STUDIA INFORMATICA 2009
Volume 30 Number 3A (85)

Krzysztof DOBOSZ, Andrzej HEFCZYC
Politechnika Śląska, Instytut Informatyki

WIZUALIZACJA DANYCH XML

Streszczenie. W artykule zaproponowano prostą metodę typowania danych oraz sposób na przypisanie do zdefiniowanych typów komponentów graficznych pozwalających na wizualizację i modyfikację danych. Dla tych komponentów zaproponowano interfejs programistyczny pozwalający na przygotowywanie własnych implementacji w języku Java. Wskazano kilka przykładowych typów danych oraz implementacji komponentów z nimi związanych.

Słowa kluczowe: XML, Swing, Java, wizualizacja danych

VISUALISATION OF XML DATA

Summary. In the article a simple method of data typing and a way to assign graphical components allowing for the visualisation and XML data modification are proposed. For these components a programming interface was offered, letting to prepare own implementations using Java language. A few sample data types were indicated and the implementation of components connected with them.

Keywords: XML, Swing, Java, data visualization

1. Wstęp

Choć język XML ma wiele zalet, to bezpośrednie manipulacje zawartością tekstową pliku w tym formacie mogą być kłopotliwe. Dokumenty XML są przede wszystkim plikami tekstowymi zapisanymi z użyciem odpowiedniej składni [1]. Z jednej strony jest to dobre, gdyż użytkownikowi znającemu ten język opisu danych wystarczy tylko edytor tekstowy do modyfikacji takiego pliku. Z drugiej jednak strony, dla osoby nieposiadającej takiej wiedzy, edycja takiego pliku tylko za pomocą edytora tekstowego może być uciążliwa. Ponadto, wszelkie

wartości wewnątrz znaczników oraz wartości atrybutów są tekstem, więc ich bezpośrednia modyfikacja za pomocą edytora tekstowego może prowadzić do powstania błędów.

Ponieważ graficzna wizualizacja danych mogłaby rozwiązać wiele wymienionych tu trudności, więc celem praktycznym autorów było opracowanie graficznego edytora dedykowanego językowi XML, który uwzględnia typowanie danych w znacznikach i atrybutach. Przypisanie typu umożliwia związanie z nim odpowiedniego komponentu prezentującego tę wartość oraz komponentu umożliwiającego modyfikację tej wartości. Użytkownikowi korzystającemu z takiego narzędzia wystarcza wiedza, że język XML zbudowany jest z hierarchicznej struktury elementów, a te mogą posiadać atrybuty oraz wartość. Przykładowo, jeśli dany atrybut posiadałby wartość logiczną – to można ją reprezentować za pomocą komponentu pola wyboru, jeżeli byłaby to wartość reprezentująca kolor – to zamiast tej wartości tekstowej mogłaby ona zostać zaprezentowana np. przez obszar zamalowany w odpowiednim kolorze itp. Dodatkowo, dedykowane dla danego znacznika czy atrybutu narzędzie modyfikacji uniemożliwiłoby wprowadzenie wartości błędnej. Zestaw typów danych i związanych z nimi komponentów graficznych może być zupełnie dowolny.

Odpowiednie narzędzie pozwalające na automatyczne uruchamianie graficznych komponentów do wyświetlania i modyfikacji danych XML zostało zaimplementowane w języku Java i udokumentowane [2]. Hierarchię znaczników reprezentuje w nim komponent drzewa, zaś listę atrybutów i ich wartości dwukolumnowa tabela. Na wartość znacznika przeznaczono oddzielny obszar graficzny. Celem autorów była odpowiednia wizualizacja wartości atrybutów i samych znaczników.

Niniejszy artykuł przedstawia bliżej tylko sposób typowania danych, wiązania typów z odpowiednimi komponentami oraz specyfikację dla przygotowywania własnych implementacji do wizualizacji wymienionych danych. Ponieważ proponowana specyfikacja przedstawiona jest w języku Java, więc w kolejnych rozdziałach pojawi się sporo odniesień do klas i metod już istniejących, a zdefiniowanych w standardowych bibliotekach Javy [3].

