STUDIA INFORMATICA 2012
Volume 33 Number 2B (106)

Piotr PAŁKA
Politechnika Warszawska, Instytut Automatyki i Informatyki Stosowanej

WYKORZYSTANIE NARZĘDZI XML DO TWORZENIA MATEMATYCZNYCH MODELI OPTYMALIZACYJNYCH[1]

Streszczenie. W pracy przedstawiona została metodyka wykorzystująca zestaw narzędzi XML do generowania matematycznych modeli optymalizacyjnych. Zaprezentowane zostały podstawowe założenia metodyki, zakładane wyniki oraz przykładowy diagram klas docelowej aplikacji. Praca powołuje się także na istniejącą, przykładową, implementację powstałą na bazie przedstawionej metodyki. Aplikacja ta posłuży do ewaluacji opracowanej metodyki.

Słowa kluczowe: narzędzia XML, modele optymalizacyjne

XML TOOLS APPLICATION TO CREATE MATHEMATICAL OPTIMIZATION MODELS

Summary. In the paper the methodology that uses a set of XML tools to generate mathematical optimization models is presented. The basic assumptions of the methodology, expected results and the class diagram of the sample application are shown. Work relies on existing, exemplary implementation built upon on the presented methodology. The application is used to evaluate the developed methodology.

Keywords: XML tools, optimization models

1. Wprowadzenie

Z punktu widzenia projektanta mechanizmów rynkowych, ważna jest możliwość wygodnego formułowania reguł, wedle których działa dany mechanizm rynkowy. Reguły te wygodnie jest formułować, używając modeli matematycznych. Model handlu wielotowarowego M^3

[1] Praca naukowa finansowana ze środków na naukę w latach 2009 – 2011, jako projekt badawczy nr N N516 375736

[9] dostarcza wygodnego sposobu pozwalającego na zapis danych rynkowych dotyczących różnych segmentów rynku. Istnieje wiele prac dotyczących modelu M^3, których autorzy formułują modele matematyczne handlu dostosowane do różnorodnych segmentów rynku (rynki energii elektrycznej, telekomunikacyjne, alokacja zasobów kolejowych i lotniskowych, rynek handlu pozwoleniami na emisję gazów cieplarnianych). Modele te formułowane są na bazie modelu handlu wielotowarowego z ograniczeniami, który stanowi podstawę standardu M^3.

1.1. Projektowanie mechanizmów rynkowych

Mechanizm rynkowy, według definicji podanej w pracy [7], interpretuje pewne sygnały (oferty) odbierane od uczestników rynku (sprzedających i nabywców), a następnie, korzystając z tzw. reguły alokacji, dokonuje przydziału towarów i korzystając z tzw. reguły opłat, przeprowadza wycenę towarów. Aby analizować właściwości takiego mechanizmu, musimy ustalić, w jaki sposób będzie określony podział ofert na przyjęte i odrzucone (czyli określić regułę alokacji) oraz w jaki sposób będą wyznaczane ceny za poszczególne towary (czyli określić regułę wyceny).

Problem projektowania mechanizmów rynkowych jest rozpatrywany w licznych pracach [4, 10, 13]. Wynikiem projektowanego mechanizmu rynkowego jest zestaw reguł, których celem jest spełnienie pewnych właściwości [10]. Właściwości te mają m.in. na celu: równe i sprawiedliwe traktowanie uczestników rynku, zapobieganie działaniom spekulacyjnym, zapewnienie zbilansowania budżetu. Aby analizować różne mechanizmy rynkowe (dla różnych segmentów rynku) i oceniać różne ich warianty (czyli różne reguły alokacji i wyceny), potrzebny jest zestaw narzędzi służących do:

- zapisu danych rynkowych dla różnych segmentów w spójny sposób. Narzędziami służącymi do tego są model M^3 i notacja M3XML [9];
- rozwiązywania różnych mechanizmów rynkowych zapisanych w notacji M3XML. Takie narzędzie (SolveM3) powstało i zostało opisane szerzej w pracy [6];
- wygodnego tworzenia nowych reguł, które będą spójne z zaproponowanymi powyżej narzędziami. Celem tej pracy jest opisanie sposobu otrzymania takiego narzędzia.

1.2. Motywacja

W pracy [6] zostało opisane narzędzie SolveM3, które pobiera dane rynkowe sformułowane za pomocą notacji M3XML (oparty na XML dialekt do zapisu danych rynkowych), dokonuje alokacji ofert i wyceny towarów (przez rozwiązanie odpowiedniego zadanie programowania matematycznego) i zwraca wyniki zapisane w notacji M3XML. Model matematyczny dostarczany do narzędzia jest sformułowany w postaci transformaty zapisanej

w języku XSLT lub XQuery. Celem transformaty jest przekształcenie danych rynkowych zapisanych w M3XML do postaci modelu odpowiedniego mechanizmu rynkowego. Najczęściej jest to model optymalizacyjny, zapisany w postaci zadania programowania liniowego (LP) bądź też całkowitoliczbowego, mieszanego (MIP). Zadania te można zapisać, używając np. języka GMPL (GNU MathProg Modelling Language) [2] czy też bardziej ogólnego języka AMPL (A Modeling Language for Mathematical Programming). Następnie zapisane zadania programowania matematycznego są rozwiązywane przez specjalne aplikacje (solvery).

