POLSKO-JAPOŃSKA WYŻSZA SZKOŁA TECHNIK KOMPUTEROWYCH

PRACA MAGISTERSKA

Nr .....

Automatyzacja generowania plików pakietu MS Office

Student/studentka	**Michał Degentysz**
Nr albumu	4870
Promotor	Prof. dr inż. Kazimierz Subieta
Specjalność	Inżynieria Oprogramowania i Baz Danych
Katedra	Systemów Informacyjnych
Data zatwierdzenia tematu	25.03.2009
Data zakończenia pracy	

Podpis promotora pracy	**Podpis kierownika katedry**
.....	.....

SPIS TREŚCI:

I. Wstęp

Żadna współczesna firma nie może obejść się bez używania oprogramowania, czy to usprawniającego, czy nawet umożliwiającego jej pracę. Zwykle można określić jego przynależność do jednego z dwóch rodzajów aplikacji – z jednej strony znajdują się narzędzia wspomagające właściwe cele biznesowe organizacji (od prostych aplikacji do drukowania faktur czy narzędzi CRM aż po potężne, tworzone na zamówienie kompleksowe systemy); z drugiej – służące ogólnie do pracy z dokumentami pakiety biurowe.

Daje się jednakże zauważyć brak uniwersalnych narzędzi, wspomagających import i eksport danych pomiędzy wyżej wspomnianymi grupami aplikacji – brak systemów-pomostów umożliwiających automatyzację tworzenia dokumentów na podstawie danych generowanych przez aplikacje jest dużym utrudnieniem dla osób zajmujących się procesami biznesowymi na styku aplikacji biznesowych i pakietów biurowych; problemem który nierzadko powoduje konieczność poświęcenia znacznych środków i czasu na przenoszenie danych z aplikacji biznesowych do dokumentów oraz w drugą stronę.

Niniejsza praca jest uzupełnieniem stworzonego przez autora narzędzia MSGenerator, będącego odpowiedzią na opisany wyżej problem; zawarty w niej jest opis sytuacji która była inspiracją dla stworzenia narzędzia; jego funkcjonalność, architektura, możliwe zastosowania oraz problemy, a także propozycje dalszego rozszerzenia funkcjonalności. Ponadto zostały opisane pokrótce technologie wykorzystane w procesie tworzenia oraz motywy stojące za ich zastosowaniem – z argumentacją uzasadniającą pominięcie niektórych możliwych ścieżek rozwoju.

Pomysł na stworzenie automatycznego narzędzia do generowania plików pakietu MS Office oraz przerzucania do nich danych był naturalną konsekwencją problemów na które autor niniejszej pracy natknął się podczas swojej pracy zawodowej. Pracując nad rozwojem aplikacji WebInwestor autor zetknął się z tematyką eksportu danych i zestawień z aplikacji biznesowej do plików tworzonych w programach Word, Excel oraz Project.

1.1 Wykorzystanie eksportu danych do pakietu Office w aplikacji WebInwestor

Pracując w firmie Impaq, autor zajmuje się między innymi rozwojem projektu WebInwestor. Jest to system przeznaczony dla sektora gazowniczego, oferujący (w dużym skrócie) całościową funkcjonalność śledzenia i zarządzania procesem inwestycyjnym, takim jak budowa/remont gazociągu czy przyłączenie odbiorcy do sieci gazowej; zarządzanie środkami stałymi spółki, tworzenie i zatwierdzanie planów inwestycyjnych czy też wczyt i powiązanie danych z systemów finansowo-księgowych z zadaniami inwestycyjnymi, z uwzględnieniem monitoringu wykonania poszczególnych zadań jak i całych planów.

Aplikacja ta została przystosowana i wdrożona w kilku obecnych na polskim rynku spółkach gazowniczych – pomimo tego, że są one odrębnymi firmami, udziałowcem wszystkich jest PGNiG; dzięki czemu część ich modelu biznesowego jest wspólna, umożliwiając wyodrębnienie uniwersalnego rdzenia aplikacji.

Użytkownicy systemu w poszczególnych spółkach w różnym stopniu korzystają z eksportu do poszczególnych programów MS Office – w każdym jednak przypadku możliwość ta stanowi ważny element funkcjonalności całego systemu.

Dolnośląska Spółka Gazownicza Sp. z o.o. (DSG) generuje umowy zlecenia na wykonanie bądź na projekt gazociągów i przyłączy do nich, a także protokoły odbioru końcowego powyższych oraz dokumenty przyjęcia środków trwałych (tzw. OT) – stosowane przez klienta szablony dokumentów zostały zmodyfikowane w taki sposób, aby następujące dane były możliwe do automatycznego wypełnienia:

- **nazwa firmy** - używana na oficjalnych dokumentach nazwa spółki zmieniała się kilkukrotnie na przestrzeni ostatnich dwóch lat; uwzględnienie jej jako zmiennej dokumentu wyeliminowało konieczność zmiany w takiej sytuacji samego szablonu – aktualnie jedynymi elementami, których zmiany wymuszają modyfikacje samego dokumentu są elementy graficzne – logo firmy – oraz zmiany w układzie treści; konfiguracja wszystkich pozostałych elementów jest sterowana za pomocą kodu

- **nazwa oddziału firmy** – DSG składa się z oddziałów: wrocławskiego, wałbrzyskiego i zgorzeleckiego – zmienne umożliwiły umieszczenie tego samego szablonu na serwerze; z danymi takimi jak ich właściwa nazwa oraz adres pobieranymi z bazy – pozwoliło to rozwiązać problem pracy z siedmioma różnymi wersjami tego samego pliku, różniącymi się jedynie nazwą; dzięki czemu aktualnie wykorzystywane są tylko dwa różne szablony. Problematyczne są jedynie dane ładowane do stopki lub nagłówka pliku – po eksporcie wymagają odświeżenia na dokumencie poprzez uruchomienia makra wbudowanego w plik
- **Dane osoby generującej zlecenie**, również pobierane z bazy – w przypadku DSG kilkunastu różnych użytkowników generuje dokumenty; wykorzystanie automatycznego eksportu pozwala na bezbłędne wypełnienie danych takich jak dział użytkownika, numer telefonu czy faksu – wszystkie są pobierane z bazy, eliminując tym samym potencjalne pomyłki
- **Adresy i parametry przyłączy** oraz odcinków gazociągu wchodzących w skład zlecenia – w przypadku dużego zlecenia, tabela prezentująca wspomniane dane może mieć nawet kilkadziesiąt wierszy; ich ręczne wypełnianie byłoby nie tylko bardzo żmudne, ale również obarczone ryzykiem popełnienia błędu
- **Skład komisji** dokonującej odbioru prac będących tematem zleceń
- **Daty utworzenia** zlecenia bądź protokołu
- **Kwota zlecenia**
- **Numery umów ramowych**, które obejmowały swym zakresem dane zlecenie

Szczególnie funkcjonalność aplikacji związana z dokumentami OT poprawiła się dzięki automatyzacji generowania dokumentów – w sytuacji, gdy wykonanie pojedynczego zadania inwestycyjnego może powodować konieczność przyjęcia nawet kilkuset środków trwałych przez inwestora, tworzenie i wypełnianie danymi właściwych druków trwałoby zdecydowanie zbyt długo.

Wielkopolska Spółka Gazownictwa Sp. z o.o. (WSG) wykorzystuje przede wszystkim raporty generowane w Excelu – pobierane z bazy danych informacje dotyczące finansowania zadań inwestycyjnych są następnie przeliczane i umieszczane w uprzednio przygotowanym formularzu.

Funkcjonalność ta daje klientowi możliwość generowania raportów i zestawień w locie, z uwzględnieniem zmian danych kiedy tylko te zostaną zapisane w bazie – jest to podstawowy sposób prezentowania danych zarządowi spółki, będąc tym samym krytycznym z punktu widzenia funkcjonalności punktem całej aplikacji. Dodatkowo wykorzystywana jest też podobna do tej istniejącej w DSG funkcjonalność generowania dokumentów OT.

Operator Gazociągów Przesyłowych GAZ-SYSTEM S.A. (OGP) korzysta z interfejsu udostępnionego przez narzędzie MS Project. WebInwestor w tym przypadku pozwala na stworzenie planu wykonania danego zadania gazociągowego, z uwzględnieniem struktury dowolnie zagnieżdżonych podzadań, a następnie na eksport całości do pliku MS Project. Również i u tego klienta dokonywane są eksporty różnych zestawień finansowych do Excela.

1.2 Problemy związane z rozwojem WebInwestora

Stworzone na potrzeby WebInwestora narzędzie eksportu danych było bardzo nieelastyczne; każdorazowa zmiana w wyglądzie czy układzie któregoś z dokumentów wymagała niewygodnych zmian w kodzie, co w konsekwencji prowadziło do niezbędnego ponownego testowania elementów nie związanych z samym eksportem – na przykład po dokonaniu jakichkolwiek zmian w szablonie któregoś z plików czy też poszerzeniu zakresu danych uwzględnionych w zleceniu.

Zastosowana architektura okazała się krótko mówiąc niepraktyczna; ponadto była nierozerwalnie związana z modelem systemu, a przez to niemożliwa do wykorzystania w innych aplikacjach – w poszczególnych wersjach WebInwestora wdrożonych u każdego z wymienionych klientów była odrębna wersja narzędzia eksportującego. W głównej aplikacji brakowało wydzielonego modułu obsługującego

tylko jej powiązania z plikami narzędzi MS Office, co wyeliminowałoby konieczność zmian w trzech różnych systemach po znalezieniu w którymś błędów.

Jednym z powodów opisanych problemów było wykorzystanie dość starej i mało elastycznej technologii; pierwsza wersja WebInwestora powstała jeszcze w latach dziewięćdziesiątych i została napisana w całości w języku PL/SQL (konsekwencja zastosowania bazy danych Oracle). Jedynym sposobem stworzenia aplikacji webowej w tej technologii było generowanie poszczególnych linii kodu HTML przez polecenia PL/SQL-a.

Jak to często bywa w długo rozwijanych aplikacjach, ostateczna wersja systemu znacznie przerosła możliwości zastosowanej architektury i doprowadziła do przemieszania warstwy prezentacji i logiki biznesowej, co w konsekwencji uzależniło funkcjonalność eksportu danych od konkretnej logiki biznesowej.

Sytuacja ta zainspirowała autora do opracowanie modelu, który byłby na tyle elastyczny, aby bez większych zmian w samej aplikacji móc dostarczyć programistom narzędzia pomagającego w implementacji wymaganej funkcjonalności, możliwie łatwo i szybko. Co więcej, autor chciał aby narzędzie to było jak najbardziej niezależne od samej aplikacji – to jest istniało w postaci przenaszalnego komponentu, który mógłby zostać łatwo dołączony do programów już istniejących, jak i nowo tworzonych.

W niniejszej pracy autor postara się udowodnić, że narzędzie MSGenerator stanowi skuteczne rozwiązanie wspomnianego problemu.

II. Wykorzystanie pakietu MS Office we współczesnych firmach

Jeśli istnieje wspólny mianownik łączący sferę IT współczesnych firm - to jest nim bez wątpienia wykorzystanie w codziennej pracy pakietów biurowych oraz dedykowanych ich biznesowej działalności specjalistycznych aplikacji.

2.1 Pakiety biurowe

Korzystając z definicji podanej w Wielkiej Internetowej Encyklopedii Multimedialnej, możemy określić pakiet biurowy jako „zestaw oprogramowania umożliwiającego wykonywanie prac biurowych o różnym stopniu skomplikowania. Najczęściej w skład pakietu biurowego wchodzą takie moduły jak edytor tekstu i arkusz kalkulacyjny; często są również dołączane inne składniki, jak baza danych, program do tworzenia grafiki prezentacyjnej czy moduł graficzny”[1].

W założeniu pakiety biurowe były projektowane wyłącznie z myślą o zastosowaniu w biurach; obecnie są one powszechnie wykorzystywane również przez użytkowników do celów prywatnych.

Aplikacje z takiego pakietu zwykle charakteryzują się podobnym interfejsem oraz wysokim stopniem zintegrowania ze sobą – polegającego między innymi na możliwości zagnieżdżania obiektów jednej z aplikacji w innej – na przykład wykorzystaniu wykresu czy tabelki utworzonych w arkuszu kalkulacyjnym w dokumencie edytowanym przy pomocy edytora teksu, bądź też skorzystaniu z możliwości oferowanych przez edytor przy edycji slajdów w narzędziu do tworzenia prezentacji multimedialnych.

Do standardowych narzędzi, jakie najczęściej można znaleźć w pakiecie biurowym zaliczają się:

- Edytor tekstu
- Arkusz kalkulacyjny

[1] http://portalwiedzy.onet.pl [13.06.2009]

- Narzędzie do budowania prezentacji multimedialnych
- Prosta baza danych

Istnieją pakiety dostępne na w zasadzie dowolną platformę systemową; niektóre przywiązane tylko do jednej (przykładem jest **iWork** firmy Apple, współpracujący jedynie z systemem Mac OS X), inne prawdziwie uniwersalne (sztandarową aplikacją tego typu jest **OpenOffice.org**, obecny na praktycznie wszystkich liczących się platformach).

Do najpopularniejszych pakietów należą:

- **Microsoft Office** (systemy Windows, Mac OS X)
- **OpenOffice.org** (systemy Windows, Max OS X, Linux, Unix)
- **IBM Lotus Symphony** (systemy Windows, Mac OS X, Linux)
- **Apple iWork** (system Mac OS X)

Dość nowym podejściem do tematyki aplikacji biurowych są pakiety biurowe online (*online office suites*); nie wymagające instalowania na lokalnej maszynie ani też ręcznego kopiowania na nią plików – do ich działania wymagana jest jedynie przeglądarka internetowa. Przykładem jest aplikacja GoogleDocs, powstała w okolicach 2006 roku.

Ten zyskujący popularność trend odpowiada koncepcji oprogramowania jako usługi, uniezależniającego aplikacje od systemów operacyjnych (*SaaS – Software as a Service*).

2.2 Perspektywy rozwoju

Trudno byłoby zaprzeczyć ogromnej popularności pakietu MS Office wśród przedsiębiorstw. W praktyce mało która firma nie korzysta z pakietów biurowych – a nawet jeśli są to rozwiązanie konkurencyjne wobec narzędzia Microsoftu (jak np. OpenOffice.org), to są one z nim w znacznym stopniu kompatybilne.

Z drugiej strony, specyficzne wymagania klientów doprowadziły do powstania rynku oprogramowania tworzonego na zamówienie, wspomagającego bądź realizującego procesy biznesowe w danej firmie.

Pomiędzy tymi dwoma warstwami pozostaje luka – zwykle aplikacje nie są projektowane z myślą o integracji z pakietami biurowymi, co potrafi prowadzić do niepotrzebnych utrudnień w realizacji zadań wykonywanych przez firmę.

Narzędzie MSGenerator trafia właśnie w tę lukę, dając możliwość bezpośredniego eksportu danych do plików MS Office; korzystając z informacji przechowywanych w bazach danych bądź tworzonych „w locie" w aplikacji – niezależnie od tego czy plikiem docelowym miałaby być umowa wypełniana danymi kontrahenta w pliku .doc, raport w postaci .xls czy też plan projektu jako .proj. – szczególnie w tym pierwszym przypadku, w zasadzie dowolny dokument tworzony na bazie jakiegoś szablonu może zostać wygenerowany automatycznie.

Co więcej, dokumenty w postaci przygotowanych szablonów w których jedynie część danych się zmienia stanowią najpopularniejszy sposób korzystania z edytorów tekstu w firmach – trudno nie docenić oszczędności czasu którą daje wyeliminowanie powtarzalnych czynności (chociażby tworzenie firmowej stopki i nagłówka w dokumencie czy danych adresowych). A takie właśnie dokumenty najbardziej zyskują w połączeniu z możliwością automatyzacji wypełniania ich danymi.

Należy tu również rozważyć fakt, że w Polsce jeszcze przez długi czas zdecydowana większość umów wszelkiego rodzaju potwierdzana będzie dokumentami papierowymi – brak w powszechnym użyciu kwalifikowanych podpisów elektronicznych – powodowany wysoką ceną kompletnego rozwiązania, oraz nierzadko nieufnością wobec nowości - powoduje, że w zasadzie każdy formalny i posiadający ważność prawną dokument musi zostać wydrukowany i podpisany – te cechy razem otwierają naturalne pole do wykorzystania aplikacji takiej jak MSGenerator; pozwalającej zaoszczędzić czas (poprzez brak konieczności ręcznego dostosowywania dokumentu przez pracownika) jak i uniknąć pomyłek (dzięki pobieraniu informacji bezpośrednio z bazy danych).

Zebrawszy razem olbrzymią popularność pakietu MS Office oraz fakt, iż większość aplikacji biznesowych (przynajmniej po stronie klienckiej) działa na platformach Windowsowych sprawiają, że MSGenerator mógłby znaleźć szerokie

zastosowanie jako rozwinięcie już istniejących jak i nowo tworzonych aplikacji biznesowych opartych na technologii J2EE. Łatwość i prostota zastosowania narzędzia – a co za tym idzie niskie koszty wdrożenia stanowią dobry argument za użyciem go w tych aplikacjach, dla których funkcjonalności byłby naturalnym rozszerzeniem, nie wymagając jednocześnie daleko posuniętych modyfikacji.

