Transformacja danych z wykorzystaniem formatu B2MML jako element integracji systemów informatycznych przedsiębiorstwa

Jacek Pękala, Konrad Gadzina
Instytut Technologii Maszyn i Automatyzacji Produkcji, Politechnika Krakowska

Streszczenie: W omawianej pracy podjęto próbę przybliżenia problematyki wymiany informacji pomiędzy różnymi podmiotami struktury informatycznej przedsiębiorstwa, która stanowi istotę ich integracji. Artykuł poświęcony jest przede wszystkim procesowi transformacji danych wymienianych między systemami. W artykule przedstawiono koncepcję oraz budowę rozwiązania pozwalającego na transformację danych. Działanie opracowanego programu przetestowane zostało na przykładowych danych wygenerowanych przez system klasy ERP o nazwie IFS.

Słowa kluczowe: ERP, transformacja, XSLT, B2MML, ISA-95

Rozwój technologii, w tym także informatycznych, wspiera tworzenie nowych rozwiązań i usprawnienie już istniejących. We współczesnych strukturach informatycznych, których architektura jest często rozproszona, zastosowanie odpowiednich narzędzi komunikacyjnych jest niezbędne. Ze względu na specyfikę pracy systemów informatycznych przedsiębiorstwa, wymiana danych między nimi musi być pozbawiona przypadkowości, a metody komunikacji muszą być z góry określone. Charakter komunikacji jest w dużej mierze determinowany przez ośrodek przekazu danych, którego rolę w przypadku systemów informatycznych spełniać może oprogramowanie pośredniczące. Wymusza ono wykorzystanie odpowiednich standardów [1, 9]. Celem pracy jest przedstawienie koncepcji rozwiązania pozwalającego na zarządzanie przepływem informacji między różnymi podmiotami informatycznymi. Wykorzystany do tego został ogólnodostępny standard przekształcania danych za pomocą arkuszy stylów XSL oraz języka B2MML. Specyfikacja języka została zbudowana na bazie standardu ISA-95, przy zachowaniu zgodności ze specyfikacją języka XML.

1. Wprowadzenie

W dobie stałego rozwoju przemysłowego i związanego z nim postępu technologicznego przedsiębiorstwa produkcyjne zmuszone są stawiać czoła wielu różnym wyzwaniom. Pochodne postępu technologicznego: automatyzacja i informatyzacja (w szczególności warstwy produkcyjnej) mają sprostać tym zadaniom. Niestety, procesy te są trudne i skomplikowane, toteż producenci borykają się z wieloma problemami w trakcie ich realizacji. Jedną z wad informatyzacji produkcji jest często występująca redundantność danych. Podłączenie każdego nowego, zinformatyzowanego podmiotu generuje zazwyczaj kilka naprawdę potrzebnych wartości, pośród wielu takich, co do których przydatności można mieć wątpliwości. W rezultacie firmy toną w morzu danych, nadal potrzebując informacji. Winna temu jest m.in. niespójna architektura informatyczna. Korzystanie z wielu systemów czy aplikacji funkcjonujących w różnych warstwach przedsiębiorstwa, które przechowują dane w oparciu o natywną strukturę powoduje powstawanie błędów komunikacyjnych [3]. Problemy te stały się źródłem poszukiwań rozwiązań kontrolujących i zarządzających wymianą danych między systemami. W odpowiedzi na te oczekiwania tworzone są standardy, których ważnej roli regulującej funkcjonowanie systemów informatycznych (i nie tylko) w przedsiębiorstwie nie sposób nie docenić. Standardy stymulują rozwój technologiczny, ponieważ zarówno ich użytkownicy, jak i dostawcy technologii, której dziedzinę działania standard opisuje, dążą do tego, by dane rozwiązanie odpowiadało specyfikacjom w normach zawartym w [6].