Wadą takiego rozwiązania jest trudność w tworzeniu nowych modeli handlu, a nawet wprowadzaniu drobnych modyfikacji modelu. Jest to spowodowane, po pierwsze, uciążliwością języków XSLT i XQuery (ich uciążliwość polega na złożonej składni oraz trudności w analizie takiego kodu, w porównaniu z klasycznymi językami programowania), a po drugie, znaczną ilością kodu, jaka powinna być stworzona nawet dla dość prostych reguł rynkowych. Celem tej pracy jest stworzenie metodyki, która określa budowę aplikacji służących do wygodnego tworzenia matematycznych modeli optymalizacyjnych, wykorzystujących dane sformułowane za pomocą modelu M^3 do zamodelowania odpowiedniego segmentu rynkowego. Celem docelowej aplikacji jest ułatwienie generowania transformat XSLT, odpowiadających konstruowanym regułom mechanizmu rynkowego. Aplikacja ta powinna zapewniać wygodny interfejs graficzny do modelowania różnych reguł alokacji i wyceny.

Istnieje wiele komercyjnych narzędzi służących do mapowania struktur XML (np. Altova MapForce), jednak nie umożliwiają one generowanie transformat XSLT na podstawie skonstruowanego modelu matematycznego mechanizmu rynkowego. Dlatego też nie zajmujemy się zastosowaniem tych narzędzi, skupiając się na projektowaniu specjalizowanej aplikacji.

2. Używane narzędzia

W punkcie tym przedstawione zostaną poszczególne narzędzia służące do obróbki modeli XML oraz standard M^3 służący do zapisu danych dla handlu wielotowarowego.

Mutli-commodity Market Model – M^3 [9] jest formalnym modelem służącym do zapisu danych dla różnorakich segmentów handlu wielotowarowego. Został stworzony jako podstawa systemu wymiany danych opartego na języku XML. Umożliwia on standaryzację komunikacji pomiędzy uczestnikami tego samego segmentu rynkowego oraz łatwy przepływ informacji między różnymi segmentami rynków. Wymiana handlowa, prowadzona w danym segmencie rynkowym, może być dokonywana na bazie agentowego paradygmatu programowania. Dzięki temu zapewnione są autonomiczność poszczególnych podmiotów rynkowych oraz rozproszona struktura informacji. Model M^3 pozwala realizować wiele różnych sposobów wymiany towarów – poza prostym przypadkiem rynku aukcyjnego oraz giełdy scentralizowa-

nej, także rynki rozproszone, w tym oparte na negocjacjach rynki kontraktów bi- i multilateralnych. W ramach standardu M^3 został opracowany język opisu danych rynkowych M3XML, będący dialektem XML.

*SolveM*3 jest aplikacją służącą do rozwiązywania matematycznych modeli optymalizacyjnych, do których dane sformułowane zostały za pomocą notacji M3XML. Aplikacja ta powstała na podstawie analizy przeprowadzonej w pracy [6]. Jako wynik zwraca dane uzupełnione o wyniki reguł alokacji ofert i wyceny towarów, także sformułowane w notacji M3XML. Wspomniane modele optymalizacyjne są modelami matematycznymi odpowiednich reguł mechanizmów rynkowych. Reguły te zapisane są za pomocą notacji XSLT, która transformuje dane rynkowe do danych odczytywalnych przez solver programowania liniowego (np. AMPL, GMPL, itp.). SolveM3 jest modułem oprogramowania, który może być wykorzystany w wielu różnych aplikacjach.

XPath [14] (ang. *XML Path Language*) służy do adresowania elementów dokumentu XML. Pierwotnym celem stworzenia tego narzędzia było dostarczenie wspólnego interfejsu dostępu do węzłów drzewa XML dla języków XSLT oraz XPointer. Poza główną funkcjonalnością adresowania, XPath pozwala również wykonywać proste operacje na danych liczbowych, ciągach znaków oraz używać podstawowych wyrażeń logicznych. Ze względu na swoją prostotę, a jednocześnie dużą funkcjonalność, jest to szeroko stosowane narzędzie.

