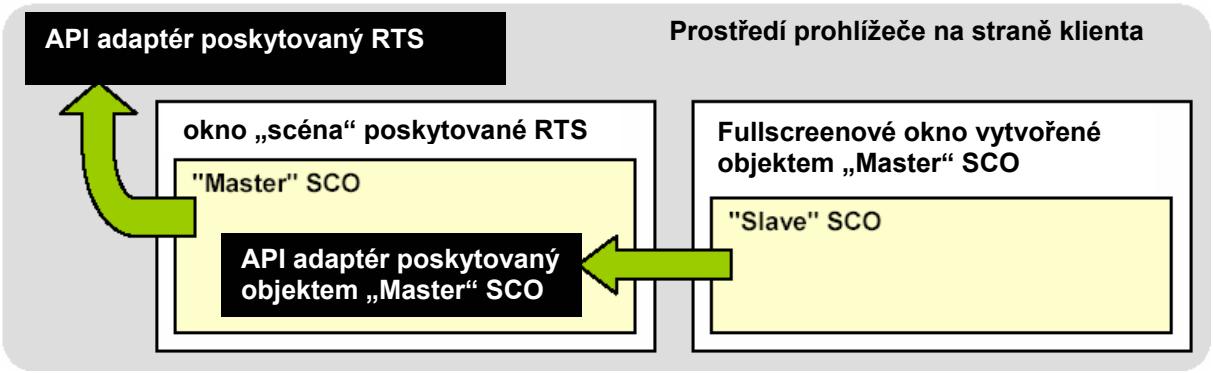
Kapitola 11 – Zobrazení objektu SCO ve fullscreenovém módu

SCO nesmí předpokládat, že poběží v okně nejvyšší úrovně, nebo se pokoušet prosadit do okna nejvyšší úrovně. Musí být navrženo tak, aby bylo kompatibilní při spouštění ve framu. Nicméně existují určité situace, kdy je žádoucí, aby SCO běželo ve fullscreenu. Určité typy obsahu, jako je například hra, mohou poskytovat nejlepší zážitek právě ve fullscreenu.

Protože SCO může být spouštěno runtimovou službou jen ve framu, jediným způsobem, jak poskytnout fullscreenový zážitek, je otevřít fullscreenové okno. Žádná ze specifikací SCORMu nebrání objektu SCO v tom, aby takové závislé popup okno otevřel, ale může to být poněkud riskantní. V každém případě nemůže SCO nechat takové okno otevřené poté, co bylo ukončeno, a to i v případě, že ukončení bylo nestandardní. Zkušenost také ukázala, že uživatelé mohou být velmi zmateni z neúmyslné aktivace dalšího okna.

Tento příklad ukazuje, jak zvládnout tento problém elegantně zvládnout. Jedná se o fullscreenovou verzi příkladu z kapitoly 10 – sledování cílů.

Je mnoho způsobů, jak toho dosáhnout. V tomto případě, vzhledem k tomu, že většina scriptů je již dostupná, je částečným řešením spustit SCO uvnitř SCO, jak ukazuje obr. 11.1.



Obr. 11.1 – Spuštění „slave“ SCO ve fullscreenovém okně

Skutečné SCO, které je spouštěno runtimovou službou – nazývejme ho třeba „master SCO“ – zde hraje úlohu miniruntimové služby pro další objekt SCO – nazvěme jej „slave“ SCO“. Master SCO spouští slave SCO v okně, které vytváří. V okamžiku, kdy je spuštěno slave SCO, začne hledat API adaptér, který je vlastně částí master SCO. Tento API adaptér, jehož instanci vytváří master SCO, zachytává volání slave SCO. Většina těchto volání je jednoduše přeposlána „skutečnému“ API adaptéru, který je poskytován runtimovou službou. Jelikož master SCO provádí inicializaci a končí vlastním způsobem, tak v okamžiku, kdy slave SCO zavolá *LMSInitialize* a *LMSFinish,* fingovaný API adaptér jednoduše vrátí „*true*“, ale nedělá vlastně nic.

