LaTeX3 nadchodzi

Marcin Woliński
Warszawa
wolinski@mimuw.edu.pl
http://www.mimuw.edu.pl/~wolinski

Streszczenie

W artykule opisano eksperymentalne pakiety dla LaTeXa 2_ε, dostępne pod adresem `http://www.latex-project.org/code/experimental/`. Pakiety te są interesujące z dwóch powodów. Po pierwsze stanowią wstępne wersje podstawowych komponentów LaTeXa 3. Po drugie są użyteczne same w sobie (choć należy pamiętać o ich eksperymentalnym charakterze).

Wprowadzenie

Od kilku lat ekipa *LaTeX3 Project* pracuje nad nową wersją systemu składu LaTeX. Punktem wyjścia jest oczywiście oryginalne dzieło Leslie Lamporta. W ramach tego projektu powstała przejściowa wersja 2_ε, obecnie stanowiąca Standardowy LaTeX. Następnym etapem rozwoju systemu będzie LaTeX 3. Obecnie dostępne są fragmenty nowego systemu w postaci pakietów dla LaTeXa 2_ε. Autorzy nazywają tę hybrydę LaTeX 2_ε*.

LaTeX3, podobnie jak poprzednie wersje, będzie działał z każdą implementacją Standardowego TeXa (lub jego następcy).

W artykule [1] autorzy wymieniają następujące zalety LaTeXa:

- ugruntowany standard adiustacji tekstu, formuł matematycznych i tabel,
- rozdzielenie abstrakcyjnej adiustacji od formatowania,
- łatwość użycia (dla autorów),
- przenośność dokumentów,
- przystosowywalność do różnych języków,
- dostępność, niski koszt `:-)`,
- wsparcie i utrzymywanie systemu przez *LaTeX3 Project Team*.

Sternicy projektu mają natomiast świadomość, że pod pewnymi względami LaTeX odstaje od większości tzw. programów DTP. Dotyczy to zwłaszcza ścisłego wiązania tekstu z elementami graficznymi, np. stosowania apli we fragmentach akapitu lub tzw. *changebars*, czyli oznaczania części akapitu linią na marginesie kolumny.

Ponadto od czasu pierwszej wersji LaTeXa pojawiły się nowe zjawiska, które zmieniają obraz branży składu. Są to przede wszystkim rosnąca popularność systemów opartych na SGML oraz fakt, że obecnie dokumenty to nie tylko papier, ale i elektroniczne formy udostępniania (hipertekst).

Aby ułatwić używanie LaTeXa do formatowania dokumentów SGML-owych, w LaTeXu 3 mają zostać uwzględnione takie pojęcia SGML, jak całostki (*entities*), atrybuty i skróty notacyjne (*short references*).

Podobnie jak w LaTeXu 2_ε nastąpiła integracja poleceń do składu kolorowego, LaTeX 3 ma integrować obsługę hipertekstu: generowania dowiązań i innych elementów potrzebnych przy tworzeniu dokumentów prezentowanych *on-line*.

Najciekawszą nowością jest próba uporania się z bardziej fundamentalnym brakiem LaTeXa. Dotychczas użytkownicy LaTeXa dzielili się na dwie grupy: autorów oraz tzw. guru, czyli osoby umiejące programować w LaTeXu. Użytkownicy, którzy nie chcieli się uczyć, jak LaTeX jest skonstruowany, mieli bardzo ograniczony wpływ na wygląd swoich dokumentów. W zasadzie byli zdani na to, co przewidział autor używanej klasy w postaci opcji i niewielkiej liczby parametrów.

Tymczasem widać trzeci naturalny typ użytkownika. Jest to typograf projektujący wygląd dokumentu (czy składacz realizujący taki wygląd według czyjegoś projektu). Taki użytkownik nie jest zainteresowany programowaniem, tylko opisem wyglądu poszczególnych elementów dokumentu.

LaTeX 3 ma zawierać mechanizm pozwalający na projektowanie formatowania poprzez zestawianie ze sobą sparametryzowanych „klocków” (szablonów). Działanie projektanta polegałoby również na wiązaniu tych elementów z abstrakcyjnymi poleceniami adiustacyjnymi występującymi w dokumencie.