2. Plik konfiguracyjny

Aby uniwersalny edytor działał prawidłowo, musi otrzymać dodatkowe dane związane z plikiem XML, którego zawartość ma być wizualizowana. W tym celu konieczne jest przygotowanie odpowiedniego pliku konfiguracyjnego. Plik ten również jest plikiem XML i dostarcza informacji o typach danych zawartych w znacznikach i ich atrybutach oraz o komponentach związanych z tymi typami. Opracowany schemat takiego pliku przedstawia się następująco:

```
<xml_config>
   <attributes>
      <attribute name="nazwa_atrybutu" typeid="nazwa_typu"/>
      ...
   </attributes>
   <tags>
      <tag name="nazwa_znacznika" typeid="nazwa_typu"
            icon="plik_ikony"/>
      ...
   </tags>
   <types>
      <type id="nazwa_typu">
         <renderer class="klasa_wyświetlacza">
            <param name="nazwa_parametru"
                    value="wartość_parametru"/>
            ...
            <list_param name="nazwa_parametru">
                   <param_value> wartość_parametru </param_value>
               <param_value> wartość_parametru </param_value>
               ...
            </list_param>
            ...
         </renderer>
         ...
         <editor class="klasa_edytora"/>
            <param name="nazwa_parametru" value="wartość_parametru"/>
            ...
            <list_param name="nazwa_parametru">
               <param_value> wartość_parametru </param_value>
               <param_value> wartość_parametru </param_value>
               ...
            </list_param>
         </editor>
         ...
      </type>
      ...
   </types>
</xml_config>
```

Plik konfiguracyjny posiada jeden element główny o nazwie *xml_config*, wewnątrz którego znajdują się trzy elementy wewnętrzne o nazwach: *attributes*, *tags* oraz *types*, definiujące odpowiednio: atrybuty, znaczniki oraz typy. Elementy te mogą być rozmieszczone w dowolnej kolejności.

Sekcja definicji atrybutów rozpoczyna się znacznikiem otwierającym *attributes*. Każdy wewnętrzny element o nazwie *attribute* odpowiada definicji jednego atrybutu, jego parametry *name* oraz *typeid* służą do określenia odpowiednio nazwy atrybutu i typu jego wartości zdefiniowanego w sekcji typów. Obecność tych dwóch atrybutów jest obligatoryjna.

Fragment określający znaczniki znajduje się w elemencie o nazwie *tags*. Posiada on wewnętrzne elementy o nazwie *tag*, opowiadające kolejnym definicjom znaczników. Pojedynczy element *tag* posiada trzy atrybuty: *name*, *typeid* oraz *icon*. Znaczenie dwóch pierwszych jest takie samo jak w sekcji definicji atrybutów, natomiast trzeci – *icon* określa nazwę pliku graficznego wraz z rozszerzeniem, który pełni funkcję ikony do reprezentacji znacznika w aplikacji narzędziowej. Wymagana jest obecność dwóch pierwszych atrybutów, w przypadku braku argumentu *icon* do wyświetlenia znacznika w komponencie drzewa użyta zostanie domyślna ikona.

Fragment rozpoczynający się znacznikiem *types* odpowiada za deklaracje typów. Przez typ rozumie się dowolny łańcuch znakowy (brak typów predefiniowanych) reprezentujący odpowiednie komponenty do wizualizacji i edycji danych oraz listę ich parametrów konfiguracyjnych. Poszczególne elementy wewnętrzne o nazwie *type* odpowiadają konkretnym typom danych. Wewnątrz elementu *type* znajdują się dwa elementy o nazwach *renderer* oraz *editor*. Zawierają one identyfikatory klas komponentów do wyświetlania i edycji, służące do obsługi wartości danego typu. Brak danego elementu spowoduje przerwanie procesu konfiguracji narzędzia. Oba elementy mają analogiczną strukturę. Posiadają one parametr o nazwie *class* wskazujący identyfikator klasy komponentu odpowiedzialnego za wyświetlanie bądź edycję. Aby poprawnie działać, identyfikator ten musi zawierać nazwę pakietu, jeśli kod bajtowy klasy o podanej nazwie wewnątrz takiego się znajduje. W trakcie uruchamiania aplikacji wykorzystującej omawiany komponent należy zadbać, żeby wirtualna maszyna Javy miała dostęp do pakietów i klas z komponentami, które mają być użyte.