Trudno byłoby wyobrazić sobie sytuację, w której w przyszłości tendencja do wykorzystania i narzędzi biznesowych, i pakietów biurowych zmalałaby. MS Office i podobne mu narzędzia stanowią niejako podstawowe narzędzia w pracy biurowej; często klient zamawiający tworzony na zamówienie system biznesowy uzależnia swoją decyzję o zaakceptowaniu projektu od możliwości jego integracji z wybranym pakietem biurowym (autentyczny przypadek z pracy zawodowej autora).

Opisane wyżej powody sugerują, że MSGenerator bądź podobne mu narzędzie z pewnością znalazłyby swoją niszę na rynku oprogramowania.

III. MS Generator jako propozycja uniwersalnego narzędzia do generowania plików

Projekt programistyczny będący uzupełnieniem niniejszej pracy magisterskiej przedstawia przykładowe rozwiązanie problemów związanych z zautomatyzowanym eksportem danych.

3.1 Założenia

System, którego funkcjonalność miałaby kompleksowo obejmować funkcjonalność eksportu danych do poszczególnych narzędzi pakietu biurowego musi spełniać szereg założeń oraz odpowiedzieć na pewne kwestie związane z tematyką – nie funkcjonuje on jako oderwany samodzielny komponent, ale uzupełnienie większego systemu.

3.1.1 Cel

Należy odpowiedzieć na pytanie, które z aplikacji MS Office zyskałyby najbardziej na możliwości eksportowania do nich danych z aplikacji czy bazy danych?

Naturalną odpowiedzią są najpopularniejsze narzędzia z pakietu biurowego; te używane najczęściej przez użytkowników systemów – czyli edytor tekstu, arkusz kalkulacyjny oraz, w nieco mniejszym stopniu, aplikacja wspomagająca zarządzenie projektami. W przypadku MS Office; Powerpoint, Access oraz Outlook są używane na tyle rzadko w kontekście współpracy z innymi aplikacjami, że próby dołączenia ich do narzędzia eksportującego byłyby pozbawione większego sensu – funkcjonalność eksportu danych do tych narzędzi byłaby pozbawiona większego sensu.

3.1.2 Dane

Aby system był możliwie elastyczny, konieczne jest opracowanie spójnego standardu danych, przy użyciu którego informacje mogłyby być łatwo przekazywane pomiędzy poszczególnymi warstwami aplikacji, a także pomiędzy różnymi aplikacjami.

Do tego celu idealnie nadaje się XML – dane w tej postaci mogą być przechowywane w pliku czy bazie danych, bądź nawet generowane na bieżąco w aplikacji – a dodatkowo przesyłane poprzez tzw. usługi sieciowe (*web service*). Drzewiasta struktura XML idealnie nadaje się do prezentacji danych takich jak komórki arkusza, tabele w dokumencie tekstowym czy też zadania w projekcie. Ponadto, dzięki wykorzystaniu XPath można takie struktury można łatwo przetwarzać już logice aplikacji.

Kolejnym etapem jest zaprojektowanie wzorca definiującego poprawną postać danych. Idealna struktura powinna być przejrzysta, spójna, skalowalna oraz na tyle elastyczna, by umożliwić jej łatwe rozszerzenie o nowe elementy. Jako jej uzupełnienie, istotny byłby też standard weryfikujący poprawność danych – opracowany jako DTD bądź XML Schema. Wykorzystanie XML znacznie ułatwia zrealizowanie tego postulatu – nie narzucając niemal żadnych ograniczeń technicznych czy formalnych pozwala na dowolne kształtowanie struktury.

3.1.3 Technologia

Jako że narzędzie eksportujące w ogromnej większości przypadków miałoby w założeniu współpracować z innymi aplikacjami – czy też jako ich integralna część, czy też tylko w roli systemu współpracującego – powinno być łatwe do wykorzystania w już istniejących projektach.

Co więcej, jako że coraz więcej systemów biznesowych tworzonych jest w postaci aplikacji internetowych, również i narzędzie do eksportu danych powinno działać jako potencjalny komponent takiej aplikacji. Dwie najpopularniejsze takie technologie to **J2EE** (obecnie jako **Java EE 5**) oraz **ASP.NET**.

Ta druga, opracowana przez Microsoft, dużo łatwiej integruje się z pakietem MS Office – stąd wniosek, że narzędzie dostosowane do pracy z Javą byłoby bardziej przydatne, gdyż tworząc aplikację ASP o wiele łatwiej implementuje się obsługę obiektów MS Office - stąd też wybór J2EE jako technologii, w jakiej został zrealizowany MS Generator.

Technologie javowe ponadto na tyle dobrze współpracują z dokumentami XML, że ich implementacja tego właśnie formatu jako nośnika informacji nie wymaga poświęcenia dodatkowych środków i czasu.

3.1.4 Funkcjonalność

Najważniejsze decyzje podczas projektowania narzędzia dotyczą zakresu funkcjonalności mającej zostać zaimplementowanej w systemie. W tej kwestii autor oparł się na rzeczywistej aplikacji – opisanym w rozdziale 1.1 WebInwestorze – i zawarł te jej elementy, które okazały się być najbardziej użyteczne.

Dla klienta najważniejszym okazała się współpraca z MS Wordem – z racji olbrzymiej ilości dokumentów generowanych przez spółki gazownicze to eksport danych do predefiniowanych szablonów okazał się najczęściej wykorzystywanym modułem – ze szczególnym uwzględnieniem możliwości tworzenia w dokumencie tabel, z racji dużego zróżnicowania ilości elementów, które trafiały do szablonu.

Nieco prostszym, ale równie ważnym obszarem było też generowanie raportów w Excelu – pozornie prosta funkcjonalność wypełniania poszczególnych komórek danymi zapewniała w praktyce olbrzymie możliwości wykorzystania mechanizmu prezentacji danych ilościowych, i to właśnie ona zwykle stanowiła odpowiedź na kolejne zgłoszenia klienta dotyczące rozszerzenia funkcjonalności.

Wreszcie, pomimo tego że integracja WebInwestora z MS Projectem wykorzystana została tylko przez jednego z klientów firmy Impaq – to potencjalne możliwości oferowane przez to połączenie zdecydowały o stworzeniu odpowiedniego algorytmu – dzięki udostępnieniu funkcjonalności tworzenia drzewiastej struktury projektu w aplikacji WebInwestor implementacja eksportu do MS Projecta nie stanowiła większego problemu, dzięki czemu użytkownicy nie musieli korzystać bezpośrednio z

Projecta tworząc projekty inwestycji, a jednocześnie otrzymywali możliwość ich ręcznej modyfikacji, już po wyeksportowaniu struktury.

Wszystkie opisane powyżej przypadki złożyły się na funkcjonalność zaimplementowaną w MS Generatorze.

3.1.5 Interfejs użytkownika

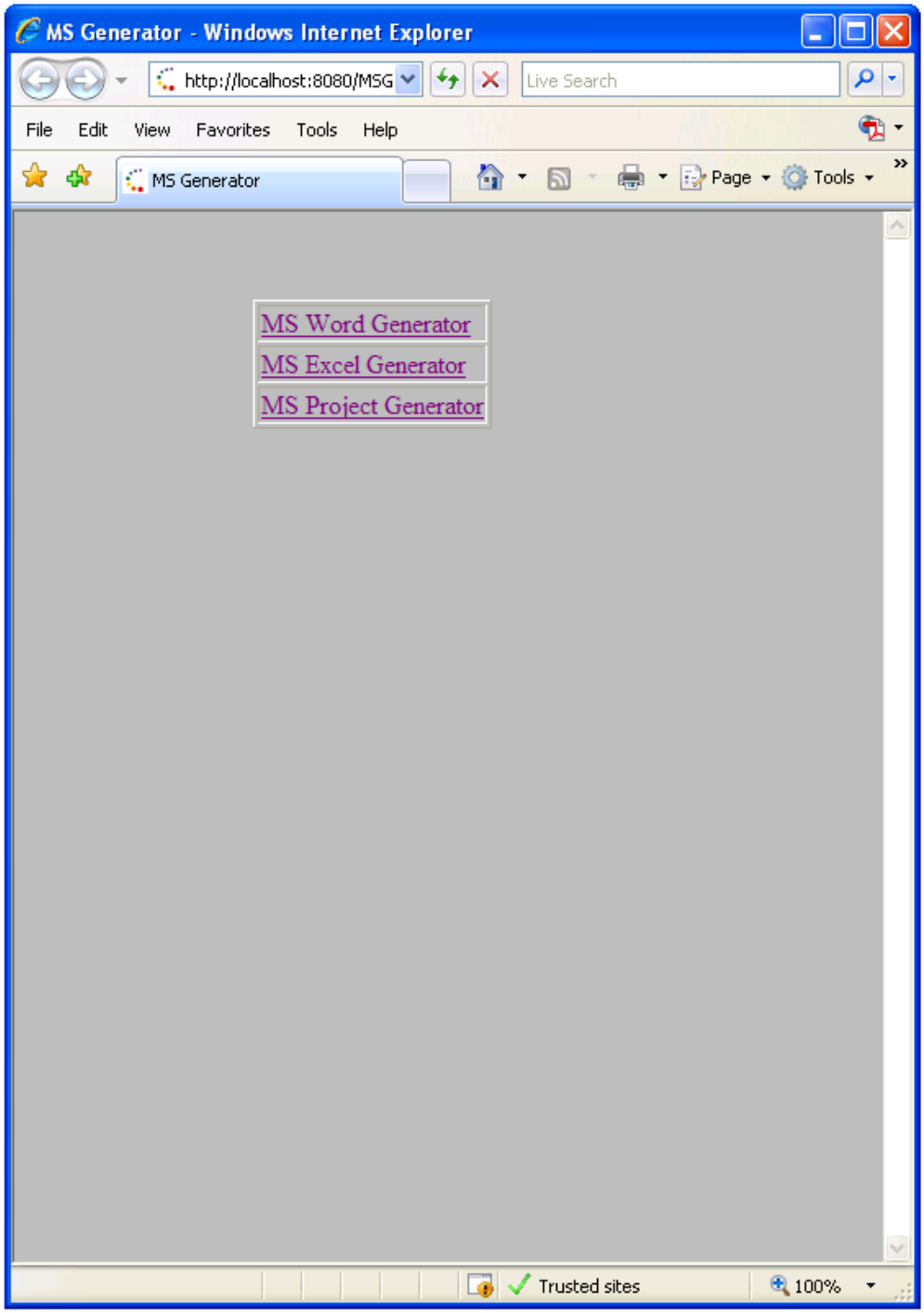
Stosunkowo mniej ważnym punktem całości jest interfejs użytkownika – niemniej jednak należy zastanowić się, w jaki sposób użytkownik będzie korzystał z udostępnionej funkcjonalności.

W systemie WebInwestor praktycznie wszystkie punkty styczności z systemem eksportu danych opierały się na funkcjonalności dostępnej pod przyciskiem bądź linkiem na formularzu aplikacji – i takie właśnie podejście przyświecało projektowaniu MSGeneratora – czyli możliwość wywołania eksportu wszędzie tam gdzie może być on potrzebny; eksport informacji nie bierze się „z powietrza" tylko zawsze dotyczy pewnych określonych danych.

Funkcjonalność w MSGeneratorze udostępniona jest poprzez metody javascript – przy odpowiednich modyfikacjach wygenerowane funkcje mogą być dostępne w zasadzie z dowolnego aktywnego elementu formularza.HTML czy .JSP.

Ewentualnie można też rozważyć interfejs na tyle sparametryzowany, aby nie wymagał bezpośredniej akcji użytkownika przy każdorazowym eksporcie danych do plików – dzięki temu możliwe byłoby wsadowe generowanie wielu plików naraz; przy przekazaniu do systemu listy danych wejściowych i wyjściowych system uzupełniłby pliki pobrane z bazy czy serwera danymi i umieścił je we wcześniej zdefiniowanej lokalizacji – byłby to przydatny element w sytuacji, gdy jakaś część funkcjonalności dostępnej w aplikacji wiązałaby się z koniecznością wygenerowania dużej liczby dokumentów.

3.2 Opis



Rysunek 1 strona startowa aplikacji MSGenerator

MSGenerator jest narzędziem służącym do eksportu danych przechowywanych w postaci dokumentów XML do plików - dokumentów wynikowych poszczególnych narzędzi pakietu biurowego MS Office – Worda, Excela oraz Projecta; wykorzystując do tego obiekty ActiveX reprezentujące wymienione aplikacje; poprzez funkcjonalność dostępną z formularza aplikacji webowej.

Narzędzie to zostało zaprojektowane jako element możliwy do wykorzystania w już istniejącej jak i nowo tworzonej aplikacji w technologii J2EE, oczywiście przy założeniu że dostarczana przez nią funkcjonalność to uzasadnia.

Przykładowa implementacja – projekt będący uzupełnieniem niniejszej pracy magisterskiej - zakłada użycie predefiniowanych styli oraz treści formularza, nie wyklucza jednak ewentualnych późniejszych modyfikacji, w większym stopniu integrujących MSGeneratora z wybraną aplikacją biznesową.

Można również wykorzystać samą przykładową implementację jako proste narzędzie do eksportu danych do plików MS Office – z wykorzystaniem ręcznie bądź automatycznie generowanych plików XML z danymi oraz (w przypadku Worda i Excela) gotowych plików – szablonów.

Zastosowanie narzędzia w najprostszej postaci (bez innej aplikacji biznesowej) od strony technicznej wymaga jedynie uruchomionego serwera aplikacji z umieszczoną na nim aplikacją MS Generator; zainstalowanych na maszynie klienckiej odpowiednich narzędzi MS Office oraz przeglądarki Internet Explorer w wersji 6.0 lub 7.0 – spełnienie tych warunków daje użytkownikowi dostęp do prostych formularzy umożliwiających wczyt danych i szablonów oraz wypełnienie dokumentów danymi.

Startowy formularz aplikacji pokazany jest na rysunku 1.

3.3. Funkcjonalność

MSGenerator obejmuje taką funkcjonalność, jaką w podstawowym zakresie dostarczają aplikacje z pakietu MS Office, zawężoną do najistotniejszych z punktu widzenia MSGeneratora funkcji.

3.3.1 MS Word

U podstaw funkcjonalności części aplikacji obsługującej Worda znajduje się wypełnianie danymi zmiennych tekstowych (***DocVariables***) umieszczonych w uprzednio przygotowanym szablonie.

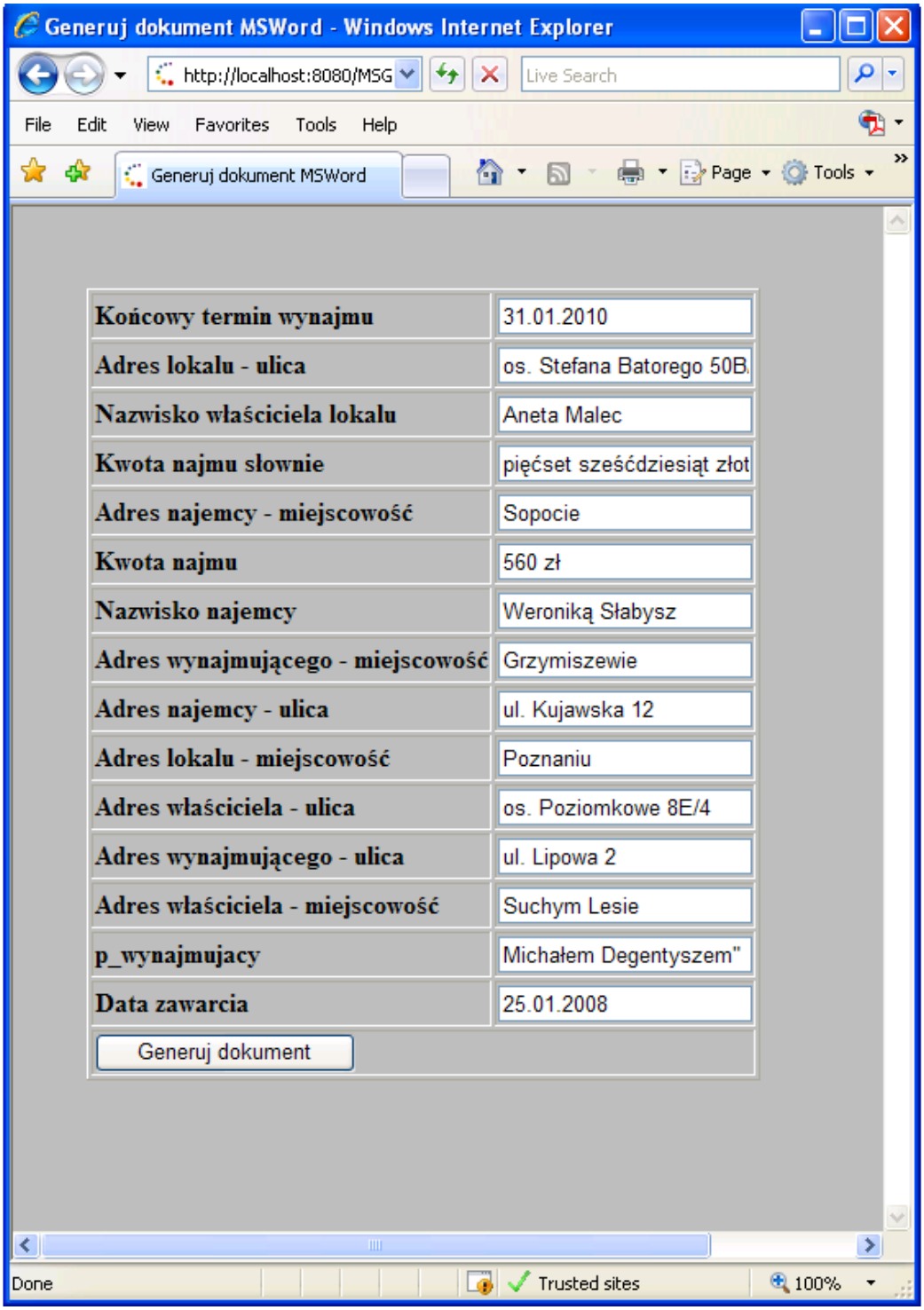
Na podstawie danych zawartych w wejściowej strukturze XML generowany jest formularz HTML z podglądem zmiennych – dzięki prezentacji poszczególnych zmiennych w postaci edytowalnych pól tekstowych umożliwia on zmianę „w locie" wartości przekazanych do dokumentu (np. zmianę rzeczownika w mianowniku na odpowiedni przypadek czy inny modyfikację formatu daty wymaganego w dokumencie – krótko mówiąc w sytuacjach, gdy źródło danych nie udostępnia ich w takiej postaci jaka jest wymagana) poprzez zastąpienie wartości z pliku wartością wpisaną przez użytkownika na formularzu (przy czym standardową akcją jest oczywiście eksport danych w takiej postaci w jakiej przyszły; bez konieczności przepisywania ich przez użytkownika) - a także ich podgląd, zanim jeszcze zostaną do dokumentu wpisane.

