

POLITECHNIKA WROCŁAWSKA

WYDZIAŁ ELEKTRONIKI

KIERUNEK: INFORMATYKA

SPECJALNOŚĆ: INŻYNIERIA SYSTEMÓW INFORMATYCZNYCH

PROJEKT INŻYNIERSKI

Budowa graficznego interfejsu
użytkownika aplikacji sieciowej z
wykorzystaniem JavaScript

Development of a web application's
GUI with the use of JavaScript

AUTOR:

Mateusz Adamiak

PROWADZĄCY PRACĘ:

dr inż. Tomasz Kubik

OCENA PRACY:

WROCŁAW, 2012

Spis treści

1. Wstęp	5
1.1. Cel i zakres projektu	6
2. Wymagania projektowe	7
2.1. Wymagania funkcjonalne	7
2.1.1. Wybór aktu prawnego	7
2.1.2. Edycja aktu prawnego	8
2.1.3. Generowanie zawartości pliku XML	8
2.2. Wymagania niefunkcjonalne	9
2.2.1. Środowisko pracy systemu	9
2.2.2. Wykorzystywane technologie	9
2.3. Cykl życia projektu	10
3. Java2Script	12
3.1. Analiza technologii	12
3.2. Zasada działania	13
3.3. Java2Script a Java	13
3.4. System dziedziczenia w zanurzonym JavaScript	13
3.5. Przegląd istniejących serwisów, w których wykorzystano Java2Script	15
4. XML i Schemat XML	18
4.1. XML	18
4.2. Schemat XML	19
4.3. Parsowanie dokumentów XML	19
5. Opis cyklu życia projektu	20
5.1. Pierwsza iteracja – wyszukiwarka semantyczna	20
5.2. Druga iteracja – ActEditor	21
5.3. Trzecia iteracja – LawCreator	24
6. Opis implementacji	26

7. Testowanie	29
7.1. Testy jednostkowe	29
8. Podsumowanie	30
8.1. Ocena projektu	30
8.2. Wnioski	31
Literatura	33
A. Instrukcja wdrożenia	34
B. Instrukcja użytkownika	35

Skróty

XSD (ang. *XML Schema Definition*)

XML (ang. *eXtensible Markup Language*)

JS (ang. *JavaScript*)

J2S (*Java2Script*)

SWT (ang. *Standard Widget Toolkit*)

MSWiA (*Ministerstwo Spraw Wewnętrznych i Administracji*)

OOP (ang. *Object-Oriented Programming*)

Rozdział 1

Wstęp

Dzięki dynamicznemu rozwojowi technologii informacyjnych Internet stał się obecnie jednym z podstawowych mediów stosowanym do wymiany danych, komunikacji interpersonalnej, publikowania zasobów cyfrowych itp. Na bazie Internetu powstała sieć WWW, wykorzystująca protokół HTTP oraz język HTML do opisu opublikowanych zasobów. Dla zwykłych użytkowników sieć WWW kojarzona jest głównie z witrynami portali, do których można sięgnąć z poziomu okna własnej przeglądarki. Jednak selekcja towarów, obsługa poczty elektronicznej, przeglądanie najnowszych wiadomości ze świata, oglądanie filmów jak również realizacja innych, niezliczonych wręcz funkcji oferowanych w sieci bywa różnie oceniana przez ich odbiorców. Poza samym faktem spełnienia zadeklarowanej roli przez daną witrynę (aplikację sieciową), na tę ocenę duży wpływ ma ergonomia i inteligencja zaimplementowanych w niej rozwiązań. Mówiąc prościej - przyjazność interfejsu użytkownika.

Historia budowy przyjaznego interfejsu użytkownika jest tak długa, jak długa jest historia rozwoju komputerów. Od chwili powstania pierwszej maszyny liczącej projektanci musieli stawiać czoło zadaniu skonstruowania odpowiedniego zestawu przyrządów, które ułatwiłyby człowiekowi interakcje z tworzonymi przez nich urządzeniami. Pojawiły się więc monitory, wyświetlacze, drukarki, klawiatury, myszki, joysticki i inne urządzenia wejścia/wyjścia. Jednak pomimo upływu czasu i wprowadzania coraz to nowszego sprzętu nie zmienił się zbytnio charakter sposobu prezentacji danych użytkownikowi, za którym kryje się oprogramowanie. Można powiedzieć, że w tym obszarze nadal obowiązuje podział na interfejs tekstowy i interfejs graficzny, przy czym ten drugi obecnie przeważa. Zapewne w przyszłości dojdzie do usprawnienia interfejsów do komunikacji na linii człowiek-maszyna, by uczynić ją bardziej naturalną i wygodną dla człowieka. Już teraz prowadzone są badania nad rzeczywistością wirtualną, czy interfejsami mózg-maszyna. Zanim jednak te rozwiązania trafią do produkcji należy dobrze wykorzystać te środki, które są obecnie dostępne.

Dla projektantów systemów, poza spełnieniem oczekiwań użytkowników, istotne jest również narzędziowe wsparcie, jakie mogą uzyskać stosując dane technologie. Tak więc obok

interesów użytkowników twórcy nowych technologii muszą brać pod uwagę interesy programistów. Ze względu na specyfikę problemu (tj. zasadę działania, wykorzystywane technologie, architekturę rozwiązań) tworzenie interfejsu użytkownika dla aplikacji sieciowych nie jest łatwe. Trudności pojawiają się już na etapie tworzenia makiet, aby urosnąć jeszcze przy próbach debuggowania i testowania. Aby je pokonać stosuje się różne techniki polegające na powiązaniu różnych technologii - tych, które umożliwiają łatwe projektowanie i testowanie interfejsów aplikacji typu desktop, oraz tych, które wykorzystywane są podczas wdrażania aplikacji sieciowych. Do takich technologii należy JavaScript. Niniejszy projekt inżynierski skupia się właśnie na wykorzystaniu tej technologii.

1.1. Cel i zakres projektu

Celem niniejszego projektu jest zbadanie możliwości technologii JavaScript oraz wykonanie implementacji graficznego interfejsu użytkownika aplikacji sieciowej oferującej dostęp do funkcji wybranego systemu.

Zakres projektu obejmuje:

1. Przygotowanie i wykonanie projektu informatycznego, a w nim:
 - dokonanie analizy wymagań,
 - implementacja logiki biznesowej aplikacji sieciowej w języku Java,
 - implementacja graficznego interfejsu użytkownika aplikacji z wykorzystaniem biblioteki SWT,
 - wygenerowanie strony HTML z zanurzoną skrypcem Javascript z wykorzystaniem technologii JavaScript (plugin do środowiska Eclipse),
 - przetestowanie działania serwisu,
 - opracowanie dokumentacji oraz instrukcji wdrożeniowej,
2. Dokonanie podsumowania wykonanych prac, a w nim opisanie zalet i wad wykorzystanej technologii.

Rozdział 2

Wymagania projektowe

Zrealizowany projekt został wykonany w oparciu o serwis EDAP na potrzeby pracy inżynierskiej na kierunku Informatyka na Politechnice Wrocławskiej. Projekt nosi nazwę LawCreator. Aplikacja stanowi narzędzie, mające na celu usprawnienie pisania aktów prawnych zgodnie ze schematem narzuconym przez ministerstwo.

Serwis EDAP [EDA] – elektroniczna forma aktów prawnych, został przygotowany przez MSWiA w związku z wejściem w życie rozporządzenia w sprawie wymagań technicznych dokumentów elektronicznych zawierających akty normatywne i inne akty prawne, elektronicznej formy dzienników urzędowych oraz środków komunikacji elektronicznej i informatycznych nośników danych (Dz. U. 2008 Nr 75, poz 451 z późn. zm.).

System EDAP pozwala na tworzenie oraz edycję aktów prawnych, wspomaga także zarządzanie oraz przechowywanie stworzonych dokumentów.

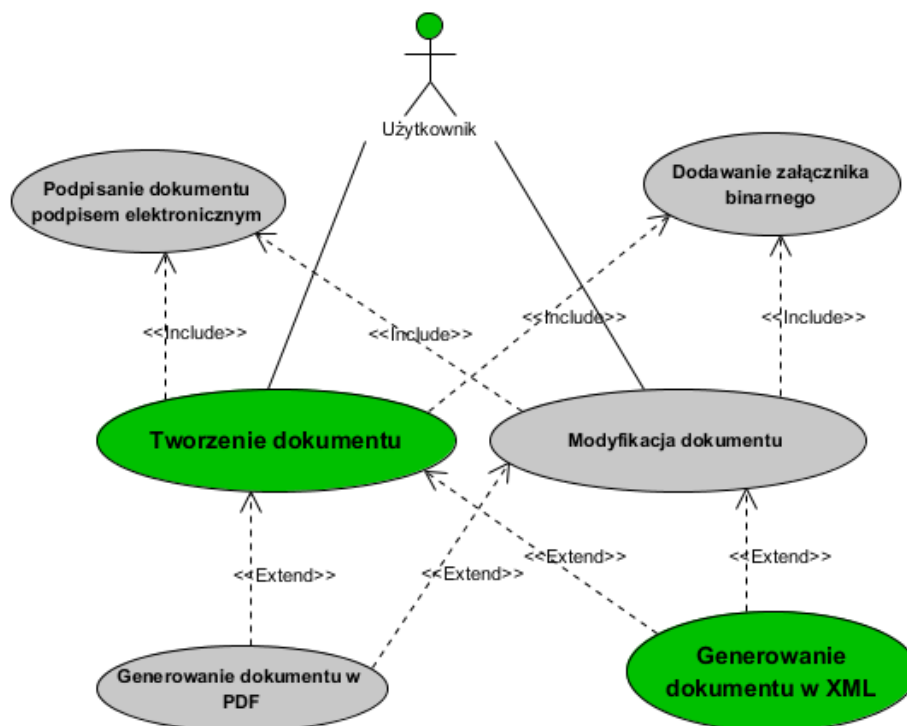
Akty prawne są generowane w postaci pliku z rozszerzeniem .xml oraz .pdf w oparciu o elektroniczny schemat aktów prawnych, przygotowany przez MSWiA w postaci pliku `schema.xsd` [SCH].