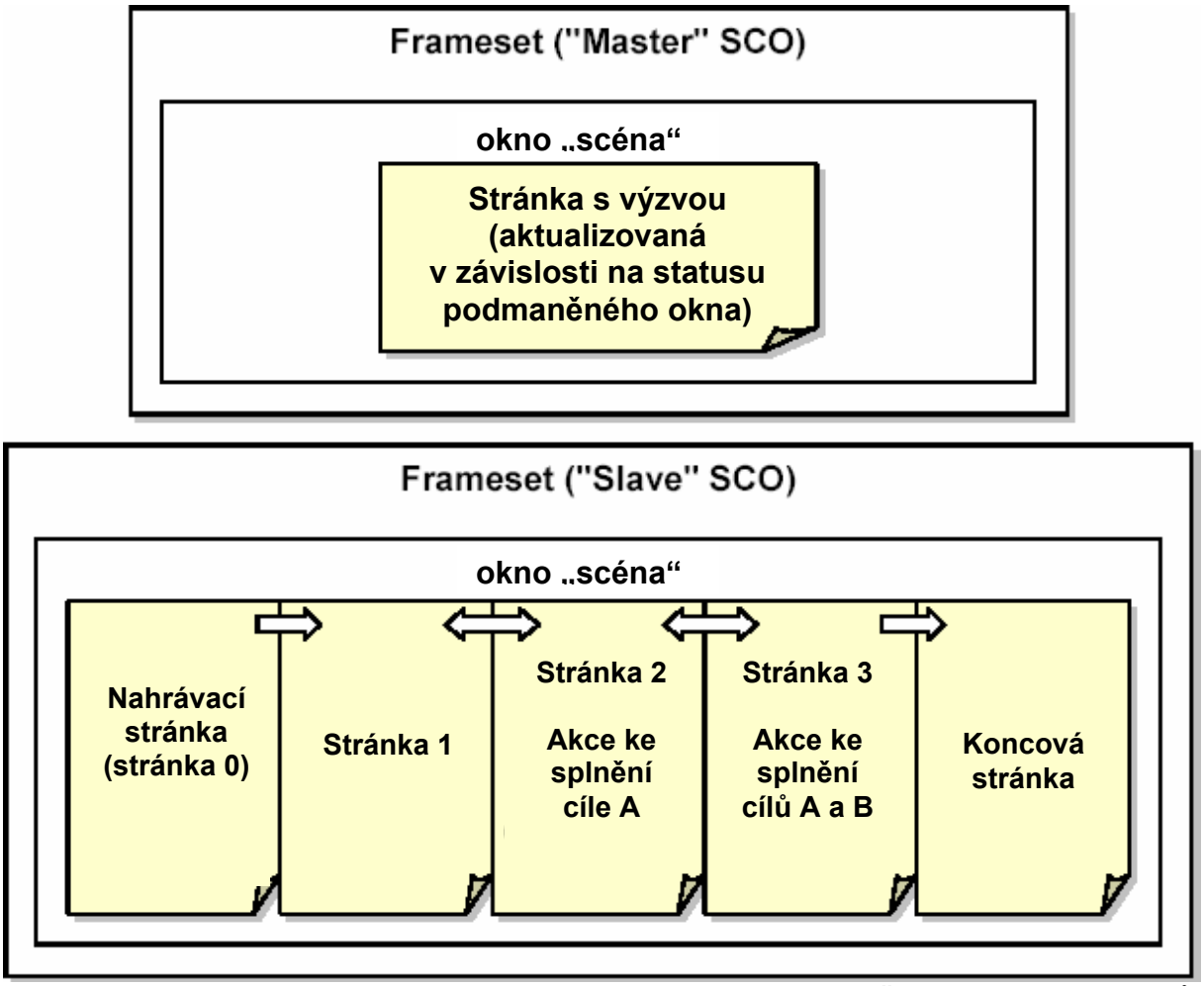
Pravidla SCORMu nedovolují objektu SCO odkazovat na jiné SCO nebo přecházet na jiné SCO v rámci stejného okna. V tomto případě nicméně nebudou pravidla porušena, protože slave SCO nenahrazuje SCO spuštěné runtimovou službou, ale je spuštěno v okně, které je „vlastněno“ tímto SCO. Runtimová služba vlastně vůbec neví, že nějaké slave SCO existuje, protože jediné SCO, které ho vidí běžet, je master SCO. A zároveň slave SCO vidí pouze runtimovou službu provozovanou master SCO a absolutně netuší o další runtimové službě mimo master SCO.

Předpoklady uživatelského rozhraní

Ve chvíli, kdy prohlížeč otvírá fullscreenové okno, neposkytuje žádné prostředky pro jeho uzavření; stránka „scéna“ prohlížeče zaplní celou obrazovku. Proto musíte poskytnout způsob, jak toto fullscreenové okno uzavřít, jinak v něm uživatel „uvízne“. Uvedený příklad proto přidává na konec stránky příkaz „Exit“. Kliknutí na „Exit“ zavírá okno. Uživatel může k jeho uzavření použít také Alt+F4, to ale není příliš elegantní. Propracovanějším příkladem může být použití jiné metody. Všimněte si, že když je master SCO z jakéhokoliv důvodu uvolněno z paměti, jeho script se ujistí, že se fullscreenové okno automaticky uzavřelo. Je zde také problém aktivace okna. Například v prostřední Windows nezabráníte tomu, aby uživatel použil Alt+Tab k aktivaci jiného okna. Také nezabráníte tomu, aby uživatel neuzavřel fullscreenové okno pomocí Alt+F4. Script v tomto příkladu užívá k dohledu nad fullscreenovým oknem funkci vyvolanou rekurzivním časovačem. Tím se zajistí, že se v okně „scéna“ objektu master SCO vždy zobrazí relevantní výzva.

Struktura SCO

Toto SCO používá dva framesety. Hlavní frameset je master SCO a zobrazuje prázdnou stránku, která je dynamicky aktualizována. Druhý frameset a asociované stránky tvoří slave SCO. Druhý frameset bude zobrazen v odděleném fullscreenovém okně, zatímco první frameset je spuštěn v okně „scéna“ runtimové služby.



Obr. 11.2 – Stránky ve SCO, které uchovává informace o sledování určitých individuálních cílů

Hlavní frameset – master SCO – je znázorněn v ukázce 11.1. stránka s výzvou, kterou používá master SCO k informování uživatele je v ukázce 11.2.