Wygenerowany formularz składa się z pól opisanych przez zawarty w danych XML synonim – natomiast w przypadku jego braku, opisem pola jest nazwa zmiennej w dokumencie, pod którą dana wartość zostanie przypisana.

Dodatkowo, możliwe jest też automatyczne wypełnianie istniejących w szablonie tabel – poprzez dodawanie do nich wierszy z określoną ilością kolumn i zawartością poszczególnych komórek. Przykładem obrazującym taką funkcjonalność mogłaby być faktura, posiadająca wiele pozycji – lub też każda inna struktura mogąca zostać przedstawiona w takiej postaci.

Poszczególne tabele w dokumencie identyfikowane są przez ich numer porządkowy, równoznaczny kolejności wystąpienia w dokumencie, natomiast zmienne przez nadaną im w dokumencie nazwę (przy czym w przypadku zmiennej o tej samej nazwie która w dokumencie występuje kilka razy, wszystkie jej wystąpienia zostaną wypełnione jednocześnie).

3.3.1.1 Przykładowy formularz z danymi



Rysunek 2 formularz z danymi eksportowanymi do MS Worda

Na rysunku 2 przedstawiono przykładowy formularz wygenerowany z pliku XML. Jak widać, w pliku źródłowym pole o nazwie **p_wynajmujacy** nie posiadało aliasu i pod taką nazwą występuje na formularzu; pozostałe pola identyfikowane są poprzez synonimy.

3.3.2 MS Excel:

Podstawową funkcjonalnością części obsługującej Excela jest wypełnianie danymi poszczególnych komórek konkretnego arkusza w pliku. Komórki są identyfikowane poprzez numeryczną reprezentację ich adresu na arkuszu – przykład poniżej:

adres komórki **B6** w pliku to row=”**2**”, col=”**6**”

Poza uzupełnianiem zawartości komórki, można też dowolnie formatować jej wygląd – przekazane w strukturze XML właściwości komórki są interpretowane na odpowiednie jej formatowanie – przy czym ich dobór jest „ograniczony” do wszystkich możliwych właściwości komórki – narzędzie interpretuje nazwę-klucz danej właściwości, jej wartość oraz to, czy wartość należy otoczyć cudzysłowem (konieczne na przykład dla niektórych wartości tekstowych), a następnie odpowiednio formatuje komórkę. Daje to prawdziwie szeroki wachlarz opcji formatujących – umożliwiając stworzenie w pełni kontrolowanego wizualnie dokumentu bez konieczności nanoszenia na niego ręcznie poprawek.

Drugim sposobem w jaki można formatować dane jest wykorzystanie uproszczonych, z góry zdefiniowanych w kodzie algorytmu własności komórki – po zawarciu odpowiednich znaczników i ich atrybutów w strukturze XML narzędzie odpowiednio je zinterpretuje i sformatuje komórki (każdą osobno), zmieniając kolor czcionki, tła, pogrubienie, pochylenie bądź podkreślenie treści; w zależności od tego które z opcji wystąpiły w pliku. Dzięki temu do wygenerowania dokumentu XML opisującego sformatowany arkusz nie jest wymagana znajomość składni poleceń udostępnianych przez interfejs Excela; wystarczy zastosować składnię interpretowaną przez MSGenerator.

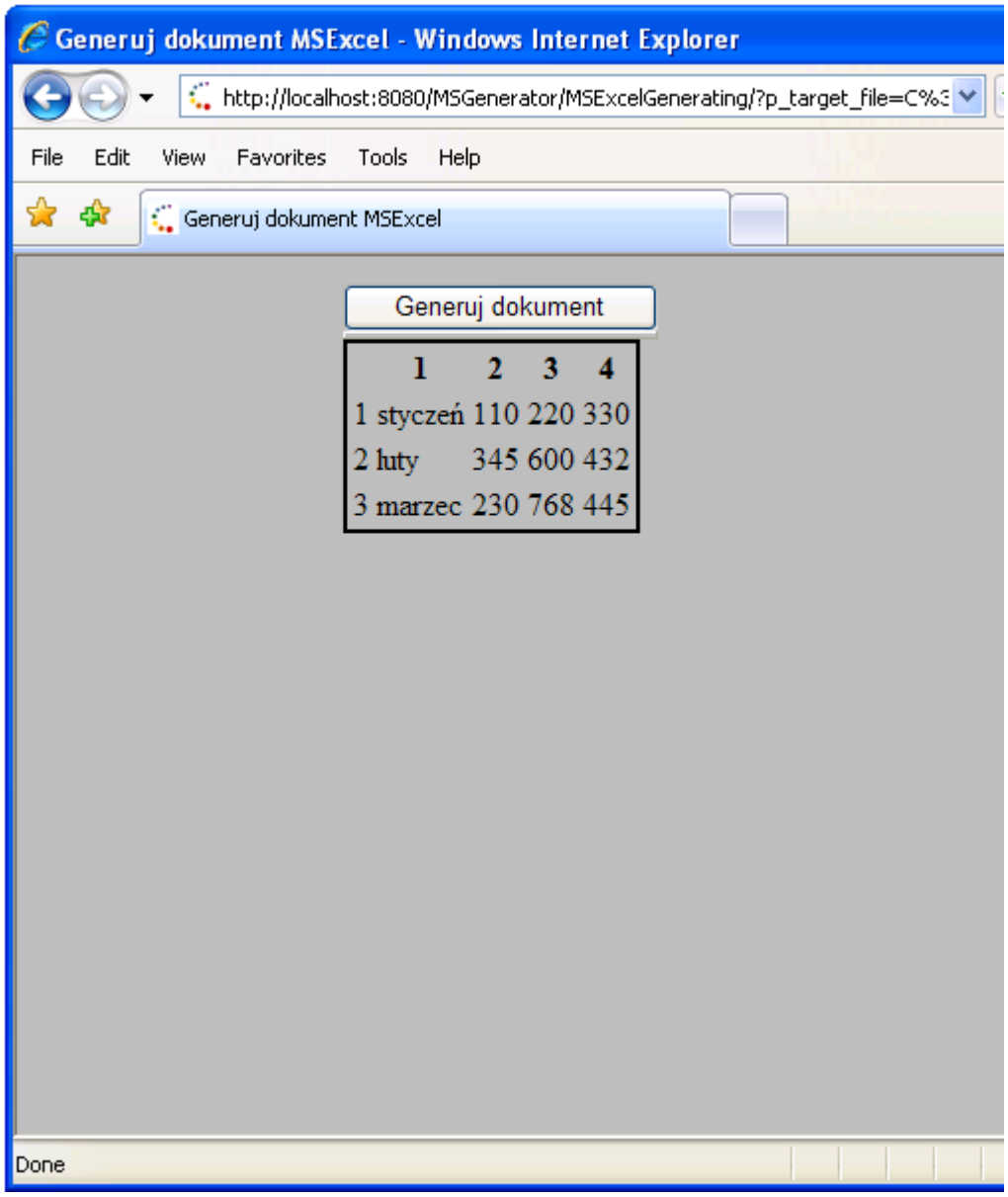
Właściwości, które można w ten sposób zmieniać to:

- numer kolumny (przyjmując 1 jako numer pierwszej kolumny)
- numer wiersza (przyjmując 1 jako numer pierwszego wiersza)
- zawartość (treść) komórki
- kolor czcionki tekstu w komórce
- kolor wypełnienia komórki
- pogrubienie czcionki tekstu w komórce
- pochylenie czcionki tekstu w komórce
- podkreślenie czcionki tekstu w komórce

Wygenerowany formularz HTML zawiera podgląd eksportowanych wartości oraz numer kolumny i wiersza do którego trafią; w takim zakresie komórek, w jakim zawarte są dane w pliku – jeśli na przykład pierwszą kolumną wypełnioną danymi będzie D to numeracja kolumn rozpocznie się od kolumny czwartej, natomiast numeracja wierszy kończy się na ostatnim wierszu występującym w pliku. Pozwala to jeszcze przed właściwym eksportem na ocenę czy dane zostały przygotowane poprawnie.

3.3.2.1 Przykładowy formularz z danymi

Wygląd formularza z przykładowymi danymi eksportowanymi do Excela pokazany jest na rysunku 3.



Rysunek 3 formularz z danymi eksportowanymi do MS Excela

3.3.3 MS Project:

Funkcjonalność części obsługującej Projecta polega na eksporcie odpowiedniej struktury projektu do MS Project. Algorytm eksportujący interpretuje takie elementy jak nazwy poszczególnych zadań, ich czas trwania, daty rozpoczęcia/zakończenia; to, czy są one kamieniami milowymi (*milestones*) oraz kolejność ich następowania po sobie. Ta

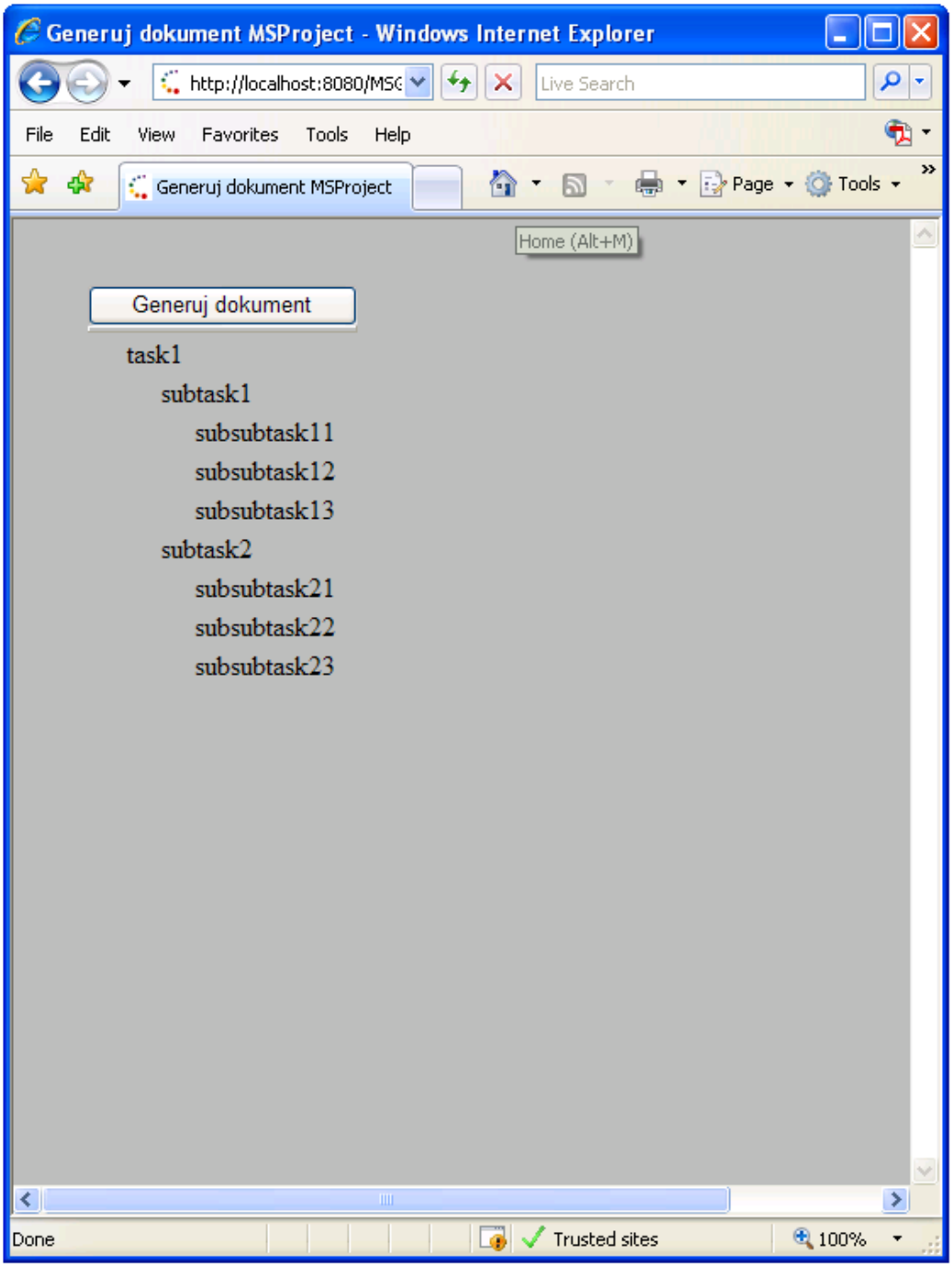
część narzędzia nie wymaga wcześniej utworzonego szablonu; cały plik projektu tworzony jest od zera.

Poszczególne zadania identyfikowane są przez kolejność wystąpienia w dokumencie XML oraz ich ewentualne zagnieżdżenie w innych zadaniach – całość struktury tworzy drzewo zadań odpowiadające całemu projektowi. Poszczególne zadania są numerowane – pozwala to na zbudowanie ścieżki wzajemnie powiązanych zadań poprzez określenie numeru zadania poprzedzającego dane zadanie.

Na wygenerowanym formularzu wynikowym dostępny jest podgląd eksportowanych zadań – prezentowane są ich nazwy oraz daty rozpoczęcia i zakończenia; dodatkowo podzadania są przesunięte w prawo względem zadań w skład których wchodzą – służy to ewentualnej weryfikacji poprawności danych jeszcze przed właściwym eksportem do MS Project.

3.3.3.1 Przykładowy formularz z danymi

Na rysunku 4 przedstawiono wygląd przykładowej struktury danych przed jej eksportem do MS Projecta.



Rysunek 4 formularz z danymi eksportowanymi do MS Projecta

IV. Architektura i technologia wykorzystane w projekcie

Narzędzie MSGenerator opiera się na kilku technologiach związanych bądź współpracujących z platformą J2EE. W tym rozdziale zostały wymienione i pokrótce opisane rozwiązania wykorzystane przy projektowaniu narzędzia eksportującego. Opisy te zostały opracowane na podstawie stron www producentów poszczególnych technologii oraz praktyki zawodowej autora niniejszej pracy.

4.1. Technologia

Jako że współczesne aplikacje biznesowe coraz częściej tworzone są jako aplikacje webowe, narzędzie MSGenerator również zostało zaprojektowane jako uzupełnienie takiej aplikacji. Jako platformę technologiczną autor wybrał jedną z dwóch aktualnie dominujących technologii stosowanych do budowy tego typu aplikacji – **J2EE**.