2.1. Wymagania funkcjonalne

Na rys. 2.1 został diagram przypadków użycia serwisu EDAP. Na zielono wyróżnione zostały funkcje systemu, które zostały zaimplementowane w LawCreator, a na szaro które nie zostały zrealizowane, m.in. ze względu na ograniczenia użytych technologii.

2.1.1. Wybór aktu prawnego

System umożliwia użytkownikowi utworzenie jednego z dokumentów, umieszczonych w Schemacie XML. Jest nim uchwała, która posiada takie elementy jak metryka, preambuła czy paragraf. Użytkownik włącza aplikację LawCreator, po czym tworzona jest w pamięci zawartość



Rys. 2.1: Diagram przypadków użycia dla użytkownika serwisu EDAP (opracowanie własne)

pliku XML generowanego aktu, a użytkownik przechodzi do jego edycji. Użytkownik ma możliwość konstrukcji dokumentu od nowa w przypadku, gdy wybierze zły rodzaj aktu lub postanowi zaniechać edycji już utworzonego dokumentu. W tym przypadku zawartość pliku XML jest wyczyszczona z pamięci.

2.1.2. Edycja aktu prawnego

Aplikacja daje możliwość wypełnienia aktu prawnego – uchwały. Edycja dokumentu jest zależna od schematu XML, ponieważ każda opcja modyfikacji aktu jest opisana w pliku `schema.xsd`. Użytkownik wybiera do modyfikacji jedynie dostępne opcje, co pozwala trzymać się reguł opisanych w schemacie.

2.1.3. Generowanie zawartości pliku XML

Serwis umożliwia wygenerowanie zawartości pliku XML zgodnego ze schematem `schema.xsd`. Użytkownik otrzymuje na wyjściu dokument w formacie XML z wypełnionymi wszystkimi polami, które wcześniej wypełnił w edytorze. Polem nazywamy tutaj zawartość elementu bądź jego atrybut wraz z wartością.

2.2. Wymagania niefunkcjonalne

System powinien prezentować użytkownikowi funkcjonalność, która pozwoli na szybkie i intuicyjne tworzenie skomplikowanych aktów prawnych.

Serwis powinien też zapewnić użytkownikowi wybór tworzenia aktu online lub offline. Narzędzie offline ma mieć postać aplikacji desktopowej, a konstruowanie aktów prawnych online ma odbywać się poprzez interfejs webowy, utworzony dzięki JavaScript.

Aplikacja LawCreator ma prezentować użytkownikowi przyjazny interfejs, który ułatwi mu korzystanie z takiego narzędzia, jak generator aktów prawnych. System w wersji offline pozwala na tworzenie dokumentów jednemu użytkownikowi na raz, natomiast wersja sieciowa aplikacji pozwala na korzystanie z jej funkcji wielu użytkownikom w tym samym czasie.

2.2.1. Środowisko pracy systemu

Wersja online programu LawCreator pozwala na działanie na każdym systemie operacyjnym, jednak ograniczeniem jest rodzaj przeglądarki, która będzie ten serwis prezentować - aplikacja ma działać w przeglądarkach, które są wspierane przez technologię J2S. Aplikacja w wersji desktopowej będzie mogła zostać uruchomiona na każdym systemie operacyjnym wspieranym przez bibliotekę SWT.

2.2.2. Wykorzystywane technologie

Poniżej zostały przedstawione języki programowania, języki znaczników, narzędzia wspomagające oraz technologie użyte w projekcie.

Java Główny język programowania, użyty w projekcie. Oprogramowana jest w nim cała logika biznesowa, opierająca się na wyświetleniu i obsłudze danych w oknie aplikacji oraz generowaniu rezultatu, jakim jest plik XML.

SWT Jest to biblioteka graficzna na platformę Java, która została użyta w systemie ActEditor do przedstawienia makiet interfejsu użytkownika. Została wybrana ze względu na kompatybilność z technologią JavaScript [J2Se]. Jest alternatywą dla takich rozwiązań jak AWT czy Swing.

Aby wyświetlić kontrolki, implementacja SWT sięga po natywne biblioteki systemu operacyjnego, korzystając z Java Native Interface (JNI). Ze względu na to, że SWT używa różnych bibliotek dla różnych platform, istnieje ryzyko wystąpienia błędów charakterystycznych dla danego systemu.

Według Eclipse Foundation [SWT] celem SWT, w odróżnieniu od Swing, było zapewnienie powszechnego API, które sięgałoby po natywne kontrolki systemu operacyjnego.

JavaScript Technologia szczegółowo opisana jest w rozdziale 3.

JavaScript Język skryptowy powszechnie implementowany w przeglądarkach internetowych. W stworzonej aplikacji do generowania aktów prawnych skrypty JavaScript są utworzone dzięki kompilatorowi Java2Script, a następnie zanurzone w HTML, dzięki czemu użytkownik uzyskuje interfejs aplikacji w postaci strony sieciowej. Szczegóły dotyczące tworzenia plików źródłowych JavaScript z klas Java są zawarte w rozdziale 3.

HTML HyperText Markup Language jest głównym językiem znaczników, służącym do wyświetlania stron internetowych w przeglądarkach.

Informacje na temat jak, gdzie i co ma być wyświetlane, zawarte są w znacznikach (tagach), zazwyczaj występujących w parach - tag otwierający i zamykający, a pomiędzy nimi zawarty jest tekst, komentarz lub inny typ tekstowy.

Język ten służy do budowania szkieletu strony, jednak konkretny wygląd i funkcjonalność serwisu oparta jest głównie na stylach CSS oraz skryptach JavaScript i PHP.

XML, XML Schema Technologie zostały opisane w rozdziale 4.

Eclipse IDE Środowisko, w którym został utworzony projekt. Jest to narzędzie, które wspiera technologię Java2Script [J2Sd].

Platforma została stworzona przez firmę IBM i udostępniona na zasadzie open-source. Nie dostarcza ona żadnych narzędzi służących do tworzenia kodu i budowania aplikacji, jednak wspiera obsługę wtyczek, które rozszerzają jej funkcjonalność.

2.3. Cykl życia projektu

Produkt został wytworzony przy użyciu modelu przyrostowego (iteracyjnego). W stosunku do modelu kaskadowego daje on o wiele większą elastyczność, dzięki powrotom do wstępnych faz projektu i pozwala reagować na opóźnienia, czyli przyspieszać prace nad innymi częściami projektu. Daje też elastyczność pod względem definiowania wymagań – możemy początkowo zdefiniować wstępne wymagania, podczas gdy końcowa specyfikacja może powstać dopiero na etapie testowania.

1. Definiowanie projektu

- a) Etap inicjowania projektu: inicjowanie projektu wymaga odpowiedzi na pytania: jaka jest tematyka przedsięwzięcia? Jakie są główne problemy zagadnienia? Jakie problemy ma rozwiązać projekt?

- b) Etap definiowania celu: definiowanie zagadnienia polega na wyspecyfikowaniu kluczowego problemu, który formułujemy jako cel główny. Identyfikuje się tutaj również cele cząstkowe, które muszą zostać zrealizowane, by osiągnąć cel końcowy, a także prognozuje się potencjalne przeszkody, które mogą wpłynąć na sukces projektu.
- c) Etap weryfikacji: w tym etapie ocenia się możliwość wykonania postanowionych wymagań oraz weryfikuje szanse na osiągnięcie sukcesu projektu.

2. Planowanie

Podsumowanie tego, co musi być wykonane w ramach projektu, aby osiągnąć cel. Tworzy się tu podział na podzadania, które są łatwe do zorganizowania i zarządzania.

3. Wykonanie planu projektu

- a) Organizacja kamieni milowych: ustalenie kolejności wykonywanych podzadań wraz z rozmieszczeniem ich w czasie. Specyfikowana jest tutaj kolejność wykonywania zadań oraz ich przewidywany czas wykonania.
- b) Wprowadzenie planu w życie: wykonanie wyspecyfikowanych zadań i dążenie do ich realizacji przed upływem ich szacowanego czasu trwania.
- c) Przetestowanie aplikacji: testowanie implementowanych części systemu. Najczęściej jest to etap iteracyjny, który jest wykonywany po zaimplementowaniu konkretnej funkcji w systemie.