W założeniu interfejs ma być naturalny dla projektanta nie będącego programistą TeXowym. Można sobie w związku z tym wyobrazić pojawienie się graficznego narzędzia pozwalającego zestawiać ze sobą elementy formatujące i określać ich parametry.

Dalsze postulaty/zapewnienia z dokumentu [1] brzmią następująco:

- Zostanie ulepszona odporność na błędy, w szczególności zostaną usunięte sztuczne ograniczenia zagnieżdżania poleceń.
- Zostanie wprowadzony ulepszony interaktywny system pomocy.
- Nowe interfejsy zostaną szczegółowo udokumentowane. W dystrybucji systemu znajdzie się obszerny zbiór przykładów, starannie dobranych tak, aby zmniejszyć czas uczenia się dla nowych użytkowników (zarówno autorów jak i projektantów).
- Kod nowego systemu będzie modularny i dogłębnie udokumentowany. Pozwoli to łatwo utrzymywać i ulepszać system.

Niestety chęć pełnej ich realizacji może sprawić, że nie ujrzymy nowej wersji LATEXa w ciągu najbliższych lat.

Mechanizm analizy argumentów poleceń

Jeden z walorów LATEXa dla autorów stanowi rozdzielenie treści (struktury logicznej) i formy (formatowania) dokumentów. Jak zostało powiedziane wyżej, w LATEXu 3 ma zostać wprowadzony mechanizm projektowania typograficznego. Mianowicie zbiór poleceń dostępnych w dokumencie będzie traktowany jako abstrakcyjna adiustacja (coś w rodzaju DTD). Natomiast na potrzeby danej klasy dokumentów polecenia te będą wiązane z odpowiednimi mechanizmami formatującymi. Żeby było to możliwe, trzeba dać projektantowi narzędzie do łatwego określania takiego powiązania.

Pierwszą przeszkodą do osiągnięcia tego celu w bieżącym LATEXu są ograniczone możliwości polecenia `\newcommand`. Pozwala ono mianowicie definiować polecenia z argumentami wymaganymi i ewentualnie jednym (pierwszym) opcjonalnym. Nie wystarcza to do zdefiniowania niektórych poleceń LATEXa, np. polecenia `\parbox`, które ma 3 argumenty opcjonalne i 2 wymagane, ani poleceń trybu `picture`, które wymagają argumentów ujętych w nawiasy okrągłe.

Ograniczenia te pokonuje nowy mechanizm pobierania argumentów poleceń (eksperymentalny pakiet `xparse`, [2]). Oto przykład definicji polecenia:

```
\DeclareDocumentCommand{\parbox}{ooomm}
  {\implementacja@parbox
      {#1}{#2}{#3}{#4}{#5}}
```

Wyspecyfikowano polecenie o nazwie `\parbox` i pięciu argumentach: 3 pierwsze są opcjonalne (`o` — *optional*), dwa następne — wymagane (`m` — *mandatory*). Poleceniem `\implementacja@parbox` zamarkowano treść definicji. Istotnym elementem jest tu to, że do wartości wszystkich argumentów (również opcjonalnych) można się odwoływać tak samo. Argumenty opcjonalne, które nie wystąpiły w danym wywołaniu, przyjmują specjalną wartość `\NoValue`.

Argumenty opcjonalne z zasady w dokumencie są podawane w nawiasach kwadratowych. Zatem następujące przykłady pokazują niektóre poprawne (syntaktycznie) użycia nowo zdefiniowanego polecenia `\parbox`:

```
\parbox[t][3cm][s]{5cc}{Tekst}
\parbox[t]{5cc}{Tekst}
```

W drugim przypadku do implementacji polecenia `\parbox` zostaną przekazane argumenty:

```
{t}\NoValue\NoValue{5cc}{Tekst}
```

Oto wszystkie oznaczenia, które mogą znaleźć się w specyfikacji argumentów podanej w poleceniu `\DeclareDocumentCommand` i analogicznym poleceniu `\DeclareDocumentEnvironment`:

`m`	wymagany
`o`	opcjonalny
`O{dflt.}`	opcjonalny z wartością domyślną
`s`	opcjonalna gwiazdka
`c`	para w nawiasie, np. `(1,23)`
`C{{x}{y}}`	para z wartością domyślną