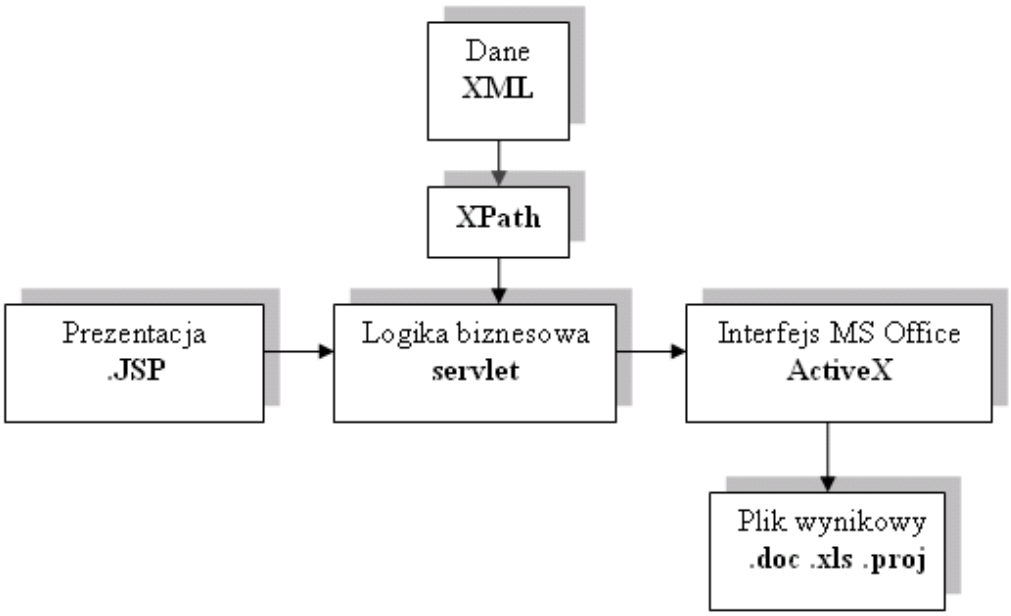
Sama aplikacja uruchomiona jest na serwerze aplikacji **jBoss** w wersji 4.1. Serwer jest przykładowy, w przypadku przystosowania MSGeneratora do współpracy z już istniejącą aplikacją nie byłoby problemu z wykorzystaniem serwera na którym ta ostatnia została umieszczona – narzędzie nie wykorzystuje żadnych cech właściwych tylko dla tego serwera.

Podstawowym komponentem narzędzia są servlety, interpretujące dane w postaci XML (bądź w postaci obiektu typu Document, bądź jako zawartość fizycznego pliku XML) i generujące formularze JSP wraz z zawartymi w nich funkcjami w języku javascript, które odpowiadają za stworzenie bądź otwarcie odpowiedniego dokumentu danego narzędzia z pakietu Office oraz wypełnienie go danymi. Pliki zawierające dane oraz docelowe szablony są wczytywane za pomocą wchodzących w skład narzędzia formularzy JSP – po jednym dla każdego z obsługiwanych narzędzi MS Office.

Wspomniane funkcje opierają się na manipulacjach kontrolkami ActiveX; poszczególne elementy w pliku XML przekazywane są jako parametry odpowiednich funkcji kontrolki.

Użycie XML jako nośnika danych miało na celu możliwość zastosowania różnych punktów wejściowych dla danych do systemu – na przykład plików generowanych przez inne aplikacje bądź tworzonych ręcznie, lub też WebSerwisów, przekazujących serializowalny obiekt Document. Kolejnym powodem ku temu była łatwość tworzenia struktury XML na podstawie danych w innej postaci – z uniknięciem problemów z kodowaniem w przypadku znaków narodowych. Taki plik można zarówno wygenerować z innego narzędzia, czy też nawet stworzyć ręcznie, gdyby zaszła taka potrzeba.

Wykorzystanie poszczególnych technologii w projekcie MSGenerator przedstawione jest na rysunku 5.



Rysunek 5 technologie wykorzystane w aplikacji MSGenerator

Wszystkie wyżej wymienione technologie są darmowe (za wyjątkiem oczywiście pakietu MS Office) – autorowi zależało na pokazaniu, że można zbudować kompletne narzędzie bez sięgania po nierzadko kosztowne rozwiązania – fakt ten miałby znaczenie wobec ewentualnego przygotowania MSGeneratora do sprzedaży (związany z brakiem konieczności zakupu licencji potrzebnych do jego stworzenia); ponadto wdrożenie go

przy rozwijaniu już istniejącej aplikacji nie wiązałoby się z żadnymi dodatkowymi kosztami.

Taki właśnie dobór narzędzi miał również być swego rodzaju dowodem na możliwość stworzenia aplikacji integrującej się z pakietem biurowym Microsoftu, bez konieczności stosowania technologii programistycznych tej firmy.

4.2 XML

Opis technologii XML został oparty na informacjach dostępnych na stronach organizacji W3C (*World Wide Web Consortium*)[2].

XML (*EXtensible Markup Language*) jest opartym na znacznikach (*tagach*) językiem zaprojektowanym jako medium do przechowywania danych. W przeciwieństwie do „dopełniającego" go HTML (służącego do prezentowania danych) nie posiada z góry zdefiniowanych znaczników – z założenia jest samoopisujący i pozwala na samodzielne tworzenie i używanie tagów, jak i w zasadzie dowolnie projektowanie struktury dokumentu. Stosunkowo największą różnicą pomiędzy nim a HTML jest to, że sam z siebie nic nie robi ani nie posiada „natywnego" interpretera (jakim jest przeglądarka internetowa dla języka HTML) – XML pełni wyłącznie rolę kontenera opakowującego dane w postaci tekstowej.

Ważny podkreślenia jest fakt, że XML pozostaje całkowicie niezależny od rodzaju sprzętu jak i technologii programistycznych – tylko i wyłącznie od decyzji programistów zależy to, czy zdecydują się wykorzystać ten sposób przechowywania danych w swoich aplikacjach; przywiązanie do żadnej konkretnej technologii nie stanowi ograniczenia dla użycia XML.

Jedną z cech XML (notabene wykorzystaną w MSGeneratorze) jest jego „przeźroczystość" dla różnych typów aplikacji – idealnie nadaje się do współdzielenia danych; na przykład pomiędzy bazą danych a aplikacją kliencką – łatwo można zaimplementować z jednej strony generowanie danych w postaci dokumentu XML, z drugiej natomiast jego odczyt – taki pomost daje w zasadzie nieograniczone możliwości

[2] http://www.w3schools.com/xml/default.asp

integracji różnych platform i technologii. Nic nie stoi na przeszkodzie, aby przykładowo z funkcjonalności aplikacji w technologii J2EE skorzystać w innej aplikacji napisanej w technologii ASP.NET, bądź też wykorzystać opartą o XML bazę danych która natywnie wygeneruje dane wyjściowe jako źródło danych dla aplikacji.

Ta właśnie cecha powoduje też, że wszelkie zmiany które mogą zajść w cyklu życia aplikacji, takie jak zmiana bazy danych, przeglądarki czy nawet systemu operacyjnego na którym dana aplikacja działa pozostają bez większego wpływu na dane w postaci dokumentów XML – te zawsze pozostają takie same. Pośrednio ma to wpływ również na wszechstronność aplikacji wykorzystujących XML jako medium do transportu czy przechowywania danych, gdyż mogą one bezprobleмowo współpracować z innymi aplikacjami; nie uwzględnionymi podczas projektowania pierwotnego systemu.

4.2.1 Struktura dokumentu XML

Przykładowy dokument XML:

```
<?XMLversion ="1.0"  encoding="UTF-8"?>
<document_variables>
    <variable>
        <name>p_data_zawarcia</name>
        <value>25.01.2008</value>
         <synonym>Data zawarcia</synonym>
    </variable>
</document_variables>
```

W pierwszej linii znajduje się deklaracja dokumentu, obejmująca w tym przypadku wersję XML oraz kodowanie danych.

Dalej znajduje się korzeń dokumentu (*root*), obejmujący całą pozostałą treść. Następnie, niżej w hierarchii są „dzieci” korzenia – które same mogą posiadać zagnieżdżone w nich „dzieci”. Dokument kończy się tagiem zamykającym korzeń.

Łatwo zauważyć, że elementy dokumentu formują dowolnie głęboko zagnieżdżone drzewo.

Jednym z nielicznych formalnych wymogów struktury dokumentu XML jest fakt, że każdy tag musi posiadać znacznik zamykający – za wyjątkiem deklaracji dokumentu, nie wchodzącej technicznie w skład zawartych w nim danych. Kolejnym jest konieczność rozróżniania w nazwach tagów małych i wielkich liter.

<value> oznacza coś innego niż **<Value>**

Ponadto, tagi muszą być zamykane w takiej samej kolejności jak otwierane. Poniżej przykład niepoprawnej konstrukcji:

<variable><name>nazwa**</variable></name>**

Całą struktura musi też posiadać dokładnie jeden korzeń – nawet jeśli jest on jedynym użytym tagiem, musi w dokumencie wystąpić.

Poszczególne tagi mogą posiadać atrybuty (w poniższym przykładzie pokazane wytłuszczonym drukiem):

<property **property_path**="Font.ColorIndex" **property_value**="3" />.

Wymogiem formalnym dla atrybutu jest konieczność objęcia jego wartości cudzysłowem bądź apostrofem.

Z punktu widzenia logiki przechowywanych danych, te same rzeczy można przedstawić i jako atrybut, i w postaci znacznika – „dziecka”; wybór jednego z tych sposobów pozostawia się osobistej preferencji programisty.

Istnieją jednakże pewne wytyczne, na podstawie których można oprzeć decyzję które treści przedstawiać w jakiej postaci – a stanowią je ograniczenia związane z atrybutami:

- Elementy mogą przybrać kilka wartości jednocześnie – atrybuty nie
- Elementy mogą zawierać w sobie strukturę drzewiastą – atrybuty nie
- Trudniej jest dodać do struktury nowy atrybut niż element – ma to znaczenia przy rozszerzaniu struktury na poczet ewentualnego rozszerzenia funkcjonalności

Stąd też logicznym wydaje się stosowanie elementów do samych danych, a atrybutów do informacji te dane opisujących – dobrym przykładem jest tu id węzła w strukturze drzewa XML, który naturalniej przedstawia się jako atrybut.

Jednym z problemów związanych z przechowywaniem danych w postaci dokumentów z tagami jest fakt, iż niektóre ze znaków mają specjalne znaczenie w strukturze dokumentu – są to symbole otwierające (**<**) i zamykające (**>**) znacznik, a także apostrof (**‘**), cudzysłów (**”**) oraz ampersand (**&**). W celu rozwiązania tego problemu stosuje się specjalne ciągi literałów zwane *entity reference*, wykorzystywane w celu zastąpienia problematycznych znaków. Przedstawia je tabela 1.

<	<
>	>
‘	'
”	"
&	&

Tabela 1 specjalne znaki XML

Formalnie jedynie znaki (**<**) i (**&**) są zabronione w treści dokumentu, jednakże do dobrej praktyki należy zastępowanie wszystkich pięciu – tym bardziej że dużo łatwiej jest w aplikacji korzystającej z dokumentu parsować go, jeśli nie zawiera cudzysłowów ani apostrofów – czyli znaków które w większości języków programowania ograniczają łańcuchy tekstowe.

Dopuszczalne jest stosowanie kombinacji cudzysłowu i apostrofu jako wartości atrybutu – zakłada się że pierwszy z nich który wystąpi traktuje się jako początek wartości; jeśli wówczas drugi wykorzysta się pomiędzy pierwszymi, to nie psuje on samej wartości – czyli dopuszczalny jest przedstawiony przykład:

<ksiazka autor='**Anthony "Tony" Stark**'>

Wytłuszczona treść stanowi poprawną wartość atrybutu, pomimo wystąpienia wewnątrz niej cudzysłowu.

Struktura dokumentu XML dopuszcza stosowanie komentarzy – otoczonych specjalnymi tagami **<!-- komentarz -->**.

Do wymienionych uprzednio różnic pomiędzy XML a HTML należy również sposób w jaki traktowane są tzw. białe spacje – o ile HTML zawsze zmienia takie znaki wielokrotnie występujące obok siebie w jeden, o tyle XML zezwala na ich powielanie i ich nie usuwa.

Z punktu widzenia danych przechowywanych w dokumencie XML w postaci sformatowanej, istotny jest sposób w jaki w dokumencie zakodowany jest symbol nowej linii – w odróżnieniu od standardu wykorzystanego chociażby w systemie Windows, gdzie nowa linia to para znaków CR oraz LF (odpowiednio *carriage return* i *line feed*); w XML nową linię reprezentuje jedynie symbol LF.

Nie należy również zapominać o właściwym kodowaniu; dokument może przechowywać treść zawierającą znaki narodowe, ale do poprawnej jej interpretacji wymagane jest zadeklarowanie i obsługa kodowania dokumentu.

4.2.2 Zasady nazewnictwa elementów w dokumencie XML

Następujące reguły odnoszą się do nazw poszczególnych elementów dokumentu XML:

- Nazwy mogą składać się z liter, cyfr oraz innych znaków
- Nazwy nie mogą zaczynać się od cyfry ani od znaku interpunkcyjnego
- Nazwy nie mogą zaczynać się od słowa XML; niezależnie od tego czy składa się ono z małych czy wielkich liter
- Nazwy nie mogą zawierać w sobie spacji

4.2.3 Walidacja

Istnieją dwa poziomy „poprawności” dokumentu XML:

- Dokument XML dobrze uformowany (*Well Formed XML Document*) – aby móc nazwać tak dany dokument, musi on spełniać następujące warunki:
  - musi posiadać korzeń
  - wszystkie elementy muszą posiadać znacznik zamykający

- musi być zachowane rozróżnienie znaczników pod kątem małych i wielkich liter
- wszystkie elementy muszą być poprawnie zagnieżdżone
- wartości atrybutów muszą zostać ujęte w cudzysłowach

- Poprawny dokument XML (*Valid XML Document*) – spełnia on wszystkie warunki dobrze uformowanego dokumentu; ponadto jest zgodny z warunkami zapisanymi w Definicji Typu Dokumentu (*Document TypeDefinition*, lub *DTD*).

W celu ułatwienia walidacji dokumentów wykorzystywane są dwa standardy plików opisujących dopuszczalną strukturę XML,

4.2.3.1 DTD

Zadaniem pliku DTD jest określenie dopuszczalnej struktury dokumentu XML. Przykładowe odwołanie do takiego pliku umieszczone jest w drugiej linijce dokumentu i wygląda następująco:

<?xml version="1.0" encoding="UTF-8"?>
<**!DOCTYPE detede SYSTEM "Detede.dtd"**>

Istnieje też możliwość dołączenia zawartości pliku DTD do samego dokumentu XML – wówczas cała jego treść zawiera się wewnątrz elementu DOCTYPE.

Przykładowy blok DTD wygląda następująco:

<!DOCTYPE msexcel_data [
<!ELEMENT msexcel_data (cell)>

<!ELEMENT cell(property)>
<!ELEMENT property (#PCDATA)>

Użycie DTD znacznie ułatwia wymianę danych pomiędzy różnymi systemami, udostępniając wygodny a jednocześnie spójny sposób weryfikacji poprawności struktury dokumentu.

4.2.3.2 XML Schema

Istnieje alternatywny wobec DTD standard definiowania struktury dokumentu, i jest nim XML Schema. Zaproponowany przez organizację W3C (*World Wide Web Consortium*), ma w założeniu zastąpić DTD jako standard określania poprawności danych XML – między innymi z racji swej rozszerzalności, zachowania zgodności impedancji z dokumentem (jest definiowany jako XML) oraz wsparcia dla określonych typów danych.

4.3 ActiveX

Opis kontrolek ActiveX oparto na informacjach zawartych na stronach www firmy Microsoft[3].