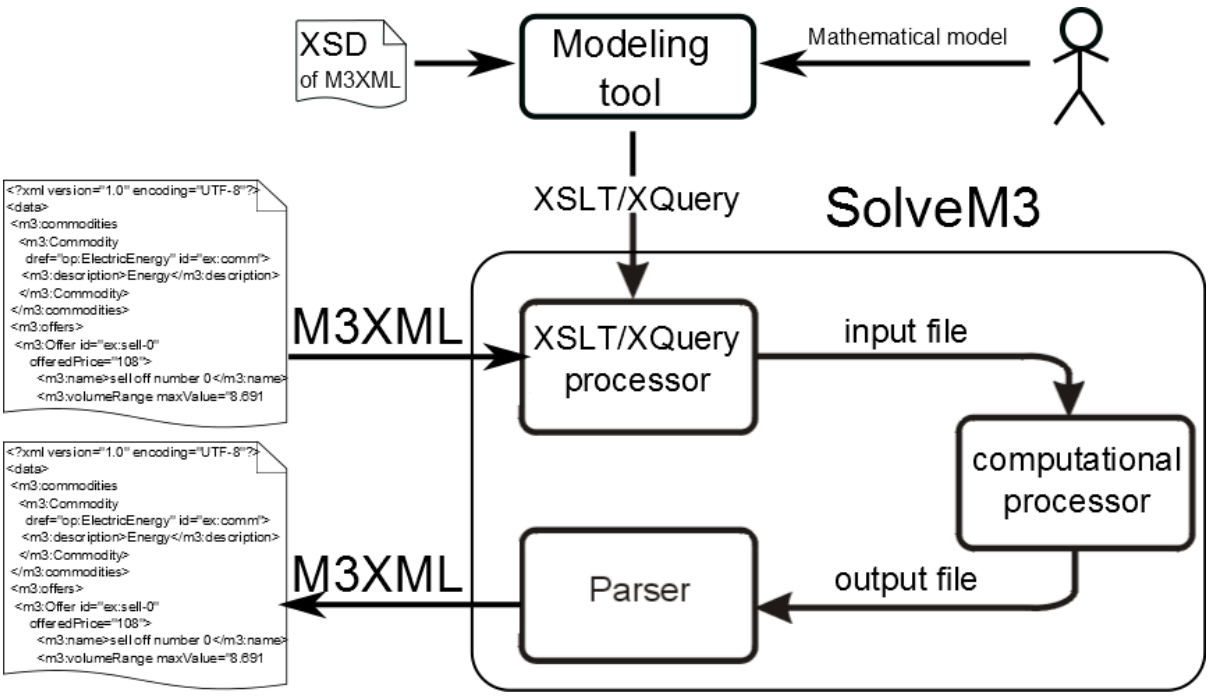
XSLT [17] jest językiem transformacji dokumentów XML. Wpływ na jego powstanie miały języki funkcyjne oparte na dopasowywaniu wzorców. Za pomocą transformacji XSLT możliwe jest przekształcenie pliku zapisanego w notacji XML do dowolnie innego formatu tekstowego (także innego pliku XML). Jest on używany w aplikacji SolveM3, jako język transformujący dane M3XML do modelu matematycznego. Język XSLT, pomimo ogromnej funkcjonalności, jest jednak skomplikowany oraz nieintuicyjny.

XQuery [15] jest językiem zapytań oraz programowania funkcyjnego. Za jego pomocą możliwe jest przeszukiwanie dokumentów XML w celu odnalezienia pewnych informacji. XQuery to narzędzie, za pomocą którego możliwe jest tworzenie czytelnych i zwięzłych zapytań, pobierających dane z dokumentu wejściowego XML i zwracających wyniki w postaci dowolnie sformułowanej notacji. Dzięki zastosowaniu podzapytań można uzyskać podobny efekt jak w przypadku transformat XSLT. Prace nad XQuery były inspirowane językami SQL (ang. *Structured Query Language*), XQL (ang. XML Query Language) oraz XPath.

XML Schema [16] stanowi opis schematu danego pliku XML. Dzięki niemu istnieje możliwość definiowania zawartości projektowanej notacji XML, a także ujednolicenia zapisu dla specyficznych zastosowań.

3. Proponowana metodyka

W tej części przedstawiony zostanie sposób rozwiązania problemu zdefiniowanego wcześniej, to znaczy trudności w wygodnym uzyskiwaniu transformat XSLT bądź XQuery oraz ich ewentualnych modyfikacji. Proponuje się, aby problem ten rozwiązać przy wykorzystaniu narzędzi służących do obróbki plików XML. Schemat przepływu danych pomiędzy narzędziami został przedstawiony na rys. 1. Użytkownik posiadający wiedzę o regułach rynkowych, które chce zaimplementować, będzie wprowadzał je za pomocą intuicyjnego interfejsu do docelowej aplikacji. Schemat XML ułatwia wyszukiwanie poszczególnych elementów zbioru M3XML. Za jego pomocą można stworzyć wygodny widok (np. w postaci drzewa) docelowego dokumentu M3XML. Następnie można nawigować po tym drzewie, a także w wygodny sposób wybierać poszczególne elementy czy atrybuty. Zakładamy, że narzędzie umożliwiać będzie eksport modelu do postaci XSLT bądź XQuery.