4. Zamykanie projektu

- a) Ocena projektu: następuje tu ocena odpowiedniości, efektywności, skuteczności projektu. Jest to przegląd osiągnięć realizowanego projektu w stosunku do początkowych oczekiwań. Odpowiada na pytania: czy rezultaty projektu były zgodne z oczekiwaniami? Czy prace zostały wykonane zgodnie z planem?
- b) Wnioski: wyciągnięcie wniosków z realizacji projektu. Opis doświadczeń zdobytych w trakcie wykonywania zadania, które mają służyć poprawie przyszłych projektów i programów.

Rozdział 3

Java2Script

3.1. Analiza technologii

Java2Script (J2S) Pacemaker [J2Sa] jest wtyczką typu open-source do środowiska Eclipse, która pozwala na konwertowanie klas języka Java do JavaScript. Wtyczka opiera się na zorientowanym obiektowo symulatorze JavaScript, który został nazwany Java2Script (J2S)Clazz. Zapewnia on pełne wsparcie OOP, włączając dziedziczenie i polimorfizm.

Głównym pomysłodawcą i konstruktorem wtyczki jest Zhou Renjian, który od 2005 roku starał się stworzyć kompilator, który mógłby skonwertować kod języka Java do JavaScript. Pomysł zrodził się dlatego, że praca z plikami źródłowymi JS bywa koszmarem, gdy osiągają one większy rozmiar, a także trudno nimi zarządzać, ze względu na brak profesjonalnych narzędzi, w porównaniu do Eclipse i Visual Studio. Brak jest też zaawansowanych narzędzi do testowania JavaScriptu, jak i jego debuggowania.

Główne funkcje i właściwości J2S [J2Sb]:

- Polimorfizm,
- Dziedziczenie,
- Implementacja interfejsów,
- Zapewnia generowanie klas anonimowych,
- Zapewnia poprawne działanie zmiennych finalnych,
- Zapewnia poprawne działanie typów klas.

3.2. Zasada działania

Zasadę działania wtyczki w porównaniu do normalnego kompilatora Java pokazuje rys. 3.1 Po zainstalowaniu J2S możemy wybrać, czy chcemy skompilować projekt używając standardowego kompilatora, czy kompilatora Java2Script. Gdy wybierzemy tę drugą opcję, J2S utworzy do każdej klasy `.java` plik `.js`. Zostanie też wygenerowany plik `.html` do każdej klasy, posiadającej metodę `main()`. Do tego pliku dołączony zostaje powstały skrypt `.js`, który w swoim ciele może już odwoływać się do innych powstałych plików źródłowych JavaScript. Plik w formacie HTML otwieramy w przeglądarce internetowej i korzystamy z funkcjonalnej aplikacji sieciowej jak ze strony internetowej.

Twórca wtyczki ocenia, że dzięki środowisku Eclipse z dołączoną wtyczką J2S można poprawnie przetłumaczyć nawet 90% plików źródłowych Java [J2Sb]. Niestety występują pewne rozbieżności, między tymi dwoma językami, których nie da się rozwiązać. Użytkownicy wtyczki muszą np. pamiętać, że pola statyczne klasy mogą nie kompilować się poprawnie, typy prymitywne: `char`, `byte`, `short`, `int`, `long`, `float`, `double` są traktowane jako `Number`, a także brak jest wsparcia dla wątków czy asynchroniczności.

3.3. Java2Script a Java

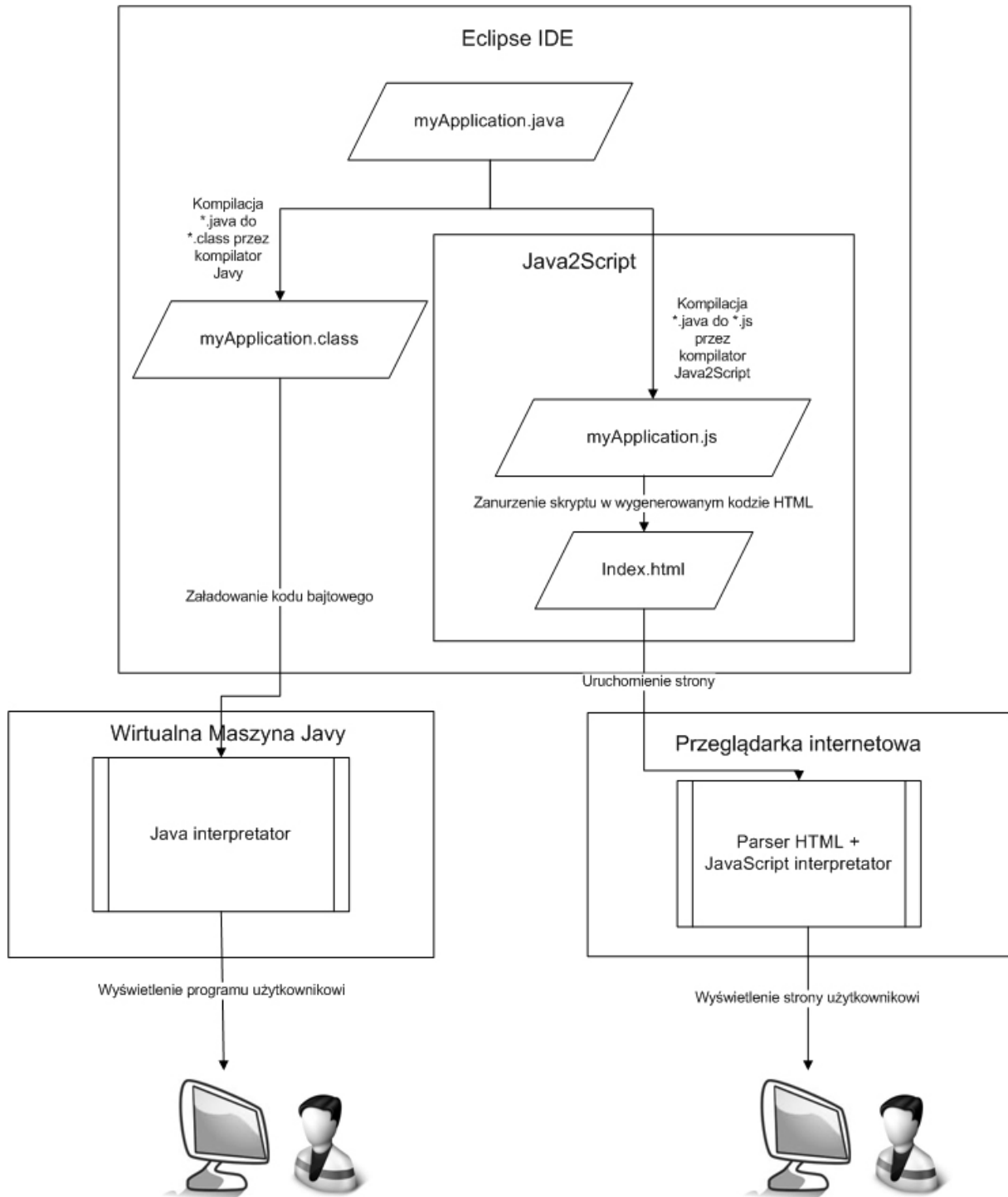
Biorąc pod uwagę to, że stosowanie się do zasad programowania obiektowego, które znamy z platformy Java, w JavaScript jest trudne i frustrujące, autor Java2Script dążył do tego, aby móc pisać aplikacje w języku w pełni wspierającym OOP, jakim jest Java i użyć go do tworzenia zorientowanych obiektowo skryptów JS.

Jednak ze względu na różnice pomiędzy językiem Java a JavaScript, wyniknęły pewne ograniczenia stosowania tej technologii. J2S wspiera wyłącznie klasy z pakietów: `java.io`, `java.util`, `java.lang`, `java.lang.reflect`, `junit`, `org.eclipse.swt` [J2Sc]. Nie można mieć jednak pewności, wiedząc że dana klasa należy do jednego z wymienionych pakietów, że klasa ta ma wsparcie Java2Script. Należy się upewnić, że widnieje w spisie wspieranych klas, zamieszczonym na stronie głównej J2S.