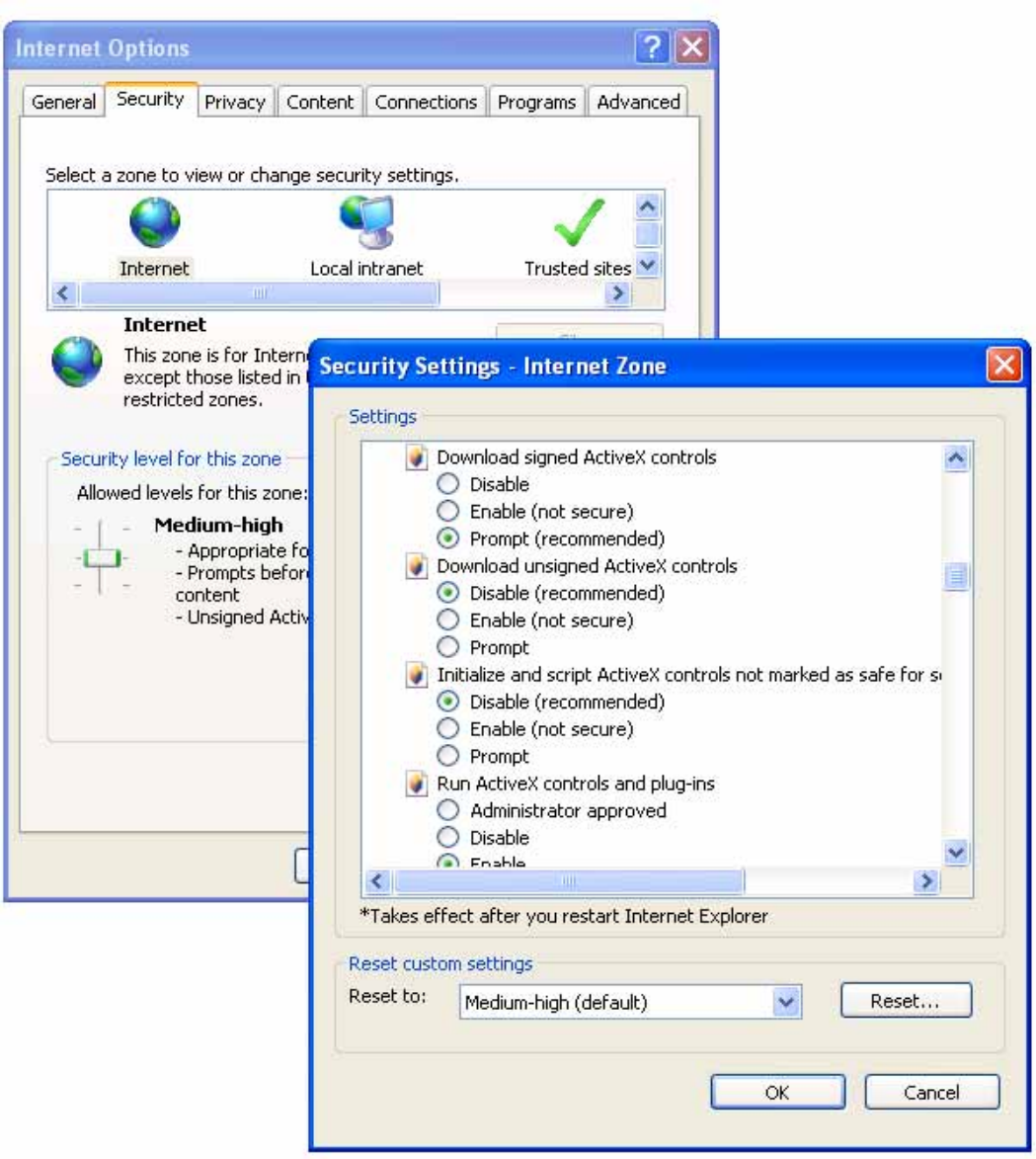
U podstaw rozwiązania stojącego za zdolnością MSGeneratora do współpracy z aplikacjami MS Office znajdują się kontrolki ActiveX. Ta opracowana przez Microsoft technologia, będąca podzbiorem komponentów COM (*Component Object Model*) opiera się na obiektach bądź komponentach, możliwych do zanurzenia na stronie www. Kontrolki takie udostępniają z góry zdefiniowaną funkcjonalność, rozszerzając możliwości samej strony czy formularza.

Technologia ta natywnie ograniczona jest jedynie do współpracy z przeglądarką Internet Explorer (wzwyż od wersji 3.0), istnieje jednak plugin do przeglądarki Firefox, umożliwiający uruchomienie na niej kontrolki – niestety, najnowsze wersje tej przeglądarki go nie wspierają.

ActiveX znakomicie rozszerza możliwości aplikacji uruchamianych z przeglądarki; daje programistom o wiele większe możliwości niż porównywalne koncepcyjnie applety javy; włączając w to również możliwość tworzenia ich w różnych językach programowania. Najważniejsze z nich to:

[3] http://www.microsoft.com

- C#
- C++
- Borland Delphi
- Visual Basic



Rysunek 6 ustawienia bezpieczeństwa przeglądarki dotyczące ActiveX

Wszystkie te możliwości niosą jednak ze sobą pewne zagrożenia – przy nieprawidłowych ustawieniach bezpieczeństwa w przeglądarce, kontrolki te ściągane są

na maszynę klienta i instalowane praktycznie bez kontroli użytkownika, co stanowi istotne ryzyko złamania zabezpieczeń systemu. Dodatkowym zagrożeniem jest to, że kontrolki raz zainstalowane są stosunkowo trudne do usunięcia.

Wymienione wyżej powody spowodowały, że Microsoft został zmuszony do wprowadzenia dodatkowych środków bezpieczeństwa, takich jak cyfrowy podpis pakietów instalacyjnych, obowiązek zadeklarowania kontrolki jako skryptowo bezpiecznej czy też na bieżąco aktualizowana lista kontrolek uznanych za niebezpieczne.

Od wersji 7.0 Internet Explorera wprowadzono mechanizm **ActiveX Opt-In** – który automatycznie blokuje kontrolki na stronie www, wymagając jawnego potwierdzenia przez użytkownika chęci ich zainstalowania i uruchomienia.

Na rysunku 6 przedstawione są niektóre z opcji ustawień bezpieczeństwa przeglądarki, związane z kontrolkami ActiveX.

Abstrahując jednakże od kwestii bezpieczeństwa, zyski jakie daje stosowanie kontrolek uzasadniają związane z nimi ryzyko – szczególnie w środowisku aplikacji znajdującej się w firmowym intranecie, dla którego można poluźnić zwykle restrykcyjną politykę bezpieczeństwa, tym samym umożliwiając wykorzystanie pełnej funkcjonalności opisywanych komponentów.

MSGenerator wykorzystuje kontrolki **Word.Application**, **Excel.Application** oraz **MSProject.Application**, dające w sumie pełną funkcjonalność odpowiadających im narzędzi pakietu MS Office. Instancje kontrolek są tworzone przez klasę javascript **ActiveXObject** i obsługiwane przez polecenia w tymże języku. Składnia i nazwy metod oraz właściwości obiektów są podobne do składni ich odpowiedników w języku Visual Basic, w którym można tworzyć makra w aplikacjach pakietu Office; dzięki czemu przy pracy z nimi można opierać się na dokumentacji dostępnej w systemie pomocy poszczególnych narzędzi pakietu.

4.4 Servlety

Opis technologii servletów został oparty na informacjach zawartych na stronach www firmy Sun Microsystems[4].

Formularze zawierające w sobie funkcje tworzące dokumenty są generowane przez servlety.

Servlety to obiekty napisane w języku java, których głównym celem jest dynamiczne przetwarzanie żądań i generowanie odpowiedzi na nie. Specyfikacja tej technologii powstała w 1997 roku w Sun Microsystems; od tamtej pory servlety sukcesywnie zdobywały popularność jako proste w użyciu i wydajne elementy umożliwiające dynamiczne generowanie stron www jak i dokumentów XML.

Jedną z immanentnych cech servletów jest ich zdolność do utrzymywania własnego stanu niezależnie od sesji serwera – jest to osiągane poprzez wykorzystanie „ciasteczek" (*cookies*) HTTP, zmiennych sesji bądź też przepisywanie adresu url.

Kolejną niewątpliwą zaletą servletów jest ich „lekkość" – na serwerze są wykonywane nie jako ciężki proces systemu operacyjnego, ale stosunkowo lekki wątek. Dodatkowo, servlet jest obiektem wielokrotnego użytku – jest ładowany na serwer tylko raz, po pierwszym wywołaniu którejś z jego usług bądź po zmianie kodu stojącego za nim. Równie przydatna jest jego zdolność do jednoczesnej obsługi wielu żądań, a także synchronizacji pomiędzy nimi – przykładem takiego wykorzystania może być chociażby oparty na nich system telekonferencyjny.

Przewagą servletów nad podobnym pod względem zastosowań protokołem CGI (*Common Gateway Interface*) jest ich zdolność do bezpośredniej komunikacji z web serwerem oraz mniejsze obciążenie serwera – wadą CGI jest tworzenie nowego wątku na serwerze związane z każdorazowym wywołaniem usługi za jego pomocą.

Servlety są umieszczone w tzw. kontenerze web, odpowiedzialnym za zarządzanie cyklem życia, mapowaniem adresów url na właściwy servlet oraz weryfikowaniem uprawnień klienta wysyłającego żądanie.

4.4.1 Cykl życia servletu

[4] http://java.sun.com/products/servlet/

1. Klasa servletu jest ładowana do kontenera servletów
2. Kontener wywołuje metodę *init*, inicjalizującą servlet
3. Po inicjalizacji, servlet może przyjmować i przetwarzać żądania
4. Kontener wywołuje metodę *destroy*, usuwając klasę z pamięci

4.4.2 Architektura servletów

Na poziomie technicznym można określić servlet jako instancję klasy java implementującą interfejs *javax.servlet.Servlet*. Zwykle jednak wykorzystuje się którąś z już istniejących implementacji wymienionego interfejsu – to jest *javax.servlet.GenericServlet* lub *javax.servlet.http.HttpServlet*.

Zainicjalizowany servlet jest gotów do przetwarzania żądań. Odpowiada za nie metoda *service (ServletRequest request, ServletResponse response)*. Z racji tego, że metoda ta może być wywoływana jednocześnie przez różnych klientów, powinna być implementowana w sposób zapewniający bezpieczną wielowątkowość.

4.5 JavaServer Pages

Opis technologii JSP został opracowany na postawie informacji zawartych na stronach www firmy Sun Microsystems[5].

JavaServer Pages (*JSP*) to wchodząca w skład specyfikacji J2EE technologia, służąca do budowania prezentacyjnej warstwy aplikacji – to jest dynamicznych stron webowych.

Główną zaletą wykorzystania JSP jest potencjalna łatwość w oddzieleniu logiki aplikacji od warstwy prezentacji, co znacznie ułatwia wielokrotne wykorzystanie elementów aplikacji i samo ich tworzenie.

Pod względem zakresu funkcjonalności, można je określić jako wysoko poziomowy abstrakt servletów – i faktycznie, na etapie kompilacji są one przekształcane w servlety. Może to zajść na jeden z dwóch sposobów – albo są przetwarzane na kod servletu w javie, kompilowany przez kompilator tego właśnie języka; albo też

[5] http://java.sun.com/products/jsp/

bezpośrednio na kod bajtowy. Istnieje też możliwość interpretowania strony .jsp „w locie" – za pomocą interpretera JSP Weaver.

Ta ostatnia możliwość co prawda wiąże się ze zwiększonym narzutem wydajnościowym, ale rekompensuje to dużo szybszym przeładowaniem strony po zmianie jej kodu (istotne dla aplikacji charakteryzujących się częstymi zmianami).

Korzystając z JSP Weavera można też wykorzystywać tzw. skryptlety (*scriptlets*) – czyli bloki kodu w języku java zawarte na stronie.jsp pomiędzy znacznikami **<% %>**, pozwalające na wywoływanie metod w tym języku oraz deklarowanie i wykorzystywanie zmiennych.

W przeciwieństwie do właściwego kodu jsp, skryptlety same w sobie nie generują kodu HTML – aby mogły wysłać jakiś tekst na stronę, potrzebne jest skorzystanie z predefiniowanej zmiennej *out* poprzez konstrukcję *System.out.println(„*jakiś tekst*")*.

Podobnie jak servlety, również strony .jsp mają dostęp do obiektów *response* oraz *request* poprzez zmienne o tej samej nazwie.

Aby skorzystać z możliwości oferowanych przez javę w skryptletach, potrzebne są dyrektywy (*directives*) – zastępujące znane z javy dyrektywy *import*. Przykładowe użycie:

<%@ page import="java.util.*" %>

Przy pomocy dyrektyw można również zawrzeć zawartość innego pliku .jsp na formularzu – do tego celu stosuje się dyrektywę **include**.

<%@ include file="hello.jsp" %>

Aby wykorzystać zmienne i metody zdefiniowane w danym miejscu strony .jsp w innym bloku skryptletu, należy posłużyć się deklaracjami – które otacza się znacznikami **<%! %>**.

4.5.1 Znaczniki JSP

Znaczniki (*tags*) JSP są podobne do znaczników HTML. Zawarte są w nawiasach <>; mogą posiadać znacznik otwierający, zamykający oraz „ciało” (*body*). Nazwy znaczników składają się z części określającej typ tagu; właściwej nazwy oraz oddzielającego obie części dwukropka.

Rozróżniamy dwa rodzaje znaczników – ładowane z zewnętrznego pliku oraz tzw. predefiniowane – w przypadku tych ostatnich ich nazwy rozpoczynają się słowem kluczowym **jsp:** W przypadku często używanego znacznika **jsp:include,** treść pliku będącego celem odnośnika nie jest ładowana od razu do strony .jsp, lecz wywołanie następuje dopiero w fazie wykonania (*runtime*) – co znacznie przyśpiesza ładowanie strony do przeglądarki klienta, nie wymuszając konieczności ściągania przez nią często niewykorzystywanego kodu.

4.5.2 Sesje JSP

Strony JavaServer Pages umożliwiają wykorzystanie sesji – czyli obiektu powiązanego z użytkownikiem przebywającym na stronie. W dużym uproszczeniu, sesja działa jak tablica, do której można wpisywać i pobierać dane – przy czym każdy użytkownik posiada swoją własną – dzięki temu rozwiązaniu możliwa jest jednoczesna praca wielu użytkowników, każdego na jego własnej instancji tej samej strony JSP.

Wpisanie danych do sesji polega na wywołaniu następującej metody:

```
<%
  session.setAttribute( "nazwa_atrybutu", wartosc_atrybutu );
%>
```

Jednym z istotnych parametrów sesji jest jej czas trwania (*age*) – określający po jakim czasie bezczynności użytkownika dane zostają utracone.

4.6 dom4j

Opis technologii dom4j został opracowany na podstawie informacji dostępnych na stronie www projektu o tej samej nazwie[6].

Dom4j jest biblioteką typu *open source* napisaną w języku java, wspomagającą pracę z dokumentami typu XML, XPath oraz XSLT. Znacznie prostsza w użyciu niż interfejs DOM, jest z nim mimo wszystko w pełni zgodna – pozwala zatem na jednoczesne korzystanie z obu.

Cała biblioteka rozpowszechniana jest z licencją pozwalającą na jej bezpłatne użycie nawet w produktach komercyjnych, które nie są dystrybuowane razem ze swoim kodem.

Algorytmy w kodzie MSGeneratora związane z obsługą XML oparte są właśnie na tej bibliotece – wykorzystana jest tu jej zdolność do stworzenia kompletnego dokumentu z reprezentującego go łańcucha tekstowego XML.

Dom4j w dużym stopniu wykorzystuje polimorfizm; dzięki niemu wszystkie klasy związane z dokumentem XML implementują interfejs Węzła (*Node*) – co stanowi oczywiste ułatwienie przy tworzeniu algorytmów opartych na Dom4j.

4.6.1 Cechy dom4j

- *Open source*
- Oparcie na interfejsach java
- Wsparcie dla kolekcji java
- Zdolność do korzystania z dowolnego parsera SAX oraz XMLFilter
- Konwersja do i z drzew DOM
- Implementacja interfejsu DOM
- Zintegrowane wsparcie dla API XPath
- Wbudowana implementacja XPath
- Wsparcie dla JAXP/TrAX
- Zdolność do przetwarzania ciągłych strumieni XML
- Zdolność do przetwarzania bardzo dużych dokumentów

[6] http://www.dom4j.org/

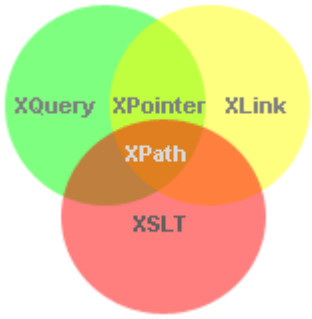
- Wsparcie dla typu danych XML Schema

4.7 XPath

Opis XPath został opracowany na podstawie informacji zawartych na stronach www organizacji W3C[7].

XPath (*XML Path Language*) jest językiem zapytań operującym na dokumentach XML. Został opracowany przez konsorcjum W3C; aktualnie wykorzystywany jest w dwóch wersjach – 1.0 oraz 2.0. W innym rozumieniu, jest to też składnia służąca definiowaniu części dokumentu w formacie XML.

Na rysunku 7 przedstawiono zależność między XPath a innymi technologiami wspomagającymi korzystanie z XML.



Rysunek 7 XPath i inne technologie XML

XPath opiera się na drzewiastej strukturze dokumentu XML; udostępnia funkcjonalność nawigowania po drzewie oraz wyszukiwania węzłów dokumentu według wybranych kryteriów. Zawiera ponad sto wbudowanych funkcji, służących głównie manipulacjom na łańcuchach tekstowych i zmiennych numerycznych, a także działaniach na poszczególnych węzłach dokumentu.

W zakresie XPath posługujemy się następującymi terminami:

- Węzły (*nodes*)– występuje siedem rodzajów węzłów. Są to:
    - Element

[7] http://www.w3schools.com/XPath/default.asp

  - Atrybut
  - Tekst
  - Przestrzeń nazw
  - Instrukcja przetwarzania
  - Komentarz
  - Korzeń drzewa (tzw. węzeł dokumentu lub *root*)
- Wartości atomowe (*atomic values*) – są to węzły nie posiadające dzieci ani rodziców
- Elementy (*items*)- są to zarówno węzły jak i wartości atomowe

Poszczególne węzły mogą pozostawać ze sobą w rozmaitych relacjach:

- Rodzice – każdy element i atrybut posiada jednego rodzica
- Dzieci – elementy mogą mieć od zera do wielu dzieci
- Rodzeństwo – tworzą je węzły posiadające wspólnego rodzica
- Przodkowie – są nimi rodzice węzła, ich rodzice i wyżej w hierarchii
- Potomkowie – dzieci węzła, ich dzieci i niżej w hierarchii

Przykładowe zapytanie dla pokazanej w podrozdziale 4.2.1 struktury wygląda następująco:

document_variables//variable/name

Zapytanie to wybiera wszystkie podwęzły typu **name** nazwanego węzła **variable**, które z kolei znajdują się gdziekolwiek wewnątrz węzła **document_variables**.

