CODEN:LUTEDX/(TEIE-3020)/1-47/(2014)

Electrical Engineering and Automation

Industrial

Implementation av SCORM 2004 API och Datamodell i .NET

Kristian Månsson

Division of Industrial Electrical Engineering and Automation
Faculty of Engineering, Lund University

Implementation av SCORM 2004 API och Datamodell i .NET

LUNDS
UNIVERSITET
Lunds Tekniska Högskola

LTH Ingenjörshögskolan vid Campus Helsingborg
Datateknik

Examensarbete:
Kristian Månsson

LTH Ingenjörshögskolan vid Campus Helsingborg
Lunds universitet
Box 882
251 08 Helsingborg

LTH School of Engineering
Lund University
Box 882
SE-251 08 Helsingborg
Sweden

Tryckt i Sverige
Media-Tryck
Biblioteksdirektionen
Lunds universitet
Lund 2014

Sammanfattning

Examensarbetet behandlar en partiell implementation av SCORM-standarden för en existerande LMS, och innefattar förutom en presentation av implementationen också en genomgång av standarden.
SCORM står för *Shareable Content Object Reference Model* och är en teknisk standard för webbaserat e-lärande. Standarden behandlar sådant som API, förpackning av kursmaterial och datamodell.
Implementationen var inte fullständig, då tidsramarna ej tillät detta. Det visade sig dock att det var fullt möjligt att implementera delar av standarden. T.ex. tillåter standarden navigering mellan olika kursmoduler, kallade SCO:s. Det går dock att begränsa användbarheten till att endast innefatta kurspaket med en kursmodul.

Nyckelord: ASP.NET MVC, webbprogrammering, SCORM

Abstract

The thesis treats a partial implementation of the SCORM standard for an existing LMS (*Learning Management System)*, and includes an overview of the standard and a presentation and analysis of the implementation.
SCORM is short for *Shareable Content Object Reference Model* and is a standard for web based e-learning. The standard specifies, among other things, API, aggregation and data model.
The implementation is not fully compliant, as the time-frames of the thesis did not allow for such an endeavour. However, a partial implementation turned out to be possible. For example does the standard impose functionality for navigation between different course modules, called SCO:s. It´s however possible to restrict the functionality to only allow for packages with single SCO:s.

Keywords: ASP.NET MVC, web programming, SCORM

Förord

Jag vill rikta ett speciellt tack till Fredrik Ovesson, Markus Wallén och Ulf Wennström på P&L Nordic AB som stöttat och handlett mig genom den praktiska delen av arbetet.

Innehållsförteckning

1 Inledning

1.1 Bakgrund

Webbaserat e-lärande har växt som fenomen de senaste åren Anledningarna är flera, det viktigaste är att webbtekniker har utvecklas de senaste åren och att fler och fler får tillgång till Internet. I själva verket har de flesta i Sverige tillgång till webben i någon form. E-lärande har fördelen att det går att enkelt sköta lärande på distans och att förmedla interaktivt läromaterial såsom filmer, animationer och liknande[6][7].

I takt med att tekniker sprids och blir allmänt tillgängliga så växer också behovet av standarder. Standarder kan vara av olika karaktär, och i kontexten av webbaserat e-lärande kan det tänkas finnas behov av både tekniska och pedagogiska standarder. En av de vanligaste standarderna (som egentligen en samling av standarder) för webbaserat e-lärande är *Shareable Content Object Reference Model* (SCORM)som är utvecklad Advanced Learning Initiative (ADL), till en början på uppdrag från det amerikanska försvarsdepartementet. Standarden är teknisk och beskriver hur kursmaterial ska kunna presenteras för en kursdeltagare och hur uppföljning, resultathantering etc. ska skötas med avseende på dataformat, datamodell osv. Standarden beskriver också hur kursmaterial ska förpackas och hur uppdelningar i olika kursmoduler- och moment kan köras i sekvens, kommunikation mellan lärandeplattform (LMS) och kursmoduler, hur testpoäng kan sparas och se ut osv. [7][14]

SCORM har gått igenom en utveckling där version 1.2 är den första släppta, 2004 (i två upplagor, 3:e och 4:e) som är den som kommer att behandlas i den här rapporten samt *Tin Can* som är en ny standard som nyligen har utvecklats och släppts i sin första version, och som utgör en vidareutveckling av SCORM.[15]

Det finns således en självklar önskan hos många företag som utvecklar kursmaterial och/eller LMS:er att implementera standarden, då detta möjliggör körning av kurserna i andra system än de egna utvecklade eller körning av alla kurser som är författade enligt standarden i de egenutvecklade LMS:erna. Ett sådant företag är P&L i Hässleholm som har kompetensutveckling som främsta verksamhetsområde. En del i detta är utvecklingen av ett webbaserat kompetenshanteringsverktyg *Competence Tool 3* (CT3) som tillhandahåller viss läroplattformsfunktionalitet.

Denna rapport syftar till att beskriva standarden ur ett tekniskt perspektiv och att beskriva en partiell implementation av standarden i CT3.

1.2 Målformulering

Målet för examensarbetet är att undersöka SCORM-standarden och sammanfatta dess tekniska aspekter i denna rapport, samt att implementera ett API och datamodell i .NET-ramverket och Javascript, anpassa detta till en kommersiell produkt efter önskemål från kund, och analysera implementationen.

1.3 Avgränsningar

Examensarbetet kommer endast att behandla SCORM-standardens tekniska aspekter, såsom språk, datamodell och implementationskrav. Standardens pedagogiska, ekonomiska eller andra aspekter kommer inte att avhandlas. Examensarbetet är också begränsat till att behandla SCORM 2004; inga tidigare eller senare versioner kommer att undersökas. Däremot kommer de att presenteras kortfattat då detta är nödvändigt för förståelsen för standardens tekniska aspekter och historiska sammanhang.

Vidare avgränsningar har att göra med att utvecklingsprojektet ingick i ett projekt för anpassning av annan mjukvara för att kunna köra vissa kurser författade enligt standarden (mer under *Metoder för utveckling* och *Genomförandet*). Dessa begränsningar innebar att delar av SCORM-standardens krav för certifiering (mer under *SCORM-standarden*) inte kunde uppfyllas.

1.4 Målgrupp

1.4.1 Studenter inom dataingenjörsvetenskap, datavetenskap och liknande

Många studenter som studerar datavetenskap eller närliggande kommer att stöta på webbprogrammering och standarder under sin utbildning. Denna rapport kan kunna användas som ett exempel på hur lösningar som innefattar standarder och webbprogammering kan hanteras utifrån ett ingenjörsmässigt perspektiv. Rapporten kan också användas som en exempelsamling till ramverket, då författaren inte besatt några kunskaper om ramverket innan arbetet började.

1.4.2 Utvecklare av SCORM-kompatibla LMS:er

Då rapporten innehåller en genomgång av SCORM-standarden likväl som ett exempel på hur delar av standarden kan implementeras, så kan enskilda utvecklare eller företag använda rapporten som en introduktion och inspiration till hur en SCORM-lösning skulle kunna se ut. Rapporten kan också ge en fingervisning till hur mycket tid som kan krävas för att utveckla ett fullständigt SCORM-stöd. Vidare finns också en samling referenser som kan stödja utvecklare.

1.4.3 Utbildande organisationer

Då SCORM-standarden är en standard för webbaserat e-lärande, är det självklart olika utbildningsorganisationer som kommer att använda sig av implementationer av standarden. Rapporten kan tjäna som en teknisk introduktion till standarden för intresserade inom sådana organisationer.

1.5 Metoder och litteraturöversikt

1.5.1 Litteraturgenomgång

För att hitta litteratur om SCORM-standarden så gjordes en litteratursökning (sökord och sökplatser finns i Appendix 1). Det visade sig dock att det finns mycket lite artiklar publicerat som behandlar de tekniska aspekterna av SCORM 2004.

Den huvudsakliga litteraturkällan är den officiella dokumentationen för standarden. Dokumentationen består av tre dokument som beskriver olika delar av standarden*: Content Aggregation Model*, *Run-Time Environment* och *Sequencing andNavigation*. Dessa dokument återkommer som huvudsakliga källor i rapporten och kommer således inte att beskrivas närmare här. ADL, som utvecklat standarden, har också förutom dokumentationen också get ut en officiell guide för utvecklare. Denna kommer också att användas, framförallt i den övergripande beskrivningen av standarden. Vidare används officiella hemsidor som källor för övergripande beskrivning av produkter och projekt, vilka kan anses pålitliga då det inte är någon kritisk analys som eftersöktes, utan endast övergripande tekniska aspekter.

Det finns också ett antal uppsatser och examensarbete från datavetenskapliga och –tekniska institutioner som behandlar ämnet. För denna rapport intressanta och relevanta resultat och metoder från dessa kommer att tas upp huvudsakligen under detta kapitel, men också under analysdelen.
Slutligen kommer ett mindre antal läroböcker utgivna av erkända förlag användas som källa för vissa övergripande beskrivningar.

1.5.2 Metod för utveckling

Då mjukvaruutveckling till stor del fortfarande får ses som ett hantverk [29], så får utvecklingsprocessen motiveras av de omständigheter som rådde vid utvecklingens gång.

Utvecklingen skedde som i ett projekt för att kunna köra kurser författade i programmet Lectora och Adobe Captivate i programmet Competence Tool 3. Målsättningen var således inte att göra en fullständig SCORM-certifiering, och då utvecklingen av en fullständig SCORM-kompatibel LMS kan beräknas ta ca 2 manår [18] så var detta mål självklart fullständigt orimligt att uppnå under den begränsade tid som examensarbetet pågick.

CT3 är, som tidigare nämnts, ett webbaserat kompetenshanteringssystem utvecklat av P&L i Hässleholm. CT3 är utvecklat i .NET med MVC-ramverket och användargränssnittet är webbaserat och således uppbyggt med HTML, CSS, Javascript och JQuery. Då kraven endast bestod i att kunna köra de i enlighet med SCORM-standarden författade kurserna samt att spara resultaten av dessa beslutades det att utveckla stödet relativt fristående. I korthet så skedde utvecklingen genom att utveckla ett javascript för att starta och kommunicera med kursen. Kursen, och scriptet som ska kommunicera med kursen ges till användaren genom att användaren blir dirigerad till en sida där kursen körs.

1.5.2.1 Krav och kravhantering

Mjukvarukrav [17] består av en systematisk nebrytning av önskemål som intressenter som de vill att mjukvara ska kunna lösa. Alla kravprocesser består kortfattat av följande delar, dock nödvändigtvis inte i den här ordningen:

- Elicitering
- Analys
- Specifikation
- Validering
- Hantering

Med elicitering menas att krav på något sätt tas fram. Detta kan göras på många sätt: intervjuer, läsning av dokumentation, workshops osv. Eliciteringen handlar om att på kunna få fram vilka önskemål intressenterna faktiskt har. Analysprocessen innebär att bryta ner kraven och att förbereda dem för specifikation och dokumentation. Frågor såsom ”Vad är det systemet ska utföra?” och ”Varför är systemet nödvändigt?” kan t.ex. ställas [20], och eventuella motsägelser mellan kraven försöks hittas. Under specifikationen bryts kraven, önskemål och annan data från de tidigare processerna ned och de faktiska kraven formuleras. Det finns en mängd olika stilar på vilka det är möjligt att presentera krav på (User Stories, Use Cases och dataflöden för att nämna några)[24]. Abstraktionsnivå och val av stil måste vägas mot vad intressenternas och projektets behov och önskemål.

En kravhanteringsprocess är central för all mjukvaruutveckling av flera anledningar bl.a.