Jak powiedziano, brakujący argument opcjonalny bez wartości domyślnej przyjmuje wartość `\NoValue`. Przy przetwarzaniu argumentów fakt ten można wykryć przy pomocy makra `\IfNoValueTF`:

```
\IfNoValueTF{#1}
  {Brak parametru}
  {Parametr ma wartość #1}
```

Wynikiem użycia argumentu oznaczonego literą `s`, czyli opcjonalnej gwiazdki, jest przekazanie na odpowiedniej pozycji wartości `\BooleanTrue` (gwiazdka obecna) lub `\BooleanFalse` (brak gwiazdki). Do sprawdzania tych przypadków służy następujące makro `\IfBooleanTF`:

```
\IfBooleanTF{#2}
  {Jest gwiazdka}
  {Brak gwiazdki}
```

Opisany mechanizm wydaje się wystarczać do obsłużenia wszystkich typów argumentów występujących w poleceniach LATEXa. W opisie nie uwzględniono dopuszczalnych wartości argumentów — to inny aspekt sprawy. Na tym poziomie na przykład ujęta w nawiasy kwadratowe lista typu klucz-wartość jest po prostu argumentem opcjonalnym.

W pierwszym przykładzie treść polecenia została tylko zamarkowana. Spróbujmy się teraz bliżej przyjrzeć możliwościom ulepszeń w tej dziedzinie.

Szablony (*templates*)

Szablony typograficzne są jednym z podstawowych konceptów LaTeXa 3 i niewątpliwie najbardziej interesującą z dotychczas udostępnionych części.

Otóż poszczególne aspekty formatowania dokumentu opisane są za pomocą sparametryzowanych *szablonów typograficznych*.

Każdy szablon należy do jakiegoś *typu* specjalizowanego do danego zadania (zwykle formatowania jakiegoś konkretnego elementu dokumentu). Można więc spodziewać się istnienia typu justification, do którego należą szablony ustalające sposób justowania akapitów. Typ heading mógłby odpowiadać za formatowanie nagłówków (pod)rozdziałów. Szablony typu caption mogłyby służyć do formatowania podpisów pod rysunkami. I tak dalej.

Oczywiście może istnieć wiele szablonów danego typu, np. w różny sposób pozwalających formatować nagłówki. Jednak nawet pojedynczy szablon zwykle przewiduje modyfikowanie jego zachowania. Otóż definicja każdego szablonu określa zestaw *parametrów* sterujących formatowaniem.

W dokumencie nie używa się bezpośrednio szablonów, ale ich *instancji*. Instancje powstają poprzez ustalenie wartości parametrów szablonu.

Rozważmy przykład związany z typem justification. Szablony tego typu określają sposób justowania tekstu, mówiąc ściślej wywołanie instancji szablonu tego typu powoduje specyficzne ustalenie parametrów składu dla następujących dalej akapitów. Na przykład wywołanie predefiniowanej instancji o nazwie `raggedright` powoduje skład w chorągiewkę:

```
\UseInstance{justification}{raggedright}
```

Natomiast wywołanie instancji `center` powoduje centrowanie każdej z linii akapitu:

```
\UseInstance{justification}{center}
```

Obie powyższe instancje pochodzą od szablonu std typu justification. Oto definicja instancji center:

```
\DeclareInstance{justification}
                    {center}{std}
  { parindent  =0pt,
    rightskip  =\fill,
    leftskip   =\fill,
    parfillskip=0pt
  }
```

Zestaw parametrów szablonu std zawiera nazwy odpowiadające niskopoziomowym parametrom TeXa sterującym składem akapitów. Uzyskanie przy ich pomocy właściwego układu wymaga zrozumienia znaczenia tych parametrów. Można sobie jednak wyobrazić, że ktoś stworzy inny szablon tego typu, prezentujący możliwe sposoby formatowania w inny, bardziej przystępny sposób. Zauważmy też, że w przypadku tego typu szablonu predefiniowane instancje typu center i raggedright zaspokajają większość typowych potrzeb.