Tabela 2 prezentuje wyrażenia wchodzące w skład składni XPath:

Węzeł	Wybiera wszystkie podwęzły węzła o podanej nazwie
/	Wybiera z korzenia drzewa
//	Wybiera węzły w dokumencie znajdujące się w bieżącym węźle; niezależnie od tego gdzie się znajdują
.	Wybiera bieżący węzeł
..	Wybiera rodzica (nadwęzeł) bieżącego węzła
@	Wybiera atrybut węzła

Tabela 2 wyrażenia XPath

W tabeli 3 zaprezentowane są przykłady wykorzystania wyrażeń XPath.

/węzeł	Wybiera wszystkie podwęzły nazwanego węzła
Węzeł/**podwęzeł**	Wybiera wszystkie elementy typu **podwęzeł** znajdujące się w danym węźle
//**zmienna**	Wybiera wszystkie węzły typu **zmienna**, niezależnie od tego gdzie w dokumencie się znajdują
Węzeł//**podwęzeł**	Wybiera wszystkie elementy typu **podwęzeł** znajdujące się w danym węźle; również te, które znajdują się w podwęzłach podwęzła
//@**atrybut**	Wybiera wszystkie **atrybuty** o podanej nazwie; niezależnie od tego gdzie się znajdują

Tabela 3 przykładowe wyrażenia XPath

Tabela 4 prezentuje wykorzystanie predykatów XPath.

/bookstore/book[1]	Wybiera pierwszy (wg Specyfikacji W3C) bądź drugi (implementacja IE 5.0 i późniejsze) podwęzeł danego typu w węźle
/bookstore/book[last()]	Wybiera ostatni element typu podwęzeł w danym węźle
//title[@lang]	Wybiera wszystkie węzły posiadające atrybut o podanej nazwie
//title[@lang='eng']	Wybiera wszystkie węzły posiadające atrybut o podanej nazwie i wartości
/bookstore/book[price>35.00]/**title**	Wybiera wszystkie elementy typu **title** spośród elementów book posiadający atrybut cena > 35 w węźle bookstore

Tabela 4 predykaty XPath

Tabela 5 pokazuje symbole wieloznaczne (*wildcards*) możliwe do zastosowania w wyrażeniach XPath.

*	Zastępuje dowolny element
@*	Zastępuje dowolny atrybut
node()	Zwraca wszystkie węzły dowolnego typu

Tabela 5 symbole wieloznaczne w XPath

Operator „|” pozwala łączyć wyniki kilku różnych wyrażeń XPath. Ponadto, ścieżka lokacyjna do danego miejsca może być względna (zaczynająca się od nazwy węzła) lub bezwzględna (rozpoczynająca się symbolem „:/” (*slash*)).

4.8 Office Open XML

Począwszy od wersji 2003 MS Office umożliwia zapisywanie plików w formacie XML. Format ten, zwany Office Open XML (także OpenXML i OOXML) znalazł zastosowanie w Wordzie, Excelu i Powerpoincie.

W odróżnieniu od dotychczasowych, binarnych formatów wykorzystywanych przez Office, OpenXML jest darmowy i otwarty. Fakt ten miał (i wciąż ma) wielkie znaczenie dla twórców innych aplikacji biurowych, które wreszcie mogły doczekać się pełnej kompatybilności z plikami pakietu Microsoftu.

OOXML szybko nabrał dużego znaczenia; o ile w Office 2003 był on niejako dodatkiem do standardowego formatu; o tyle w wersji 2007 stał się domyślnym sposobem zapisywania plików.

4.8.1 Struktura plików OOXML

Dokument zapisany w formacie OOXML tworzy cały pakiet plików, zgodny ze standardem *Open Packaging Convention*. OPC wykorzystuje mechanizmy wywodzące się z formatu ZIP; został zaprojektowany do przechowywania plików XML razem z plikami w innych formatach, które wspólnie tworzą skompresowany kontener.

Plik w postaci OOXML składa się zatem z pliku stanowiącego właściwą treść oraz wszelkich elementów dodatkowych, takich jak pliki graficzne, video, wykresy i temu podobne.

4.9 Narzędzia

MSGenerator został stworzony przy użyciu zintegrowanego środowiska programistycznego Eclipse z rozszerzeniem Europa w wersji 3.3.1. Jako serwer aplikacji wykorzystany został jBoss 4.0. Serwer został uruchomiony na maszynie z systemem Windows XP SP 3; do testów wykorzystano przeglądarkę Internet Explorer 7.0.

Przykładowe pliki XML zostały utworzone w edytorze kED 2.1.4.0 – wykorzystanym z racji wygodnego operowania różnymi kodowaniami tekstu.

4.10 Algorytmy

MSGenerator wykorzystuje stosunkowo prosty algorytm. Przekazany do narzędzia plik XML odpowiadający strukturze danego typu dokumentu MS Office zostaje przekonwertowany na obiekt typu Document – przy czym pobierana jest tylko jego treść, sam plik nie zostaje fizycznie zapisany na serwerze.

Następnie, z powstałych obiektów zapytania XPath wyłuskują jednostkowe dane, które z kolei tworzą kolekcje bazowych obiektów, takich jak zmienna dokumentu, zadanie projektu bądź komórka arkusza Excela – za tą część odpowiada centralna klasa systemu, nazwana XMLReader. Zbierane są zarówno dane „pojedyncze" (jak na przykład zmienne dokumentu) w kolejności ich występowania, jak i zagnieżdżone w sobie, w sposób rekurencyjny (struktura projektu) czy też wreszcie kolekcje (dane do tabeli w dokumencie).

Zmienne dokumentu Worda wyszukiwane są przez wyrażenie XPath:

document_variables//variable/name

Następnie; dla każdej znalezionej zmiennej kolejne zapytania szukają jej wartości

document_variables//variable[name='elem_name']/value

oraz synonimu

document_variables//variable[name='elem_name']/synonym.

Oba zapytania wykorzystują nazwę zmiennej jako klucz i w rezultacie znajdują zawierające te elementy kolekcje.

Tablice Worda tworzone są podobnie – zapytanie

document_variables//table

znajduje wszystkie zdefiniowane w dokumencie tabele – są one identyfikowane przez będące atrybutami numery, odpowiadające numerowi porządkowemu ich wystąpienia w docelowym pliku.

Następnie w dokumencie wyszukiwane są znaczniki reprezentujące wiersze danej tabeli – realizuje to zapytanie

document_variables//table[@number='table_number']//row

W znalezionych w ten sposób wierszach zapytanie

el.getUniquePath()+//cell//value

wyszukuje komórki – przy czym **el** to znaleziony wiersz, a metoda **getUniquePath()** zwraca bezwzględną ścieżkę do niego w dokumencie XML. Same komórki identyfikowane są przez ich numer wiersza i kolumny – również przechowywane jako atrybuty.

Dane dla arkusza Excela wyszukiwane są w odpowiadającym mu dokumencie XML poprzez zapytanie

msexcel_data/cell

Znalezione obiekty przeszukiwane są pod kontem obowiązkowych i opcjonalnych atrybutów, zaś same komórki identyfikowane przez numer wiersza i komórki – odpowiednio atrybuty **row** i **col**.

Dokument reprezentujący treść pliku Projecta wymaga innego podejścia niż dwa opisane powyżej – z racji dowolnie głębokiego zagnieżdżenia w sobie poszczególnych zadań (*tasków*) konieczne było zastosowanie rekurencyjnych metod do ich odczytu.

Zapytanie XPath

msproject_data/task

wyszukuje zadania znajdujące się najwyżej w hierarchii projektu. Dla każdego ze znalezionych wywoływana jest kolejna metoda, w podobny do opisanego powyżej sposób znajdująca podzadania i nadająca im właściwy poziom wcięcia (*indent level*).

Powstałe kolekcje obiektów następnie przekazywane są do metod wykorzystujących je do wygenerowania funkcji javascript operujących na odpowiedniej kontrolce ActiveX – które z kolei wykonują żądane operacje, zwracając w rezultacie utworzony bądź zmodyfikowany dokument.

Modyfikowane pliki wynikowe Excela i Worda (plik Projecta jest tworzony od zera) również nie trafiają na serwer – są wypełniane treścią w pamięci operacyjnej serwera, a następnie otwierane na maszynie klienta, który może je zapisać na dysku czy po prostu wydrukować.

4.11. Opis przykładowej implementacji

MSGenerator w wersji zaprezentowanej jako projekt będący uzupełnieniem niniejszej pracy magisterskiej jest prototypem, z czym wiążą się niestety pewne ograniczenia i uproszczenia.

Jako przykładowe źródło danych wejściowych dla systemu autor wybrał plik – jako najprostszy do testowania i implementacji; brak jest interfejsu umożliwiającego wywołanie funkcjonalności narzędzia dla danych w postaci klasy *document* – jednakże zaimplementowane metody klasy XMLReader, realizujące właściwą funkcjonalność czytania danych z obiektu XML Document dopuszczają parametry wejściowe w takiej postaci (pod kątem przyszłego rozszerzenia funkcjonalności).

Istnieją różnice w składni poleceń obsługiwanych przez kontrolki ActiveX w różnych wersjach językowych poszczególnych aplikacji pakietu Office (na przykład, MS Project w wersji polskiej posiada parametr „dni”, natomiast w angielskojęzycznej – „days”) – prototyp dostosowany został tylko do pakietu w wersji anglojęzycznej.

Kolejnym uproszczeniem względem ewentualnej wersji produkcyjnej jest brak innej niż podstawowa – oparta na wyjątkach javy – walidacji wprowadzonych danych – nie wykorzystano DTD ani XML Schema. Elementy umieszczone w poszczególnych plikach XML niezgodnie z założeniami algorytmów, po wczytaniu mogłyby dać nieprzewidziane efekty – nie ma to jednak wpływu na działanie narzędzia dla poprawnych danych.

Zastosowano uproszczony styl graficzny zaszyty na stałe w pliku css; w finalnej implementacji zapewne zaimplementowano by bardziej elastyczne rozwiązanie, pozwalające na zastosowanie stylu właściwego dla aplikacji docelowej.

W zależności od wybranej aplikacji z pakietu MS Office, narzędzie wyświetla odpowiedni formularz służący do wprowadzania plików z danymi oraz, w przypadku

narzędzia współpracującego z Excelem i Wordem, pliku do docelowego do uzupełnienia wczytanymi danymi. Servlet na podstawie danych z plików generuje kolejny formularz z osadzoną funkcją javascript – której wykonanie generuje bądź modyfikuje odpowiedni plik.

Dla celów prezentacji narzędzia wybrano trzy zestawy danych:

Dokumentem **.doc** jest szablon przykładowej umowy najmu pokoju w mieszkaniu – pokazujący teoretyczną funkcjonalność aplikacji wykorzystywanej przez agencję nieruchomości.

Jako dokument **.xls** autor wybrał trzymiesięczne rozliczenie kosztów czynszu, rachunków i kwoty odstępnego dla wynajmu mieszkania.

Przykładem projektu eksportowanego do MS Project jest uproszczony opis czynności wchodzących w skład procesu budowy domu mieszkalnego.

4.12. Wymagania:

Dla poprawnego działania narzędzia po stronie klienta wymagany jest odpowiedni komponent pakietu MS Office (narzędzie przetestowano na wersjach 2000 oraz 2003 pakietu) oraz przeglądarka Internet Explorer w wersji 6.0 bądź 7.0; z poprawnie skonfigurowanymi ustawieniami bezpieczeństwa – aplikacja wymaga przyznanych uprawnień do uruchamiania skryptów javascript oraz otwierania obiektów ActiveX.

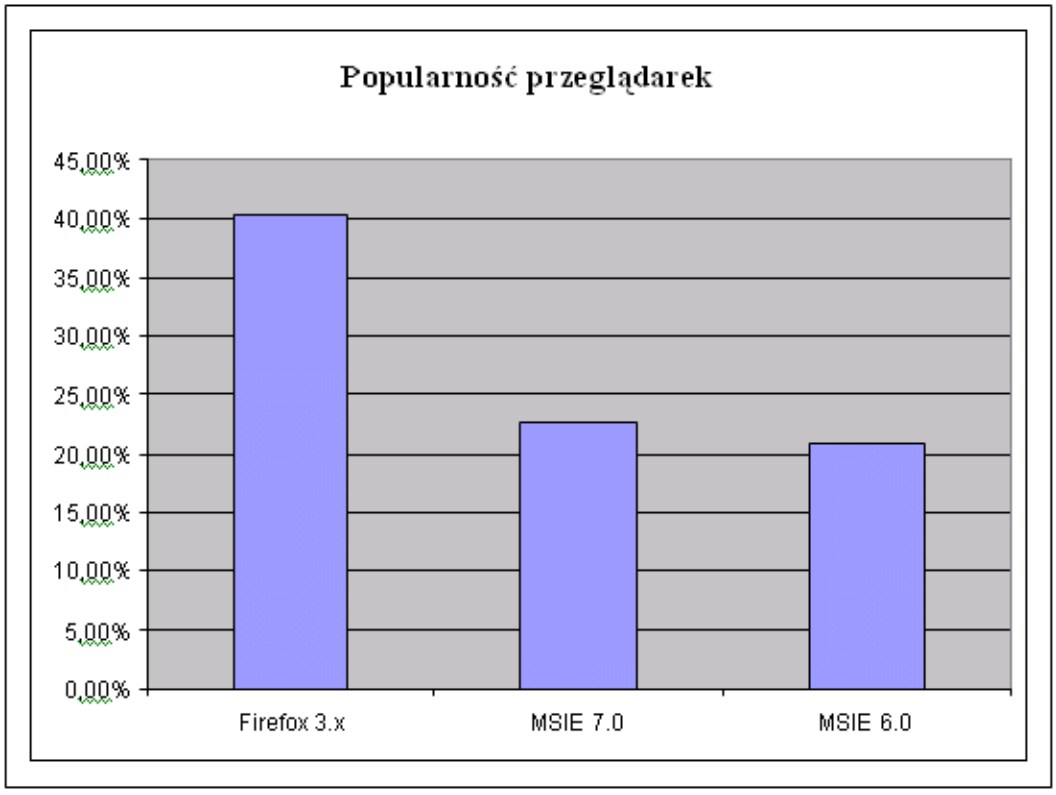
Wymagania sprzętowe są takie same jak dla samego pakietu Office – MSGenerator nie dodaje swojego narzutu wydajnościowego.

4.13 Przeglądarki:

Aktualnie **Internet Explorer 6.0** lub **7.0** są jedynymi przeglądarkami współpracującymi z narzędziem MSGenerator. Powodem tego ograniczenia są restrykcyjne założenia w innych przeglądarkach (Opera, Firefox od wersji 3.0) które

uniemożliwiają dostęp aplikacji do pełnej ścieżki pliku ładowanego na formularzu, tłumacząc to względami bezpieczeństwa – przez co uniemożliwiają wczytanie jego zawartości bez zapisania pliku na serwerze. W przypadku Firefoxa można to ograniczenie obejść, wymaga to jednak ręcznej edycji konfiguracji przeglądarki – co w środowisku produkcyjnym i potencjalnie dużej ilości użytkowników nie byłoby praktycznym rozwiązaniem.

W firmach korzystających z oprogramowania Microsoftu (a raczej tylko takie mogłyby być zainteresowane narzędziem współpracującym z MS Office), w zdecydowanej większości przypadków stosowaną do pracy przeglądarką jest Internet Explorer – zatem ograniczenie to nie stanowi dużego problemu.



Rysunek 8 ranking popularności przeglądarek [17.05.2009]

Na rysunku 8 pokazano ranking popularności przeglądarek sporządzony przez firmę Gemius[8], obejmujący przełom marca i kwietnia 2009 roku. Jak widać, Internet Explorer w wersjach 6.0 i 7.0 (a zatem w wersjach współpracujących z MSGeneratorem) ma w sumie prawie 45% pokrycia na rynku. Co więcej, jako że badanie obejmowało wszystkich użytkowników, należy się spodziewać, że w konserwatywnym środowisku korporacyjnym popularność przeglądarek Microsoftu jest jeszcze większa.