Wewnątrz elementu *renderer* lub *editor* mogą znaleźć się fragmenty konfigurujące narzędzie, jest to lista elementów o nazwach *param* oraz *list_param*. Pierwszy z nich służy do przesyłania parametrów skalarnych typu prostego (*boolean, char, int, long, float, double, String*), drugi natomiast pozwala przesłać parametr indeksowany – lista parametrów skalarnych. W przypadku parametru skalarnego wystarczy umieścić po znaczniku *param* atrybut o nazwie *name* z nazwą parametru oraz jako wartość tekstową podać wartość tego parametru. Z parametrem indeksowanym postępujemy podobnie, z tą różnicą, że zamiast wartości parametru umieszczonego w wartości tekstowej znacznika, liście wartości opowiada ciąg elementów *param_value* z atrybutami *value* deklarującymi odpowiednią wartość. Elementy *list_param* oraz param mogą występować w dowolnej kolejności, nawet naprzemiennie.

3. Interfejs programistyczny

Ideą opracowanego narzędzia jest możliwość wykorzystania samodzielnie przygotowanych komponentów graficznych dla wizualizacji określonych danych. Kod bajtowy takich komponentów w opracowywanym narzędziu jest dynamicznie wywoływany z odpowiednich plików za pomocą mechanizmu refleksji [4].

Aby przygotować własne komponenty, należy zdefiniować w języku Java dwie klasy, implementujące odpowiednie metody interfejsu: *XMLEditorCellRenderer* – dla komponentu służącego do wyświetlania danych oraz *XMLEditorCellEditor* – dla komponentu służącego do edycji danych. Oba te interfejsy są pewną modyfikacją interfejsów *TableCellRenderer* oraz *TableCellEditor* dostępnych w ramach biblioteki Swing [5], a służących do tworzenia klas wyświetlających oraz edytujących dane w komórkach tabeli *JTable*. Jako obszar służący

do wyświetlenia graficznej reprezentacji danej oraz do jej edycji domyślnie przyjęto prostokątne pole komórki komponentu tabeli z biblioteki Swing. Jednakże, nic nie stoi na przeszkodzie, aby komponent służący do edycji uruchamiany był w niezależnym oknie.

3.1. Interfejs komponentu wyświetlania

Interfejs komponentu wyświetlania reprezentowany jest przez klasę *XMLEditorCellRenderer* bazującą na klasie *TableCellRednerer*. Posiada on dwie metody o następujących nagłówkach:

- *Component getTableCellRendererComponent(JTable jTable, String value, boolean isSelected, boolean hasFocus) throws Exception*,
- *void updateUI()*.

Pierwsza z nich zwraca komponent obsługujący wyświetlanie wartości przekazanej przez parametr *value*. Dodatkowo przekazywana jest referencja do komponentu tabeli, w której znajduje się edytowana komórka. Parametr *isSelected* określa, czy komórka jest zaznaczona, a ostatni *hasFocus* informuje, czy komórka posiada podświetlenie. Dwa ostatnie parametry wymuszają na komponencie odpowiednie jego wyświetlenie. W przypadku sytuacji błędnej, gdy klasa nie jest w stanie zinterpretować przekazanej wartości, zostaje wygenerowany wyjątek. W takim wypadku dla wyświetlenia wartości w komórce zostanie użyty domyślny wyświetlacz ciągów znakowych z czerwonym tłem sygnalizującym błąd, jeśli taka opcja została włączona.

Druga z wymienionych metod odświeża wygląd komponentu w przypadku zmiany dynamicznej tematu wyglądu całej aplikacji (np. z MS Windows na Motif). Jeśli nie ma potrzeby zapewnienia dynamicznej zmiany wyglądu, wtedy można zostawić metodę pustą.