Všechny stránky, které tvoří slave SCO jsou zcela identické se stránkami z kapitoly 10 – sledování cílů, a není proto nutné je znovu zařadit. Je zde jedna výjimka: Koncová stránka z ukázky 11.2 je oproti předcházejícímu příkladu modifikována, protože musí obsahovat zařízení uživatelského rozhraní k uzavření fullscreenového okna.

```
<html>
<head>
<script src="SCORMGenericLogic.js" type="text/javascript" language="JavaScript">
</script>

<script type="text/javascript" language="JavaScript">
// If all goes well this local object named API will become the
// SCORM API adapter found and used by the "slave" SCO
var API = null;
function dummyInitialize(){ return "true" }
function dummyFinish(){ return "true"}

// This part of the script ensures that the user knows
// what's going on in 3 different states:
```

```
// (1) while the SCO is creating the full frame popup window
// (2) while the full frame window is displayed, in case the
//     user inadvertently activates another window
// (3) after the full frame window closes.
// It uses a timer to control the display of a prompt in the "main"
// window.

var zPopupStageTimerID = null;
var zbOtherWindowPromptShown = false;

function DisplayPrompt() {
  var s = '<html><head>';
  s += '<style type="text/css">body {background-color: silver; color: blue;';
  s += 'font-size: small;font-family: Verdana, Arial, Helvetica, Sans-Serif}';
  s += '</style></head><body><h3>%s</h3><p>%s</p>';
  s += '</body></html>';
  var re = /%s/
  for (i=0; i < DisplayPrompt.arguments.length; i++){
    s = s.replace(re,DisplayPrompt.arguments[i]);
  }
  frameMyStage.location.href = "dummypage.htm";
  frameMyStage.document.open();
  frameMyStage.document.write(s);
  frameMyStage.document.close()
}

function PopupStageFocus() {
  if ((zwndSlave) && (!zwndSlave.closed)) {
    zwndSlave.focus()
  }
}

function PopupStageFocusTimer() {
  // called by timer while a popup stage is open
  if ((zwndSlave) && (!zwndSlave.closed)){
    if (!zbOtherWindowPromptShown){
        DisplayPrompt('This content was launched in another window.',
          'Click <a href="javaScript:window.parent.PopupStageFocus()">'+
            'here</a> to reactivate it.');
        zbOtherWindowPromptShown = true;
    }
  } else {
    KillStageFocusTimer();
    frameMyStage.location.href = "dummypage.htm"
    if (zbOtherWindowPromptShown){
        DisplayPrompt("Finished",
          "The full screen content window is now closed.")
        zbOtherWindowPromptShown = true;
    }
    zbOtherWindowPromptShown = false;
  }
}

function StartStageFocusTimer() {
  if (zPopupStageTimerID!= null) {
    clearInterval(zPopupStageTimerID)
  }
  zbOtherWindowPromptShown = false;
  zPopupStageTimerID= setInterval("PopupStageFocusTimer()", 500);
}

function KillStageFocusTimer() {
  if (zPopupStageTimerID != null) clearInterval(zPopupStageTimerID);
```

```
}

// The actual work begins here
function init() {
  DisplayPrompt("One moment, please...","Preparing a window...");
  SCOInitialize();
  if (APIOK()){
    API = new Object();
    // We are handling overall SCO initialize and finish in this script
    // Make the methods called by the slave no-ops
    API.LMSInitialize = dummyInitialize;
    API.LMSFinish = dummyFinish;
    // Add the functions instantiated by the generic script to this API adapter
    API.LMSSetValue = zobjAPI.LMSSetValue;
    API.LMSGetValue = zobjAPI.LMSGetValue;
    API.LMSCommit = zobjAPI.LMSCommit;
    API.LMSGetLastError = zobjAPI.LMSGetLastError;
    API.LMSGetErrorString = zobjAPI.LMSGetErrorString;
    API.LMSGetDiagnostic = zobjAPI.LMSGetDiagnostic;
  }
  // Now launch the actual content in full screen window
  var url = "frameset.htm" ; // Note: could be read from a parameter.
  // Open window with a blank page because it may take a long
  // time to load the actual content; psychologically it is
  // better to see the window come up as fast as possible.
  zwndSlave = window.open("dummypage.htm","SCOFullScreenStage","fullscreen=yes");
  zwndSlave.location.href = url;
  zwndSlave.focus();
  StartStageFocusTimer()
}

function SCOSaveData(){
  KillStageFocusTimer();
  // Make sure that the slave window is not left behind
  if ((zwndSlave) && (!zwndSlave.closed)) {
    zwndSlave.close()
  }
}

</script>

<title>Sample Full Screen SCO</title>
</head>
<frameset rows="100%,*" border="0"
      onload="init()"
      onunload="SCOFinish()"
      onbeforeunload="SCOFinish()">
>
<frame name="frameMyStage" title="Explanation" src="loadpromptpage.htm" />
<frame src="dummypage.htm" />
</frameset>
</html>
```

Obr. 11.1 – master SCO

```
<html><!-- saved as file "loadpromptpage.htm" -->
<head>
<style type="text/css">
body {background-color: silver;
 color: blue;
 font-size: small;
 font-family: Verdana, Arial, Helvetica, Sans-Serif}
</style>
<title></title>
</head>
<body>
<h3>One moment, please...</h3>
<p>Preparing a window...</p>
</body>
</html>
```