1.1. Standard ISA-95 i język B2MML

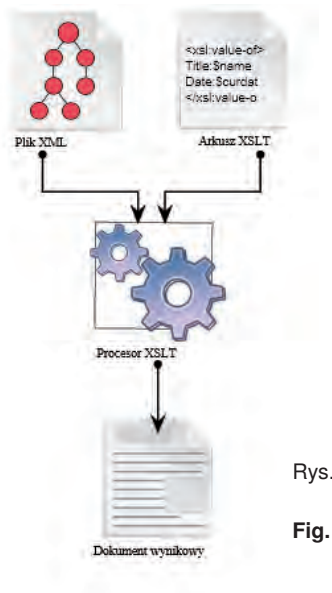
ANSI/ISA-95 Enterprise-Control System Integration to międzynarodowy standard zatwierdzony przez grupę producentów, dostawców systemów i ich opiniodawców. To opisana w kilku dokumentach, składająca się z pięciu części metodyka integracji. Standard nie przedstawia jednak technicznego rozwiązania problemu, za to określa fundamenty pod jego realizację. Część pierwsza standardu zawiera zbiór podstawowych pojęć oraz modeli przydatnych do integracji systemów produkcyjnych z systemami zarządczymi, które mają na celu poprawić komunikację między nimi. Część druga stanowi rozwinięcie części pierwszej. Zawiera dalsze szczegóły oraz przykłady wyjaśniające i ilustrujące pojęcia wprowadzone w części pierwszej, a także przygotowanie do budowy interfejsu międzysystemowego. Część trzecia definiuje modele czynności produkcyjnych, a także związany z nimi obieg informacji. Koncentruje się ona na opisie, analizie i ujednolicaniu czynności procesu produkcyjnego na poziomie systemów realizacji produkcji (MES), co w efekcie ma usprawnić wymianę i przetwarzanie informacji pochodzących z warstwy produkcyjnej. Część czwarta przybliża właściwości i zastosowanie modeli obiektowych omówionych w części trzeciej. Modele obiektowe i właściwości stanowią podłoże do zaprojektowania i wdrożenia interfejsu międzysystemowego. Część piąta formułuje modele transakcyjne przepływu informacji pomiędzy systemami. Modele transakcyjne dostarczają nie tylko opisów transakcji, ale także wyjaśniają stosowną procedurę ich dokonywania.

Jeśli przyjąć, że standard ISA-95 prezentuje teorię dotyczącą integracji systemów zarządzania (ERP) i systemów

odpowiedzialnych za realizację produkcji (MES), to za jego funkcjonalną implementację uznać należy język B2MML (*Business to Manufacturing Markup Language*). W pracy [2] autor podaje prostą definicję języka B2MML jako opartą na języku XML implementację standardu ANSI/ISA-95. Język B2MML zawiera zbiór schematów XML zapisanych w języku XSD, w których zawarte są zaczerpnięte z treści standardu ISA-95 definicje modeli obiektowych. Celem nadrzędnym stawianym językowi jest pośredniczenie w procesie integracji systemów poprzez konwersję danych i struktury wiadomości przesyłanych pomiędzy tymi systemami. Połączenie XML i ISA-95 przynosi wiele wymiernych korzyści w procesie transferu informacji. Poza otwartością, prostotą i niezależnością, schematy XML są łatwo adaptowalne do potrzeb wymiany danych, która wymaga zachowania jednolitości i spójności struktury danych. Znaczącą zaletą języków XML i B2MML jest czytelność informacji, wynikająca z przejrzystej struktury. Należy jednak pamiętać, że język B2MML nie jest standardem, a pewną interpretacją standardu, która w drobnych szczegółach może być inaczej rozumiana przez różnych dostawców systemów i użytkowników.

1.2. Transformacja XSL

XSLT jest akronimem oznaczającym *eXtensible Stylesheet Language Transformations*. To bazujacy na XML język przekształceń dokumentów XML. Pozwala na przetłumaczenie dokumentów z jednego formatu XML m.in. na dowolny inny format, zgodny ze składnią XML, w tym także na omawiany wcześniej B2MML. Dzięki dużej prostocie, łatwości implementacji i powszechnemu stosowaniu XML jako standardu dla zapisu informacji, XSLT jest uniwersalnym narzędziem znajdującym zastosowanie w wielu rodzajach oprogramowania. XSLT to należący do rodziny języków XSL – *eXtensible Stylesheet Lanuguage*, co w wolnym tłumaczeniu oznacza „rozszerzalny język stylów". Obok wspomnianego języka transformacji XSLT, do grupy tej należy język XPath, służący do nawigacji po dokumentach XML oraz język XSL-FO, odpowiedzialny za formatowanie dokumentów XML. Rozwijany przez konsorcjum W3C język XSLT doczekał się już wersji 2.0, choć powszechnie stosowana i rekomendowana jest wersja 1.0 [8].



Rys. 1. Schemat procesu transformacji [8]

Fig. 1. Transformation process schema [8]

Na rys. 1. przedstawiono poglądowy schemat procesu transformacji. Danymi wejściowymi dla procesora XSLT jest źródłowy dokument XML oraz arkusz stylów XSL, określający sposób transformacji dokumentu XML. Arkusz stylów składa się z szablonów. Każdy szablon opisuje, jak zamieniać pewien fragment dokumentu wejściowego na fragment dokumentu wyjściowego. Procesor zaś to aplikacja, która potrafi interpretować arkusz XSLT i na podstawie danych wejściowych wygenerować dokument wyjściowy. Wykonanie transformacji polega na wywołaniu szablonu pasującego do konkretnego elementu. W uzupełnieniu do powyższego opisu należy dodać, iż dokument XML nie jest plikiem „płaskim". Ma strukturę drzewa, dane w niej przechowywane są zhierarchizowane. XSLT stosuje szablony do elementów drzewa pasujących do zadanych wzorców, a zatem XSLT zawiera zbiór reguł opisujących przekształcenie jednego drzewa XML w nowe. Procesor w procesie transformacji tworzy nowe drzewo.