Rys. 1. Schemat przepływu danych podczas rozwiązywania reguł mechanizmu rynkowego. Na górze rysunku przedstawiono projektowane rozwiązanie
Fig. 1. Schema of the data flow during mechanisms rules solving. At the top the proposed solution is depicted

3.1. Wymagania

Proponowana aplikacja powinna dostarczać modeli matematycznych dla narzędzia SolveM3, sformułowanych w języku XSLT bądź XQuery. Powinna oferować wygodny interfejs użytkownika, pozwalający na definiowanie abstrakcyjnego modelu matematycznego

w postaci definicji zbiorów, parametrów, zmiennych, ograniczeń oraz funkcji celu [3]. Powyższe elementy tworzą matematyczny model optymalizacyjny. Poszczególne elementy powinny być opisane za pomocą dwoistej notacji: po pierwsze, powinny mieć interpretację matematyczną, po drugie, powinny odnosić się do notacji M3XML i wskazywać dane, które sprecyzują zawartość docelowego modelu. Odniesienia te powinny być sformułowane przez adresowania XPath, elementy transformat XSLT oraz zapytania XQuery. Dane dotyczące struktury dokumentu M3XML będą pobierane na podstawie schematu XML. Zakładamy także, że poszczególne elementy zapisu XML mogą zostać ukryte przed użytkownikiem i zastąpione wygodnymi elementami graficznymi. Takie podejście zostało zastosowane w innych narzędziach modelowania matematycznego (np. w AIMMS [1]). Wymagania te mogą zostać spełnione dzięki odpowiedniej definicji klas, zawierających odpowiednie elementy poszczególnych narzędzi XML.

3.2. Model konceptualny

Zakładamy, że istnieje klasa abstrakcyjna zawierająca definicję ogólnego elementu modelu matematycznego. Istnieje także odpowiednie odwzorowanie danych ze zbioru M3XML, sformułowane w notacji XPath, XSLT lub XQuery. Zakładamy także, że każdy z elementów modelu będzie generował pewien fragment kodu XSLT/XQuery, a kolekcja tych elementów, powiązana dodatkowymi zależnościami (odpowiednich reguł wynikających z konstrukcji modelu matematycznego), stworzy, po pierwsze, właściwy model matematyczny, a po drugie, kompletny kod XSLT/XQuery. Kod ten, zastosowany do odpowiedniego modelu rynku, sformułowanego za pomocą modelu M^3 i zapisanego za pomocą notacji M3XML, wygeneruje poprawny plik dla odpowiedniego solvera. Ponadto, kod ten będzie mógł zostać zastosowany jako model reguły mechanizmu rynkowego do narzędzia SolveM3.

Abstrakcyjny *element* powinien zawierać: pewną nazwę oraz odwołanie do odpowiedniego elementu (bądź elementów) pliku M3XML. Odwołanie to powinno mieć postać zapytania XPath, gdy będziemy rozważali bezpośrednie stosowanie danych zawartych w notacji M3XML, bądź też odpowiednie przekształcenie tych danych za pomocą transformacji XSLT czy XQuery. Na podstawie powyższej analizy możemy zaproponować odpowiednie pola dla abstrakcyjnej klasy element:

- Nazwa (unikalna)
- Element XML:
  - wskaźnik XPath na odpowiedni element, atrybut węzeł bądź zbiór węzłów. Stosowany, gdy odpowiedni element zbioru M3XML odpowiada elementowi docelowemu (bez konieczności dodatkowych transformacji);

- przekształcenie XSLT, którego zadaniem jest proste przekształcenie fragmentu zbioru M3XML na wartości odpowiednie dla danego modelu;
- zapytanie XQuery, które zwróci odpowiedni wynik; jego zadanie jest analogiczne do zadania przekształcenia XSLT.

Zbiór dziedziczy po abstrakcyjnej klasie element. Definiuje on odpowiedni zbiór wykorzystywany do definicji pozostałych elementów modelu. Zbiory są budowane na podstawie danych pobranych z plików M3XML. Zbiory posiadają zdefiniowane indeksy, które służą do indeksacji poszczególnych ich elementów. Indeksy te są wykorzystywane przy definicji innych zbiorów, parametrów, zmiennych oraz ograniczeń, gdy te posiadają dwa lub więcej wymiary. Wartości elementów zbioru są budowane na podstawie zawartości zbioru M3XML.

Parametr reprezentuje skalar, wektor bądź (wielowymiarową) macierz danych. Do reprezentacji poszczególnych wymiarów używane są indeksy, zdefiniowane na poziomie zbiorów. Wartości parametrów są budowane na podstawie danych pobranych z plików M3XML. Dziedziczy po klasie element.