- Minskar tiden som krävs för utveckling eftersom det krävs mindre tid att ändra på ett krav än på en färdig implementation
- Underlättar verifikation och acceptans av färdig produkt
- Reducerar riskerna för alla intressenter

Med en kravhanteringsprocess menas en fastställd process för arbete med elicitering, analys, specifikation, validering och hantering av krav. [24]

1.5.3 SCORM-standarden

För att undersöka standarden så gjordes en genomgång av hela dokumentationen för standarden samt en läsning på den officiella hemsidan. Vidare söktes också efter examensarbeten som behandlar ämnet ifrån olika perspektiv. De tekniska aspekterna tas upp under SCORM-standarden under *2.1Teknisk bakgrund*, här behandlas endast bakgrunden till standarden.

1.5.3.1 Bakgrund

ADL (*Advanced Distributed Learning Initiative*) är ett företag som startades som ett initiativ av det amerikanska försvaret för att skapa standarder för e-läranade, alltså undervisning som är helt eller delvis sker genom interaktion med dator eller liknande[6][9]. För att distribuera och administrera kurser krävs en plattform kallad en LMS (Learning Management System). En vanlig översättning av LMS är *läroplattform*och dessa två begrepp kommer att användas synonymt i rapporten. En LMS har till syfte att tillhandahålla testning, registrering, schemaläggning, kommunikation, uppföljning och publicering av kursmaterial för e-lärande.

1.5.3.2 Standarden

Utvecklingen av standarden skedde mot bakgrunden av att det fanns en önskan om att ha ett samlat sätt att kunna leverera kursmaterial och kunna återanvända material samt att kunna ha ett fastställt kommunikationssätt mellan LMS och kurser. Detta arbete slutade i SCORM-standarden, som nu finns i 2 versioner: 1.2 och 2004. De huvudsakliga lösningarna som togs fram innebär ett standardiserat sätt att paketera kursmaterial samt ett API för kommunikation. Kommunikationen sker mellan s.k. SCO:er (*Shareable Content Objects)* (som ibland kommer att kallas *kursmoduler*) som består av olika typer av läromaterial som är samlat i en enhet som anropar LMS:en via API:et. Det framtogs också en standard för att organisera flera kursmoduler och kurser i olika hierarkier och sekvenser. [14]

Version 2004 finns i två upplagor. Skillnaden mellan dessa är inte så stor och ligger huvudsakligen i behandlingen av sekvenser av kursmoduler och navigering mellan dessa. Kortfattat kan mer data sparas globalt i 4:e upplagan och möjliggör delning mer data mellan olika kurser.

1.5.3.3 Framtid - Tin Can-projektet

2013 släpptes en ny standard som är en vidareutveckling av standarden. Projektet heter *Tin Can* (numer känt som *Experience API* eller *xAPI*) [23] och syftar till att lösa några av de problem [15] som SCORM-standarden ansågs ha av vissa.

2 Genomförandet

2.1 Teknisk bakgrund

2.1.1 Internet, World Wide Weboch HTML

Internet har funnits sedan 60-talet, och består av ett globalt nätverk av datorer som kommunicerar via protokollen i TCP/IP- modellen. Den stora kommersiella spridningen av Internet skedde i och med införandet av World Wide Web i början av 90-talet, då även många privatpersoner började få internetuppkopplingar. WWW är en teknik för att underlätta informationsöverföring över internet, bl.a. genom att kunna presentera bilder och formaterad text tillsammans för användaren [10]. WWW består av ett antal protokoll och mjukvara (såsom webbläsare) för kommunikation mellan datorer på Internet[22].

Språket som används för att bygga upp sidorna som presenteras för användare är HTML (Hypertext Markup Language) som är standardiserat av The World Wide Web Consortium (W3C) [22] Genom en hierarkisk uppbyggd struktur av så kallade ”taggar”, där taggarna beskriver vilken typ av informationen det rör sig om såsom text, länkar bilder osv., kan en webbläsare tolka och presentera datan för användaren. Eftersom HTML är standardiserat kan alla användare av system som kan köra en webbläsare som klarar av att tolka HTML också kunna ta del av de webbsidorna.

Det finns två huvudsakliga versioner av HTML, 4, från 1997, och 5, från 2012. 5:e versionen är en vidareutveckling men innefattar det mesta från den 4:e versionen.

Det har blivit allt vanligare att bygga system där delar eller hela system är uppbyggt kring webbteknologi i motsats till system och applikationer som bygger på att applikationerna kör lokalt för varje användare. Med webbapplikationer menas här att användargränssnittet består av webbsidor. Anledningarna till att webbapplikationer blivit vanligare är flera, bl.a. [8]

- Det är lättare att distribuera till flera användare då WWW är plattformsoberoende. Alla system som har webbläsare har tillgång till applikationen.
- Lättare att underhålla då t.ex. uppdateringar bara behöver göras på webbservern.
- Minskad risk för dataförluster då datan kan sparas centralt.

2.1.1.1 Klient och server

Men hur skiljer sig en webbapplikation från en applikation som körs lokalt?

Då en applikation som körs lokalt kan vara uppbyggd på en mängd olika sätt, innebär en webbapplikation alltid i grunden server/klientförhållande. Att en applikation är en webbapplikation innebär att klienten, alltså den delen av applikationen som tillhandahåller gränssnittet för användaren, begär tillgång till resurser av servrar. Det sker alltså alltid i grunden en förfrågan av klienten och ett svar från servern, även om vissa av de dynamiska delarna (mer om detta nedan) delvis gör att viss funktionalitet sker på klientsidan. I en webbapplikation utgör webbläsaren klienten och webbservern server.

2.1.1.2 Webbservern

Servern som tillhandahåller klienterna med data i en webbmiljö kallas för en webbserver. Den i allmänhet mest använda webbservertekniken är Apache, vilken fungerar för både Linux/UNIX såväl som för Windowsmiljö. Den mest använda tekniken i windowsmiljö är däremot *Internet Information Services (IIS)*[5].

Webbservern ser till att klienten navigerar rätt bland sidorna och dess olika innehåll, samt ser till att sessionsvariabler och kommunikation med eventuell databas sker. Olika tekniker tillåter också programkod att verka på server-sidan. Ett sådant teknikramverk är ASP.NET.

2.1.2 Webbprogrammering

Till en början kunde en webbbapplikation endast hantera statiska sidor, vilket innebar att det var svårt att härma ett beteende som fanns hos lokalt körda applikationer. Idag finns det tekniker för ett mer dynamiskt beteende. Dynamisk webb innebär att viss funktionalitet kan köras lokalt på webbläsaren, att det finns tekniker för att spara tillståndet för en användarsession och att begäran av information av klienten till servern kan ske på olika sätt.

Då utvecklingen som denna rapport behandlar skedde mot bakgrund av ASP.NET med MVC och javascript så kommer en översikt av teknikerna nedan. En fullständigt uttömmande genomgång är självklart varken möjlig eller genomförbar i den här rapporten, men ett försök att fånga de viktigaste koncepten på en högre nivå har gjorts.

2.1.2.1 Javascript

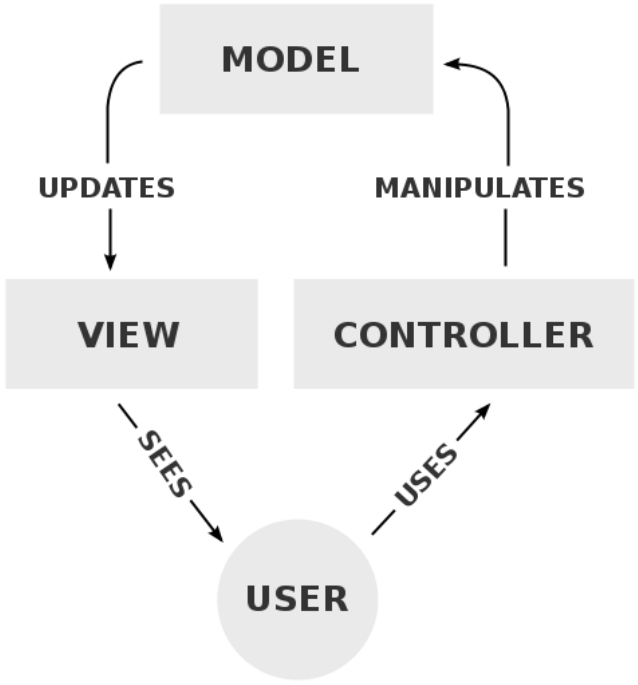
Javascript är ett objektorienterat, icke-kompilerat språk (alltså ett scriptspråk), som bland annat kan köras av webbläsare. Syntaxen liknar C-språkets och namnet till trots har det inget att göra med språket Java. På en webbsida kan det användas för att t.ex. ändra på hur sidan ser ut, navigera till nya sidor, läsa av inmatningar från användaren, interagera på olika sätt med webbläsaren osv. [26]. Det kan också användas för att starta s.k. *ajax-anrop*, vilket innebär att javascriptet kan kommunicera med servern i bakgrunden utan att ändra på hur

sidan ser ut eller beter sig. En vanlig datastruktur som används för kommunikationen är XML, och kommunikationen kan ske både synkront likväl som asynkront. [27]

2.1.2.2 C# och .NET

C# är ett objektorienterat, kompilerat språk som är en vidareutveckling av språken C och C++ utvecklat av Microsoft. Syntaxen är inspirerad av dessa språk och därmed relativt lik. Språket har många likheter med C++, t.ex. att den har, i likhet med Java, en *garbage collector*, vilket innebär att programmeraren inte behöver hantera allokeringen av minne. C# är ett fristående språk men används i första hand i .NET-ramverket. [10]
Sedan 2002 har Microsoft utvecklat ramverket .NET i syfte att tillhandahålla ett fullständigt ramverk för utveckling av såväl fristående applikationer som webbapplikationer. Ramverket består i grunden av ett stort bibliotek som möjliggör körning av flera olika språk mot en virtuell maskin, som hanterar sådant som minneshantering och hårdvarukopplingar. Biblioteket innefattar också klasser som möjliggör webbutveckling på olika sätt. Inräknat i ramverket är också en utvecklingsmiljö, *Visual Studio¸* där utveckling i dessa olika språk kan göras. [28]

2.1.2.3 ASP.NET MVC



Figur1: *MVC-arkitekturen*Källa: [19]

MVC, eller *Model-View-Controller,* är en mjukvaruarkitektur som har blivit mycket spridd i och med att den lämpar sig väl för webbprogrammering.

Arkitekturen består av tre koncept eller delar: *model, view och controller*. Model består av en viss mängd data och används för att hålla ett tillstånd som användaren har försatt applikationen i. Något sätt att hantera tillstånd i webbapplikationer krävs alltid då klient-serverkommunikation på webben är tillståndslös. Modellen kan bestå av egna klasser eller vara direktkopplad till en databas. View är ett sätt att presentera modellen för användaren samt ge denne en plattform för interaktion med applikationen, och består av en HTML-sida med eventuella script. Controllern består av olika s.k. *Actions* som är funktioner som anropas av användaren via något gränssnitt i View:n. Controllern uppdaterar modellen och även hur modellen ska presenteras för användaren samt vilken View som ska användas.

Traditionellt har *routingen*, alltså mappningen mellan adresser och de faktiska webbsidorna, skett som en spegling till var de ligger i webbserverns minne rent fysiskt. Med MVC frångås detta. Ett exempel på en vanlig routing-adress i en MVC-applikation är ”controller/action/id” där *controller* är controllerklassen som ska instansieras, *action* är metoden i klassen som anropas samt *id* som kan innehålla parametrar. Ett exempel på parametrar i webbsammanhang är s.k. *querystrings* som är parametrar vars värde skrivs direkt i adressfältet via t.ex. ett script på sidan.