4.14 Struktura plików XML

4.13.1 Dokumenty będące źródłem dla Worda

Przykładowa struktura danych:

```
<?XMLversion ="1.0"  encoding="UTF-8"   standalone="no"?>
<document_variables>
        <table number="1" columns="2">
                <row>
                        <cell>
                                <value>Jan Nowak</value>
                        </cell>
                        <cell>
                                <value>AAA 123456</value>
                        </cell>
                        <cell>
                                <value>Właściciel mieszkania</value>
                        </cell>
                </row>
                <row>
                        <cell>
                                <value>Kazimierz Iksiński</value>
                        </cell>
```

[8] http://www.ranking.pl/ [17.05.2009]

```
                    <cell>
                        <value>BBB 654321</value>
                    </cell>
                    <cell>
                        <value>Wynajmujący</value>
                    </cell>
                </row>
            </table>
            <variable>
            <name>p_data_zawarcia</name>
            <value>25.01.2008</value>
                <synonym>Data zawarcia</synonym>
            </variable>
            <variable>
            <name>p_wynajmujacy</name>
            <value>Michałem Degentyszem</value>
            </variable>
            <variable>
            <name>p_wynajmujacy_miejscowosc</name>
            <value>Grzymiszewie</value>
                <synonym>Adres wynajmującego - miejscowość</synonym>
            </variable>
</document_variables>
```

Całość struktury zawiera się w tagu **<document_variables>**. Niżej w hierarchii mogą znajdować się dwa rodzaje tagów:

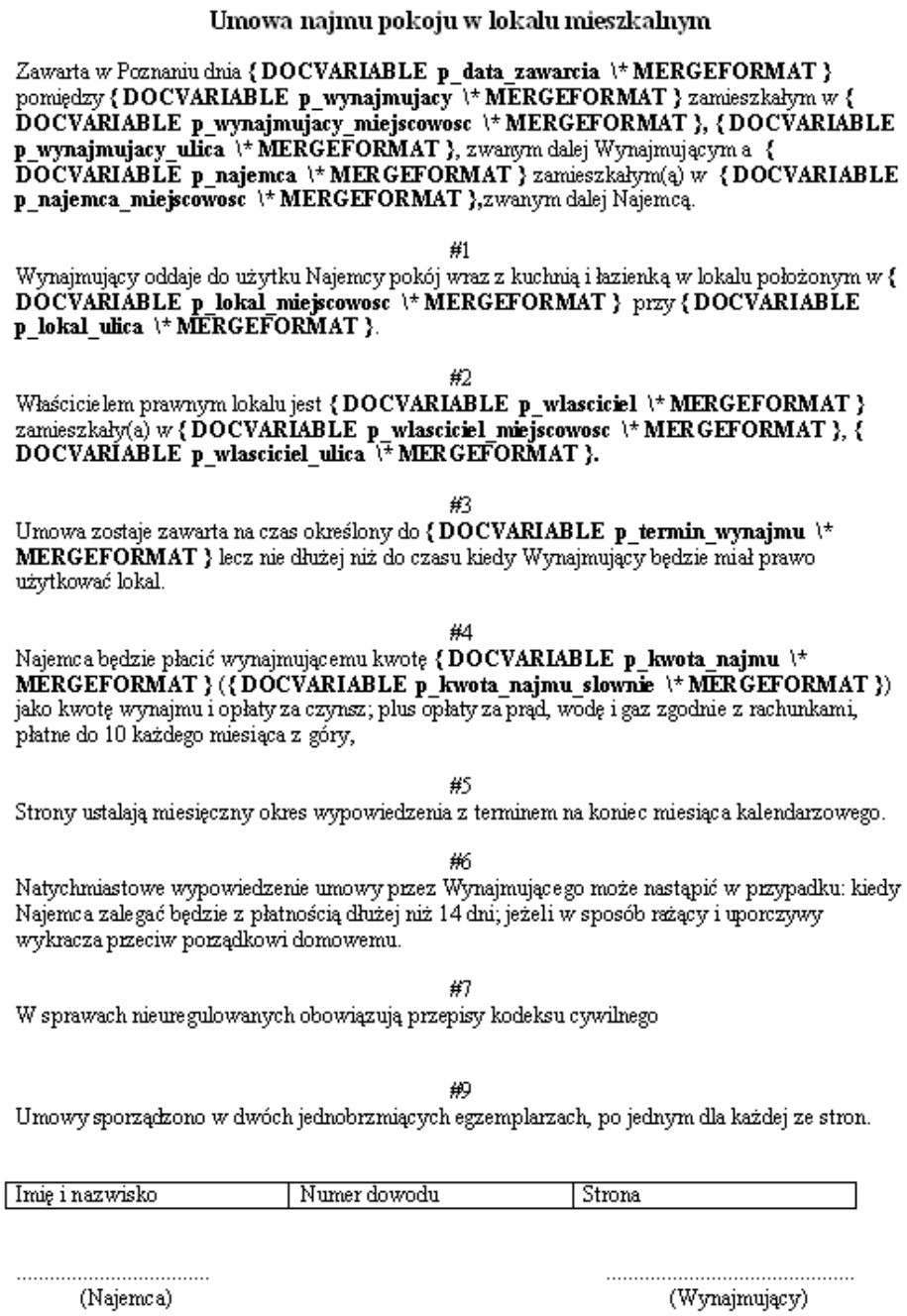
- **<table>** - struktura ta opisuje dane tabeli znajdującej się w docelowym dokumencie. Posiada następujące atrybuty:
    - **number** - pole obowiązkowe, jest to numer porządkowy tabeli w dokumencie Worda; biorąc pierwszą tabelę jako „1” oraz licząc od góry
    - **columns** - pole obowiązkowe, określa liczbę kolumn w tabeli

Struktura **<table>** się z tagów **<row>**, te z kolei z poszczególnych **<cell>** - których liczba w danym wierszu jest równa wartości atrybutu **columns**. **<cell>** składa się z tagu **<value>** , opisującego zawartość konkretnej komórki w tabeli.

- **<variable>** - struktura opisująca konkretne pole tekstowe w dokumencie. Może posiadać następujące struktury poniżej:
  - **<name>** - pole obowiązkowe; jest to nazwa zmiennej tekstowej w dokumencie
  - **<value>** - pole obowiązkowe; wartość do wpisania w pole tekstowe
  - **<synonym>** - pole opcjonalne; opis pola, który pojawi się na formularzu generującym dokument

4.14.1.1. Przykładowy formularz z wpisanymi docvariables

Na rysunku 9 pokazano przykładowy szablon Worda, z wyświetlonymi kodami poszczególnych zmiennych tekstowych.



Umowa najmu pokoju w lokalu mieszkalnym

Zawarta w Poznaniu dnia **{ DOCVARIABLE p_data_zawarcia * MERGEFORMAT }** pomiędzy **{ DOCVARIABLE p_wynajmujacy * MERGEFORMAT }** zamieszkałym w **{ DOCVARIABLE p_wynajmujacy_miejscowosc * MERGEFORMAT }, { DOCVARIABLE p_wynajmujacy_ulica * MERGEFORMAT }**, zwanym dalej Wynajmującym a **{ DOCVARIABLE p_najemca * MERGEFORMAT }** zamieszkałym(ą) w **{ DOCVARIABLE p_najemca_miejscowosc * MERGEFORMAT },**zwanym dalej Najemcą.

#1

Wynajmujący oddaje do użytku Najemcy pokój wraz z kuchnią i łazienką w lokalu położonym w **{ DOCVARIABLE p_lokal_miejscowosc * MERGEFORMAT }** przy **{ DOCVARIABLE p_lokal_ulica * MERGEFORMAT }**.

#2

Właścicielem prawnym lokalu jest **{ DOCVARIABLE p_wlasciciel * MERGEFORMAT }** zamieszkały(a) w **{ DOCVARIABLE p_wlasciciel_miejscowosc * MERGEFORMAT }, { DOCVARIABLE p_wlasciciel_ulica * MERGEFORMAT }.**

#3

Umowa zostaje zawarta na czas określony do **{ DOCVARIABLE p_termin_wynajmu * MERGEFORMAT }** lecz nie dłużej niż do czasu kiedy Wynajmujący będzie miał prawo użytkować lokal.

#4

Najemca będzie płacić wynajmującemu kwotę **{ DOCVARIABLE p_kwota_najmu * MERGEFORMAT }** (**{ DOCVARIABLE p_kwota_najmu_slownie * MERGEFORMAT }**) jako kwotę wynajmu i opłaty za czynsz; plus opłaty za prąd, wodę i gaz zgodnie z rachunkami, płatne do 10 każdego miesiąca z góry,

#5

Strony ustalają miesięczny okres wypowiedzenia z terminem na koniec miesiąca kalendarzowego.

#6

Natychmiastowe wypowiedzenie umowy przez Wynajmującego może nastąpić w przypadku: kiedy Najemca zalegać będzie z płatnością dłużej niż 14 dni; jeżeli w sposób rażący i uporczywy wykracza przeciw porządkowi domowemu.

#7

W sprawach nieuregulowanych obowiązują przepisy kodeksu cywilnego

#9

Umowy sporządzono w dwóch jednobrzmiących egzemplarzach, po jednym dla każdej ze stron.

Imię i nazwisko	Numer dowodu	Strona

....

(Najemca) (Wynajmujący)

Rysunek 9 szablon Worda z widocznymi zmiennymi tekstowymi

4.14.2. Dokumenty będące źródłem dla Excela – wersja uproszczona

Przykładowa struktura danych:

```
<?XMLversion ="1.0"  encoding="UTF-8"      standalone="no"?>
<msexcel_data>
        <cell col="1" row="2" font_color="6"
                                    interior_color = "3"
                                    content="1"/>
        <cell col="2" row="2" bold="1" content="2"/>
        <cell col="3" row="2" content="3" italic="1" underline="0"/>
        <cell col="1" row="3" content="1" underline="1"/>
        <cell col="2" row="3" content="2"/>
        <cell col="3" row="3" content="3"/>
        <cell col="1" row="4" content="1"/>
        <cell col="2" row="4" content="2"/>
        <cell col="3" row="4" content="3"/>
</msexcel_data>
```

Korzeniem struktury jest tag **<msexcel_data>**. Niżej w hierarchii znajdują się tagi **<cell>**, które mogą posiadać następujące atrybuty:

- **col** – oznacza numer kolumny (licząc od 1)
- **row** – oznacza numer wiersza (licząc od 1)
- **content** – treść komórki
- **italic** – wartość „1” oznacza pochylenie tekstu w komórce
- **bold** – wartość „1” oznacza pogrubienie tekstu w komórce
- **underline** – wartość „1” oznacza podkreślenie tekstu w komórce
- **interior_color** – wartość numeryczna reprezentująca kolor wypełnienia komórki
- **font_color** - wartość numeryczna reprezentująca kolor czcionki w komórce

4.14.3 Dokumenty będące źródłem dla Excela – wersja zaawansowana

Przykładowa struktura danych:

```
<?XMLversion ="1.0"  encoding="UTF-8"   standalone="no"?>
<msexcel_data>
        <cell col="1" row="2" content="1">
                <property property_path="Font.ColorIndex"
                        property_value="3"
                        with_quote="false"/>
                <property property_path="Font.Bold"
                        property_value="1"
                        with_quote="false"/>
        </cell>
        <cell col="2" row="2" content="2">
                <property property_path="Font.Italic"
                        property_value="1"
                        with_quote="false"/>
        </cell>
</msexcel_data>
```

Całość struktury zawiera się w tagu **<msexcel_data>**. Niżej w hierarchii znajdują się tagi **<cell>**, które mogą posiadać następujące atrybuty:

- **col** – oznacza numer kolumny (licząc od 1)
- **row** – oznacza numer wiersza (licząc od 1)
- **content** – treść komórki

Każda z komórek może posiadać dowolną ilość elementów **<property>**, opisujących formatowanie komórki. Tag ów posiada następujące atrybuty:

- **property_path** – ścieżka do danego typu elementu podlegającego formatowaniu – na przykład do rodzaju czcionki czy koloru tła.
- **property_value** – wartość opisująca formatowanie (np. „1” – dla pogrubienia czcionki)

- **with_quote** – przyjmuje wartość true (dla property_value będących wartościami numerycznymi) bądź false (dla pozostałych).

4.14.4 Dokumenty będące źródłem dla Projecta

Przykładowa struktura danych:

```
<?XMLversion ="1.0"  encoding="UTF-8"   standalone="no"?>
<msproject_data>
        <task name="task1">
                <task name="subtask1" >
                        <task name="subsubtask11"
                                start_date="2009-02-18"
                                end_date="2009-02-19">
                        </task>
                        <task name="subsubtask12"
                                start_date="2009-02-19"
                                end_date="2009-02-20"
                                milestone="true">
                        </task>
                        <task name="subsubtask13"
                                start_date="2009-02-20"
                                end_date="2009-02-22">
                        </task>
                </task>
                <task name="subtask2" >
                        <task name="subsubtask21"
                                start_date="2009-02-22"
                                end_date="2009-02-23">
                        </task>
                        <task name="subsubtask22"
                                start_date="2009-02-23"
                                end_date="2009-02-25">
                        </task>
```

```
                    <task name="subsubtask23"
                          start_date="2009-02-25"
                          end_date="2009-02-26">
                    </task>
               </task>
          </task>
</msproject_data>
```

Całość struktury zawiera się w tagu **<msproject_data>**. Niżej w hierarchii znajdują się struktury objęte tagami **<task>**; te z kolei posiadają następujące atrybuty:

- **name** – atrybut obowiązkowy; nazwa zadania
- **start_date** – atrybut obowiązkowy w tasku na najniższym poziomie zagłębienia; data rozpoczęcia zadania
- **end_date** - atrybut obowiązkowy w tasku na najniższym poziomie zagłębienia; data zakończenia zadania
- **milestone** – atrybut opcjonalny; wartość “true” oznacza, że dane zadanie jest „kamieniem milowym”.

W danym tasku mogą być zagnieżdżone kolejne – ograniczenie „głębokości” struktury jest takie samo jak to zaimplementowane w MS Project.

V. Problemy związane z opracowaniem uniwersalnego narzędzia

Jak każde narzędzie mające w założeniu być jak najbardziej elastycznym i wszechstronnym, także i MSGenerator wymusza konieczność rozwiązania pewnych problemów, które wyniknęły podczas jego tworzenia.

.

5.1 Obsługa plików

Zastosowany sposób traktowania plików (zarówno binarnych szablonów pakietu MS Office jak i plików z danymi w postaci XML) pozwala na uniknięcie uploadowania ich na serwer – w przypadku tych ostatnich pobierana jest jedynie ich treść jako serializowalny obiekt Document. Rozwiązanie to pozwala uniknąć niepotrzebnego narzutu sieciowego, koniecznego do wysłania i zapisania całego pliku na serwerze, podobnie jak i dodatkowej warstwy aplikacji odpowiadającej za obsługę pliku po stronie serwera – w takim wypadku plik musiałby zostać umieszczony w bazie danych lub dyskowym repozytorium.

Takie podejście niesie niestety ze sobą ryzyko naruszenia bezpieczeństwa aplikacji – wspomniana już kwestia wysyłania całej ścieżki pliku na serwer może doprowadzić do ryzyka ujawnienia niepożądanych danych – na przykład przechwycenia nazwy konta użytkownika, w sytuacji gdy plik znajdowałby się np. w katalogu **C:\Documents and Settings\mojekonto\My Documents\pjwstk**.

Obecnie jedynie Microsoft w przeglądarce Internet Explorer (i tylko w wersji niższej niż najnowsza 8.0) pozwala na dostęp serwera do pełnej ścieżki – można ponadto z całą pewnością stwierdzić, że taka funkcjonalność już nie zostanie przywrócona w kolejnych wersjach tej przeglądarki.

Problem ten można by obejść przesyłając plik na serwer – dzięki tej możliwości nawet wystąpienie wyżej opisanej sytuacji nie pozostawiłoby narzędzia MSGenerator bezużytecznym; wymagałoby jedynie zmian związanych z przetwarzaniem pliku; tak, aby umożliwić pobieranie struktury XML bezpośrednio z serwera.

5.2 Wymagania licencyjne

Wykorzystanie kontrolki ActiveX do manipulacji plikiem wymaga, aby na każdej maszynie z której będzie wywoływane narzędzie był zainstalowany pakiet MS Office. Nie wydaje się to być problemem zbyt poważnym – z założenia osoba chcąca wygenerować plik powinna mieć możliwość jego otworzenia, zatem powyższe wymaganie i tak byłoby spełnione.

Wiąże się to oczywiście z koniecznością posiadania odpowiedniej liczby licencji na pakiet Office – ale jest to koszt którego i tak nie dałoby się uniknąć posługując się narzędziami pakietu biurowego w bezpośredni sposób.

5.3. Dlaczego nie .xml?

MSGenerator w obecnej postaci nie współpracuje z plikami tworzonymi przez pakiet Open Office – pomimo teoretycznej zgodności narzędzi z tego pakietu z ich odpowiednikami z MS Office w wersji 2003, aplikacja Writer (odpowiednik Worda) nie pozwala na manipulację polami tekstowymi (*docvariables*) – co wyeliminowało najważniejszą funkcjonalność części związanej z obsługą dokumentów tekstowych, czyli automatyzację wypełniania szablonów danymi. Problem ten wpłynął na decyzję autora o dopasowaniu narzędzia tylko pod pakiet MS Office.

Próba generowania plików pakietu MS Office w formacie *Office Open XML* (dostępnym od wersji Office 2003) również nie powiodła się - składnia plików tworzonych przez Worda (a dokładniej jej cześć dotycząca pól tekstowych) jest niespójna i nielogiczna, co całkowicie uniemożliwiło implementację wypełniania pól tekstowych – element z „wypełnionym" polem był innym elementem niż „puste" pole w szablonie. Ponadto, wciąż powszechnie używane są pakiety Office w wersjach wcześniejszych niż 2003 – a dzięki użyciu binarnego formatu pliku, zachowana jest wsteczna zgodność.