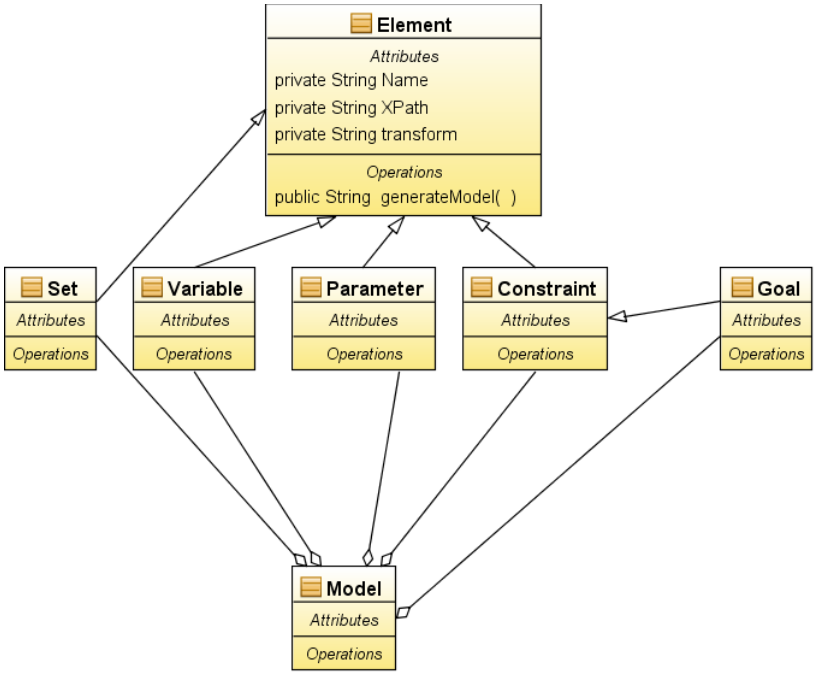
Zmienna decyzyjna reprezentuje zmienną modelu optymalizacyjnego. Definiuje się ich zakres oraz charakter (zmienne binarne, całkowitoliczbowe, rzeczywiste). Zmienne mogą tworzyć wektory bądź wielowymiarowe macierze, do ich tworzenia używane są indeksy. Dziedziczy po klasie element.

Ograniczenia zadania programowania liniowego lub mieszanego, których ogólna postać przedstawia się następująco: $Ax \le b$, gdzie A jest macierzą parametrów, x jest wektorem zmiennych, zaś b jest wektorem parametrów (prawych stron ograniczeń). Ograniczenia każdego zadania programowania liniowego lub mieszanego można zapisać w taki sposób. Tak jak poprzednich w przypadkach, ograniczenia także mogą występować jako wektory bądź macierze ograniczeń. Dziedziczą po klasie element.

Funkcja celu definiuje funkcję, której optymalne wartości będą poszukiwane podczas optymalizacji. Dziedziczy po klasie ograniczenia.

W końcu *Model*, który agreguje poszczególne elementy i tworzy model matematyczny. Pozwala także na eksport modeli w postaci transformaty XSLT bądź zbioru zapytań XQuery.

Z przeprowadzonych rozważań wynika bazowy diagram klas dla problemu (patrz rys. 2). Zakłada się, że aplikacja dostarczy wygodnego narzędzia do definiowania zapytań XPath, elementów transformat XSLT oraz zapytań XQuery. Przykład takiego narzędzia, wykorzystującego schematy XML, został wykorzystany w aplikacji AIMMS [1]. Narzędzie to, na podstawie pliku schematu, tworzy graficzną reprezentację drzewa DOM. Na jego podstawie można skonstruować zapytania XPath.



Rys. 2. Schematyczny diagram klas docelowej aplikacji
Fig. 2. Schematic class diagram for an application

3.3. Przykładowe wykorzystanie

Rozważmy istnienie następującego fragment pliku M3XML. Przedstawione są w nim trzy oferty sprzedaży towaru o nazwie `"ex:energia"`. Oferty te są prezentowane przez dwa podmioty rynkowe o identyfikatorach `"ex:s1"` oraz `"ex:s2"`:

```
<m3:offers>
<m3:Offer id="ex:s1-offer1" offeredPrice="37.00">
  <m3:offeredBy ref="ex:s1"/>
  <m3:volumeRange minValue="30" maxValue="60"/>
  <m3:ElementaryOffer>
   <m3:offeredCommodity shareFactor="1" ref="ex:energia"/>
  </m3:ElementaryOffer>
 </m3:Offer>
 <m3:Offer id="ex:s2-offer1" offeredPrice="23.00">
  <m3:offeredBy ref="ex:s2"/>
  <m3:volumeRange minValue="10" maxValue="30"/>
  <m3:ElementaryOffer>
   <m3:offeredCommodity shareFactor="1" ref="ex:energia"/>
  </m3:ElementaryOffer>
 </m3:Offer>
 <m3:Offer id="ex:s1-offer2" offeredPrice="42.00">
  <m3:offeredBy ref="ex:s1"/>
  <m3:volumeRange minValue="0" maxValue="50"/>
  <m3:ElementaryOffer>
   <m3:offeredCommodity shareFactor="1" ref="ex:energia"/>
  </m3:ElementaryOffer>
 </m3:Offer>
</m3:offers>
```