2.1.2.4 Utvecklingsverktyg

Microsoft Visual Studio är utvecklingsmiljön som är utvecklad för att användas med .NET-ramverket och innehåller förutom kodhanterings- och kompileringsverktyg också exempel och wizards för skapande av olika sorters koddokument. Den övergripande strukturen innebär att ett antal *Project* ingår i en *Solution*. Solution:en är vanligen den övergripande enheten som innefattar allt som ska utvecklas i ett projekt, medan ett project är mera specifikt som ger rätt stödfunktionalitet. Dessa kan t.ex. vara av typen webservice, kodbibliotek, databaskopplingar osv.

2.1.2.5 Web services

Web Service, eller webbmetod, är ett sätt att kommunicera plattformsoberoende mellan olika webbapplikationer och servrar. En vanlig datastruktur som används för kommunikationen är XML och metoderna kan anropas med ajax. Webbmetoder i ASP.NET innebär att ett attribut sätts till en metod som exponerar den som XML-webbmetod.

2.1.2.6 Iframes

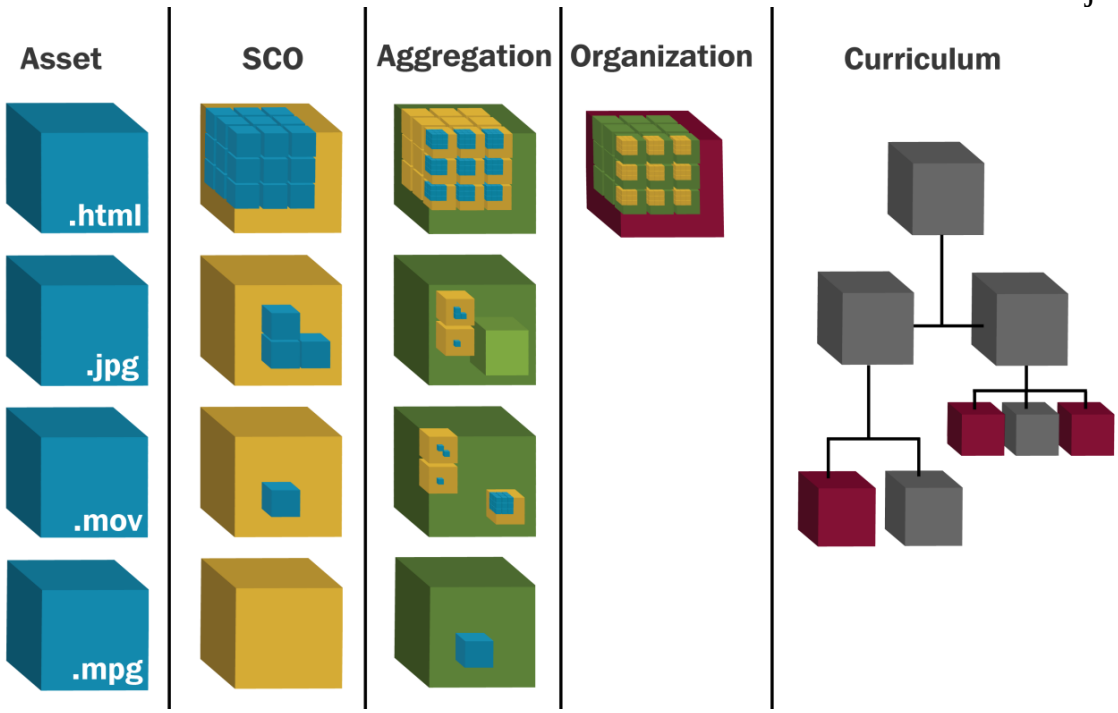
En *IFrame* är en del i ett HTML-dokument som i korthet innebär en ”sida-i-sida”, och fungerar som ett eget HTML-dokument. Detta är dock inte längre rekommenderat i HTML5.

2.2 SCORM-standarden

2.2.1 Inledning

SCORM-standarden består som tidigare nämnts av en samling standarder för webbaserat e-lärande. Den kan grovt sägas svara på följande frågor (vilka behandlas mer noggrant nedan): hur en kurs ser ut, hur kursmoduler ska kommunicera med läroplattformen, hur den är förpackad, hur en läroplattform ska packa upp en kurs, hur en läroplattform ska kommunicera med en kursmodul och hur läroplattformen ska kunna hantera sådant som poängsättning, spara en kursdeltagares framsteg etc. samt enligt vilka regler läroplattformen ska kunna navigera mellan kurser och i enlighet med vilka sekvenser.[1][2][3][4]

En översiktsbild för en SCORM-kurs med alla dess delar kan se ut som följer:



Figur 2: *Components of SCORM Content*Källa: [4]

Bilden beskriver en kurs olika delar. Samtliga delar kommer att behandlas; däremot kommer nivån *Sharable Content Object* (SCO) att vara central. Mer om detta nedan.

Med *Asset* menas helt enkelt sådant som kan presenteras i en webbläsare. I exempelbilden finns exemplen HTML-sida, en jpg-bild och två olika filmformat. En *Asset* kommunicerar inte med LMS:en.

En SCO utgörs av en samling av sådana *Assets* tillsammans med något sätt för SCO:n att kunna kommunicera med LMS:en via API:et.

En *Aggregation* utgör inget fysiskt objekt utan är en struktur som finns beskriven i manifestet (mer om det under *Aggregering, förpackning och manifest*) som medföljer varje SCO, som beskriver grupperingar av SCO:s. Dessa grupperingar kan användas för regler för sekvenser av SCO:s. *Organization* respektive *Curriculum* organiserar aggregationer av kurser på högre nivå.

Ett så kallat *Content Package* kan innehålla allt från enstaka SCO:s upp till mängder med SCO:s som är organiserade på olika sätt utifrån dessa nivåer. [1][4]

2.2.2 SCO

Ett av de mest centrala begreppen i standarden är SCO. En SCO är det minsta objektet som en LMS kan tillhandahålla en användare och det är SCO:n som ansvarar för att kommunicera med LMS:en via API:et [4]. Syftet med att ha ett objekt såsom en SCO är enligt ADL att kunna använda samma SCO:er i flera olika kurssammanhang. [1][14]T.ex. kan en SCO som visar en film om hur säkerhetsutrustning såsom reflexväst, hjälm och skyddsskor och som består av en film och ett flervalsprov efter filmen, användas i flera olika kurser på ett företag som vill utbilda sina anställda i säkerhetsregler. Det kan då tänkas att det finns flera olika kurser som är mer specifika beroende på vad den anställde arbetar med, men att det finns en SCO, såsom beskriven ovan, är gemensam för alla. SCO:n är enligt standarden endast skyldigt att leta upp en instans av API:et samt att anropa funktionerna *Initialize* och *Terminate* på instansen av javascriptsobjektet som API:et utgör. Men SCO:n kan också anropa andra metoder (mer under *API, Datamodell* och *Impementation*). [2]LMS:ens skyldighet gentemot SCO:n är att tillhandahålla ett API och en datastruktur som kan spara och tillhandahålla all data som SCO:n kräver.

2.2.3 Content package och manifest

Ett *Content Package* består av antingen en .zip- eller en .pif-fil och innehåller minst en SCO och en manifestfil. Paketet kan dock innehålla i princip hur många SCO:er som helst. [1]

Manifestfilen består av en .xml-fil och innehåller data som beskriver innehållet i paketet, vilket innehåll som ska levereras till användaren och när det ska levereras. Det är upp till LMS:en att kunna tolka denna fil och följa informationen. Manifestet kan sägas vara uppdelat uppdelat i tre delar: en del som beskriver hur SCO:erna är organiserade med avseende på *Organization* och *Aggregation*, en del som listar och identifierar SCO:er och Assets och slutligen metadata. [1]

Figur 3*: Översikt över innehållet i en manifestfil* [1]

2.2.3.1 Metadata

Metadatan består i sin enklaste form av två element *schema* och *schemaversion.* Dessa innehåller ”ADL SCORM” respektive vilken version av SCORM som åsyftas, i detta fall ”SCORM 2004”. [1][4]

2.2.3.2 Organizations

Varje element under varje organisation består av ett *item* vilket identifierar en SCO eller en Aggregation. Det är inom varje Organisation som sekvenser och navigering sker. [1]

2.2.3.3 Resources

SCORM-innehåll består av filer som kan visas i en webbläsare, exempelvis HTML-sidor, flashobjekt, videofiler etc. En gruppering av sådant innehåll utgör en *Resource*. Det finns två typer av Resources: *SCO Resource* och *Asset Resource.* Det är här som det första filen som ska presenteras för användaren i en SCO anges. Detta görs genom att peka ut en HTML-fil i attributet ”href”. Det som krävs för att en asset ska vara en SCO förutom attributen är som tidigare nämnts att det måste finnas ett javascript som kan kommunicera med API:et. Detta kan anges som en egen fil. [4]

I exemplet [4] nedan syns en organisation och två resources, en SCO och en asset. SCO:ns första fil är här alltså ”load.html” och ”apiwrapper.js” innehåller scriptet för kommunikation.

```
<organizations>
<organization>
<title>Example Course</title>
<item identifier="EXAMPLE-SCO" identifierref="SCO-RESOURCE">
<title>Example SCO</title>
</item>
</organization>
</organizations>
<resources>
<resource identifier="SCO-RESOURCE" adlcp:scormType="sco" type="webcontent"
```

```
href="load.html" >
<file href="load.html" />
<file href="example.jpg" />
<file href="example.swf" />
<file href="example.mp3" />
<dependency identifierref="LESSON_COMMON" />
</resource>
<resource identifier="LESSON-COMMON" adlcp:scormType="asset"
type="webcontent" >
<file href="common/apiwrapper.js" />
<file href="common/common.js" />
<file href="common/logo.jpg" />
</resource>
```

2.2.4 API

Centralt för standarden är API:et. LMS:en har en skyldighet att tillhandahålla ett sådant för SCO:n, och används för att kommunicera med LMS:en och därmed utgöra en koppling till datastrukturen som läroplattformen ska tillhandahålla. Detta ska göras genom en instans av ett javasscriptobjekt som exponerar ett antal funktioner. All typ av kommunikation via API:et samt felhantering är standardiserad. [2]

2.2.4.1 Script

Instansen som ska exponera funktionerna till SCO:n ska döpas till ”API_1484_11” och SCO:n ska implementera script som letar upp instansen. Den ska klara av att leta sig uppåt i flera hierarkier, t.ex. om SCO:n körs i en Iframe och API:et ligger i HTML-sidan som IFrame:n ligger i.[2]

2.2.4.2 Initialisering och avbrott

SCO:n har endast skyldighet att anropa två av funktionerna, *Initialize* och *Terminate.* Dessa talar om för LMS:en att användaren har startat respektive avslutat kursmodulen, alltså att SCO:n startar och avslutar sessionen. [2]

2.2.4.3 Datakommunikation

Två metoder, *SetValue(element, argument)* och *GetValue(element)* ska användas för att sätta respektive hämta data från LMS:en. Elementet i datamodellen pekas ut genom en sträng som med punktnotation pekar ut rätt objekt. [2]

Exempel: Ett anrop av SetValue(”cmi.max_score”, 4) ålägger LMS:en att spara talet 4 som max_score för SCO:n och användarsessionen.