3.4. System dziedziczenia w zanurzonem JavaScript

Technologia J2S ma na celu konwersję klas środowiska Java do skryptów JavaScript. Autor wtyczki zaznacza różnice między podejściem do programowania obiektowego w przypadku natywnego JavaScript w porównaniu do tego, jak OOP jest realizowane na platformie Java [J2Sb]. Podstawowe różnice to:

- brak idei klas i metod abstrakcyjnych,



Rys. 3.1: Zasada działania JavaScript (opracowanie własne)

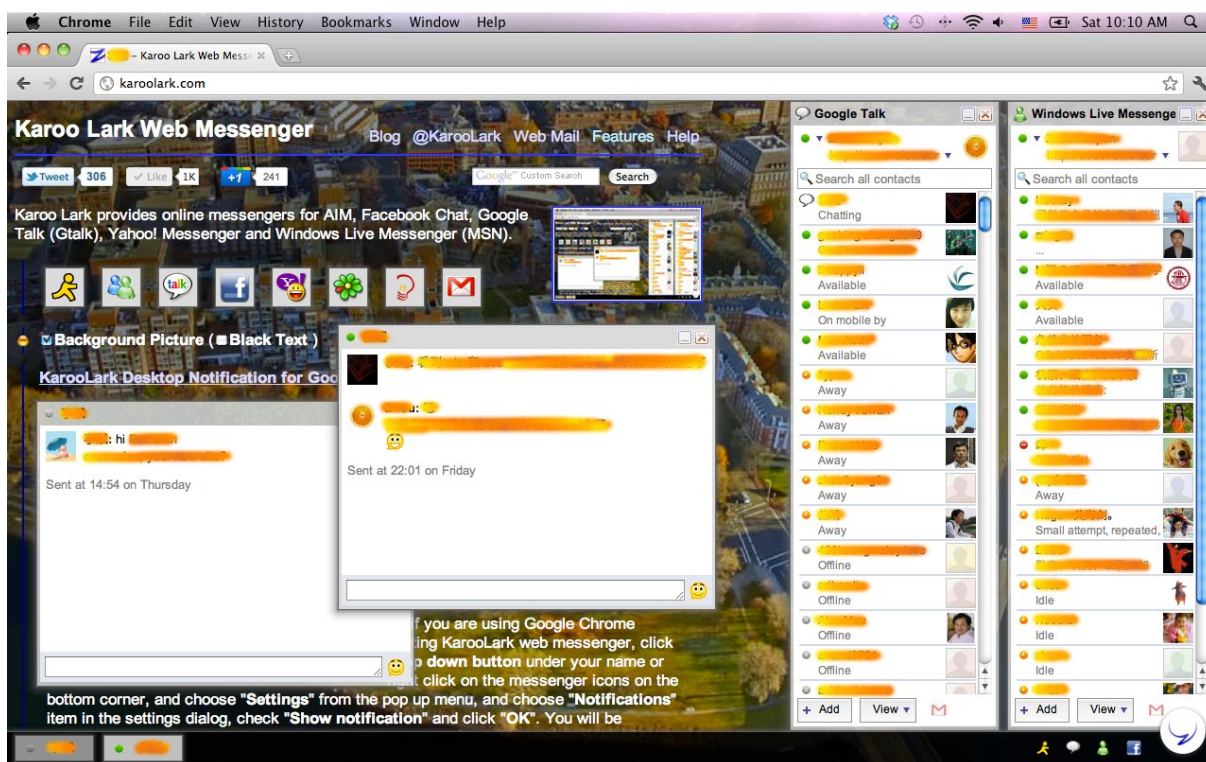
- idea konstruktorów w JavaScript jest bardzo zaciemniona, chyba że założymy, że każda funkcja może być rozpatrywana jako konstruktor,
- metody mogą być łatwo przeciążane bez wsparcia polimorfizmu,

- w przypadku przeciążenia nie ma jednak możliwości odwołania się do metody nadrzędnej,
- brak kontroli nad dostępem do pól obiektu,
- występowanie bibliotek JavaScript jest rzadkością,
- JavaScript pozwala na bezpośrednią konwersję bardzo wielu typów.

3.5. Przegląd istniejących serwisów, w których wykorzystano Java2Script

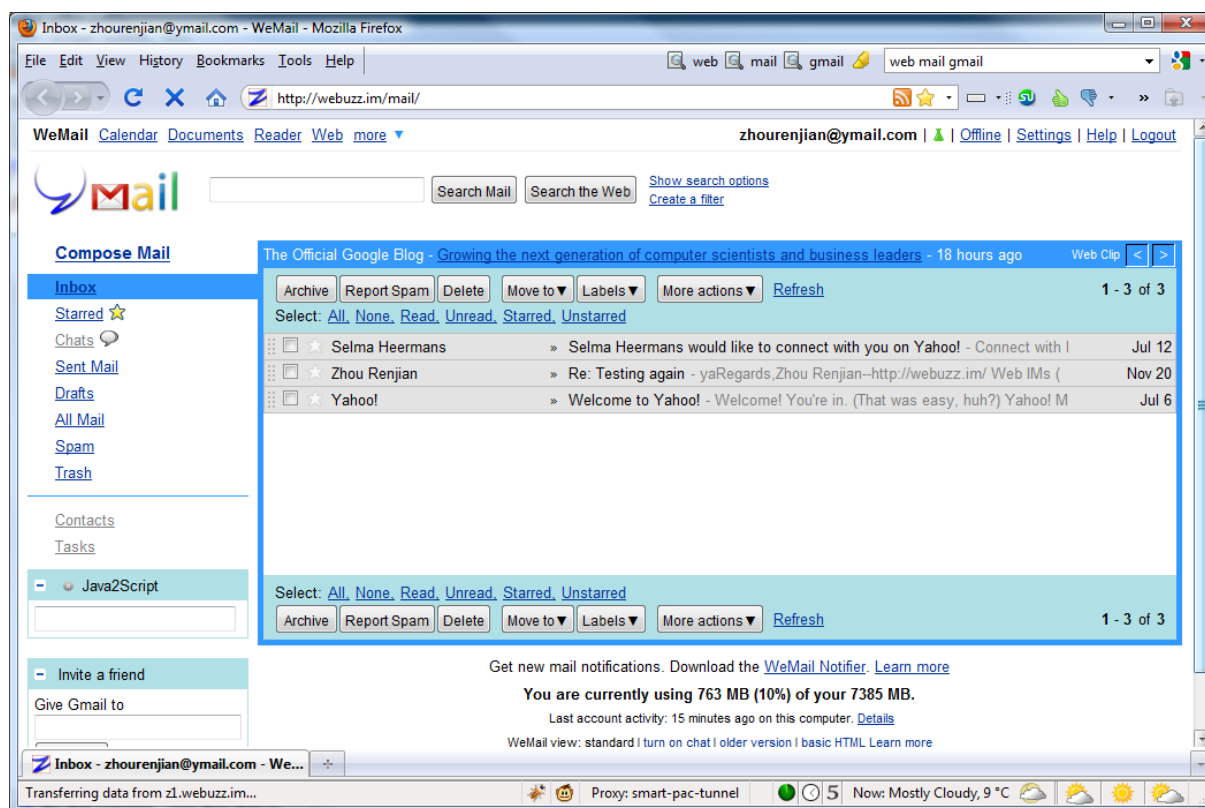
Karoo Lark Web Messenger

Rys. 3.2 zawiera zrzut ekranu z interfejsem użytkownika aplikacji. Serwis udostępnia wersje



Rys. 3.2: Interfejs użytkownika aplikacji Karoo Lark Web Messenger
(<http://karoollark.com/images/karoollark-web-messenger.jpg> dostęp z dnia 5 grudnia 2012r.)

online słynnych komunikatorów, takich jak Facebook Chat, Google Talk czy Windows Live Messenger (MSN). Głównymi zaletami takiego rozwiązania są: łatwość wyboru komunikatorów, których chcemy używać oraz łatwe przełączanie się między nimi. Jednak aplikacja posiada również inną zaletę. Wyobraźmy sobie sytuację, gdy jesteśmy uczniem lub studentem i korzystamy z komputera należącego do jednostki edukacyjnej. Niektóre z nich blokują usługi



Rys. 3.3: Interfejs użytkownika aplikacji Lemon Dove Web Mail Client (<http://lemondove.com/images/dove.png> dostęp z dnia 5 grudnia 2012r.)

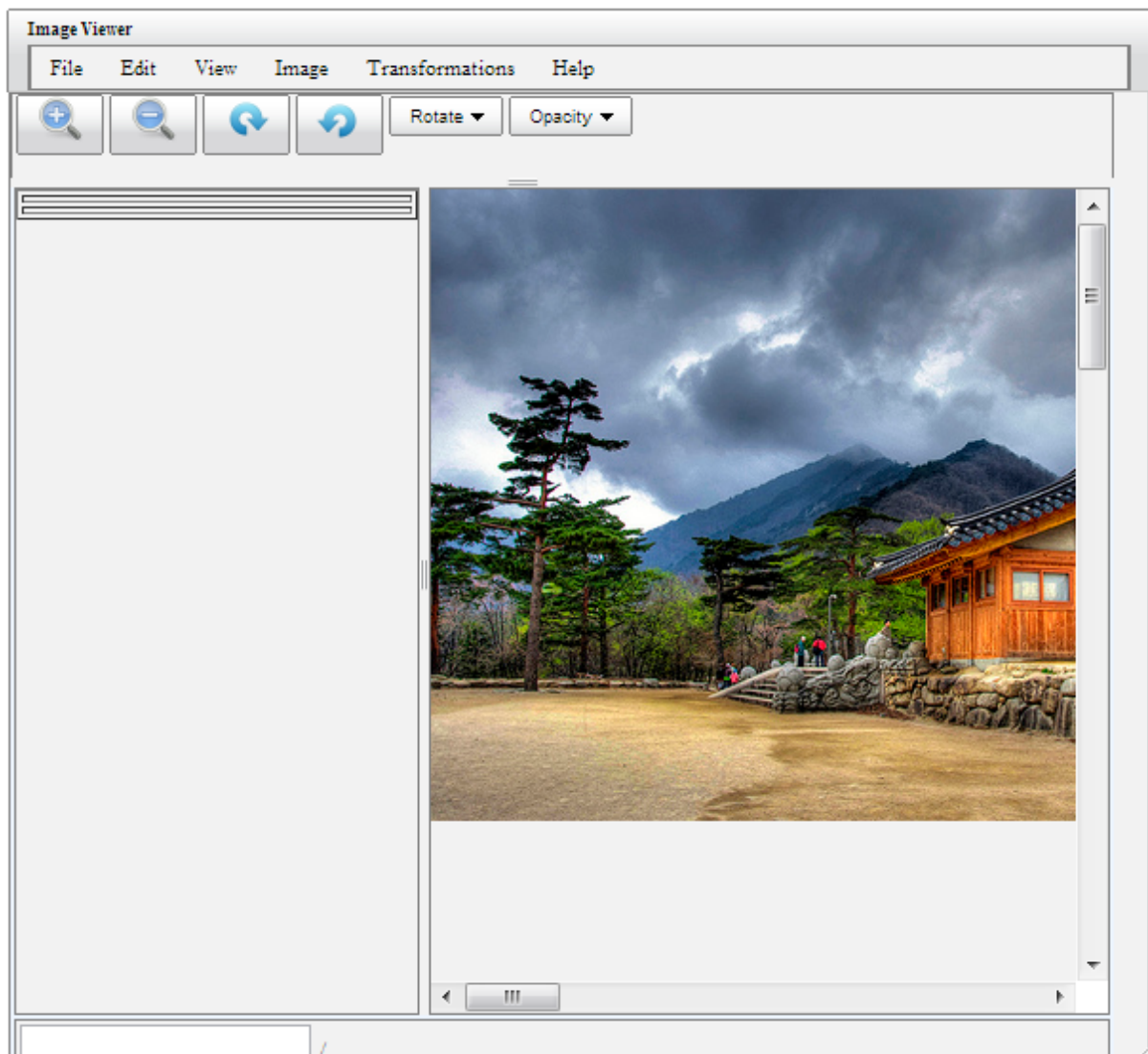
komunikatorów przy użyciu zapór ogniowych. Jednak zazwyczaj nie blokują one stron internetowych, takich jak karoolark.com. Dzięki temu serwisowi możemy bez obaw zalogować się do interesujących nas komunikatorów.