[Uwaga: typ justification, szablon std i omówione wyżej instancje są faktycznie zdefiniowane w [4]. Pojawiający się dalej typ heading wymyślam na motywach dyskusji toczących się na liście LATEX-L.]

Omówione powyżej szablony zachowują się jak polecenia bez argumentów. Parametrów szablonu nie da się nazwać argumentami, jako że są one takie same we wszystkich wywołaniach danej instancji. Tymczasem często pewne „parametry” szablonu powinny być ustalane z osobna dla każdego użycia danej jego instancji. Wyobraźmy sobie, że mamy instancję szablonu typu heading, której parametry dobrano właściwie do składu tytułów podrozdziałów (krój i stopień pisma, interlinia, sposób wypisywania numerów, itd.). Ta instancja będzie wielokrotnie wywoływana w dokumencie w celu formatowania nagłówków poszczególnych podrozdziałów. Zatem pewien jej „parametr” (treść tytułu) będzie inny przy każdym użyciu. Byłoby więc rozsądnie, gdyby ta instancja zachowywała się jak makro z (co najmniej) jednym argumentem.

Niech szablony typu heading będą wyposażone w 4 argumenty: flagę, wskazującą, czy nagłówek ma być numerowany; tekst nagłówka; wariant tekstu do żywej paginy i wariant tekstu do spisu treści.

Jeżeli teraz przyjmiemy, że dany jest jakiś szablon tego typu, i że utworzyliśmy instancję tego szablonu o nazwie mysection, można łatwo zdefiniować polecenie `\section` o takim znaczeniu, jak w LaTeXu 2_ε:

```
\DeclareDocumentCommand{\section}{som}
  {\UseInstance{heading}
    {mysection}{#1}{#3}{#2}{#2}}
```

Po poleceniu `\section` może stać gwiazdka, informacja o tym fakcie zostanie przekazana do szablonu jako pierwszy argument. Tekst nagłówka (wymagany) zostanie przekazany do szablonu jako drugi argument. Wreszcie opcjonalny argument polecenia posłuży zarówno jako wariant tekstu do paginy, jak i do spisu treści.

Jeżeli jednak ktoś chciałby mieć możliwość zadawania oddzielnie tekstu do żywej paginy i do spisu treści, nic prostszego, jak sformułować taką definicję:

```
\DeclareDocumentCommand{\section}{soom}
```

```
{\UseInstance{heading}
  {mysection}{#1}{#4}{#2}{#3}}
```

To polecenie ma dwa opcjonalne argumenty, które zostają przekazane odpowiednio do szablonu nagłówków.

W odróżnieniu od tworzenia instancji, które w zasadzie nie wymaga umiejętności pisania makr, definiowanie szablonu jest działalnością stricte programistyczną.

Składa się ono z kilku kroków. Jak powiedziano, szablon musi należeć do jakiegoś typu. Trzeba więc wybrać istniejący typ, lub zdefiniować nowy. Definicja typu określa jedynie jego nazwę i liczbę argumentów. Oczywiście jednak wspólne dla wszystkich szablonów danego typu powinno być również znaczenie poszczególnych argumentów i przeznaczenie całego szablonu.

Definicja szablonu zawiera deklarację parametrów i kod (makra TeXowe), który ma być wykonywany przy każdym wywołaniu szblonu (czy ściślej, jego instancji). Każdy z parametrów ma określony typ. W przykładzie powyżej występowały parametry typu wymiar, możliwe są jednak i inne: liczba, napis, polecenie z zadaną liczbą argumentów, wartość logiczna, instancja szablonu zadanego typu. W kodzie szablonu można odwoływać się do parametrów za pomocą ich nazw określonych w specyfikacji i do argumentów wywołania instancji przy pomocy zwykłych symboli argumentów: `#1`, …, `#9`.

Parametry liczbowe i wymiarowe szablonów zadaje się wyrażeniami w stylu standardowego pakietu calc — w postaci wyrażeń z operatorami arytmetycznymi `+`, `-`, `*` i `/`. W związku z tym ich wyliczanie jest bardziej czasochłonne niż zwykłych przypisań. Dlatego wartości parametrów wyliczane są w momencie definiowania instancji, czyli tylko raz (chyba, że jawnie zażądano wyliczania w momencie użycia instancji, co ma sens na przykład dla wymiarów określonych w jednostkach `em`).