Wspomniany wyżej problem „mutacji” kodu pliku obrazuje poniższy fragment kodu XML pliku w formacie MS Word 2003 – pole tekstowe w niewypełnionym szablonie wygląda następująco:

```
<w:p wsp:rsidR="00DA6430" wsp:rsidRDefault="000C59A9">
<w:r>
        <w:t>Imię: </w:t>
</w:r>
<w:r>
        <w:fldChar w:fldCharType="begin"/>
</w:r>
<w:r>
        <w:instrText>
                DOCVARIABLE  p_imie  \* MERGEFORMAT
        </w:instrText>
</w:r>
<w:r>
        <w:fldChar w:fldCharType="end"/>
</w:r>
</w:p>
```

Ten sam fragment pliku z już wypełnioną zmienną **p_imie** znacznie różni się od pierwowzoru:

```
<w:p wsp:rsidR="00DA6430" wsp:rsidRDefault="000C59A9">
<w:r>
        <w:t>Imię: </w:t>
</w:r>
<w:fldSimple w:instr=" DOCVARIABLE  p_imie  \* MERGEFORMAT ">
        <w:r wsp:rsidR="00BD26E9">
                <w:t>
```

Michał Degentysz

</w:t>

</w:r>

</w:fldSimple>

</w:p>

Jak widać, w strukturze zaszły duże zmiany; zniknęły niektóre tagi (na przykład **<w:instrText>**), inne zmieniły się (**<w:fldChar>** na **<w:fldSimple>**) – co więcej, zmiany te nie wydają się być w żaden sposób możliwe do sztucznego zreplikowania – wypełnienie zmiennej wartością powoduje utworzenie nowego paragrafu o konkretnym ID, definiowanym przez nieznany, a zatem niedostępny dla zewnętrznego narzędzia algorytm.

Wszystko to powoduje, że budowanie przez algorytm kodu XML mającego odwzorowywać plik z wypełnionymi zmiennymi tekstowymi jest na tyle trudne, aby zwrócić się w stronę prostszego i sprawdzonego rozwiązania.

Wymienione wyżej problemy sprawiły, że najprostszym sposobem modyfikacji pliku Worda jest działanie na obiekcie ActiveX, obsługiwanym poprzez metody javascript.

Bezpośrednią manipulację na strukturze XML można by mimo wszystko zastosować w przypadku plików Excela. Fragment tego pliku – konkretnie kod jednej karty – wygląda następująco:

```
<Worksheet ss:Name="Sheet1">
 <Table ss:ExpandedColumnCount="2" ss:ExpandedRowCount="2" x:FullColumns="1"
x:FullRows="1">
<Row>
   <Cell>
        <Data ss:Type="Number">11</Data>
  </Cell>
   <Cell>
        <Data ss:Type="Number">22</Data>
  </Cell>
  </Row>
```

```
 <Row>
        <Cell ss:Index="2">
                <Data ss:Type="Number">33</Data>
        </Cell>
 </Row>
</Table>
 <WorksheetOptions xmlns="urn:schemas-microsoft-com:office:excel">
  <Selected/>
  <Panes>
        <Pane>
                <Number>3</Number>
                <ActiveRow>2</ActiveRow>
                <ActiveCol>1</ActiveCol>
        </Pane>
  </Panes>
  <ProtectObjects>False</ProtectObjects>
  <ProtectScenarios>False</ProtectScenarios>
 </WorksheetOptions>
</Worksheet>
```

Z punktu widzenia MSGeneratora najistotniejszą cześć stanowią znaczniki **<Cell>**. Jak widać na powyższym przykładzie, znaczniki te wchodzą w skład znacznika **<Row>**; bez problemu można by w pustym dokumencie wstawić taki zestaw znaczników i tym samym wypełnić go danymi.

Odnośnie części generującej arkusz Excela należy zwrócić uwagę na fakt, że poszczególne komórki ani wiersze nie są poindeksowane w sposób bezwzględny – to kolejność w jakiej znajdują się w dokumencie determinuje „adres" komórki w arkuszu. W przypadku kiedy komórka znajduje się za wierszem bądź komórką pustą, używany jest atrybut „**index**" – wskazujący na przesunięcie danej komórki względem pierwszego wiersza lub komórki.

Na przedstawionym poniżej fragmencie arkusza zapisanego w aplikacji **Calc** pakietu OpenOffice.org jasno widać, że poszczególne komórki są prezentowane w ten

sam sposób – zatem narzędzie które wypełniałoby danymi arkusz Excela, mogłoby również korzystać z tego samego algorytmu aby generować pliki zgodne z pakietem biurowym firmy Sun Microsystems. Ta możliwość została szerzej omówiona w rozdziale 6.1 niniejszej pracy.

```
<ss:Worksheet ss:Name="Arkusz1">
        <Table ss:StyleID="ta1">
                <Column ss:Span="1" ss:Width="64.2614"/>
                <Row ss:Height="12.8409">
                        <Cell>
                                <Data ss:Type="Number">11</Data>
                        </Cell>
                        <Cell>
                                <Data ss:Type="Number">22</Data>
                        </Cell>
                </Row>
                <Row ss:Height="12.8409">
                        <Cell ss:Index="2">
                                <Data ss:Type="Number">33</Data>
                        </Cell>
                </Row>
        </Table>
        <x:WorksheetOptions/>
</ss:Worksheet>
```

VI. Dalsze rozszerzenie funkcjonalności

Funkcjonalność zaimplementowana w narzędziu MSGenerator w żadnym wypadku nie wyczerpuje możliwości obsługi narzędzi MS Office udostępnionych poprzez kontrolki ActiveX.

Dla narzędzia eksportującego do Worda logicznym rozszerzeniem funkcjonalności wydaje się być wprowadzenie dalszych elementów, w przypadku których sensowna byłaby możliwość do ich automatycznego dodania w wybranym miejscu w dokumencie – na przykład całych paragrafów z treścią czy też odnośników HTML. Przydatne również byłoby udostępnienie możliwości zdalnego formatowania wyglądu wybranego paragrafu w tekście – na przykład wykorzystane w aplikacji WebInwestor przekreślanie wybranych elementów (automatyzacja często występujących w oficjalnych dokumentach miejsc „niepotrzebne skreślić").

Istnieje również szerokie pole dla rozszerzenia potencjalnych możliwości narzędzia w kwestii formatowania wyglądu tabel w dokumencie – z pewnością można by znaleźć zastosowanie dla modyfikowania szerokości wybranych kolumn (obecnie szerokość jest ustawiana automatyczna) bądź formatu tekstu w poszczególnych wierszach/kolumnach.

Osobną kwestią techniczną jest wypełnianie danymi pól tekstowych znajdujących się w stopce bądź nagłówku strony – mechanizmy zastosowane w Wordzie uniemożliwiają automatyczne wypełnienie ich wartościami w taki sam sposób jak tych umieszczonych w ciele dokumentu (precyzując, pozwalają na ich wypełnienie, lecz dane te bez ręcznego odświeżenia wartości pola pozostają niewidoczne).

Rozwiązaniem byłoby dodawanie do dokumentu makra, które, wywołane po wypełnieniu dokumentu odświeżyłoby zawartość owych pól – najwygodniejszą opcją byłoby oczywiście jego automatyczne dołączenie do dokumentu i wykonanie po jego utworzeniu. W tym przypadku na przeszkodzie stałaby kwestia bezpieczeństwa aplikacji – trudno wyobrazić sobie organizację, w której polityka bezpieczeństwa dopuszczałaby możliwość automatycznego uruchamiania makr w aplikacjach pakietu biurowego.

Należałoby również rozważyć kwestię tworzenia nowego dokumentu – to jest bez konieczności posiłkowania się szablonem źródłowym. Kontrolka Word daje możliwość stworzenia takiego dokumentu, który można by następnie wypełnić danymi.

Z pewnością przydatnym rozszerzeniem algorytmu obsługującego Excela byłoby automatyczne generowanie wykresów – taką funkcjonalność kontrolka arkusza kalkulacyjnego również udostępnia, i jest to jedno z potężniejszych potencjalnych zastosowań – można wyobrazić sobie pomocniczą aplikację służącą do prezentacji danych w takiej właśnie postaci, wykorzystującą silnik aplikacji MS Excel; bez potrzeby stosowania innych, nierzadko drogich narzędzi udostępniających taką funkcjonalność.

Część dotycząca formatowania poszczególnych komórek arkusza jest w zasadzie kompletna - obejmuje pełną funkcjonalność udostępnioną przez MS Excel; co więcej, w przyszłości może obsłużyć elementy formatowania, których dzisiejsze wersje Excela jeszcze nie oferują; dzięki bezpośredniemu interpretowaniu poleceń javascript udostępnionych przez kontrolkę – a to wszystko bez jakichkolwiek modyfikacji w kodzie algorytmu.

Wykorzystane w MSGeneratorze elementy projektu aplikacji MS Project w dość niewielkim stopniu wykorzystują możliwości udostępnione przez to narzędzie; dalszą drogą w rozwoju algorytmu eksportującego byłoby uwzględnienie takich elementów jak zasoby, stopień wykonania poszczególnych zadań czy też ich koszt. Dodatkowo, interesująca byłaby też opcja modyfikowania i uaktualniania już istniejących dokumentów projektu – czynność praktycznie niezbędna podczas prowadzenia projektu z wykorzystaniem narzędzia Microsoftu.

Od szerszej niż tylko funkcjonalna strony, warte rozważenia byłoby również rozszerzenie współpracy narzędzia na inne pakiety biurowe niż MS Office. I OpenOffice.org firmy Sun, i iWork Apple'a mają swoją bazę użytkowników; niezależnie natomiast od systemu operacyjnego z którego potencjalni klienci korzystają, kwestia eksportu danych z aplikacji biznesowej do pakietu biurowego wciąż pozostaje aktualna.

Naturalne też wydaje się uwzględnienie w rozwoju MSGeneratora innych przeglądarek – dodanie obsługi Mozilli Firefox, Opery czy Apple Safari (wymieniając tylko najpopularniejsze) z pewnością pomogłoby w stworzeniu jeszcze bardziej uniwersalnego narzędzia.

Niestety na przeszkodzie ku temu ponownie staje kwestia obsługi kontrolek ActiveX przez przeglądarki inne niż Internet Explorer (odpowiedni plugin istnieje do Firefoxa, niestety tylko do wersji 2.x) – jednakże dla aplikacji które miałby współpracować głównie z dokumentami Excela czy Projecta, można by wykorzystać tworzenie dokumentów w formacie Office Open XML i pominąć całkowicie kontrolki, zdając się na bezpośrednie generowanie kodu pliku. Ponadto nic nie stałoby na przeszkodzie, aby również dokumenty Worda, do których narzędzie miałoby eksportować wyłącznie tabele korzystały z powyższej metody.

Elementem, który raczej na pewno znalazłby zastosowanie w narzędziu eksportującym byłby także generator XML; przetwarzający dane w „surowej" postaci (na przykład pobrane z bazy danych) na pliki w postaci akceptowanej przez MS Generator. Z racji tego, że najbardziej złożoną, drzewiastą strukturę posiadają pliki XML będące źródłem danych dla narzędzia eksportującego do MS Projecta, to właśnie tam takie narzędzie byłoby najbardziej potrzebne. Jakkolwiek byłby to element nie związany z właściwą funkcjonalnością narzędzia, to stanowiłby przydatne uzupełnienie całości, jeszcze bardziej upraszczając implementację MSGeneratora w istniejącej aplikacji biznesowej.

Poruszając kwestię integracji narzędzia eksportującego z aplikacją, w której funkcjonalność miałby rozszerzać MSGenerator, warte uwagi byłoby umożliwienie parametryzowania wyglądu poszczególnych formularzy – na przykład poprzez sterowanie generowaniem podglądu danych na formularzu obsługującym właściwą funkcję generującą – czy to w postaci surowej (pomocnej dla developerów) czy też sformatowanej (dla klientów).

W przykładowej implementacji składnia niektórych poleceń udostępnianych przez kontrolki poszczególnych aplikacji uzależniona jest od językowej wersji zainstalowanego na maszynie klienckiej pakietu Office – aby poprawić uniwersalność narzędzia należałoby więc dodać mechanizm rozpoznający właściwą wersję i dobierający na jej postawie odpowiednią składnię – brak tego składnika mógłby doprowadzić do trudnych w wykryciu błędów w działaniu narzędzia.

Wreszcie, dużym ułatwieniem dla developerów podczas pracy nad niezawodnością działania narzędzia byłoby zaimplementowanie wzorców dokumentów

XML w postaci DTD bądź XMLSchema – uprościłoby to w znacznym stopniu walidację wejściowych plików w systemie.

Jak wykazano powyżej, wciąż istnieje szerokie pole do popisu w kwestii poszerzenia funkcjonalności MSGeneratora; temat eksportu danych do narzędzi MS Office jest na tyle szeroki że znacznie wykraczałby poza zakres projektu, mającego w założeniu być tylko studium wykonalność tej koncepcji. Narzędzie poszerzone o większość lub nawet wszystkie z wyżej wymienionych funkcjonalności mogłoby być może zaistnieć jako potencjalny produkt z szansą na sprzedaż.

6.1 OpenOffice.org

Z punktu widzenia klienta, który chciałby swoje rozwiązanie biznesowe oprzeć wyłącznie na darmowym oprogramowaniu, najistotniejszym rozwinięciem funkcjonalności MSGeneratora byłoby z pewnością dostosowanie go do współpracy z którymś z darmowych pakietów biurowych - wśród których najpopularniejszym jest obecnie OpenOffice.org.

Biorąc po uwagę pakiety biurowe będące konkurencją dla MS Office, aplikacja ta wyjątkowo nadaje się do współpracy z MSGeneratorem. Podobnie jak narzędzie Microsoftu, OpenOffice.org posiada zestaw swoich kontrolek ActiveX – więc rozwiązanie obejmujące ten pakiet mogłoby korzystać z tego samego algorytmu co przykładowa implementacja, oczywiście z uwzględnieniem różnic w składni poleceń interpretowanych przez poszczególne kontrolki.

Kolejną cechą wspólną dla obu omawianych pakietów biurowych są dwa sposoby, za pomocą których można zdalnie tworzyć dokumenty – to jest bądź poprzez wykorzystanie kontrolek ActiveX do modyfikacji binarnych plików, bądź też przez wykorzystanie kompatybilności obu pakietów z plikami w formacie XML – w postaci kodu dokumentu generowanego przez algorytm w javie.

Ta pierwsza możliwość jest niestety ograniczona jedynie do systemów z rodziny Windows – z powodu braku odpowiedniej implementacji ActiveX na platformach Linux i

Mac Os X[9] nie byłoby możliwości skorzystania z funkcjonalności udostępnionej przez kontrolki na maszynach klienckich z zainstalowanymi tymi właśnie systemami – to ograniczenie dotknęłoby oczywiście tylko użytkowników OpenOffice.org, i tylko w sytuacji, gdy nie byłoby możliwości skorzystania z opcji generowania plików XML.

Pomimo wszystkich opisanych wad, OpenOffice.org z pewnością jest pakietem biurowym nadającym się do współpracy z narzędziem eksportującym dane. Wszędzie tam, gdzie jest wykorzystywany, brak współpracy ze zmiennymi tekstowymi najwyraźniej nie jest żadnym ograniczeniem, a pod wszystkimi innymi względami spełnia swoje funkcje w takim samym stopniu jak MS Office. Oczywistą zaletą takiego połączenia jest możliwość stworzenia gotowego rozwiązania, nie wymagającego żadnych płatnych licencji.

[9] http://wiki.services.openoffice.org/ [16-06-2009]

VII. Podsumowanie

Niniejsza praca daje wgląd w tematykę automatyzacji korzystania z pakietów biurowych, na podstawie prototypu działającego narzędzia. Zostały w niej poruszone okoliczności oraz potrzeba powstania takiej aplikacji, związane z mechanizmem eksportu danych korzyści oraz problemy wynikłe podczas jej tworzenia.

Coraz powszechniejsze zastosowanie systemów informatycznych do wspomagania obsługi procesów w firmach nieuchronnie prowadzi do konieczności integracji różnych aplikacji. Kwestią czasu jest pojawienie się innych narzędzi typu MSGenerator – i niewykluczone, że korzystać one będą z podobnych rozwiązań i algorytmów.

Funkcjonalność zaimplementowana w narzędziu MSGenerator to jedynie najistotniejsze z funkcji poszczególnych aplikacji pakietu udostępnionych poprzez ich interfejs COM. Możliwości MSGeneratora z pewnością można by rozszerzyć do dużo szerszego spektrum – ale nie to było celem projektu, a pokazanie, że można stworzyć uniwersalne narzędzie służące do integracji na linii aplikacja biznesowa – pakiet biurowy.

Spis prac cytowanych

[1] http://portalwiedzy.onet.pl

[2] http://www.w3schools.com/xml/default.asp

[3] http://www.microsoft.com

[4] http://java.sun.com/products/servlet/

[5] http://java.sun.com/products/jsp/

[6] http://www.dom4j.org/

[7] http://www.w3schools.com/XPath/default.asp

[8] http://www.ranking.pl/

[9] http://wiki.services.openoffice.org/

Dodatki

Dodatek A: Spis rysunków

Dodatek B: Spis tabel