Obr. 11.2 – Iniciační stránka s výzvou zobrazovaná v průběhu otvírání fullscreenového okna

```
<html><!-- saved as file "endpage.htm" -->
<head><title>---</title></head>
<body>
<script type="text/javascript" language="JavaScript">
var s = "<p>Status: ";
var stat = "";
// Get the status for the SCO
stat = window.parent.SCOGetValue("cmi.core.lesson_status");
switch(stat) {
  case "": s += window.parent.SCOGetDiagnostics("");break;
  case "completed": s += "completed (you visited every navigable page)";break;
  default: s += stat
}
s += "</p><p>Summary of how you did on objectives:</p>"
// Get the status for one objective, and turn it
// into some text to display
var objectiveID = "myUniqueObjectiveName001";
stat = window.parent.SCOGetObjectiveData(objectiveID, "status");
if (stat != "completed") stat="not completed";
s += '<p>Status for objective A: ' + stat + '</p>';
// Get the status for the other objective
// and append that to the text to display
objectiveID = "myUniqueObjectiveName002";
stat = window.parent.SCOGetObjectiveData(objectiveID, "status");
if (stat != "completed") stat="not completed";
s += '<p>Status for objective B: ' + stat + '</p>';
document.write(s)
</script>
<p><a href="JavaScript:window.parent.close()">Exit</a></p>
</body>
</html>
```

Obr. 11.3 – Koncová stránka se zařízením uživatelského rozhraní k zavření fullscreenového okna

Kapitola 12 – Balení SCORM konformního obsahu

Když už jsme nějaká SCO vytvořili, bylo by hezké vidět je v akci v prostředí pro distribuci. SCORM specifikuje, jak zabalit Vaše objekty SCO tak, aby mohly být agregovány, skladovány, kopírovány, přemisťovány, archivovány, nahrávány a případně distribuovány k uživateli prostřednictvím jakéhokoliv SCORM konformního management systému. Balíček může obsahovat jedno nebo více SCO.

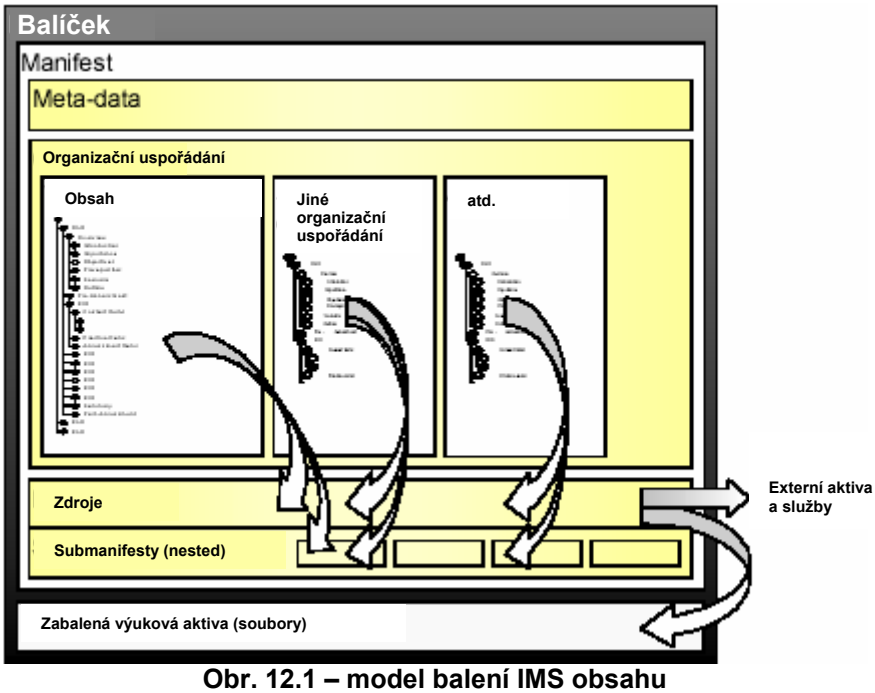
Organizační model IMS & SCORM obsahu

IMS Global Learning Consorcium (www.imsglobal.org) vyvinulo specifikace balení pro výukový obsah, které poskytují užitečný plán organizace obecného obsahu ve formě manifestu zahrnutého do balíčku. Manifest se používá ke katalogizaci obsahu balíčku, ale také k jeho popisu prostřednictvím metadat. Manifest může být také užitečný pro představu o tom, jak je obsah organizován. Balení obsahu SCORM 1.2 je založeno na specifikacích IMS.

Manifest IMS balení je XML dokument, který obsahuje několik částí.

- Metadata popisující balíček
- Uspořádání: žádná, jedna nebo více strukturálních map popisujících způsob, jakým je obsah organizován. Každá položka takové mapy může odkazovat na zdroj v balíčku. Manifest SCORM obsahu pro distribuci ke koncovému uživateli musí obsahovat alespoň jedno uspořádání.
- zdroje, které specifikují skutečné kusy obsahu, jež mohou být použity. Na stejný zdroj může odkazovat více než jedna položka organizačního uspořádání. Zdroj také může mít vlastní metadata. Typicky je SCO popsáno prostřednictvím zdroje.

- Sub-manifesty (sub-manifests, nested manifests), které popisují podskupiny obsahu v balíčku. Sub-manifest může mít vlastní metadata, organizační uspořádání a zdroje.



Obr. 12.1 – model balení IMS obsahu

Například pokud byste chtěli zobrazit organizační uspořádání jako obsah kurzu, výběrem položky z obsahu můžete spustit nebo otevřít odpovídající zdroj.