Z tego fragmentu zbioru M3XML chcemy wyciągnąć: zbiór ofert sprzedaży oraz parametr oznaczający ceny ofertowe za poszczególne towary. Prezentowane poniżej fragmenty wyge-

nerowanych danych są zgodne z notacją AMPL, jednak należy pamiętać, że są to tylko przykłady i docelowa aplikacja może używać innej notacji.

Aby uzyskać zbiór wszystkich ofert, należy zastosować następujące zapytanie XPath:

```
/m3:offers/m3:Offer/@id.
```

Zakładamy, że elementami tego zbioru będą unikalne identyfikatory poszczególnych ofert. Dzięki temu zapytaniu XPath otrzymamy zbiór wszystkich ofert. Aby ograniczyć zakres ofert do ofert sprzedaży towaru `ex:energia`, należy wzbogacić zapytanie o warunki:

```
/m3:offers/m3:Offer[m3:ElementaryOffer/m3:offeredCommodity/@shareFactor = '1'
           and  m3:ElementaryOffer/m3:offeredCommodity/@ref = 'ex:energia']@id.
```

Aby uzyskać odpowiedni obiekt klasy zbiór, należy zaaplikować do niego to zapytanie XPath. Załóżmy, że zbiór ten nosi nazwę `sellOffers`. Wówczas jego definicja wygląda następująco: `set sellOffers := ex:s1-offer1 ex:s2-offer1 ex:s1-offer2;`

Aby uzyskać parametr oznaczający cenę ofertową sprzedaży poszczególnych ofert, należy wykorzystać odpowiedni iterator uprzednio uzyskanego zbioru. Iterator ten ma postać:

```
/m3:offers/m3:Offer[m3:ElementaryOffer/m3:offeredCommodity/@shareFactor = '1'
             and  m3:ElementaryOffer/m3:offeredCommodity/@ref = 'ex:energia']
```

Oprócz powyższego iteratora należy dobrać odpowiednie wyrażenie XPath, odpowiadające za wartości, które ma postać `@offeredPrice`, oraz wyrażenie odpowiadające za klucze, które ma postać `@id`. W ten sposób otrzymujemy całość zapytania, które zwraca wektor klucz-wartości, czyli pożądany przez nas parametr (wektor):

```
sellPrice :=
ex:s1-offer1  37.00
ex:s2-offer1  23.00
ex:s1-offer2  42.00;
```

Jednak nie zawsze zapytanie XPath wystarcza, aby poprawnie zdefiniować elementy modelu. Przykładem może być potrzeba definicji sieci telekomunikacyjnej, w której istnieją węzły połączone łączami telekomunikacyjnymi. Budowa takiego modelu w M^3 jest oczywista: należy zdefiniować węzły sieci (`m3:node`) oraz krawędzie (`m3:arc`), które będą reprezentowały poszczególne łącza. Aby zamodelować połączenie pomiędzy węzłami, należy wprowadzić referencję do odpowiednich węzłów w elemencie `m3:predecessor` oraz `m3:succesor` węzła `m3:arc`:

```
<m3:node id="ex:nodeA" dref="ex:CommunicationNode"/>
<m3:node id="ex:nodeB" dref="ex:CommunicationNode"/>
...
<m3:arc id="ex:arcA-B" dref="ex:BandwidthArc">
     <m3:predecessor ref="ex:nodeA"/>
      <m3:successor ref="ex:nodeB"/>
 </m3:arc>
```

Jednak przejście do definicji parametru oznaczającego macierz incydencji takiej sieci nie jest już sprawą trywialną. Aby utworzyć taką macierz, należy zastosować m.in. odpowiednią transformatę XSLT:

```
<xsl:choose>
  <xsl:when test="key('arc_key', $id_edge)/m3:predecessor/@ref = $id_node">1</xsl:when>
  <xsl:when test="key('arc_key', $id_edge)/m3:successor/@ref = $id_node">-1</xsl:when>
  <xsl:otherwise>0</xsl:otherwise>
</xsl:choose>
```

gdzie `'arc_key'` oznacza odpowiedni klucz XSLT odpowiadający elementom a nazwie `m3:arc`, `$id_edge` oznacza iterator zbioru krawędzi, a `$id_node` oznacza iterator zbioru węzłów.