Lemon Dove Web Mail Client

Serwis ten jest darmowym klientem poczty elektronicznej. Zapewnia połączenie z najpopularniejszymi skrzynkami pocztowymi, takimi jak Google Mail, Hotmail, Yahoo! Mail, czy AOL/AIM Mail. Główną zaletą tego rozwiązania jest to, że możemy zalogować się do konkretnej skrzynki pomimo zapory ogniowej, która może blokować taką usługę. Serwis zaprojektowany jest z myślą o uruchamianiu go na różnych systemach operacyjnych i w różnych sytuacjach - czy w domu, czy w biurze z włączoną zaporą. Zapewnia też dość intuicyjne korzystanie z aplikacji dla użytkowników poczty Gmail, ze względu na podobieństwo interfejsu, co widać na rys. 3.3.

Advanced Image Viewer

Aplikacja pozwala użytkownikowi załadować obraz z Internetu, obejrzeć i manipulować nim online. Wspierane są takie operacje, jak obrót, zbliżenie, import, eksport, zaawansowane filtry



Rys. 3.4: Interfejs użytkownika aplikacji Advanced Image Viewer (opracowanie własne)

(np. rozmycie), wykrywanie krawędzi i wiele innych. Aplikacja przeznaczona jest dla użytkowników, którzy chcą w sposób zaawansowany manipulować danym obrazem w pełni po stronie klienta serwisu. W odróżnieniu do większości projektów z użyciem J2S, nie używa biblioteki SWT do wyświetlenia interfejsu. Używa natomiast YUI z frameworkiem Raphaël – rys. 3.4 przedstawia interfejs użytkownika aplikacji Advanced Image Viewer.

Rozdział 4

XML i Schemat XML

4.1. XML

XML to prosty i elastyczny język znaczników, który został zaprojektowany z myślą o tym, by stawić czoła reprezentacji danych w strukturalizowany sposób [XML]. Wywodzi się z technologii SGML, która jednak była o wiele bardziej zawiła, a zadaniem, jakie postawili sobie twórcy języka XML, było uproszczenie SGML-a tam, gdzie się tylko dało.

Język XML oferuje nam m.in. sformalizowany zapis informacji, pozwala zapisać wszystko, co może być wyrażone tekstem, a dzięki regularnej składni i przyjętym standardom – przejrzystość dla człowieka oraz prostotę przetwarzania dla komputera. Wszelkiego typu parsery (procesory) XML są już zaimplementowane w wielu programach (np. przeglądarkach internetowych).

Aby utworzyć poprawny dokument XML, należy trzymać się określonych reguł. Poprawny składniowo dokument powinien np. zawierać deklarację XML, która musi być umieszczona na samym początku pliku. Musi też zawierać dokładnie jeden element główny, zwany korzeniem. Wszelkie elementy, które zawierają dane, muszą być otoczone dwoma znacznikami - początkowym i końcowym, natomiast elementy puste stanowią wyjątek i posiadają tylko jeden tag.

XML znajduje swoje zastosowanie w wielu dziedzinach informatyki [PLO]. Jedną z nich jest Internet. Choć na dzień dzisiejszy panuje tutaj HTML, to istnieją jeszcze takie rozwiązania jak WML oraz XHTML, które są dialektami XML.

Za pomocą XML-a możemy również przyspieszyć wyszukiwanie informacji w Sieci, ponieważ umożliwia on przedstawienie zasobów dostępnych w Internecie. Możemy tu wymienić takie technologie jak RDF, OWL.

XML ułatwia nam też konstruowanie dokumentów, które mają pewną, określoną strukturę, lecz bywa ona zmienna lub słabo ustalona. W takim przypadku XML radzi sobie o wiele lepiej z reprezentacją danych w porównaniu z relacyjnymi bazami danych.

Można stwierdzić, że XML znalazł już swoje miejsce w informatyce, a jako rozwiązanie dość młode, jest stale rozbudowywany. Zmiany te odbijają się w pewnym stopniu na twórcach implementacji opartych na XML, jednak nie są one na tyle radykalne, by było to większym problemem.

4.2. Schemat XML

Reguły, które precyzują tworzenie dokumentów XML, bywają niewystarczające dla użytkowników, którzy chcieliby narzucić własne zasady budowania dokumentu XML. Do tego celu służą takie technologie jak Document Type Definition (DTD), Simple Object XML (SOX) czy Schemat XML (XSD).

Schemat XML [Tra] pozwala na formalne opisanie elementów, które mają znaleźć się w dokumencie XML, a także związki między elementami i atrybutami. Opis ten umożliwia sprawdzenie poprawności (walidacji) generowanego dokumentu.

Schematy umożliwiają precyzyjne deklarowanie typów danych, pozwalają na definiowanie fragmentów modelu, które później mogą być wiele razy wykorzystywane w innych jego fragmentach, umożliwiają zdefiniowanie zbiorów elementów wraz z określeniem rodzaju tego zbioru (czy kolejność elementów jest ważna, czy wszystkie elementy muszą być użyte) i ilości jego wystąpień.

4.3. Parsowanie dokumentów XML

Istnieją dwa podejścia do implementacji parserów, na których opiera się operowanie na dokumentach XML. Są nimi DOM i SAX.

Parser DOM [DOM] dokonuje przekształcenia dokumentu XML w drzewo DOM. Każdy element lub atrybut XML jest węzłem w tym drzewie i posiada powiązany z nim interfejs. Każde drzewo DOM zawiera korzeń (węzeł główny), który jest nadrzędny w stosunku do pozostałych obiektów. Dostępne wersje parserów DOM udostępniają użytkownikom interfejsy, które można implementować tak, by uzyskać pożądany efekt.

Metoda SAX [SAX] polega na wczytywaniu dokumentu XML znacznik po znaczniku, przy każdym z nich zgłaszając zdarzenie, które należy obsłużyć. Parsery SAX nigdy nie tworzą struktury drzewa dokumentu XML, jednak użytkownik może oprogramować obsługi zdarzeń, które są zgłaszane przez parser, w taki sposób, aby takie drzewo utworzyć.

W obu przypadkach użytkownik może dostosować implementację parsera do własnych potrzeb. W parserze DOM powinien implementować interfejsy powiązane z konkretnymi węzłami, natomiast w parserze SAX powinien implementować obsługę zgłaszanych zdarzeń.

Rozdział 5

Opis cyklu życia projektu

5.1. Pierwsza iteracja – wyszukiwarka semantyczna

1. Definiowanie projektu

- a) Etap inicjowania projektu: pierwsza iteracja projektu dotyczyła implementacji graficznego interfejsu użytkownika aplikacji sieciowej, jaką był serwis semWeb. Tematyką projektu semWeb było wykorzystanie metadanych i sieci semantycznych do wyszukiwania, integracji i weryfikacji urzędowych danych posiadających aspekt czasowo-przestrzenny.
- b) Etap definiowania celu: celem projektu miało być stworzenie interfejsu aplikacji sieciowej, która posiadała już pełną funkcjonalność. Interfejs miał łączyć się z bazowym systemem, jakim był semWeb. Dostarczał on endpoint, umożliwiający zarządzanie dziedzinową bazą wiedzy. Serwis bazowy był oparty na fasadowym wzorcu projektowym, co było jednym z powodów, dla którego został on wybrany jako potencjalna baza. Wzorzec fasadowy zapewniał łatwość integracji modułu interfejsu z resztą aplikacji.