Szablony w pewnym stopniu są podobne do obiektów z obiektowych języków programowania, jednak podobieństwo to jest dość powierzchowne. Co prawda mamy do czynienia z rodzajem „dziedziczenia" — polega ono na tym, że można utworzyć „uszczegółowienie" szablonu poprzez ustalenie części jego parametrów. Tworząc instancję takiego pochodnego szablonu wystarczy zadać pozostałe parametry. Drugie podobieństwo szablonów do obiektów polega na tym, że atrybut (parametr) obiektu (szablonu) też może być obiektem (instancją szablonu).

Jednak szablony nie mają same w sobie jakiegoś stanu, nie przechowują informacji (poza wartościami parametrów, ale te są stałe dla instancji). Są to obiekty „aktywne". Więc raczej są to specyficznie sparametryzowane funkcje (czy też procedury).

Projektując szablon trzeba podjąć szereg decyzji. Po pierwsze, wszystkie szablony tego samego typu muszą mieć te same argumenty. Dlatego zdefiniowanie typu szablonów wymaga dogłębnego przemyślenia zbioru argumentów właściwych dla danego zadania.

Oczywiście zbiór i dopuszczalne wartości parametrów mogą być radykalnie różne dla różnych szablonów, nawet jeśli należą one do tego samego typu. Należy jednak unikać szablonów, w których przy danej wartości jakiegoś parametru jakieś inne parametry tracą sens. W takim przypadku zapewne należy rozbić szablon na dwa, realizujące poszczególne warianty. Na przykład zapewne nie warto upychać w jednym szablonie makr generujących nagłówki wystawione (jak `\section`) i „wciągnięte" w akapit (jak `\paragraph`). W tym przypadku parametry określające światła w pionie wokół nagłówka wystawionego nie dotyczą nagłówka typu `\paragraph`.

Podobnie nie należy przeciążać szablonu zbyt wielkim zbiorem parametrów. Potrzeba wprowadzenia dużej liczby parametrów zwykle wskazuje, że lepiej byłoby jakąś część działań wydzielić w oddzielny szablon, który w całości stałby się parametrem zasadniczego.

Elastyczność systemu opartego na szablonach zależy od dobrego ich sparametryzowania i oczywiście nie jest zapewniona przez sam mechanizm. Można by nawet uzyskać podobne efekty programując w bardziej „klasyczny" sposób. Wydaje się jednak, że wprowadzenie jednolitego mechanizmu manipulowania „klockami" typograficznymi znakomicie uprzystępnia system dla projektanta nie będącego programistą.

Nowa procedura wyjścia (*output routine*)

Nowa wersja LaTeXowej procedury wyjścia, czyli części systemu odpowiedzialnej za budowanie stron, umożliwia wielołamowy skład dokumentów z rozmieszczaniem środowisk ruchomych (*floats*). Nowy algorytm dopuszcza dowolną liczbę łamów równej szerokości. Łamy na ostatniej kolumnie mają być wyrównywane (choć nie zostało to jeszcze zaimplementowane).

Zmiana liczby łamów może następować jedynie od nowej strony. (Może to ograniczenie też zostanie usunięte, gdy autorzy zaimplementują wyrównywanie łamów?)

Środowiska ruchome umieszczane są w prostokątnych obszarach rozciągających się przez dowolną liczbę łamów, na górze i na dole kolumny. Możliwe są więc np. środowiska zajmujące dwa łamy na dole trzyłamowej kolumny.

Marginalia obecnie zaimplementowano tylko w składzie jednołamowym, ale ma to zostać rozszerzone na przypadek ogólny. Jak rozumiemy, w takim przypadku marginalia ze wszystkich łamów mają pojawiać się wspólnie na zewnętrznym lub wewnętrznym marginesie kolumny.

Przypisy mogą być alternatywnie umieszczane: u dołu odpowiedniego łamu, gromadzone na dole ostatniego łamu na stronie lub (nowość) na marginesie.