4. Przykładowa implementacja

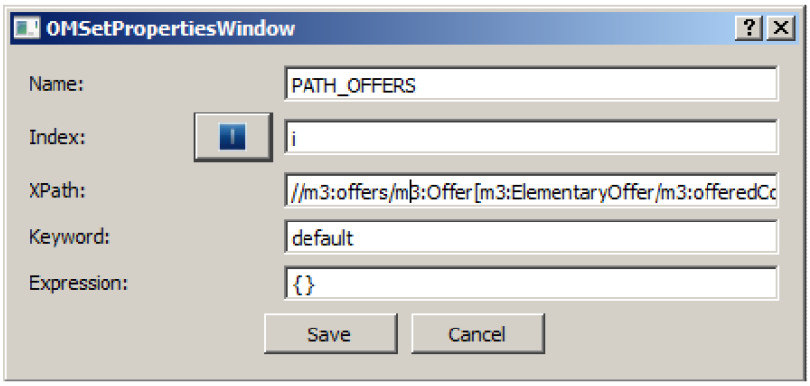
Przykładowa realizacja części założeń przedstawionych w niniejszej pracy została zaimplementowana w aplikacji *GeneratorXSLT*, opisanej dokładniej w pracy [11]. Do implementacji narzędzia posłużył autorowi język C++, do implementacji graficznego interfejsu użytkownika zastosowany został framework Qt. Na rysunku 3 przedstawione zostało przykładowe okno aplikacji GeneratorXSLT.

4.1. Ewaluacja rozwiązania

W pracy [11] autor przeprowadza ewaluację aplikacji GeneratorXSLT. Do ewaluacji użyty został model BCBT (ang. *Balancing Communication Bandwidth Trade*), służący do realokacji zasobów w sieci telekomunikacyjnej [12]. Jest to model handlu wielotowarowego, z dodatkowymi ograniczeniami dotyczącymi ustalania zależności pomiędzy łączami a tworzonymi na żądanie ścieżkami. Łącza są to istniejące połączenia pomiędzy poszczególnymi węzłami sieci telekomunikacyjnej, natomiast ścieżka jest to połączenie pomiędzy dwoma węzłami, które nie muszą być połączone bezpośrednim łączem. Model BCBT jest modelem infrastrukturalnym, posiada nietrywialny model matematyczny, tak więc przetestowanie działania aplikacji dla tego modelu udowodni jej użyteczność.

Autor pracy [11] stwierdza, że możliwe jest zamodelowanie danego problemu, korzystając z aplikacji GeneratorXSLT. W aplikacji zamodelowano regułę alokacji BCBT. Wytworzona na podstawie modelu matematycznego transformata, zastosowana do przykładowego pliku M3XML zawierającego opis specyficznego problemu dotyczącego handlu na rynku zasobów telekomunikacyjnych, jest poprawna. Plik M3XML powstał na podstawie jednego z eksperymentów przeprowadzonych w pracy [12], jego fragmenty zostały opisane w pracy [5]. Zastosowanie otrzymanej transformaty XSLT do aplikacji SolveM3 daje poprawne wyni-

ki, zgodne z tymi opisanymi w pracy [12]. Na podstawie tego można stwierdzić, że aplikacja działa poprawnie i spełnia założenia metodyki.



Rys. 3. Przykładowe okno aplikacji – definicja zbioru
Fig. 3. Exemplary application window – set definition

Aplikacja GeneratorXSLT, będąca pierwszym podejściem do implementacji sformułowanych w przedstawionej metodyce założeń, spełnia najważniejsze postulaty dotyczące wykorzystania narzędzi XML do tworzenia matematycznych modeli optymalizacyjnych. Jest to pierwsza, przykładowa, implementacja, która posiada pewne wady, jak na przykład konieczność ręcznego wprowadzania fragmentami kodu XPath czy XSLT. Jednak poza tymi fragmentami kodu, aplikacja generuje dużą część kodu XSLT automatycznie, wstawiając wprowadzone ręcznie fragmenty w odpowiednie miejsca. Przedstawiona w pracy [11] ewaluacja pokazuje prostotę formułowania nowych oraz modyfikacji istniejących modeli matematycznych. Jednocześnie udowodniona jest poprawność sformułowanej metodyki.

5. Podsumowanie

Modelowanie nowych i modyfikacja istniejących reguł mechanizmów rynkowych jest bardzo ważne elementy projektowania rynku. Aby utworzyć poprawny mechanizm rynkowy, spełniający własności pożądane zarówno przez jego projektanta, jak i operatora, a także przez uczestników danego rynku, należy zamodelować wiele pośrednich reguł i przetestować ich własności. Ze względu na potrzebę testowania różnych rozwiązań, wydaje się, że warto zastosować cykliczny proces projektowania mechanizmów rynkowych. W tym celu wydaje się, że potrzebny jest odpowiedni język do modelowania danych rynkowych, a także wygodnego narzędzia służącego do modelowania reguł mechanizmów rynkowych. Dzięki modelowi M^3 oraz zaproponowanej w niniejszej pracy metodyce mamy możliwość prowadzić cykliczny proces projektowania mechanizmów rynkowych.