Místo na zdroje jádra může položka v organizačním uspořádání odkazovat na sub-manifest a tudíž na zbytek celého organizačního uspořádání. Může to být velice užitečné, pokud jsou například stejné uspořádané kusy obsahu použity na více místech kurzu. Tím se také stává balení organizačních uspořádání ve formě orientovaných grafů vhodnější. Protože jsou soběstačné a mohou nést metadata, mohou být sub-manifesty také využity pro identifikaci částí balíčku, které lze rozbalit a opětovně použít v jiném kontextu.

Tonto model balení je základem pro strukturu obsahu a organizačního uspořádání SCORMu 1.2. SCORM 1.2 rozšiřuje definici specifikací dodatečných datových elementů. V XML dokumentu jsou tato rozšíření identifikována předponou *adlcp:* ve jméně.

Kam vložit metadata

SCORM vyžaduje pro balíček metadata. Vkládají se na nejvyšší úroveň manifestu. SCORM také navíc umožňuje odkázat na separátní soubor s metadaty zahrnutý v balíčku. Abyste vyhověli standardu, můžete buď zahrnou metadata přímo v manifestu, nebo použít ve SCORMu definované rozšíření určené k odkazování na externí soubor s metadaty. Nelze učinit obojí zároveň, tzn. nemůžete mít metadata v řádku manifestu a odkaz na soubor metadat. SCORM také povoluje doplňková metadata, jak je to definováno schématem IMS.

Vytvoření balíčku SCORM

Vytvořte XML *manifest.* Je to XML soubor s hlavičkou specifikovanou ve SCORMu 1.2.

- Přidejte sekci *resources* (zdroje) a popište každý výukový objekt (SCO) jako element *resource* v sekci zdrojů.
- Neodkazujete-li na externí zdroje, identifikujte všechny soubory, které jsou potřeba pro každý výukový objekt jako element *file* v každém *resource*.
- Pokud několik výukových objektů užívá stejné soubory (například nějakou grafiku), než byste několikrát opakovali seznam souborů, zvažte použití elementu *dependency* a zajistěte, aby tento element odkazoval na společný *resource*, který představuje soubor položek.
- Přidejte sekci *organizations* nad sekci *resources.*
- Vytvořte jeden nebo více stromů organizační struktury, obsahující elementy *item,* které odkazují na zdroje.
- Nad sekci organizačních uspořádání přidejte sekci *metadata.*
- Přidejte elementy metadat, jak je specifikováno ve SCORMu 1.2 k popisu balíčku a jeho obsahu.

Vytvořte soubor zip, který obsahuje manifest, soubory kontroly XML (XSD, ap.) a všechny další soubory zahrnuté v balíčku. Můžete použít podadresáře, ale manifest musí být v základním adresáři adresářového stromu. Pro další instrukce viz příručka Nejlepší praktiky balení IMS obsahu. SCORM konfomita a interoperabilita vyžaduje, abyste zahrnuli soubory XSD, které jsou specifikovány ve SCORMu 1.2. Nejsou to však nejaktuálnější XSD zaslané na web stránku IMS.

Cesty k obsahu a adresářům

Když už je SCORM obsah, který jste vytvořili, instalován v LMS nebo v úložišti, pravděpodobně neskončí ve stejném adresáři, ve kterém byl při vývoji, nebo s ním nebudou spojena stejná přístupová práva. Proto musíte v uspořádání interních odkazů svého webového obsahu postupovat podle následujících pravidel:

- všechny použité soubory, na které je odkazováno jakýmkoliv webovým obsahem v balíčku, musí být v adresáři, který je buď základním adresářem balíčku, nebo potomkem takového adresáře.
- všechny odkazy musí být relativní. Nesmíte použít pevný formát cesty ať už ke kořenovému adresáři disku, nebo ke specifické diskové jednotce či svazku. Například hodnoty atributů *href* a *src* v HTML souborech nesmí začínat „/“, nebo specifikovat relativní cestu, která by mohla vést zpět hlouběji, než k základnímu adresáři balíčku.