Potencjalnymi przeszkodami w rozwoju projektu były problemy z importem plików o rozszerzeniu `.jar` do projektu. Powodem tego jest obecna implementacja wtyczki JavaScript, która opiera się na wczytywaniu skompilowanych klas środowiska Java, ale tylko tych, które umieszczone są bezpośrednio w projekcie, lub należą do konkretnych, wbudowanych bibliotek, jak `java.lang`.

- c) Etap weryfikacji: z uwagi na to, że nie dało się dołączyć plików `.jar` do projektu w taki sposób, by kompilator JavaScript mógł prawidłowo załadować klasy oraz skonwertować je do plików z rozszerzeniem `.js`, projekt nie przeszedł pozytywnie przez fazę weryfikacji i został odrzucony.

Brak możliwości dołączania zewnętrznych archiwów `.jar` oraz wbudowanych bibliotek, takich jak `java.sql`, uniemożliwiło połączenie się z bazą danych. Konsekwencją tego był

brak możliwości zbudowania aplikacji, która spełniałaby minimalne wymagania zawarte w etapie definiowania projektu.

Rozwiązaniem mogłoby być zbudowanie aplikacji, która zajmowałaby się jedynie obsługą połączenia z bazą danych, jednak problemy ze zbudowaniem komunikacji między aplikacjami przeważały o niepowodzeniu tej części projektu.

W związku z tym kolejnym krokiem w cyklu życia projektu było przejście do kolejnej iteracji i zdefiniowanie nowego problemu.

5.2. Druga iteracja – ActEditor

1. Definiowanie projektu

- a) Etap inicjowania projektu: druga wersja aplikacji miała dotyczyć implementacji graficznego interfejsu użytkownika aplikacji sieciowej, jaką był serwis EDAP. Jego tematyką było tworzenie dokumentów prawnych, ich edycja oraz zarządzanie już istniejącymi dokumentami.

Projekt ułatwia skonstruowanie wybranego dokumentu prawnego zgodnie ze schematem narzuconym przez MSWiA. Dokument generowany jest w postaci pliku o formacie XML, a wszelkie obostrzenia i zasady jego konstruowania zawiera plik `schema.xsd`. Używając serwisu EDAP, użytkownik może także wczytać już istniejący plik z rozszerzeniem `.xml` i zmodyfikować go.

- b) Etap definiowania celu: celem projektu było stworzenie aplikacji sieciowej, zawierającej główną funkcję serwisu EDAP, czyli konstruowanie dokumentu prawnego w postaci pliku XML na podstawie schematu zawartego w pliku `schema.xsd`. Aplikacja powinna posiadać mechanizm wczytywania danych ze schematu XML oraz takiego ich przetworzenia, by w razie zmian w schemacie, nie ingerowałoby to w stabilność i niezawodność aplikacji.

Aby osiągnąć cel główny, należało wykonać 3 podstawowe kroki: odpowiednio wczytać zawartość pliku ze schematem XML, następnie stworzyć interfejs użytkownika, który wyświetlałby dostępne pozycje do wypełnienia przez użytkownika, z których potem generowany byłby plik XML.

Głównymi potencjalnymi problemami były operacje na plikach, takie jak wczytywanie i generowanie, które nie są wspierane przez JavaScript. Oprócz tego problemem mogło się okazać odpowiednie wczytanie zawartości schematu XML, gdyż nie można było skorzystać z zewnętrznych bibliotek, które by to ułatwiały. Powodem tego był brak możliwości dołączenia takich bibliotek w taki sposób, by JavaScript mógł je załadować.

- c) Etap weryfikacji: projekt przeszedł pozytywnie etap weryfikacji z uwagi na realną możliwość stworzenia aplikacji, której celem byłoby skonstruowanie dokumentu prawnego w postaci pliku XML.

Projekt zakłada jedynie generowanie takiego pliku, nie jego modyfikację. Kolejnym założeniem jest brak generowania samego pliku XML, a jedynie wyświetlenie jego zawartości w aplikacji. Oba założenia wynikają z braku możliwości operacji na plikach, które nie są wspierane przez wtyczkę JavaScript.

2. Planowanie

Aby osiągnąć cel, należało wykonać trzy podstawowe podzadania. Należały do nich:

- a) wczytanie schematu XML, który jest odpowiedzialny za dobrze sformatowany plik XML;
- b) konwersja wczytanych danych na obiekty języka Java;
- c) zbudowanie graficznego interfejsu użytkownika, prezentującego formularze do wypełniania ciała dokumentu;
- d) zaimplementowanie generowania zawartości pliku XML na podstawie wypełnionych formularzy.

3. Wykonanie planu projektu

- a) Organizacja kamieni milowych: pierwsza iteracja zajęła sporą część czasu, przeznaczanego na wykonanie projektu, stąd został jedynie miesiąc na wykonanie drugiego planu.

Organizacja harmonogramu wyglądała następująco: najwięcej czasu – ok. 2 tygodnie – zostało przeznaczone na pierwsze dwa podzadania, jakimi były wczytanie schematu XML oraz jego odpowiednia konwersja. Kolejny tydzień został przewidziany na zbudowanie interfejsu i generatora zawartości pliku XML. Ostatni tydzień został przeznaczony na testowanie oraz dokumentację.

- b) Wprowadzenie planu w życie: plan zakładał początkowo skupienie się na odpowiednim wczytaniu danych z pliku `schema.xsd`. Początkowo ładowanie znaczników odbywało się za pomocą rzędu instrukcji `if/else` i sprawdzano, czy w danej linii wystąpiło dane słowo kluczowe. Jednak po bardzo szybkim wzroście liczby linii kodu oraz ogólnym jego zaciemnieniu, zdecydowano, że lepszym rozwiązaniem będzie wywoływanie zdarzenia na podstawie znacznika z nim związanego, które następnie ma zostać obsłużone. Metoda ta nosi nazwę SAX (Simple API to XML) i jest opisana w punkcie 4.3.

Głównym celem wczytywania danych było utworzenie konkretnych obiektów XML według znaczników z nimi związanych. Ponieważ schemat XML zawiera wiele zagnieżdżeń znaczników, a zawartość jest przetwarzana linia po linii, stąd wynikła potrzeba pamiętania znacznika nadrzędnego, do którego należą znaczniki potomne w danej chwili przetwarzane. Rozwiązaniem tego problemu było zastosowanie wbudowanej w języku Java kolekcji `Stack`. Wywołana obsługa zdarzenia mogła mieć jedną z trzech możliwości: stworzyć obiekt rodzica i dołożyć go na wierzch stosu, stworzyć obiekt dziecka, odłożyć

obiekt ze stosu, dołączyć do niego dziecko i dołożyć obiekt rodzica z powrotem na stos, a przy znacznikach zamykających odłożyć obiekt ze stosu i dodać do rodzica lub rejestru.

Istnieją trzy rejestry, których rolą jest zapamiętanie konkretnych obiektów według ich nazw. Rejestry mogły przechowywać dane o elementach struktury (znacznikach), ich atrybutach oraz typach. Z powodu licznych referencji do obiektów, których definicje znajdowały się w różnych miejscach w dokumencie `schema.xsd`, rejestry pozwalały odtworzyć później taką strukturę. Odbywało się to dzięki temu, że z rejestru można było pobrać obiekt definicji po jego nazwie. Gdy pojawił się znacznik, mówiący o referencji do innego obiektu (deklarujący jego wystąpienie), konstruowana była jedynie namiastka o danej nazwie, a po wczytaniu całego pliku, namiastki te były uzupełniane, wczytując obiekt z rejestru.

Zastosowane podejście miało tę wadę, że uzupełnienie namiastek musiało zostać ograniczone do ilości poziomów, ponieważ pewne elementy struktury (np. litera, punkt, tiret), mogły zawierać w sobie referencje do samych siebie, co powodowało nieskończony poziom zagnieżdżenia. Wynikające trudności zostały obsłużone przez wprowadzenie maksymalnego poziomu zagnieżdżenia takiego drzewa.

Podczas wczytywania Schematu XML generowane były zdarzenia, które następnie były obsługiwane przez konkretne metody. Generowanie zdarzenia odbywało się na podstawie słów kluczowych, zawartych w jednej linii dokumentu `schema.xsd`. Jeśli dane słowa kluczowe zostały zawarte we wczytywanej linii, zwracano identyfikator zdarzenia, a następnie według wartości identyfikatora była wybierana metoda do jego obsługi.

Trzeba wspomnieć, że tego rodzaju parser jest już z powodzeniem zaimplementowany na platformę Java, jednak w przypadku ActEditor nie mógł zostać użyty. Powodem był brak możliwości dołączenia do projektu plików `.jar`, ponieważ ich import nie jest wspierany przez technologię Java2Script. Przyczyny takiego stanu rzeczy będą opisane w rozdziale podsumowanie. Zapadła więc decyzja skonstruowania własnego parsera dokumentu XML, który mógłby wczytać jego zawartość do pamięci programu. Ponieważ uniwersalny procesor przetwarzający dokumenty XML jest bardzo rozbudowany, w przypadku ActEditor obsługiwał on tylko niektóre elementy, a całość była okrojona i dostosowana do zawartości pliku `schema.xsd`.

Ostatecznie implementacja parsera Schematu XML zajęła łącznie około 700 linii kodu, a i tak nie obsługiwała wszystkich zdarzeń. Problemy, które cały czas pojawiały się w trakcie pracy z dokumentem `schema.xsd`, bardzo wydłużyły planowany czas realizacji zadania, jakim było jego wczytanie do pamięci programu.