3.2. Interfejs komponentu edycji

Interfejs narzędzia edycji *XMLEditorCellEditor* bazuje na klasie *TableCellEditor*. Znaczenie poszczególnych metod tego interfejsu jest następujące:

- *void addCellEditorListener(CellEditorListener l)* – metoda ma na celu dodanie obiektu nasłuchującego zdarzenia generowane przez narzędzie edycji,
- *void removeCellEditorListener(CellEditorListener l)* – metoda usuwa zarejestrowanego słuchacza zdarzeń narzędzia edycji,
- *void cancelCellEditing()* – metoda ta kończy edycję oraz anuluje wszelkie wprowadzone zmiany, a wewnątrz niej następuje wygenerowanie zdarzenia *ChangeEvent* oraz wywołanie metody *editingCanceled()* z tym zdarzeniem, jako parametrem dla wszystkich obiektów słuchaczy,

- *boolean stopCellEditing()* – metoda ta kończy edycję oraz zatwierdza wprowadzone zmiany, a wewnątrz niej następuje wygenerowanie zdarzenia *ChangeEvent* oraz wywołanie metody *editingStopped()* z tym zdarzeniem, jako parametrem dla wszystkich obiektów słuchaczy,
- *Component getTableCellEditorComponent(JTable jTable, String value, boolean isSelected) throws Exception* – metoda zwraca komponent obsługujący edycję wartości przekazanej przez parametr *value*, dodatkowo przekazywana jest referencja do komponentu tabeli, w której to znajduje się edytowana komórka. Ostatni parametr *isSelected* określa, czy komórka jest zaznaczona. W przypadku sytuacji błędnej, gdy klasa nie jest w stanie zinterpretować przekazanej wartości, zostaje wygenerowany wyjątek. W takim wypadku do edycji wartości w komórce zostaje użyty domyślny edytor ciągów znakowych z czerwonym tłem sygnalizującym błąd, jeśli taka opcja została włączona,
- *String getCellEditorValue()* – metoda ta zwraca wartość uzyskaną po edycji,
- *String getDefaultCellEditorValue()* – metoda zwraca domyślną wartość danego typu danych. Jest wykorzystywana w momencie dodawania nowych atrybutów,
- *void updateUI()* – metoda odświeża wygląd komponentu w przypadku zmiany dynamicznej tematu wyglądu całej aplikacji. Jeśli nie ma potrzeby zapewnienia dynamicznej zmiany wyglądu, wtedy można zostawić metodę pustą.

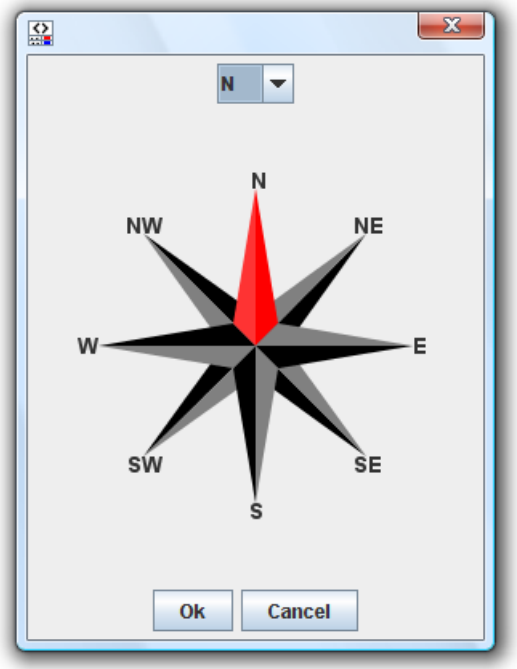
4. Przykładowe typy danych i komponenty

W celu sprawdzenia poprawności funkcjonowania mechanizmu dynamicznego wywoływania kodu bajtowego komponentów przygotowywanych na podstawie opracowanej specyfikacji, zaproponowano kilka podstawowych oraz często spotykanych typów danych oraz zaimplementowano dla nich komponenty, służące do wyświetlania i edycji ich wartości. Implementacje te wykorzystują istniejące już komponenty z pakietu Swing, opakowując je w odpowiednie elementy wymaganych interfejsów. Zaproponowane typy to:

- typ znakowy – do wyświetlania oraz edycji jego wartości służą klasy *StringRenderer* oraz *StringEditor*. Pierwsza z nich wykorzystuje w swojej implementacji komponent *JLabel*, a druga – *JTextField*. Oba komponenty posiadają jeden parametr konfiguracyjny typu *String*, który ustawia sposób wyrównania tekstu. Może przyjmować on jedną z trzech wartości: LEFT, RIGHT, CENTER oznaczające odpowiednio: wyrównanie do lewej strony, prawej strony, wycentrowanie tekstu,
- typ logiczny – do jego obsługi należy wybrać klasy *BooleanEditor* i *BooleanRenderer*, obie bazują na komponencie *JCheckBox*,