W główkach i stopkach stron można odwoływać się do poleceń `\mark` dla każdego z łamów. W szczególności np. do pierwszego z pierwszego łamu i ostatniego z ostatniego. Ale możliwe są też bardziej skomplikowane układy, np. niezależna żywa pagina dla każdego z łamów z osobna.

Podpisy w środowiskach ruchomych (np. rysunkach i tabelach), uzyskiwane poleceniem `\caption`, są przechowywane w postaci listy leksemów aż do momentu umieszczenia wyniku składu środowiska na kolumnie. Pozwala to na decydowanie o sposobie umieszczenia i składu podpisu na tym etapie, a więc np. wypełnienie podpisem wolnej przestrzeni obok rysunku.

Nowy algorytm pozwala dopuścić umieszczanie środowisk ruchomych u góry łamu, w którym jest powołanie na dane środowisko (poprzedni algorytm uznawał, że to miejsce jest przed miejscem powołania). Planowane jest również rozszerzenie dopuszczające umieszczenie obiektu na poprzedzającej stronie należącej do tego samego rozwarcia (czyli na parzystej poprzedzającej nieparzystą).

Inna możliwość polega na ręcznym rozmieszczeniu środowisk ruchomych. Algorytm może zapisać swoje decyzje w pliku z rozszerzeniem `.fpl`. Plik ten można skopiować z rozszerzeniem `.fpc` i wyedytować zmieniając niektóre lub wszystkie decyzje programu. Miejsca umieszczenia obiektów wskazuje się, podając numery stron i nazwy obszarów. W następnym przebiegu LaTeXa algorytm rozmieści środowiska według specyfikacji z pliku.

Najciekawszą nowością są próby uzyskania składu na siatce (pakiet xo-grid), czyli zachowania registru, czyli padania na siebie linii tekstu w wielu łamach i na wielu kolumnach (np. na rozwarciu). W parametrach algorytmu zadaje się odległość poziomych linii siatki (zwykle równą interlinii pisma podstawowego). Powoduje to, że algorytm nadaje każdemu obszarowi na obiekty ruchome wysokość będącą wielokrotnością „oczka” siatki. Ponadto dostępne jest polecenie powodujące „przyciągnięcie” bieżacego wiersza dokumentu do najbliższej linii siatki. Wszystkie polecenia i środowiska generujące zmienne odstępy zostają wzbogacone o wywołanie tego polecenia tak, aby zwykłe akapity tekstu miały linie bazowe na siatce. Ten zabieg oczywiście powoduje inne niż zwykle zasady rozciągania zmiennych odstępów pionowych. Teraz może się zdarzyć, że dwa odstępy na tej samej stronie zostaną rozciągnięte w różnym stopniu.

Nowy algorytm, zdaniem autorów, działa już dosyć stabilnie. Nie ma jednak jeszcze ustabilizowanego interfejsu użytkownika.

Podsumowanie

Przedstawiony obraz nowego LaTeXa jest jeszcze bardzo fragmentaryczny. Niemniej jednak, pewne nowe możliwości są bardzo ekscytujące i warte uwagi. Budzi respekt ogrom pracy, jaką wyznaczyli sobie członkowie ekipy LaTeXa.

Na stronach WWW projektu można znaleźć więcej przykładów szablonów typograficznych (np. do składu inicjałów i przypisów). Przykłady pochodzące od osób spoza ekipy są prezentowane na liście dyskusyjnej LATEX-L i można je znaleźć w jej archiwach.

Bibliografia

[1] Frank Mittelbach, Chris Rowley *The LaTeX3 Project.* Dostępny w postaci pliku `ltx3info.tex` w dystrybucji LaTeXa.

[2] Frank Mittelbach, Chris Rowley, David Carlisle *The xparse package.* Ten i następne materiały są dostępne ze strony WWW Projektu.

[3] David Carlisle, Frank Mittelbach *The template package.*

[4] Frank Mittelbach *The h&j package.*

[5] Frank Mittelbach *Formatting documents with floats. A new algorithm for LaTeX 2ε*.*

[6] Frank Mittelbach *The galley package.*

[7] Frank Mittelbach *The xfootnote package.*

[8] Frank Mittelbach *The xinitials package.*

◇ Marcin Woliński
`wolinski@mimuw.edu.pl`