2.2.4.4 Felhantering

API:et ska tillhandahålla funktionerna *GetLastError, GetErrorString* och *GetDiagnostic.* Dessa ska returnera senaste felkod, en beskrivning av det senaste felet samt i vissa fall information om hur problemet kan åtgärdas. Varje metodanrop, utom de ovan nämnda, ska sätta eller nollställa felkoderna. [2]

Exempel: SCO:n anropar funktionen *Terminate* innan ett anrop till *Initialize* har gjorts. Felkoden sätts därmed till 103 och felsträngen till ”Termination Before Initialization”.

2.2.5 Datamodell

Datamodellen består av ett antal objekt som LMS:en ska kunna spara för en användarsession. Hur denna ska implementeras med avseende på programspråk, databastekniker osv. ligger utanför SCORM-standarden, men strukturen och datatyperna är specificerade. Specifikationerna för datatyperna behandlar möjligheter för läsning och skrivning, precision för flyttal, omfång för heltal, längd på strängar och tillåtna tecken. [2] Namnrymden är *cmi* och står alltså längst fram i adresseringen, som består av en lista delad med punkter, och åtföljs av en adress till rätt objekt. Det finns dataobjekt som består av listor, alltså ett ett-till-mångaförhållande. Dessa kallas för *collection* ochadresseras med en siffra. T.ex. är objektet *cmi.objectives* en collection och åtkomsten till ett id till sådan adresseras det med *cmi.objetives.n.id* där n är en siffra.

Det finns även tre nyckelord som är reserverade för specifika betydelser.

1. *_version* talar om för SCO:n vilken version av datamodellen som LMS:en har implementerat.
2. *_count* talar om hur många element det finns i en *collection*.
3. *_children* ska bestå av en kommaseparerad lista av alla dataobjekt som ligger under det objektet som get-anropet sker på som stöds av implementationen (i SCORM 2004 ska dock hela datamodellen implementeras).

Det finns inget krav för en SCO att använda något av objekten överhuvudtaget (det enda kravet för en SCO är att anropa funktionerna *Initialize* och *Terminate*), däremot måste en LMS tillhandahålla hela datamodellen enligt specifikationen.

Då det inte finns utrymme för att gå igenom samtliga objekt, följer nedan en översikt med exempel, kategoriserade efter några konceptuella kategorier.

2.2.5.1 Allmänna dataobjekt

Det finns ett antal allmänna dataobjekt såsom *learner_name* som är en sträng som håller namnet på den som för tillfället tar kursen. Andra exempel är *comments_from_learner,* som sparar fritext från kursen, och *launch_data* som består av data som kan finnas i manifestet. Det finns också ett objekt för att spara sessionens tid *cmi.session_time.*

2.2.5.2 Prestation, poäng etc.

Ett antal dataobjekt i modellen används för att se hur den som kör kursen presterar. Ett centralt dataobjekt är *cmi.interactions* som består av interaktioner med olika delar av kursen såsom svar på en flervalsfråga eller fritextsvar. I en *interaction* ingår ett id och en typ. Typerna är fördefinierade

av standarden och kan bestå av t.ex. *true-false, choice* eller *matching.* Datatypen är beroende av vilken typ som anges. Objekten *cmi.interactions.n.objectives* sparar information om vilka mål de olika interaktionerna kan ha och *cmi.interactions.n.correct_responses.n.pattern* är data kring hur ett korrekt svar ska se ut (datatypen beror på typen på interaktionen).

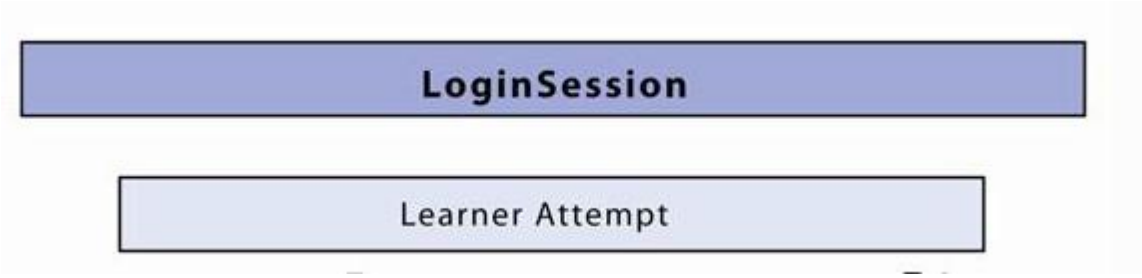
Poäng kan sparas för hela SCO:n i objektet *score* och detta objekt har *raw, scaled, min* och *max* under sig. Det finns också ett *score-objekt* under varje *objective.*

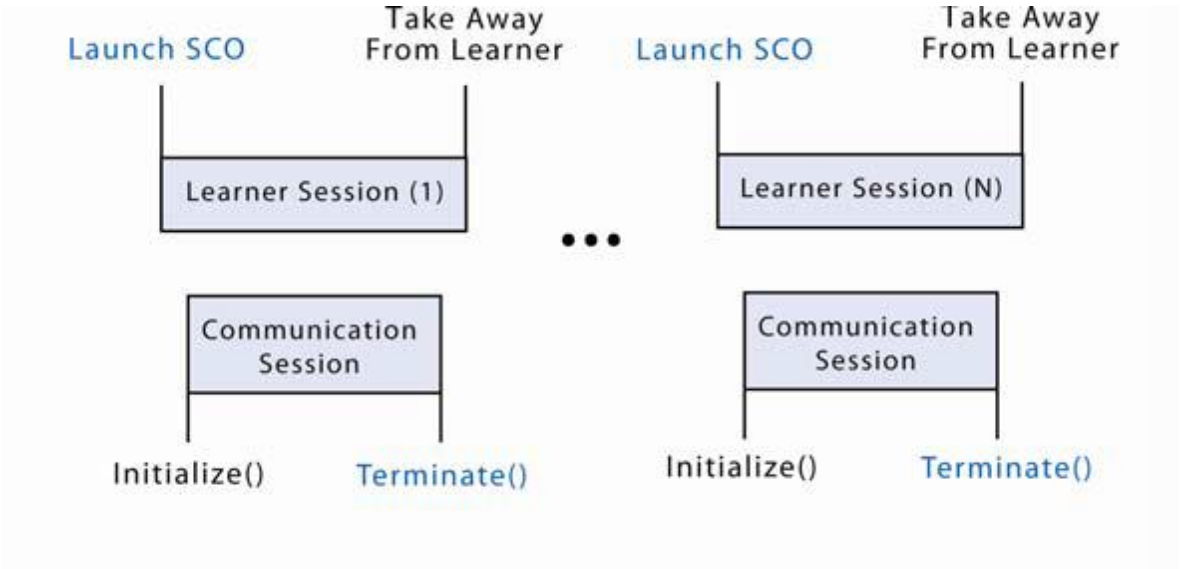
Objektet *cmi.completion_status* kan ha värdena *completed, incomplete* eller *unknown* och anger hur långt användaren har kommit i SCO:n.

cmi.completed_threshold anger om SCO:n ska anses vara avklarad, alltså att *completion_status* ska sättas till *completed.* Är detta inte angett ska SCO:n sköta detta. *completion_status* finns också för objekten *i objectives.* Det som kan avgöra om en SCO eller ett *objective* ska ses som avslutat kan vara sådant som hur många sidor som är besökta, hur många frågor som har svarats på, att användaren trycker på en specifik knapp osv. Objektet *success_status* finns också för både SCO:n som helhet likkväl som för varje *objective.* Detta kan ta värdena *passed, failed* eller *unknown* och anger om användare har blivit godkänd eller inte på kursmodulen respektive på ett *objective.*

2.2.5.3 Avslut, paus etc.

Varje SCO kan hantera användare och sessioner på olika sätt, men LMS:en ska enligt standarden kunna hantera flera sessioner, flera försök på samma kursmodul samt att kunna pausa en session för att kunna uppta den vid ett senare tillfälle. *cmi.suspend_data* innehåller data som SCO:n själv anger och behöver således inte kunna tolkas av LMS:en, men ska likväl levereras vid återupptagande av en session. Ett liknande objekt är *cmi.location* som anger var i SCO:n användaren befinner sig. Vidare ska alla interaktioner, poäng osv. som sparas av SCO:n under en session sparas och levereras vid återstart.





Figur 3*: Tidslinje över kommunikation mellan SCO och LMS* [1]

2.2.5.4 Funktionen Commit

Ett anrop till funktionen från SCO:n avser att spara all cachad data för session till databasen. Men eftersom all cachad data ändå måste sparas vid ett avslut så kan denna funktion mer ses som en säkerhetsåtgärd av SCO:n för att minska risken för dataförluster vid ett onormalt avbrott t.ex. att internetuppkopplingen försvinner.

2.2.5.5 Datatyper

Det finns ett antal huvudsakliga datatyper som kan ha mer eller mindre specificerade begränsingar. De huvudsakliga datatyperna är strängar, heltal, flyttal, boolska tal, datumtider, tidsintervall och uppräkningar och dessa kan ha vidare krav på sådant som precision, positiva eller negativa tal, antal tecken osv.

2.2.6 Navigering och sekvenser

Det är inte möjligt för en SCO att få tillgång till en annan SCO:os data. Däremot finns det dataobjekt som styr sekvenser av SCO:er och det finns också global data. Regler för navigering anges i manifestfilen och avser i vilken ordning som SCO:erna kan ges till användaren. Exempelvis finns läget *choice*, där LMS måste ge användaren möjlighet att starta SCO:erna i valfri ordning. SCO:n kan också ha mer komplicerade regler där vissa villkor ska vara uppfyllda gällande poäng, genomgångna *objectives* eller liknande. Navigering sker genom att objektet *adl.nav.request* sätts med ett navigationsanrop. Exempel på navigationsanrop är *continue* vilket talar om för LMS:en att nästa SCO i sekvensen ska levereras till användaren och*{target=<ACTIVITY_ID>}jump*där ett hopp till nästa aktivitet (t.ex. en SCO) begärs.[2][4]

2.3 Kund, krav och testning

2.3.1 Beskrivning av kund

P&L är ett utbildningsföretag beläget i Hässleholm som bl.a. utvecklar och säljer ett webbaserat kompetensstyrningssystem som heter *Competence Tool 3 (CT3).* CT3 används bl.a. för att analysera kompetenser och kompetensutvecklingsbehov för större företag. Den fungerar även som ett kursadministrationsverktyg. I Kursadministrationsfunktionaliteten ingår att kunna köra kurser författade i ett verktyg som heter *Authoring Tool (AT),* också utvecklat av P&L. Kurserna författade av AT körs för användaren såsom en e-kurs där innehållet består av exempelvis en serie filmer, bilder, flervalsfrågor osv.

Då SCORM-standarden möjliggör körande av kurser författade med vilket författarverktyg som helst som tillämpar standarden, fanns ett önskemål om att anpassa funktionaliteten hos CT3 för att möjliggöra just detta, så att det liknar körandet av kurser författade med AT. CT3:ans struktur innefattade redan uppföljning, administration, distribution osv. så implementationen sågs som ganska begränsad. Endast de delar av SCORM-standarden som ansågs absolut nödvändiga för att anpassa körningen av kurserna till CT3:an skulle implementeras. Det ålades författaren av rapporten att göra en studie och därefter i samråd med kund ställa krav som innebar att detta mål skulle uppnås.