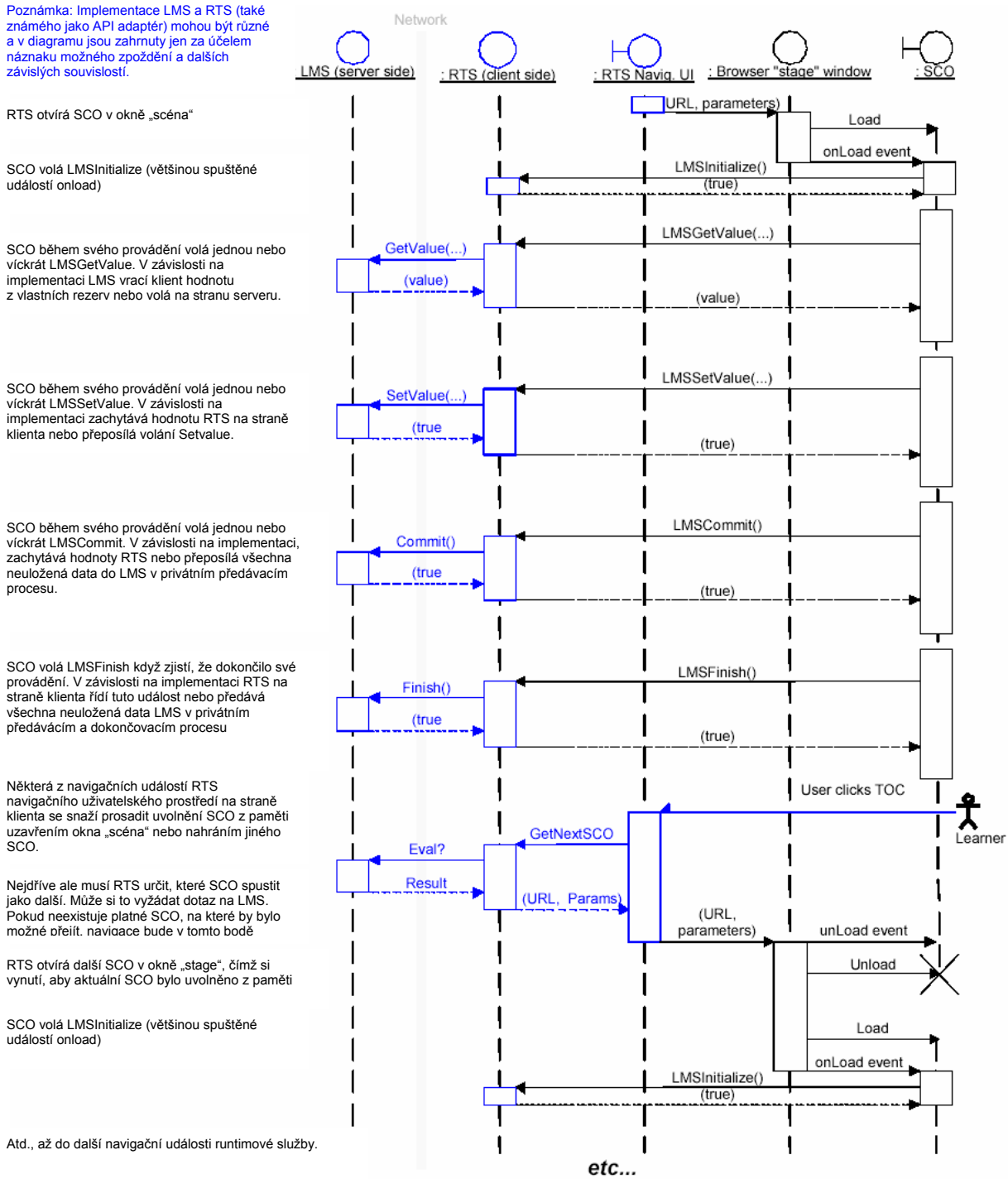
SCORM 1.2 definuje element matadat Location jako povinný. Nicméně jelikož balíček může být přenášen ze systému do systému, jediná hodnota, která pro takový element dává smysl, je „.“, což znamená „kdekoliv se tato metadata ocitnou“.

Uspořádání objektů SCO

Můžete asociovat SCO s jakoukoliv položkou ve stromové struktuře organizačního uspořádání. Nicméně si uvědomte, že jednoduchý sekvenční model pro SCORM 1.3 pravděpodobně dovolí objekty SCO pouze pro uzly listů ve stromu.