- typ numeryczny – podczas wyboru narzędzi do obsługi wartości tego typu mamy alternatywę – można wybrać zestaw klas *SliderEditor* oraz *SliderRenderer* bazujących na komponencie suwaka (*JSlider*) lub drugie rozwiązanie, którym jest użycie klas *SpinnerEditor* oraz *SpinnerRenderer*. Klasy te posługują się komponentem *JSpinner*, który działa podobnie jak pole tekstowe, jednak dodatkowo posiada dwa przyciski, umożliwiające dekrementacje lub inkrementację wartości. Obydwa zestawy narzędzi konfigurowane są w podobny sposób. Można do nich przesłać trzy parametry typu całkowitego: maksymalną, minimalną i domyślną wartość. W przypadku korzystania z edytora *SpinnerEditor* możliwe jest wykorzystanie klasy *StringRenderer* do jego prezentacji. Użycie klas *SpinnerEditor* oraz *SpinnerRenderer* umożliwia ustawienie wyrównania tekstu przez odpowiedni parametr, którego znaczenie jest analogiczne jak dla edytora ciągu znakowego. Jednocześnie *SpinnerEditor* umożliwia także ustawienie wartości skoku inkrementacji lub dekrementacji za pomocą odpowiedniego parametru typu *int*,
- typ daty – dla niego zostały zaimplementowane komponenty: *SpinnerDateEditor* oraz *SpinnerDateRenderer*. Bazują one na komponencie *JSpinner*. Obie posiadają parametr konfiguracyjny formatujący sposób wyświetlania daty,
- typ listy wartości – do edycji takiego typu danej należy użyć klasy *ComboBoxEditor*. Wykorzystuje ona komponent listy rozwijalnej do edycji wyboru wartości. Posiada też parametr konfiguracyjny, którego lista wartości będzie odpowiadać zawartości listy rozwijalnej. Jako komponentu wyświetlania najwygodniej jest użyć klasy *StringRenderer*,
- typ koloru – do jego obsługi stworzone zostały klasy *ColorRenderer*, *ColorEditor*. Nie posiadają parametrów konfiguracyjnych. Klasa *ColorRenderer* wykorzystuje *JLabel* z wykreślonym prostokątem w odpowiednim kolorze do jego prezentacji, natomiast edytor korzysta z komponentu *JColorChooser* do pełnej modyfikacji koloru.

Wymienione typy danych oraz związane z nimi komponenty graficzne stanowią tylko propozycję autorów, a ewentualny użytkownik ma w kwestii ich wykorzystania zupełną dowolność. W celu przetestowania elastyczności zaproponowanej metody do wizualizacji nietypowych danych zaproponowano pewien niestandardowy typ danych – typ kierunku. Przygotowano dla niego dwa komponenty graficzne oparte na klasach *DirectionEditor* oraz *DirectionRenderer*. Klasy te nie posiadają parametrów konfiguracyjnych. Obsługują one wartości oznaczające kierunki geograficzne. Użytkownik może wybrać jedną z głównych wartości (N, E, S, W) lub jedną z wartości pośrednich (NE, NE, SW, SE) (rys. 1).



Rys. 1. Komponent wyboru kierunku geograficznego
Fig. 1. Component for geographical direction

Czasami zdarza się, iż wewnątrz znacznika niepożądana jest obecność wartości tekstowej. W takiej sytuacji przydają się klasy *NullEditor* oraz *NullRender*, gdyż blokują one możliwość edycji wartości pustej. Powinno się je wykorzystywać tylko dla wartości znaczników. W swojej implementacji korzystają z komponentu *JTextField*.

5. Uwagi końcowe

Efektem praktycznym pracy jest powstanie uniwersalnego narzędzia opartego na maszynie wirtualnej Javy i umożliwiającego edycję plików XML. Ponieważ narzędzie to, podobnie jak klasy do wyświetlania i edycji danych, ma postać komponentu programowego, więc można wykorzystywać go w różnych aplikacjach. Zaimplementowany komponent charakteryzuje się dużą uniwersalnością oraz rozszerzalnością. Umożliwia dynamiczne wywoływanie innych komponentów (związanych z konkretnymi typami danych) z zewnętrznych plików z kodem bajtowym. Dlatego tez skalowanie tego narzędzia odbywa się przez proste zmiany w pliku konfiguracyjnym i dołączanie do jego katalogu plików z kodem bajtowym wymaganych komponentów wyświetlania i edycji danych.