2.3.2 Krav

Från början fanns det ett önskemål från kund som kan sammanfattas i ”vi vill egentligen bara kunnaköra en SCORM-kurs i CT3”. Efter analys, vilken bestod i dels att reda ut hur dessa kurser såg ut, alltså genom att studera exempelkurser författade i Adobe Captivate och Lectora och dels genom en grundlig läsning av standarden och andra källor på Internet vilka hade syfte att ge en teknisk överblick över standarden, kunde kraven specificeras. Specifikationen bestod i att

- Kunna förmedla enskilda kursmoduler författade enligt SCORM 2004 4th edition via CT3 till användare
    - Kod för uppladdning av fil
        - Tolka manifest-filen för kursen
        - Kursmaterialet ska hamna på webbservern
- Användare ska kunna köra kursmodulen i CT3
    - Endast information om kursmodulen är genomförd eller ej, samt godkänd eller ej ska sparas till CT3:ans databas.
- SCORM-kurser kan bestå av flera moduler (SCO:s) men endast stöd för en SCO i taget utvecklas.

2.3.2.1 Adobe Captivate och Lectora
Adobe Captivate och Lectora är kommersiella författarverktyg som används för att författa interaktivt läromaterial. Båda stödjer HTML-objekt samt flash-animationer, filmer, ljud etc. Dessa stödjer också flera olika standarder, bland dessa export till SCORM-standarden[32] [33].

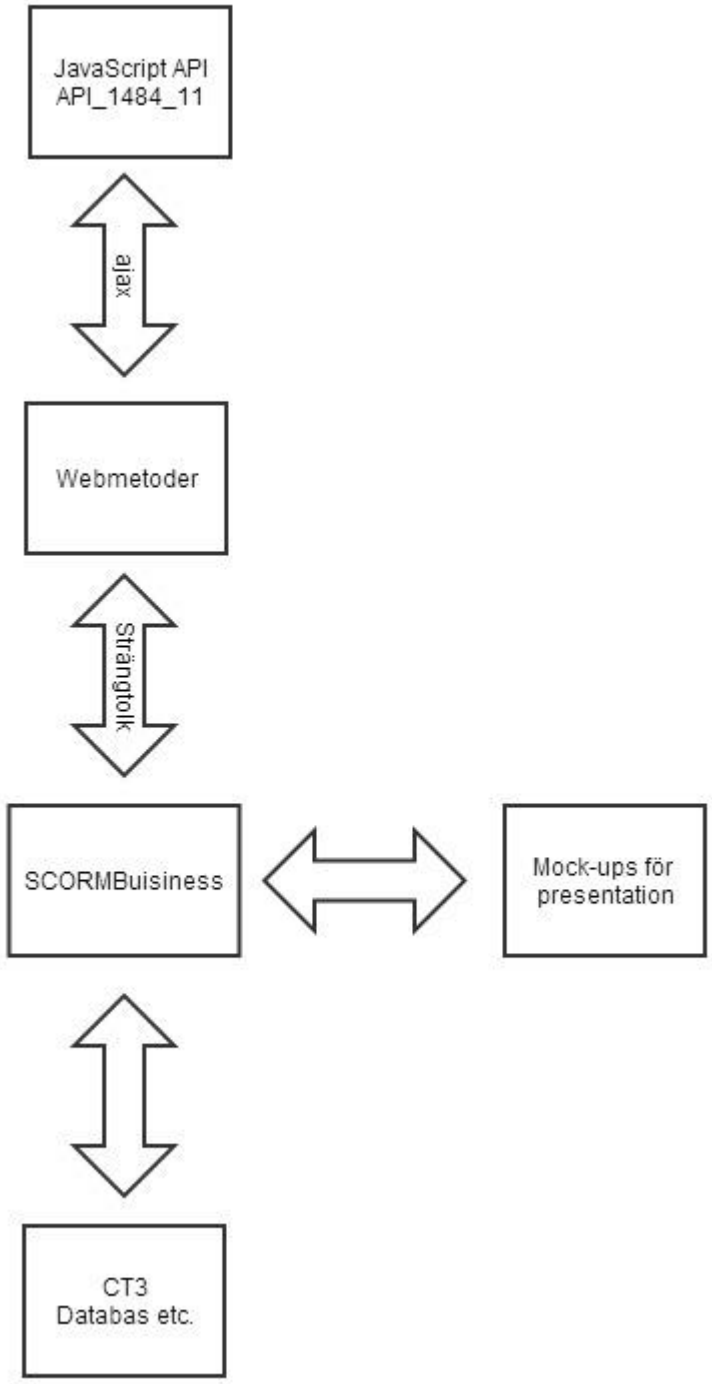
2.3.3 Kravhantering
Under projektets gång utfördes ingen kravhantering i enlighet med någon vedertagen process, utan sköttes helt ad hoc.

2.3.4 Testning
Testningen skedde också den helt ad-hoc och kan således inte sägas vara fullständigt utförd. Testningen skedde endast på följande sätt

- Kodgranskning av annan utvecklare som ej var insatt i SCORM-standarden, däremot var insatt i CT3
- Test mot exempelkurser författade av rapportförfattaren själv
- Test mot två stycken exempelkurser tillhandahållna i tredje hand från kund.

2.4 Grundarbete

2.4.1 API
Enligt standarden ska ett API tillhandahållas kursmodulen (alltså den enskilda SCO:n). Detta API består av tre delar: ett javascript som exponerar en instans av en klass som innehåller de enligt standarden fastställda metoderna, ett antal webmetoder som javascriptet anropar via synkrona ajax-anrop samt ett en strängtolk som tolkar argumenten från webmetoderna. Denna strängtolk anropar också en klass *SCORMBuisiness*, mer om detta nedan.



Figur 3: Översikt över implementationen. Egen bearbetning.

2.4.1.1 Strängtolk

SCORM-API:t kommunicerar som tidigare nämnts genom anrop till datamodellen där dataobjektets pekas ut genom en punknotation i en hierarkisk struktur, men antingen kommandot *get* eller *set.* Anropet till API:et sker genom att kursen skickar en sträng som argument. Exempel:

- Kursdeltagaren svarar på ett flervalstest. Denne får 2 poäng för detta och kursmodulen

1. SCO:n anropar API-instansens metod *SetValue* med argumenten ”cmi.score” och 2.
2. Javascriptmetoden anropar en webmetod *sendToParser* med argumenten ”cmi.score” som element, ”2” som argument, false som get.
3. Parsern tolkar strängen ”2”
4. Parsern letar upp rätt objekt i datastrukturen och sätter värdet.
5. True returneras och alla felkoder nollställs.

- Kursen begär kursdeltagarens namn för att visa i en ruta.
    1. SCO:n anropar API-instansens metod *GetValue* med argumenten ”cmi.learner_name”.
    2. Javascriptmetoden anropar en webmetod *sendToParser* med argumenten ”cmi.learner_name” som element, tom sträng som argument, true som get.
    3. Parsern letar upp rätt objekt i datastrukturen och returnerar värdet om det är satt.
    4. Är värdet inte satt returneras ”false” och felkoder sätts.

Strängtolken tar strängargumentet som pekar ut datamodellsobjektet som ska refereras och bryter ner det i sina beståndsdelar. Detta sker genom att jämföra strängargumentet med de olika enligt standarden tillåtna dataobjekten, och sedan antingen hämta eller sätta dataobjektets värde.

2.4.1.2 Javascript

Javascriptet kan grovt delas upp i ett antal delar.

1. Inläsning av variabler som används för att hålla identiteter för session och modul. Dessa variabler används av CT3 för att kunna peka ut vilken session och kursmodul ifrån vilken anropen kommer. Eftersom det är en webapplikation måste servern på något sätt kunna identifiera vilken användare det är som kör kursen samt vilken kurs det är (då flera kurser kan köras samtidigt). Detta hålls i dessa variabler på servern. När användaren navigerar till sidan som innehåller kursen och scriptet och dessa laddas, skrivs användarsessionens och modulens ID ut på sidan och läses in av javascriptet. Dessa skickas sedan med i varje anrop till servern.
2. Inläsning av adresser till webmethods. Varje anrop sker via ajax till en webmethod. När sidan som tillhandahåller kursen laddas skrivs adresserna in i ett antal div:ar. Dessa läses sedan in av javascriptet och hålls i variabler för att peka ut rätt adresser till webmetoderna.
3. Själva instansen av API:et i enlighet med SCORM-standarden. Denna består av 7st. metoder
    a. Initialize. Metoden tar en tom sträng som argument och väntar i 3 sekunder på att kursen ska vara körklar, annars skickas ett felmeddelande. Varje metod i API:t returnerar felmeddelanden som alla är specificerade i dokumentationen. Exempelvis

returneras felet ”103 – Already Initialized” kursen försöker anropa Initalize mer än en gång. Om allting går rätt till anropas en annan metod i javascriptet som också den heter intialize som i sin tur gör ett ajax-anrop. Mer om dessa metoder nedan.

b. Terminate. Metoden skickar ett fel om kursen försöker anropa metoden innan Initialize är anropad. Om anropet däremot är riktigt kommer SCO:n att få tillbaka värdet ”true” och därmed avslutas. Det sker en navigering till en annan sida.
c. GetValue kontrollerar om API:t är initialiserat och inte terminerat och använder sedan funktionen parse som i sin tur gör ett ajaxanrop till en webmetod.
d. SetValue fungerar på precis samma sätt som GetValue, fast med en ytterligare parameter.
e. Commit gör ett anrop till metoden commit under 4.
f. GetLastError och
g. GetErrorStringGetDiagnostic fungerar på liknande sätt.

4. Speglade metoder till API:et för att gör ajax-anropen. Dessa består av
   a. parse(sessID, modID, element, argument, get). Denna funktion gör ett synkront anrop till webetoden med samma namn och skickar med id:n för användarsessionen (alltså den för SCO:n), vilket id som själva SCO:n/kursmodulen har som körs samt datamodellsobjektet som avses, argumentet för detta om det är ett *setanrop* samt en boolesk variabel som visar om det är ett *get*- eller setanrop som avses.
   b. commit(sessionID, moduleID) gör ett synkront ajaxanrop till webmetoden med samma namn och skickar också den med session- och modulid.
   c. initialize(sessionID, moduleID) liknande som b.
   d. terminate(sessionID, moduleID)liknande som b.
   e. getErrorCodeAndString(sessionID, moduleID). Metoden slår ihop resultaten av *getErrorCode* och *getErrorString* nedan.
   f. getErrorCode(sessionID, moduleID) se b.
   g. getErrorString(sessionID, moduleID, errorCode) se b.
   h. getDiagnostic(sessionID, moduleID, errorCode) se b.
5. Felkoder som kan hanteras direkt i javascriptdelen av API:t, de flesta har med initialisering och terminering att göra.
6. Kod för att skriva ut debuginformation direkt i fönstret. Dessa funktioner skriver ut alla anrop och svar som kursen gör till och får från API-instansen.

2.4.1.3 Felhantering

All felhantering är standardiserad och felkoder med tillhörande strängar som beskriver problemet ska skickas via API:et. Vissa fel kommer dock att hanteras direkt via javascriptet på sidan. Detta gäller framförallt sådant som

har med anrop av *Initialize* och *Terminate* att göra. Den mesta andra felhanteringen sköts av strängtolken som kan analysera felaktiga anrop. Slutligen är datamodellen uppbyggd på ett sådant sätt att felaktiga värden i många fall upptäcks direkt när de ska sättas eller läsas.

2.4.2 Presentation av kursmodul/SCO på webbsida

Användaren kör kursen i ett HTML-objekt som kallas för en Iframe. Iframe:n innehåller en instans av javascript-API:et och sidan som Iframe:nligger i och sidan hämtar kursen som ska köras, samt skriver in alla värden som behövs för körning.