Dodatek: Různorodé zdroje a poznámky k implementaci

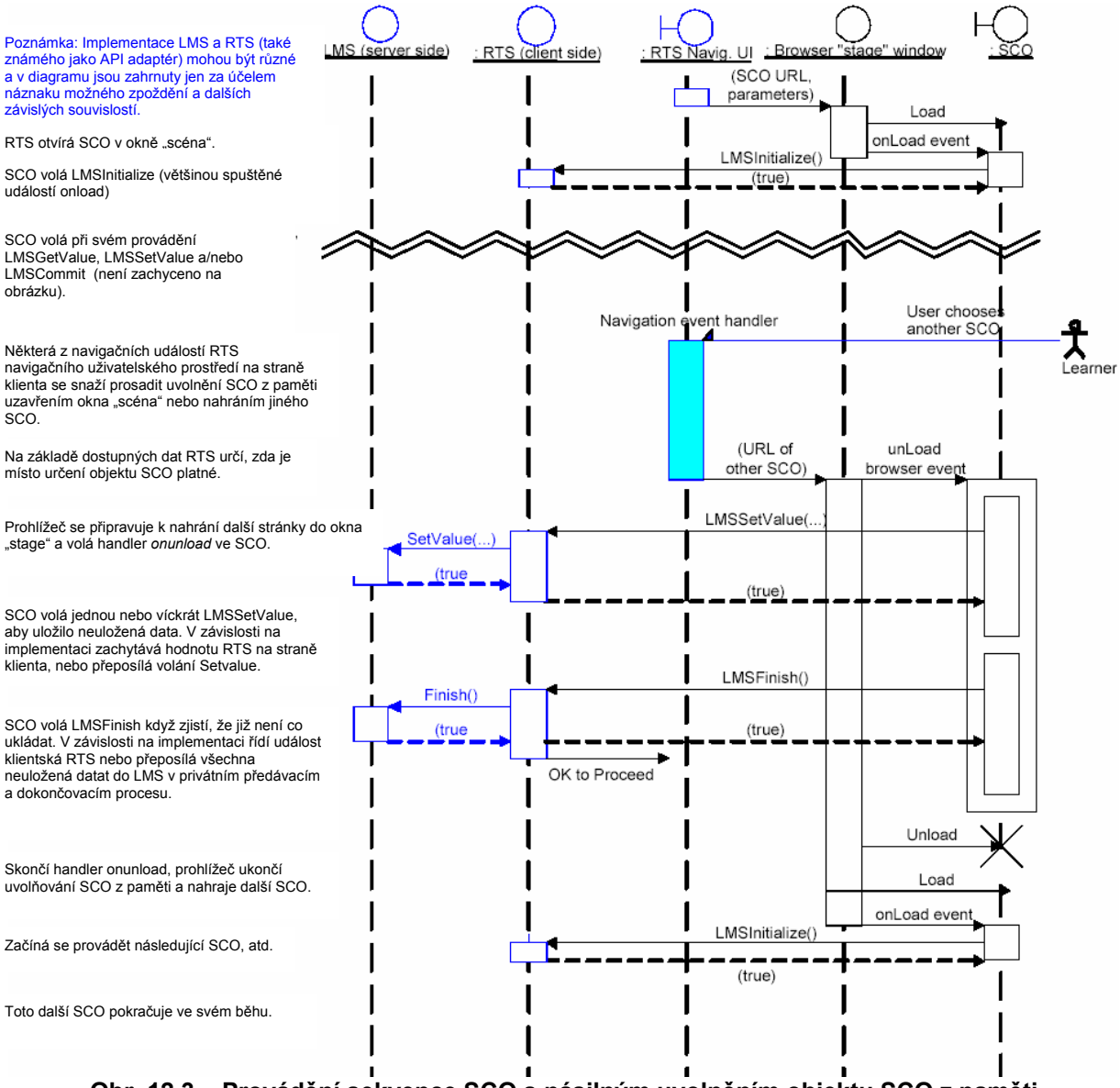
Sekvenční diagram



Obr. 12.2 – Provádění sekvence SCO bez nečekaných událostí

Obrázek 12.2 užívá nástin UML k vyobrazení životního cyklu objektu SCO spuštěného runtimovou službou. Tento příklad není možná zrovna obvyklý. Předpokládá se v něm, že SCO skončí normálně (ať už si pod tímto pojmem autor SCO představuje cokoli) a hlásí všechna data LMS ještě před tím, než uživatel pomyslí na to, že by se přesunul na další SCO. Ve skutečnosti může být SCO

kdykoliv násilně uvolněno z paměti, jak je tomu na obrázku 12.3. Pozn.: RTS znamená „runtimová služba“ (runtime service).



Obr. 12.3 – Provádění sekvence SCO s násilným uvolněním objektu SCO z paměti

Nejpravděpodobnější scénář je ten, že prohlížeč rychle uvolňuje SCO z paměti kdykoliv se objeví nějaká událost prohlížeče, nad níž SCO nemá kontrolu. Stává se to například, když si uživatel vybere jiné SCO. Diagram ukazuje, jak by mohl takový scénář vypadat při implementaci. Všimněte si, že prohlížeč už zpracovává URL dalšího SCO – a kdo může tušit co dál – přesto, že SCO je informováno o tom, že je uvolňováno z paměti. Proto je rozumnou myšlenkou ukládat veškerá kritická data dokud se SCO provádí, než čekat na událost unload.

Ukázka manifestu balíčku SCORMu 1.2

XML manifest jednoduchého SCO

Tento manifest může být použit společně se soubory, které utvářejí vícestránkové SCO z výše uvedeného příkladu. Vše, co je potřebné k dokončení balíčku, je sada XML souborů se schématy, na které jsou odkazy v xsi:schemaLocation, a máte kompletní balíček SCORM 1.2. Všimněte si, že každý soubor, který SCO využívá,