Na podstawie przeprowadzonej ewaluacji przykładowego narzędzia, powstałego na podstawie przeprowadzonej uprzednio analizy, wydaje się, że zaproponowana w pracy metodyka jest poprawna. Ponadto, można stwierdzić, że jest możliwe utworzenie narzędzia do generowania, wymaganych do operacji na plikach M3XML, transformat bądź zestawów zapytań. Narzędzie takie niewątpliwie ułatwi pracę poświęconą na modelowanie reguł rynkowych. Ponadto, pozwoli na ponowne użycie pewnych wzorców projektowych, które, na podstawie istniejących modeli optymalizacyjnych, ułatwią tworzenie nowych.

BIBLIOGRAFIA

1. Bisschop J., Roelofs M.: Aimms – Language Reference. Lulu.com, 2006.
2. GLPK (GNU Linear Programming Toolkit), http://www.gnu.org/software/glpk.
3. Haftka R. T., Gürdal Z.: Elements of structural optimization. Springer, 1992.
4. Hurwicz L., Reiter S.: Designing Economic Mechanisms, Cambridge University Press, 2006.
5. Kacprzak P., Kaleta M., Pałka P., Smolira K., Toczyłowski E., Traczyk T: Application of open multi-commodity market data model on the communication bandwidth market. Journal of Telecommunications and Information Technology, No. 4, 2007, s. 45÷50.
6. Kacprzak P., Kaleta M., Pałka P., Smolira K., Toczyłowski E., Traczyk T.: Procesor decyzyjno-obliczeniowy dla rynkowego modelu danych M3. Kozielski Stanisław et al. (red.): Bazy Danych Rozwój Metod i Technologii. Architektura, metody formalne i zaawansowana analiza danych. Wydawnictwa Komunikacji i Łączności, 2008, s. 215÷226.
7. Krishna V.: Auction Theory. Academic Press, 2010.
8. Kaleta M., Pałka P., Toczyłowski E., Traczyk T.: Electronic trading on electricity markets within a multi-agent framework. Lecture Notes In Computer Science, Springer, Vol. 5796, Springer, Heidelberg 2009, s. 788÷799.
9. Kaleta M., Traczyk T. (eds.): Modeling Multi-commodity Trade: Information Exchange Methods. Advances in Intelligent and Soft Computing, Vol. 121, 2012.
10. Shoham Y., Leyton-Brown K.: Multi-agent Systems: Algoritmic, Game-Theoretic, and Logical Foundations. Cambridge University Press, 2009.
11. Sienicki K.: Zastosowanie narzędzi języka XML do generowania modeli optymalizacyjnych. Praca inżynierska, Politechnika Warszawska, 2011.
12. Stańczuk W., Lubacz J., Toczyłowski E.: Trading links and paths on a communication bandwidth markets. Journal of Universal Computer Science, Vol. 5(14), 2008, s. 642÷652.

13. Williams S.: Communication in Mechanism Design. Cambridge University Press, 2008.
14. XPath – XML Path Language website, http://www.w3.org/TR/xpath/.
15. XQuery – XML Query Language website, http://www.w3.org/TR/xquery/.
16. XSD – XML Schema website, http://www.w3.org/XML/Schema.
17. XSL Transformations website, http://www.w3.org/TR/xslt.

Wpłynęło do Redakcji 10 stycznia 2012 r.

Abstract

From the viewpoint of the market mechanisms designer, the formulation of the rules, according to which the market mechanism will work, should be convenient. These rules are formulated using mathematical models. Multi-commodity market model M^3 provides a convenient way to notation of data on different market segments. There are many papers on the M^3 model, in which the authors formulate mathematical models adapted to various trade segments (electrical energy markets, telecommunications, airport and railway resource allocation, the market for trading permits to emit greenhouse gases, distributed markets, etc.). These models are formulated on the basis of the multi-commodity market model, enriched with additional trade restrictions. The literature describes the SolveM3 tool that collects market data formulated using the M3XML notation, solves the corresponding mathematical model and returns the results in the M3XML notation. The mathematical model is formulated in the form of transformations written in XSL or XQuery notation. The disadvantage of this approach is the difficulty in creating new trade models, and modification of existing. This is due inconvenience of the XSLT and XQuery languages, and a considerable amount of code. The proposed methodology assumes the use of XML tools to generate mathematical optimization models, based on data notated using the M3XML dialect. The basic assumptions of the methodology and sample target application class diagram are presented. Work also relies on existing, exemplary implementation built upon the presented methodology. Analysis of the functionality of the application will serve as the basis for the methodology evaluation.

Adres

Piotr PAŁKA: Politechnika Warszawska, Instytut Automatyki i Informatyki Stosowanej, ul. Nowowiejska 15/19, 00-665 Warszawa, Polska, P.Palka@ia.pw.edu.pl