2.4.3 Datamodell

Datamodellen implementerades genom att direkt konstruera en klasstruktur utefter de krav ställts enligt standarden, med ett ”rot-objekt” som kallas *cmi*. Collections, alltså 1-till-många-förhållanden implementerades som en egen datastruktur, kallad just Collection, som implementerar interfacet *IEnumerable*, vilket innebär att det är en struktur som bl.a. kan itereras över. De flesta datatyper finns med som klasser, som tillhandahåller funktionalitet och implementerar de begränsningar och beteenden som standarden satt upp. Ofta innebär detta överlagringar av i C#/.NET befintliga datatyper. Reguljära uttryck har använts för att styra specifika text- och sifferformat.

2.4.3.1 Dumpning av hela datastrukturen till databas

Hela datastrukturen är serialiserbar, vilket innebär att den kan representeras som en xml-fil. Anledningen till detta är att det på detta vis som tillståndet och all data sparas till databasen.

2.5 Anpassning till CT3 och andra krav

2.5.1 Endast en SCO per paket

CT3 innehåller funktionalitet för uppladdning av filer och även uppackning av .zip-filer på webbservern. Vid uppackningen läses manifestfilen av med metoder från den statiska klassen *SCORMManifestReader*. Den innehåller ett antal metoder som kontrollerar att paketet endast innehåller endast en SCO. Uppladdning av paket med annan struktur förhindras.

2.5.2 Ingen navigation mellan SCO:er

Uppföljning, administration osv. sköts av CT3, vilket innebär att navigeringsstrukturen som manifestfilen tillhandahåller inte tas hänsyn till. Ej heller behandlas navigeringsanrop från SCO:n. Detta gör att kurser som är beroende av navigering inte kan köras. Detta kan enkelt göras genom att hindra uppladdning av kurs som innehåller mer än en SCO.

2.5.3 Vissa delar av datan som kursen genererar behandlas separat

Gällande uppföljning av en kursmodul stödjer CT3:ans datastruktur endast begränsade delar av datan som kursmodulen genererar. Detta gör att implementationens kommunikation med CT3:ans databas är anpassad efter

detta. Anpassningen består i att identifiera de data som kursen genererat hitta motsvarande data i CT3:ans datastruktur. Exempelvis använder sig CT3:ans allmänna kursmodulsstruktur också av sådant som *poäng* och *maxpoäng*. SCORM-standarden använder sig av olika sorters poängsättning, bl.a. *raw_score, max_score* och *scaled_score.* I detta fall undersöks om maxpoängen är satt av kursmodulen, och i sådant fall sätts poäng och maxpoäng i CT3, och om den skalade poängen är satt av kursen så sätts maxpoängen i CT3 till 100 och skalpoängen multipliceras med 100 och sättstill *poäng.*

2.6 Fristående applikation för presentation

2.6.1 Mock-up av datamodell och användare

Då CT3:an tillhandahåller sådant som användare, databas och session har implementationen som åsyftas i denna rapport gjort s.k. ”mock-ups” av dessa. Det innebär att det inte finns någon databaskoppling, utan all data sparas och hämtas i en instans av klassen som innehåller hela datastrukturen.
Stora delar av det som CT3:an tillhandahåller såsom användargränssnitt, användarvariabler, sessionshantering osv. har också hårdkodats in för att kunna presentera implementationen som en fristående applikation. Som exempel kan nämnas att sessions- och personid:n är hårdkodade.

3 Analys

3.1 Standarden

3.1.1 Möjligheter att implementera delar av standarden

Det visade sig finnas möjligheter att begränsa implementationen av standarden. Detta kan tydligast ses gällande uppackning och datamodell.

3.1.1.1 Datamodell

Datamodellen i implementationen är fullständig i den meningen att strukturen finns där. Däremot följer den inte alltid standardens krav på precision på flyttal och följer inte alltid heller vissa andra begränsningar på vilka värden de olika objekten i datastrukturen kan ha. Implementationen visar dock att det ändå går att köra vissa kurser utan en fullständigt korrekt datamodell.

3.1.1.2 API

Det funktioner som API:et exponerar mot SCO:n måste samtliga vara implementerade, eftersom de flesta alltid anropas. Däremot fanns det stora möjligheter att välja hur datan som skickas via API:et behandlas.

3.1.1.3 Inläsning av Content Package

Eftersom paketeringen av SCORM-kurser innehåller en manifestfil kan det enkelt undersökas av inläsningsrutinerna om det finns. Det går därmed att begränsa vilka paket som kan accepteras av LMS:en.

3.1.2 Säkerhetsaspekter

Det finns ett problem med uppbyggnaden av API:et som är direkt uppenbar: det går nämligen att med grundläggande kunskaper i javascript och hur SCORM-standarden är uppbyggd att göra egna anrop till API:et t.ex. genom att använda ett i webbläsaren inbyggt verktyg för javascriptdebugging. Exempelvis är det möjligt att efter att en kursmodul är genomförd, och användaren inte uppfyllt målen, ändå anropa *set*-funktionen med argumenten ”cmi.success_status” och ”passed”.

3.2 Implementationen

3.2.1 Vilka delar saknas för en fullständig SCORM-implementering

Implementationen av standarden är inte fullständig; nedan följer en översikt över det som saknas för en fullständig implementation.

3.2.1.1 Datamodell

Vissa delar av datamodellen har inte noggrant undersökts att de uppfyller varje del av standardens krav på datatyper och hur de ska bete sig. Exempelvis finns krav på åtkomst som inte har tagits hänsyn till. Andra delar som har med aggregering, organisation och sekvenser har inte implementerats, då endast stöd för enstaka SCO:er har implementeras.
Exempel:

3.2.1.2 API

Samtliga anrop som har med sekvenser och navigering kommer inte att kunna behandlas av implementationen. Vidare kan det förekomma andra anrop som inte hanteras på ett korrekt sett då ingen fullständigt implementering av datamodellen gjorts.

3.2.1.3 Felhantering

I och med att inte hela datamodellen implementerats kommer inte alla möjliga fel att heller kunna hanteras av implementationen. Felhantering som har med sekvenser och organisationer av flera SCO:er kan inte heller de hanteras då stöd för detta inte utvecklats. Vidare har inte heller alla felkoder och korrekta meddelanden som tillhör dessa koder utvecklats. Det finns en risk att SCO:er använder sig av dessa felkoder för att göra sekundära API-anrop, för att anpassa sig till en delvis felaktig implementation, vilket inte kommer att fungera.

3.2.1.4 Content Packages med fler än en SCO

Eftersom standarden möjliggör många olika sorters strukturer på kurser får inskränkandet ses som en stor begränsning.

3.2.2 Val av tekniker

Valet av tekniker som gjort kan kritiseras på flera punkter utifrån de alternativa lösningar som finns. De teknikval som gjorts återspeglar den oerfarenhet av både .NET i stort likväl som webbprogrammering som

rapportens författare haft vid utvecklingen, vilket fått denne att använda sig av de mest uppenbara lösningarna direkt. Vidare kan sägas att MVC-arkitekturens principer inte utnyttjats (alltså där vy, datamodell och kontroller av dessa har sina fasta roller), utan det handlar snarare om en direktkommunikation mellan med vy-API-datamodell och först därefter kopplad till CT3 som utnyttjar MVC.

Det finns också ett ramverk kallat *Entity-Framework* för .NET som innebär ett enklare sätt att bygga datamodeller. Detta hade dock inneburit en ytterligare arbetsinsats för att koppla till CT3:ans databas.

3.2.3 Exempel och förslag på alternativa lösningar

3.2.3.1 Direktkopplad datamodell med lättviktsramverk– exempel Python med Django

Ett möjlig alternativ lösning hade varit att göra en helt fristående applikation som hade kommunicerat med CT3:ans databas på något annat sätt än att integrera det i den applikationen. I sådant fall hade ett annat ramverk kunnat användas, t.ex. ett som är mindre omfattande och som går snabbare att utveckla i ”från scratch”, då det inte krävs att man behöver sätta in sig ett (åtminstone delvis) stort ramverk.

Django är ett sådant ramverk för webutveckling som bygger på MVC-arkitektur. Det använder sig av språket Python och är open source [31].

3.2.3.2 Färdig implementation av API – exempel SCORM Cloud

En annan mer radikal lösning hade varit att anpassa en färdig implementation [30] till miljön. Fördelarna med detta är uppenbara, förutom att utvecklingstiden med största sannolikhet kortas, så är testning och verifiering redan skött. Däremot så tillkommer en kostnad av t.ex. licensavgifter. Om dessa överskrider kostnaden för utvecklingen kan dock låtas vara osagt.

3.3 Måluppfyllelse

3.3.1 Implementation

Målet med implementationen var att implementera ett API för att en SCO ska kunna kommunicera med en SCO,

3.3.1.1 Testning

Det finns med största sannolikhet en stor mängd buggar i implementationen som ej har upptäckts.
Däremot kan sägas att eftersom det övergripande målet med implementationen var att kunna starta och köra kursmodulerna författade i enlighet med SCORM, så är testningen i det fallet fullgod. Varför? Jo, eftersom det enligt standarden för att en kursmodul ska kunna köras endast krävs att den ska kunna *kommunicera* med ett API. I det hänseendet får enstaka kursmodulstester kunna sägas vara fullgoda då dessa är författade enligt

standarden. Vad som inte är testat, och därmed inte kan verifieras, är en fullständig genomgång av samtliga möjliga anrop och svar, samt samtliga möjliga felmeddelanden.

3.3.2 Övergripande beskrivning standarden

Ett av målen var att ge en övergripande beskrivning av standarden som kan utgöra en teknisk översikt och introduktion. Vid en ytlig jämförelse mot den officiella dokumentationen och programmerarhandledningen[4] så kan det hävdas att samtliga relevanta delar av standarden behandlats i genomgången.

3.4 Krav och kravhantering

Att kravhanteringen inte följde någon vedertagen processberodde på framförallt tre faktorer

- Projektet innefattade bara en person (rapportförfattaren) gällande implementationen av standarden vilket gjorde att det inte var möjligt att hitta en passande kravprocess
- Kraven var till en början mycket vaga
- Det fanns en stor kunskapsbrist hos alla intressenter. Denna avhjälptes endast i samband med projektets avslutande.

Däremot sköttes löpande formulering och validering av krav på ett sådant sätt att projektet aldrig hamnade i ett sådant läge att arbetet inte kunde fortsätta. Att inte följa någon vedertagen process hade också fördelen att projektet kunde utföras enskilt och på ett mycket anpassningsbart sätt. När oklarheter rörande vilka delar av standarden som skulle implementeras kunde detta mycket snabbt utredas med t.ex. en intervju eller möte.

Nackdelarna med ett sådant förfarande är uppenbara. Det saknas t.ex. dokumentation kring hur kraven ändrats under projektets gång, kraven som står med i denna rapport är kraven så som de såg ut efter att examensarbetet avslutats. Validering av krav kunde inte heller genomföras på ett vedertaget sätt, utan gjordes också de utifrån det färdiga resultatet.

4 Slutsatser

4.1 Tidsaspekter

Så som tidigare nämnts är en fullständig implementation av standarden ej möjlig i kontexten av ett examensarbete. Delarna som implementerats innebär ett API som är kapabelt att kommunicera med en kursmodul och en tillräckligt omfattande implementation av datamodellen för att hantera de dataelement som styrs av den

En mycket grov uppskattning av det arbete som återstår skulle kunna vara i den storleksordning som tidigare nämnts, alltså ca 1-1,5 manårs arbete till. Detta är dock en uppskattning som bygger helt på rapportförfattarens förståelse av dokumentationen, resultatet av arbetet samt övrig läsning av andra källor som behandlar ämnet.