Po zaimplementowaniu parsera przystąpiono do budowy interfejsu, którego implementacja również nie była najprostsza. Użytkownik miał do dyspozycji drzewo DOM dokumentu XML, zawierającego możliwe do wypełnienia tagi. Po naciśnięciu na dany element drzewa

makieta interfejsu zostawała uaktualniona i pojawiał się formularz, pozwalający na wypełnienie znacznika. Po zatwierdzeniu zmian, znacznik zostawał dodany jako element aktu prawnego, który miał później być wygenerowany w postaci zawartości dokumentu XML. Trudnością tutaj okazało się to, że Schemat XML nie opisuje w sposób ścisły i spójny wszystkich elementów drzewa DOM. Dodatkowo okrojona wersja parsera nie pozwalała na uzyskanie wszystkich pożądaných efektów, jakimi były np.: wstawienie zagnieżdżonego znacznika w dowolnym miejscu zawartości znacznika nadrzędnego, czy ograniczenie ilości wystąpień danego znacznika w dokumencie.

Problemy ze zbudowaniem w pełni funkcjonalnej aplikacji spowodowały, że podjęto decyzję o zmianie podejścia do tematu generowania aktów prawnych. Głównymi przyczynami takiego stanu rzeczy były:

- brak możliwości dołączenia gotowych implementacji procesora dokumentów XML, ze względu na ograniczoną funkcjonalność JavaScript,
- zbudowanie uniwersalnego parsera dokumentów XML jest czasochłonne,
- nieścisłości w strukturze Schematu XML utrudniają jego wczytanie,
- automatyzacja wyświetlania formularzy względem wczytanej zawartości Schematu XML jest bardzo trudna, ze względu na konstrukcję takiego schematu.

5.3. Trzecia iteracja – LawCreator

1. Definiowanie projektu

- a) Etap inicjowania projektu: ostateczna wersja aplikacji dotyczyła zbudowania aplikacji do generowania aktu prawnego, jakim jest uchwała.
- b) Etap definiowania celu: aplikacja LawCreator jest okrojoną wersją serwisu ActEditor, która pozwala na wygenerowanie uchwały w postaci zawartości dokumentu XML. Konstruowanie takiego aktu opiera się o zasady i reguły, opisane w `schema.xsd`.

Głównym potencjalnym problemem było ograniczenie czasowe, którego przyczyną były narzuty czasowe, powstałe w poprzednich iteracjach.

- c) Etap weryfikacji: projekt przeszedł pozytywnie etap weryfikacji z uwagi na to, że część programu została już zaimplementowana w poprzedniej iteracji, a zmiana podejścia do tematu generowania aktów prawnych, czyli skupienie się na skonstruowaniu edytora do tylko jednego rodzaju takiego aktu, jakim była uchwała, miała na celu przyspieszenie prac i zakończenie sukcesem całego projektu.

2. Planowanie

Ostateczna wersja wymagań została zamieszczona w punkcie 2. Podział projektu na podzadania wyglądał następująco – należało zbudować makietę interfejsu użytkownika aplikacji, która ułatwiałaby nawigowanie po jej funkcjach, zbudować formularze dla konkretnych elementów ustawy oraz zaimplementować parser, który wczytane dane z formularzy przekonywałoby na dokument XML.

3. Wykonanie planu projektu

- a) Organizacja kamieni milowych: z uwagi na ograniczenia czasowe, nie zdecydowano się na określenie konkretnych ram czasowych dla poszczególnych pozadań, ponieważ każde zadanie musiało zostać wykonane najszybciej, jak było to możliwe.
- b) Wprowadzenie planu w życie: szczegóły implementacyjne zostały umieszczone w rozdziale 6.
- c) Przetestowanie aplikacji: opis metod testowania oraz użytych do tego technologii został zawarty w rozdziale 7.

4. Zamykanie projektu

- a) Ocena projektu oraz podsumowanie zostały umieszczone w rozdziale 8.

Rozdział 6

Opis implementacji

Projekt został podzielony na moduły, których klasy miały podobne przeznaczenie. Moduły, które wyodrębniono, to:

- `pl.wroc.pwr.student.lawcreator` – główny pakiet, który zawiera klasę `Application`, odpowiedzialną za rozruch aplikacji,
- `pl.wroc.pwr.student.lawcreator.registry` – zawiera rejestr elementów struktury dokumentu,
- `pl.wroc.pwr.student.lawcreator.utils` – zawiera klasę `Mapper`, odpowiedzialną za mapowanie znaczników,
- `pl.wroc.pwr.student.lawcreator.view` – odpowiedzialny za wyświetlanie makiet interfejsu użytkownika,
- `pl.wroc.pwr.student.lawcreator.xml` – posiada klasę `Parser`, która zajmuje się przetwarzaniem elementów XML; pakiet zawiera też interfejsy i implementację interfejsów skojarzonych z węzłami dokumentu XML.

W odróżnieniu od sposobu implementacji w iteracji drugiej (rozdział 5.2), dostępne elementy dokumentu XML nie zostały wczytane ze schematu, a wprowadzone za pomocą metody `createElements()` w klasie `Parser`. Do uzupełnienia informacji o elementach struktury służą natomiast metody `createAttribute()` oraz `createType()`. Dzięki temu uzyskano pełny zestaw informacji o danym węźle, na który składały się informacje o elemencie, jego atrybutach i typach tych atrybutów.

Zmianę podejścia do uzupełnienia danych wymusiło to, że schemat XML zawierał komentarze, które też niosły ze sobą informacje o zasadach tworzenia dokumentu. Mówiły np. o tym, czy dany element może posiadać ciało (innymi słowy – może nieść pewną treść między znacznikami), czy też nie. Nowe podejście pozwoliło na rozwiązanie tego problemu.

Podobnie jednak do poprzedniej iteracji, istnieje rejestr w postaci klasy `Registry`. Ułatwia to rozwiązanie problemu referencji do elementów. W trakcie tworzenia elementów w metodzie `createElements()` każda definicja elementu jest zapisywana w rejestrze, stąd jeśli któryś element posiada referencję do któregoś z nich, może go pobrać z rejestru. Dzięki zastosowaniu rejestru możemy też łatwo odbudować strukturę drzewa dokumentu.

Zarówno element, atrybut, jak i typ dokumentu XML posiadają swoje interfejsy, którymi są odpowiednio `Element`, `Attribute` oraz `Type`, a ich implementacje zawierają dodatkowo słowo `Impl` (czyli implementacją interfejsu `Type` jest `TypeImpl`).

Istnieje istotne rozróżnienie na obiekty pochodzące ze schematu XML i te, które mają znaleźć się w wygenerowanym dokumencie XML. Obiekty, reprezentujące zawartość schematu XML, są wcześniej wymienionymi implementacjami interfejsów i niosą ze sobą informacje jedynie o strukturze, nie zaś o tym, co ma być umieszczone jako wartość znacznika czy atrybutu. Pozwalają na poprawne wyświetlenie formularzy, które następnie wypełnia użytkownik. Natomiast obiekty, które mają nieść informacje, które będą zawarte w wygenerowanym przez program dokumencie XML, reprezentują klasy `Node` i `Attribute` z pakietu `pl.wroc.pwr.student.lawcreator.xml.nodes`. Posiadają one informacje o nazwie znacznika lub atrybutu oraz jego wartości. Co ważne, posiadają metodę `toXML()`, która pozwala na łatwe generowanie pliku XML – wystarczy wywołać tę metodę na elemencie, będącym korzeniem drzewiastej struktury XML.

Interfejs graficzny użytkownika aplikacji został zbudowany w następujący sposób. Po lewej stronie użytkownik ma do dyspozycji drzewo elementów, które mogą znaleźć się w dokumencie. Niektóre z nich posiadają elementy zagnieżdżone i aby do nich dotrzeć należy rozwinąć dany element w drzewie. Po naciśnięciu na daną pozycję istnieją dwie możliwości – albo użytkownik zostaje poinformowany, że element nie posiada atrybutów do wypełnienia (formularz jest pusty), albo pojawia się formularz z dostępnymi polami do wypełnienia. Niektóre z nich (np. atrybut `Data`), są ograniczone pod względem zawartości, tzn. muszą spełniać warunki umieszczone w schemacie XML. Atrybut `Data` musi spełniać warunek formy zapisu, która prezentuje się tak: `RRRR-MM-DD`. Jeśli użytkownik wypełni takie pole w zły sposób, zostanie o tym poinformowany w wyświetlonym oknie dialogowym. Jeśli formularz został wypełniony, można go zapisać - wtedy zostanie on dodany do drzewa, które jest po prawej stronie. Drzewo to reprezentuje elementy, które zostały wypełnione i zapisane przez użytkownika. Zostaną one umieszczone w wygenerowanej postaci XML. Jeśli użytkownik zdecyduje, że dany element został niepoprawnie wypełniony, może powrócić do jego edycji poprzez naciśnięcie na odpowiednią pozycję w drzewie po prawej stronie. Ostatnim ważnym elementem interfejsu jest przycisk „Generuj XML”, po którego naciśnięciu pojawia się okno z uzupełnioną uchwałą w postaci XML. Tę zawartość można skopiować i wkleić do nowego dokumentu XML.