Pomimo swej uniwersalności opracowany edytor ma pewne ograniczenia. Przede wszystkim wspiera dokumenty, które nie posiadają wewnętrznych elementów poprzedzielanych tekstem. Zaprojektowane narzędzie pozwala na załadowanie pliku XML w całości korzystając z mechanizmu działającego na podstawie obiektowego modelu dokumentu (ang. *Docu-*

ment Object Model), z czym może wiązać się duże zapotrzebowanie na pamięć operacyjną [7, 8].

Z przetwarzaniem plików XML nieodłącznie związane są mechanizmy precyzowania struktury samego dokumentu, które pozwalają na jego walidację, jak na przykład XML Schema. Ponieważ zawartość takiego dokumentu w postaci tekstowej może być dość trudna w edycji, więc można wykorzystać odpowiedni graficzny edytor [9] i zintegrować go z narzędziem opisywanym w niniejszym artykule. Opracowane narzędzie można również rozbudować o obsługę przestrzeni nazw w dokumentach XML.

BIBLIOGRAFIA

1. Harold E. R., Means W. S.: XML in a Nutshell. Wyd. 3, O'Reilly 2004.
2. Hefczyc A.: Opracowanie i implementacja w języku Java uniwersalnego graficznego edytora umożliwiającego modyfikację dowolnych typów danych opisanych w pliku XML. Praca dyplomowa magisterska, Politechnika Śląska, Instytut Informatyki, Gliwice 2007.
3. Java Platform Standard Edition 6 API Specification,
    1. Witryna: http://java.sun.com/javase/6/docs/api/index.html (sierpień 2008)
4. Eckel B.: Thinking in Java. Wyd. 3, Helion, Gliwice 2003.
5. Bielecki J.: Java 4 Swing. Helion, Gliwice 2000.
6. Dobosz K. (red.).: Laboratorium programowania w języku Java. Wydawnictwo Politechniki Śląskiej, Gliwice 2004.
7. Harold E. R.: Processing XML with Java™: A Guide to SAX, DOM, JDOM, JAXP, and TrAX. Addison Wesley, 2002.
8. McLaughlin B., Edelson J.: Java i XML. Wyd. 3, Helion, Gliwice 2007.
9. Karkowski Ł.: Opracowanie i implementacja w języku Java uniwersalnego, graficznego edytora plików zgodnych z XML Schema. Praca dyplomowa magisterska, Politechnika Śląska, Instytut Informatyki, Gliwice 2007.

Recenzent: Dr inż. Maciej Bargielski

Wpłynęło do Redakcji 16 stycznia 2009 r.

Abstract

This article presents how to visualize data stored in text files XML in the simple way. Through the visualisation given, authors understand not only their graphical presentation, but also the possibility of modification with the help of graphical adequate tools. To this purpose a simple method of XML data typing was suggested. Also a method for assigning graphical components to the defined data types is described. The scheme of the configuration file was described in the chapter two of the article. In the next chapter for these components a programming interface was offered, thanks to which it is possible to prepare own implementations. The interface consists of two sets of methods: one for the component of the visualisation, second for the component given to the edition.

Implementation of several graphical components was done, then their dynamic invocation (using mechanism of reflection) was tested. Components were prepared for created, simple data types. Proposed data types are: character, logical, numerical, dates, lists of value and the colour. The elasticity of the suggested solution was presented on the basis of the type of geographical direction and component connected with it (fig. 1).

The suggested tool can be integrated with different components like e.g. tool for the edition of XML Schema files.

Adresy

Krzysztof DOBOSZ: Politechnika Śląska, Instytut Informatyki, ul. Akademicka 16, 44-100 Gliwice, Polska, krzysztof.dobosz@polsl.pl
Andrzej HEFCZYC: Politechnika Śląska, Instytut Informatyki, ul. Akademicka 16, 44-100 Gliwice, Polska, speeddemon@kopernet.org