4.2 Möjlighet att arbeta enskilt

Det största problemet för en utvecklare som inte besitter tidigare erfarenhet av varken webprogrammering i .NET, dynamisk webprogrammering med javascript eller insikt i SCORM-standarden eller dess dokumentation blir att denna måste lägga mycket tid på att studera och lära sig dessa. Dessutom kunde inte något fullständigt open-sourceproject som implementerade SCORM-standarden hittas. Ett sådant projekt hade kunnat ge en vägvisning till hur en implementationen kunnat se ut. Vidare är det också mycket svårt att hitta exempelkurser författade enligt standarden (med vissa undantag[1]). En stor mängd sådana hade gett möjlighet till mer omfattande testning av implementationen. Däremot kunde implementationen testas genom egenförfattade kurser med Adobe Captivate. Nackdelen med detta förfarande är att dels författaren själv inte har någon insikt i hur kursmoduler författas generellt, alltså vilka delar av standarden som normalt sett utnyttjas. Även om hela standarden ska implementeras för att kunna sägas vara fullt SCORM-kompatibel, så kan det i kontexten av detta examensarbete fortfarande vara intressant att ta hänsyn till om vissa delar av standarden i stort inte utnyttjas, eftersom implementationen som här behandlas inte är fullständig. Det är möjligt att sådana undersökningar utförts, men några sådana har inte hittats[2]. En annan nackdel att använda egenförfattade kurser som tester är att det kan finnas en omedveten tendens att författa kursmodulen på ett sådant sätt att den fungerar mot implementationen.
Vad gäller kravhanteringen så visade det sig också att försöket att arbeta enskilt med en relativt fristående implementation påverkade sättet på vilket krav togs fram. En slutsats som kan dras är att ett arbete med en implementation av en standard av den här omfattningen där önskemålen från kundens sida till en början är vaga eftersom de inte själva är insatta i standarden, är att arbetet med krav tenderar till att skötas ad-hoc och drivet av utvecklaren.
Det visade sig också att testningen innebar ett stort problem. Anledningen till detta kan tänkas bero på att det från början inte fanns någon klarhet i hur eller vad som skulle testas. Detta kan sammankopplas med samma faktorer som utgjorde problem för krav och kravhantering. En slutsats att dra av detta är att

[1] http://scorm.com/scorm-explained/technical-scorm/golf-examples/ och http://www.ostyn.com/standards/scorm/samples/proddingSCOwrap.htm#howto Hämtade (20141219)
[2] Se Appendix 1

det finns stora svårigheter gällande verifikation av en implementation av en standard som inte är tänkt att implementeras fullständigt.

4.3 Möjlighet att frikoppla applikationen

Där kraven på implementationen innebar att sådant som distribution, databas, användare, sessioner osv. sköts av ett system som redan besitter funktionalitet för detta, och själva kopplingen till detta system inte ingår arbetet, så fanns det ett behov att frikoppla implementationen för att göra det möjligt att presentera den som en fristående applikation. Detta gjordes möjligt genom att simulera bakgrunden som systemet utgör genom ett antal hårdkodade variabler och statiska funktioner. Dessa har till syfte att tillhandahålla liknande värden som systemet annars kulle tillhandahålla. Exempelvis finns det identifikationsvariabler för användare och sessioner som på detta sätt hårdkodats.

Vidare gäller att på sådant sätt som javascript-delen av API:et är implementerat, alltså i en fristående fil med anrop mot webmetoder, går att utveckla helt fristående från all annan utveckling i ett liknande projekt. Eftersom datamodellen är uppbyggt genom en klasstruktur, kan också denna utvecklas fristående.

Det som inte har gått att frikoppla är distribution och administration av kurserna. I presentationsversionen av implementationen är detta löst genom att hårdkoda in en adress för lokal placering av SCO:n som ska köras och att uppföljningen sker genom debug-information och liknande. Detta är självfallet inte något som går att använda praktiskt och därmed kan inte de delarna sägas vara lyckat frikopplade.

Slutsatsen av allt ovanstående kan sägas vara att kommunikationen mellan SCO:n och själva datastrukturen kan utvecklas var för sig. Om detta är optimalt eller inte beror på vilket ramverk som är valt (mer om detta under *Alternativa lösningar*), men det är möjligt. Uppföljning, lagring, administration osv. är därmed funktionalitet som måste utvecklas i själva läroplattformens systemkontext.

4.4 Förslag på vidare utveckling och undersökningar

4.4.1 Fullständig implementation av SCORM LMS för CT3

En självklar vidare möjlig utveckling är att slutföra implementationen av de delar som saknas. Detta behöver göras i två steg: ordentlig testning av det som redan är implementerat för att identifiera de möjliga problem med framförallt datamodellen som kan uppstå och möjliggöra uppackning och körning av kurser som består av mer än en SCO.

4.4.2 Hur används standarden generellt?

En jämförande undersökning bland liknande implementationer som denna, alltså en delvis implementation, hade kunnat ge information om vilka delar av standarden som används mest, hur de används och om vissa delar inte alls utnyttjas. Ett annat sätt att göra detta på hade kunnat vara att undersöka en stor mängd kurser författade enligt standarden och t.ex. genom körningar genom något slags analysverktyg exempelvis identifiera vilka slags anrop som görs via API:et.

4.4.3 Går det attanpassa SCORM-implementationer till Tin Can?

Eftersom Tin Can-projektet (numera *Experience API eller xAPI)*[23] är en vidareutveckling av SCORM så kan en jämförande undersökning mellan standarderna göras för att ta fram en strategi för att anpassa en SCORM-implementation liknande den, som är beskriven i den här rapporten, till en dito för xAPI.

4.4.4 Fördelar och nackdelar med val av andra ramverk och tekniker

En intressant undersökning hade kunnat vara att undersöka implementationer som använder sig av olika tekniker och ramverk. Detta hade t.ex. kunnat ge jämförelser mellan olika ramverks för- och nackdelar. En sådan undersökning hade också kunnat avslöja brister i standarden, då olika tekniker kan tänkas stöta på olika problem.

4.4.5 Jämförelse mellan en annan implementation som använder sig av samma eller liknande tekniker

En liknande undersökning som den ovan är att jämföra en annan implementation av standarden som är gjord med liknande tekniskt ramverk. Då detta är ett sätt att eliminera flera variabler samtidigt hade detta kunnat ge information kring hur olika faktorer kan tänkas påverka utvecklingstid, hur kodstandarder och dokumentation skiljer sig åt osv.

5 Referenser

5.1 Dokumentation

[1]*Sharable Content Object Reference Model (SCORM) 2004 4th Edition Content Aggregation Model (CAM) Version 1.1,* 2009. Advanced Distributed Learning (ADL) , 2009

[2]*Sharable Content Object Reference Model (SCORM) 2004 4th Edition Run-Time Environment (RTE) Version 1.1,* 2009. Advanced Distributed Learning (ADL) , 2009

[3]*Sharable Content Object Reference Model (SCORM) 2004 4th Edition Sequencing and Navigation(SN) Version 1.1*. Advanced Distributed Learning (ADL), 2009

[4]*SCORM Users Guide for Programmers, SCORM 2004 4th Edition Version 10*. Advanced Distributed Learning (ADL) , 2011

5.2 Litteratur, uppsatser och artiklar

[5] Blom, J., Ljung, J. *Dynamisk Webbprogrammering - Varför väljer systemutvecklande organisationer ASP.NET?*Linköpings Universitet 2004

[6] Jenderhag, P.,Carlsson, G., *Learning Management Systems - Vilka användbarhetsfaktorer, funktioner och designförslag bör beaktas vid val av lärplattform?* Högskolan i Halmstad, 2008

[7] Mingkun, Y. *Retrofit Learning Management System To Use SCORM ?*Högskolan i Halmstad, 2011

[8] Shen H., Yang Z., Sun C. *Collaborative Web Computing: From Desktops to Webtops*IEEE Distributed Systems Online IEEE Computer Society vol 8 nr 4 2007

[9] Skoglöf, J. . *Standards för e-Learning – en översikt av nästa generations e-Learning baserad på etablerade standards, infrastruktur och processer.*Täby: LearnTech AB, 2001.

[10] Watson, Karli, *Beginning Visual C#*, Birmingham: Wrox Press Ltd 2002

[11] Weiss, M.A. *Data Structures & Problem Solving using Java*. Boston: Pearson Eaudication 2010

[24] Lauesen, S. *Software Requirements Styles & Techniques* Boston: Addison-Wesley2002

[25] Löfberg, M., Molin, P. *Web vs. Standalone Application – A maintenance application for Business Intelligence* BlekingeTekniskaHögskola 2005

[26] Flanagan, D. *JavaScript - The Definitive Guide, 5th Edition*, O'Reilly 2006

[29] McConnell, S. *Professional Software Development: Shorter Schedules, Better Projects, Superior Products, Enhanced Careers*Boston : Addison-Wesley 2004

5.3 Internetkällor

[12] ADL Official Homepage
http://www.adlnet.gov/
(Hämtad20140105)

[13] ADL Overview
http://www.adlnet.gov/overview/
(Hämtad 20140105)

[14] ADL SCORM Official Homepage
http://www.adlnet.org/scorm/
(Hämtad20140105)

[15] An Introduction to the Tin Can API
http://www.thetrainingbusiness.com/softwaretools/tin-can-api/ (Hämtad 20140105)

[16] ASP.NET MVC Overview
http://www.asp.net/mvc/tutorials/older-versions/overview/asp-net-mvc-overview (Hämtad 20140105)

[17] Guide to the Software Engineering Body of Knowledge
http://www.computer.org/portal/web/swebok/html/ch2 (Hämtad 20140105)

[18] HTML Working Group
http://www.w3.org/html/wg/(Hämtad 20140105)

[19] Model-View-Controller
http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller
(Hämtad 20140105)

[20] Requirements Engineering from an HCI Perspective
http://www.interaction-design.org/encyclopedia/requirements_engineering.html#heading_Analysis_html_pages_100760
(Hämtad 20140105)

[21] Rustici Software. SCORM EngineThe Business Case

http://scorm.com/scorm-solved/scorm-engine/scorm-engine-business-case/ (Hämtad 20140105)

[22] What is HTML? http://www.w3.org/standards/webdesign/htmlcss (Hämtad 20140105)

[23] What is in the Tin Can API? http://tincanapi.com/overview/ (Hämtad 20140105)

[27] Excerpt from *Beginning Ajax* http://www.wrox.com/WileyCDA/Section/id-303217.html (Hämtad 20140122)

[28] Overview of the .NET framework http://msdn.microsoft.com/en-us/library/zw4w595w(v=vs.110).aspx (Hämtad 20140122)

[30] SCORM Cloud API http://scorm.com/scorm-solved/scorm-cloud-features/scorm-cloud-api/ (Hämtad 20140125)

[31] Django https://www.djangoproject.com/(Hämtad 20140125)

[32] Adobe Captivate Officiell hemsida http://www.adobe.com/products/captivate.html (Hämtad 2014-03-22)

[33] Lectora Officiell hemsida http://lectora.com/e-learning-software/ (Hämtad 2014-03-22)

6 Appendix I: Sökord och sökplatser

Sökord: SCORM, Shareable Context Object Reference, LMS, e-learning, ADL, Tin Can, xAPI, Experience API

Sökplatser: Google, IEEE XPlore, LOVISA, uppsatser.se

7 Appendix II: Ordlista med återkommande förkortningar och några andra begrepp

API – Application Programming Interface. En standardisering av gränssnitt för hur olika system eller hur olika delar i samma system ska interagera med varandra.