Za wyświetlanie informacji na ekranie i obsługę zdarzeń wywołanych przez użytkownika, takich jak wybór pozycji w drzewie, czy naciśnięcie przycisku, odpowiada obiekt `MainWindow`, tworzony przy starcie aplikacji w metodzie `main()`. Utworzenie tego obiektu powoduje wyświetlenie głównego widoku aplikacji. Aby stało się to możliwe, najpierw inicjalizowane są komponenty, które mają zostać wyświetlone, a następnie wywoływana jest metoda `open()` obiektu klasy `Shell`. Po otwarciu okna wywoływana zostaje metoda `dispose()` na obiekcie `MainWindow`. Ponieważ biblioteka SWT nie używa odśmieccacza, to programista musi panować nad zwalnianiem i czyszczeniem obiektów z pamięci. Stąd też metoda `dispose()`, która posiada w swoim ciele pętlę sprawdzającą, czy obiekt powinien zostać odśmiecony, czy też nie.

Rozdział 7

Testowanie

7.1. Testy jednostkowe

Testy jednostkowe zostały napisane przy użyciu technologii JUnit.

Rozdział 8

Podsumowanie

8.1. Ocena projektu

Przedstawiona praca dotyczyła technologii JavaScript oraz budowy interfejsu użytkownika aplikacji sieciowej. Dołożono wszelkich starań, by projekt zakończył się sukcesem, co widać po ilości iteracji – korygowano zakres projektu tak, aby można było zaprezentować funkcjonalny serwis. Niestety mimo wszelkich starań, nie udało się wygenerować poprawnie działającej strony HTML z zanurzonym skryptem JavaScript.

Prezentowana aplikacja, choć prosta w założeniu, okazała się być dużo bardziej złożona, a napotykanne problemy uniemożliwiły osiągnięcie początkowych wymagań.

W pierwszej iteracji zakładano, że uda się stworzyć interfejs użytkownika i dołączyć go do już istniejącej logiki biznesowej systemu webSem. Jak się okazało – było to zadanie bardzo trudne, ze względu na szereg wykorzystywanych klas Java, które nie były wspierane przez J2S. Sukces można było osiągnąć przez zbudowanie aplikacji proxy, która zajmowałaby się jedynie obsługą bazy danych. Jednak próba skomunikowania aplikacji skończyła się niepowodzeniem i podjęto decyzję o przejściu do kolejnej iteracji.

W kolejnym podejściu problem znowu wydawał się mało skomplikowany i do tego realny z punktu widzenia JavaScript. Wymagał on jedynie operacji na ciągach znaków, co nie wymagało używania zewnętrznych plików `.jar`. Jednak napisanie własnego parsera XML nie jest sprawą prostą. Próba ujednolicenia wyświetlanych formularzy na podstawie wczytanych danych ze schematu zakończyła się niepowodzeniem z powodu pewnych nieścisłości pliku `schema.xsd`. W ramach tej iteracji powstała jednak namiastka aplikacji, która prezentowała użytkownikowi wybór dokumentu, który mógł stworzyć oraz edytować, natomiast nie udało się doprowadzić do tego, by sprawdzić poprawność wprowadzonych danych, a także wygenerować akt prawny w postaci XML.

Ostatecznie w ramach projektu zaprezentowano aplikację offline, której celem jest ułatwienie tworzenia dokumentu prawnego, jakim jest uchwała. Zostały spełnione założenia dotyczące

funkcjonalności aplikacji, jakimi były utworzenie i edycja aktu prawnego – uchwały, a także generowanie zawartości dokumentu w formacie XML.

8.2. Wnioski

Technologia Java2Script jest interesująca pod względem idei. Twórca chciał połączyć ze sobą zorientowaną obiektowo platformę Java z wykorzystywanym w stronach internetowych JavaScriptem. Efektem tego jest wtyczka do Eclipse, która umożliwia tworzenie stron HTML z zagnieżdżonym skryptem JavaScript, który wcześniej został wygenerowany z plików źródłowych `.java`. Posiada to swoje niewątpliwe zalety:

- możliwość generowania stron HTML, używając jedynie natywnego kodu Java,
- brak konieczności znajomości składni JavaScript do tworzenia skryptów,
- generowane pliki `.js` zachowują zasady programowania zorientowanego obiektowo,
- łatwość debuggowania kodu, dzięki użyciu środowiska Eclipse,
- możliwość umieszczania natywnego kodu JavaScript w klasach Javy.

Zachowując jednak obiektywizm – rzadko spotyka się strony, które są stworzone przy użyciu tej technologii. Oprócz kilku przykładowych serwisów zamieszczonych na głównej stronie Java2Script, bardzo ciężko jest znaleźć stronę internetową, która korzystałaby z tego rozwiązania. Przyczyn takiego stanu rzeczy może być kilka, zaczynając od małej liczby osób zainteresowanych tym tematem, którzy zdolni są pomóc w rozwoju technologii, poprzez małą aktywność na grupach dyskusyjnych do braku wsparcia wielu wbudowanych bibliotek Java i błędów, na które napotykamy się raz po raz.

Java2Script potrafi zachowywać się nieprzewidywalnie, niestabilnie i niedeterministycznie. Przykładem takiego zachowania jest np. używanie słuchaczy do obsługi zdarzeń wywoływanych przez użytkownika. Można spotkać się z tym, że na różnych wersjach wtyczki i na różnych środowiskach IDE, aplikacja potrafi działać dobrze, potrafi nie obsługiwać zdarzeń a nawet w ogóle się nie wyświetlać, nie informując użytkownika nawet o błędach w konsoli. Bywa tak, że pracując z Java2Script, po małej zmianie w kodzie, jakim jest przekształcenie jednej większej funkcji na mniejsze, aplikacja potrafi przestać się wyświetlać w przeglądarce i nie pomaga powrót do poprzedniego rozwiązania. Takie cechy technologii skutecznie mogą zniechęcić użytkownika, stąd być może małe zainteresowanie na forach i grupach dyskusyjnych.

Tworzenie aplikacji w Java2Script nie należy do najprostszych, ale jeśli ktoś chce stworzyć prostą aplikację, która nie będzie potrzebowała wielkiej logiki biznesowej, to jest to rozwiązanie dość wygodne – szczególnie dla osób nie posiadających biegłej znajomości HTML i JavaScript.

Jednak do bardziej złożonych serwisów, które dodatkowo potrzebują pewnego przepływu interfejsów na kształt stron internetowych, jest to technologia, która potrafi utrudnić budowanie takiego serwisu.

Budowanie aplikacji, jaką był ActEditor, a później LawCreator, pozwoliło również zgłębić temat dokumentów XML oraz schematów XML. Język XSD pozwala na zdefiniowanie struktury dokumentu XML i stanowi alternatywę dla DTD. W odróżnieniu od DTD schematy umożliwiają precyzyjne deklarowanie typów danych, pozwalają też na jednokrotne definiowanie powtarzających się fragmentów modelu.

Schemat XML, który regulował zasady generowania zawartości XML w przypadku ActEditor oraz LawCreator nosił nazwę `schema.xsd` i był skonstruowany przez MSWiA. Nie można go chyba jednak uznać za udany. Jak sama nazwa wskazuje, schemat powinien precyzyjnie opisywać to, co, gdzie i jak może zostać użyte. Jednak w tym przypadku opis ten bywa zaciemniony i niejasny. Weźmy pod uwagę np. to, że nie ma jasno opisane, który element może posiadać ciało (treść elementu), a który nie. Inną nieścisłością jest to, że pewne elementy powinny być zagnieżdżone tylko w konkretnej, ustalonej kolejności, natomiast dostępny schemat dość niejasno określa tego typu powiązania. Kolejnym przykładem jest to, że w pewnym miejscu elementu sygnatura aktu jest atrybutem, a w innym miejscu jest już elementem – tutaj można zacząć zadawać sobie pytanie, czy taki zabieg był celowy, czy jest to zwykłe przeoczenie.

Literatura

[DOM] http://www.brainbell.com/tutorials/xml/how_the_dom_works.htm.

[EDA] http://bip.msw.gov.pl/portal/bip/185/18658/edytor_aktow_prawnych_edap__wersja_z_dnia_31_marca

[J2Sa] <http://j2s.sourceforge.net/>.

[J2Sb] <http://j2s.sourceforge.net/articles/oop-in-js-by-j2s.html>.

[J2Sc] <http://j2s.sourceforge.net/document.html>.

[J2Sd] <http://j2s.sourceforge.net/overview.html#j2s-features>.

[J2Se] <http://j2s.sourceforge.net/overview.html#j2s-swt>.

[PLO] http://www.ia.pw.edu.pl/~traczyk/pdf/ploug2003_art.pdf.

[SAX] <http://www.sqlmag.com/article/xml/how-sax-works>.

[SCH] <http://crd.gov.pl/xml/schematy/edap/2010/01/02/schemat.xsd>.

[SWT] http://wiki.eclipse.org/faq_is_swt_better_than_swing

[Tra] http://www.ploug.org.pl/konf_01/materialy/pdf/traczyk.pdf.

[XML] <http://pl.wikipedia.org/wiki/xml>.

Dodatek A

Instrukcja wdrożenia

Dodatek B

Instrukcja użytkownika