musí být uveden v manifestu. SCORM test suite tuto skutečnost momentálně neověřuje – jediné, co ověřuje je skutečnost, zda soubory, které jsou v manifestu uvedeny, skutečně existují. Vhodným postupem je vytvoření kompletního inventáře souborů, požadovaných v manifestu. Také si všimněte, že SCORM 1.2 vyžaduje mnoho datových elementů, které jsou ve specifikacích balení obsahu IMS (IMS Content Packaging) a metadat volitelné. Tento manifest můžete opakovaně použít k balení jakéhokoliv jednoduchého SCO, jako např. jednostránkového SCO a grafiky, kterou toto SCO využívá, nebo třeba pro jednostránkový SCO wrapper, ve kterém může být vložený soubor Flashe. Části manifestu, které nahradíte vlastními daty, jsou vyznačeny tučně. Všimněte si, že element <location> je v metadatech SCORMu vyžadován, nicméně vždy by měl být „.“.

```
<?xml version="1.0"?>
<manifest identifier="SampleContentManifest" version="1.2"
 xmlns="http://www.imsproject.org/xsd/imscp_rootv1p1p2"
 xmlns:adlcp="http://www.adlnet.org/xsd/adlcp_rootv1p2"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.imsproject.org/xsd/imscp rootv1p1p2
 imscp_rootv1p1p2.xsd
 http://www.imsglobal.org/xsd/imsmd rootv1p2p1 imsmd rootv1p2p1.xsd
 http://www.adlnet.org/xsd/adlcp rootv1p2 adlcp rootv1p2.xsd">
  <metadata>
    <schema>ADL SCORM</schema>
    <schemaversion>1.2</schemaversion>
    <lom xmlns="http://www.imsglobal.org/xsd/imsmd_rootv1p2p1">
      <general>
        <catalogentry>
          <catalog>Your favorite Catalog</catalog>
          <entry>
            <langstring>Samples-SCO12-A</langstring>
          </entry>
        </catalogentry>
        <title>
          <langstring>Sample Multiple page SCO</langstring>
        </title>
        <description>
          <langstring>This multi-page SCO implements the following features:
(a) You can navigate back and forth among the SCO pages without leaving the SCO.
(b) Pages do not contain complex scripts.
(c) Can be forcibly unloaded while on any page.
(d) Illustrates how a page can set the SCO status to "completed" when it is
reached.</langstring>
        </description>
        <keyword>
          <langstring>Sample</langstring>
        </keyword>
        <keyword>
          <langstring>SCO</langstring>
        </keyword>
      </general>
      <lifecycle>
        <version>
          <langstring>1.0</langstring>
        </version>
```

```
        <status>
          <source>
            <langstring xml:lang="x-none">LOMv1.0</langstring>
          </source>
          <value>
            <langstring xml:lang="x-none">final</langstring>
          </value>
        </status>
        <contribute>
          <role>
            <source>
              <langstring xml:lang="x-none">LOMv1.0</langstring>
            </source>
            <value>
              <langstring xml:lang="x-none">author</langstring>
            </value>
          </role>
          <centity>
            <vcard>
BEGIN:vCard
FN:Claude Ostyn
ORG:Click2learn, Inc.
END:vCard
            </vcard>
          </centity>
          <date>
            <datetime>2002-02-26</datetime>
          </date>
        </contribute>
      </lifecycle>
      <metametadata>
        <metadatascheme>ADL SCORM 1.2</metadatascheme>
        <language>en-US</language>
      </metametadata>
      <technical>
        <format>text/html</format>
        <location type="URI">.</location>
        <requirement>
          <type>
            <source>
              <langstring xml:lang="x-none">LOMv1.0</langstring>
            </source>
            <value>
              <langstring xml:lang="x-none">Browser</langstring>
            </value>
          </type>
          <name>
            <source>
              <langstring xml:lang="x-none">LOMv1.0</langstring>
            </source>
            <value>
              <langstring xml:lang="x-none">Microsoft Internet Explorer</langstring>
            </value>
          </name>
          <minimumversion>4.01</minimumversion>
        </requirement>
      </technical>
      <rights>
        <cost>
          <source>
            <langstring xml:lang="x-none">LOMv1.0</langstring>
          </source>
          <value>
```

```
            <langstring xml:lang="x-none">No</langstring>
          </value>
        </cost>
        <copyrightandotherrestrictions>
          <source>
            <langstring xml:lang="x-none">LOMv1.0</langstring>
          </source>
          <value>
            <langstring xml:lang="x-none">Yes</langstring>
          </value>
        </copyrightandotherrestrictions>
        <description>
          <langstring>Can be used freely but author retains copyright.</langstring>
        </description>
      </rights>
      <classification>
        <purpose>
          <source>
            <langstring xml:lang="x-none">LOMv1.0</langstring>
          </source>
          <value>
            <langstring xml:lang="x-none">Educational Objective</langstring>
          </value>
        </purpose>
        <description>
          <langstring>Recognize a SCO Packaging Manifest</langstring>
        </description>
        <keyword>
          <langstring>SCO</langstring>
        </keyword>
        <keyword>
          <langstring>Package manifest</langstring>
        </keyword>
      </classification>
    </lom>
  </metadata>
  <organizations default="One">
    <organization identifier="One">
    <title>Sample Multiple page SCO</title>
      <item identifier="item1" identifierref="SCO1" isvisible="true">
        <title>Sample Multiple page SCO</title>
      </item>
    </organization>
  </organizations>
  <resources>
    <resource identifier="SCO1" type="webcontent"
      adlcp:SCORMtype="SCO" href="multipage_SCO_sample.htm">
      <file href="multipage_SCO_sample.htm"/>
      <file href="dummypage.htm"/>
      <file href="page1.htm"/>
      <file href="page2.htm"/>
      <file href="page3.htm"/>
      <file href="SCORM1_2Generic.js"/>
    </resource>
  </resources>
</manifest>
```

O autorovi

Claude Ostyn byl po několik let spojen s Výukovými technologickými standardy jako přispěvatel a člen různých pracovních skupin Komise výukových technologických standardů Institutu pro elektrické a elektronické inženýrství a konsorcia IMS Global Learning. Byl také členem týmu technických konzultantů, který napomáhal při vývoji specifikací SCORMu.

Byl u Click2learn od těch raných dob, kdy se společnost ještě nazývala Asymetrix. Řídil vývoj a implementaci toho, co bylo novým druhem podpůrného hypertextového systému s nejmodernějšími on-line tutoriály. Později dodal do ToolBooku takové ty multimediální záležitosti a naprojektoval CBT vydání ToolBooku, první e-learningový vývojový nástroj s užitím průvodců, šablon a inteligentních, kontextově senzitivních objektů namísto kódování a vývojových diagramů, který se později vyvinul v Click2learn Instructor.

Jeho zázemí zahrnuje i filmové vzdělání, dlouhodobé pracovní pobyty v odlehlých aljašských destinacích, kde působil jako odborník na školách v etnografické, instruktážní i komerční videoprodukci, v individuálním výcviku dospělých v oblastech sahajících od výroby filmů až k tabulkovým procesorům, titul ze Stanfordu a příležitostné vaření dle vynikajícího receptu na čokoládovou pěnu, jenž začíná slovy „vezměte nějakou dobrou, tmavou belgickou čokoládu...“