AJAX–Asynchronious JavaScript and XML. Ramverk för kommunikation mellan klient och server där sändning och mottagande av data från klientsidan inte kräver en fullständig omladdning av sidan (vilket det I normalfallet gör).
LMS – Learning Management System. Lärandeplattform. System för att tillhandahålla funktionalitet för e-lärande.
Open Source – Innebär att källkoden är fritt tillgänglig. Det behöver dock inte betyda att produkten får användas fritt.
SCO – Shareable Content Object. Del av SCORM-standarden. Se bl.a. under *2.2.2 SCO*
XML – Exstensible Markup Language. Standard för hierarkisk datastruktur. HTML bygger på denna standard.

8 Appendix III: Kod- och testexempel

8.1 Exempel 1: Inläsning av SCO

1.

```
/// <summary>
/// Indicates wether theres one and only one SCO in the package, SCORM
2004 3rd and 4th ed.
/// </summary>
/// <param name="baseDirectoryPath">The base directory of the unpacked
package</param>
/// <returns>True if theres one and only one SCO, false
otherwise</returns>
publicstaticbool HasOnlyOneSCO(string baseDirectoryPath)
{
return
HasOnlyOneSCO(baseDirectoryPath,(XNamespace)ADLCP_NAMESPACE_STRING,(XName
space)XMLNS_NAMESPACE_STRING, SCORM_TYPE_ATTRIBUTE_STRING);
}
privatestaticbool HasOnlyOneSCO(string baseDirectoryPath, XNamespace
adlcp, XNamespace xmlns,string scormTypeAttributeString)
{
      XDocument manifest = XDocument.Load(Path.Combine(baseDirectoryPath,
MANIFEST_FILE_NAME));

//Select all elements named "resource" and count elements with attribute
"adlcp:scormType" with value "sco" (There can be only one!)
      IEnumerable<XElement> resources =
        from element in manifest.Root.Descendants(xmlns +"resource")
        where element.Attribute(adlcp + scormTypeAttributeString).Value
== SCO_STRING
        select element;

return resources.Count()==1;
}
publicstaticstring[] SCORM_2004_STRINGS ={
"SCROM 2004 3rd Edition",
"SCROM 2004 4th Edition"};
```

2.Test

Inläsningen misslyckas. Genomgång av manifestfiler från några olika SCO:er visar att dessa inte följer standarden gällande namngivning för indikering av version.
3. Korrigering: Tillägg av hittade strängar

```
publicstaticstring[] SCORM_2004_STRINGS ={
"SCROM 2004 3rd Edition",
"SCROM 2004 4th Edition",
"2004 3rd Edition",
"2004 4th Edition"};
```

4. Test
Ytterligare efterforskning på hemsidor visar att även "CAM 1.3" används i vissa fall.
5. Korrigering: Ytterligare tillägg

```
publicstaticstring[] SCORM_2004_STRINGS ={
"SCROM 2004 3rd Edition",
"SCROM 2004 4th Edition",
"2004 3rd Edition",
"2004 4th Edition",
"CAM 1.3"
};
```

8.2 Exempel 2: Strängtolk

Strängtolken består i stort sett av en enda lång villkorssats med olika hjälpmetoder. Exemplet visar de 80 första raderna i denna tillsammans med en hjälpmetod. Anropsmetoden "parse" visas också.

```
publicstring parse(string element,string argument,bool get)
{
string[] elements = element.Split('.');
int maxIndex = elements.Count()-1;
int n=0;//Dummy
return ParseCMI(argument, elements, n, get);
}



privatestring ParseCMI(string argument,string[] elements,int n,bool get)
{
if(elements[0]=="cmi")
{
//version
if(elements[1]=="_version"){return argCheckNoSet(get, CMI._version);}

//children
elseif(elements[1]=="_children"){return argCheckNoSet(get,
CMI._children);}
//BEGIN comments_from_learner
elseif(elements[1]=="comments_from_learner")
{
if(elements.Length ==2)
{
                        cmiError.setDiagnostic("cmi.comments_from_learner
is a collection. Specify index or method to be called...");
return undefinedElement();
```

```
}
elseif(elements[2]=="_children"){return argCheckNoSet(get,
Comment._children);}
elseif(elements[2]=="_count"){return argCheckNoSet(get,
root.comments_from_learner.count().ToString());}
//Kolla om index är en siffra
elseif(!checkIfNumber(elements[2])){return"false";}
//BEGIN Comment
else
{return ParseLearnerComment(argument, elements,ref n, get);}
//END comment
}

privatestring argCheckNoSet(bool get,string returnStr)
{
if(get)
{
                cmiError.setError(0);
return returnStr;
}
else
{
                cmiError.setError(404);
return"false";
}
}
```

8.3 Exempel 3: Kontakt mellan API och SCO

För att SCO:n och API-instansen ska kunna kommunicera krävs initialisering av de båda. Initialiseringen av SCO:n genom den genom API-instansen exponerade funktionen fungerade dock inte direkt, utan en timeout var tvungen att läggas till. Misstänker missmatch mellan laddningstider för sidan som tillhandahåller API och SCO.

```
var API_1484_11 =new API();

API.prototype.Initialize =function(emptyStr){
if(debug){ write("Called Initialize");}
if(isInitialized){
        setError(103);
return"false";
}
else{
        isInitialized =true;
        retStr = initialize(sessionID, moduleID);
if(debug){ write("API returned: "+ retStr);}
return retStr;
}
}
```

1.Test: SCO:n ger felmeddelande, API:et kan inte hittas.

2. Korrigering: Misstänker att SCO:n eller sidan inte hinner laddas klart. Tillägg av document.onload

```
$(document).ready(function() {
if (isInitialized == false) {
alert(communicationErrorText);
      }});
```

3. Test: Fungerar i vissa fall, några testkurser går fortfarande inte att ladda.

4. Korrigering: Tillägg av 3s timeout för inväntning av SCO

```
//Check against isInitialized for 3 seconds. Otherwise show warning
var seconds =0;
    setTimeout(function(){
if(isInitialized ==false){
alert(communicationErrorText);
}
},3000);
```

8.4 Exempel 4: Serialisering av datamodell till databas

För att spara data till en cache i databasen så serialiseras - alltså görs till en samlad datafil, i detta fallet en XML - hela datastrukturen och sparas till databasen. CMI är namnet på klassen som utgör roten i datamodellens träd. Person-objektet är en nyckel, Session är ett objekt som representerar vilken "försök" eller "gång" det är personen genomför kursmodulen och SCORMModule är objektet som håller SCO:n.

```
/// <summary>
/// Commits datacache and updates completion status, success status,
time, suspension_data and exit state.
/// </summary>
/// <param name="root"></param>
/// <param name="person"></param>
/// <param name="session"></param>
/// <param name="module"></param>
/// <returns></returns>
publicbool SerializeDataCache(CMI root, Person person, Session session,
SCORMModule module)
{
//Serialize
string serial = SerializeToXml<CMI>(root);
//Insert string
try
{
using(SqlConnection c = conn.GetConnection())
{
using(SqlCommand comm =new SqlCommand(@"BEGIN TRANSACTION
                                                    UPDATE testResult
                                                    SET dateEnd =
@dateEnd
                                                    WHERE id = @id
                                                        UPDATE
person_session_module_SCORMdata
                                                    SET
completion_status = @completion_status, success_status = @success_status,
suspension_data = @suspension_data, @exit_state = exit_state, cache =
@cache
```

```
                                                    WHERE id = @id
                                                   COMMIT", c))
{
            comm.Parameters.Add("@id",
System.Data.SqlDbType.UniqueIdentifier).Value = AttemptId;
            comm.Parameters.Add("@dateEnd",
System.Data.SqlDbType.DateTime).Value = DateTime.Now;
            comm.Parameters.Add("@exit_state",
System.Data.SqlDbType.NVarChar).Value = root.exit;
            comm.Parameters.Add("@completion_status",
System.Data.SqlDbType.NVarChar).Value = root.getCompletionStatus();
            comm.Parameters.Add("@success_status",
System.Data.SqlDbType.NVarChar).Value = root.getSuccessStatus();
            comm.Parameters.Add("@cache",
System.Data.SqlDbType.NVarChar).Value = serial;
            comm.Parameters.Add(new SqlParameter("@suspension_data",
root.suspend_data ==null?"": root.suspend_data));
            comm.ExecuteNonQuery();
}
}
}
catch(Exception e)
{
        System.Diagnostics.Trace.WriteLine(e.Message);
returnfalse;
}
returntrue;
}

privatestring SerializeToXml<T>(T value)
{
      StringWriter writer =new
StringWriter(CultureInfo.InvariantCulture);
      XmlSerializer serializer =new XmlSerializer(typeof(T));
      serializer.Serialize(writer, value);
return writer.ToString();
}

/// <summary>
/// Gets the serialized object string from database and deserializes it
/// </summary>
/// <param name="person"></param>
/// <param name="session"></param>
/// <param name="module"></param>
/// <returns>The root of the SCORM datastrucure cache.</returns>
public CMI GetCache(Person person, Session session, SCORMModule module)
{
//get string
string cacheStr;
      CMI cache =null;
try
{
using(SqlConnection c = conn.GetConnection())
{
using(SqlCommand comm =new SqlCommand(@"SELECT cache
                                                   FROM
person_session_module_SCORMdata
                                                   WHERE id = @id", c))
{
            comm.Parameters.Add("@id",
System.Data.SqlDbType.UniqueIdentifier).Value = AttemptId;
            cacheStr =(string) comm.ExecuteScalar();
```

```
            cache = Deserialize(cacheStr);
}
}

}
catch(Exception e)
{
        System.Diagnostics.Trace.WriteLine(e.Message);
}
return cache;
}

private CMI Deserialize(string xmlSerial)
{
      XmlSerializer serializer =new XmlSerializer(typeof(CMI));
      StringReader reader =new StringReader(xmlSerial);
return(CMI)serializer.Deserialize(reader);
}

privatestring SerializeToXml<T>(T value)
{
      StringWriter writer =newStringWriter(CultureInfo.InvariantCulture);
      XmlSerializer serializer =new XmlSerializer(typeof(T));
      serializer.Serialize(writer, value);
return writer.ToString();
}
```

8.5 Exempel 5: Modellobjektet "Score"

Exempel på implementation av datamodellsobjekt genom en klass. Klassen klarar av att hålla de två olika sorters poäng som krävs av standarden likväl som max- och minpoäng.

```
publicclass Score : SCORMDataModelObject
{

publicconststring _children ="scaled,"+
"raw,"+
"min,"+
"max";

privatefloat scaled =0, min =0, max =0, raw =0;
privatebool scaledSet =false, minSet =false, maxSet =false, rawSet
=false;

public Score(){}

//Setters

publicbool trySetScaled(float scaled){
if(scaled >=-1&& scaled <=1)
{
this.scaled = scaled;
                scaledSet =true;
returntrue;
}
elsereturnfalse;
}
```

```
publicvoid setRaw(float raw){
            rawSet =true;
this.raw = raw;
}

publicvoid setMin(float min)
{
            minSet =true;
this.min = min;
}

publicvoid setMax(float max)
{
            maxSet =true;
this.max = max;
}

//Getters

publicfloat getScaled(){returnthis.scaled;}
publicfloat getRaw(){returnthis.raw;}
publicfloat getMin(){returnthis.min;}
publicfloat getMax(){returnthis.max;}

//is

publicbool isSetScaled(){return scaledSet;}
publicbool isSetRaw(){return rawSet;}
publicbool isSetMin(){return minSet;}
publicbool isSetMax(){return maxSet;}
}
```