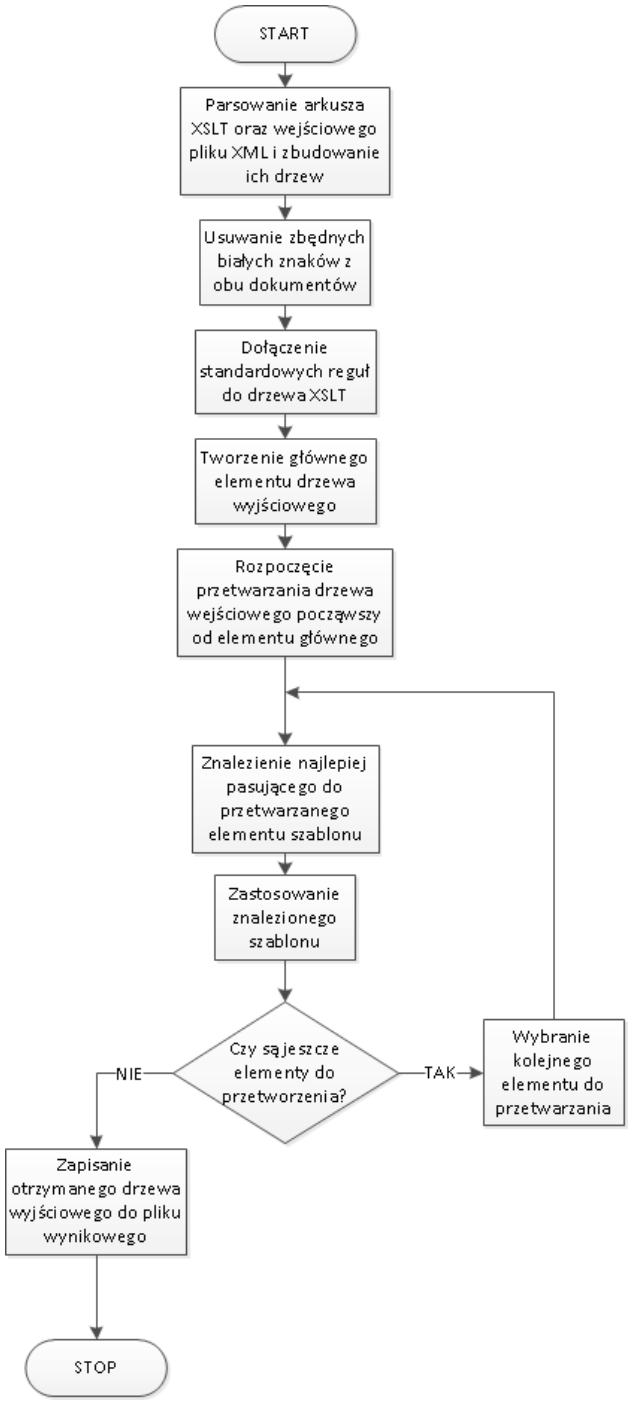
XSLT to język deklaratywny – opisuje co, a nie jak ma być przetworzone. Oznacza to, że treść szablonu jest wyrażeniem opisującym zawartość dokumentu wynikowego [7]. Nie istnieją imperatywne polecenia, które umożliwiają modyfikację stanu, wypisanie czegokolwiek na wyjście itp.

1.2.1. Algorytm transformacji [8]

Mechanizm działania procesora XSLT (rys. 2.) podczas procesu transformacji można podzielić na dwie zasadnicze części. W pierwszej dokument XML jest przygotowywany do przekształcenia, w drugiej wykonywana jest procedura transformacji. W kroku przygotowawczym dokonywane jest przede wszystkim parsowanie arkusza XSLT oraz źródłowego dokumentu XML. W wyniku parsowania utworzone zostają ich drzewa. Następnie z dokumentów usuwane są nadmiarowe białe znaki, a w dalszej kolejności do drzewa XSLT dołączane są standardowe reguły.

Po spreparowaniu dokumentów procesor przechodzi do zasadniczej części transformacji. Najpierw tworzony jest główny element drzewa wyjściowego, a następnie przetwarzane są elementy drzewa wejściowego, począwszy od elementu głównego, w efekcie czego zwracane jest drzewo wyjściowe. W ramach utworzonego drzewa wyjściowego każdy element drzewa wejściowego przetwarzany jest następująco:

- Znajdowany jest najlepiej pasujący szablon. Ze wszystkich szablonów pasujących do przetwarzanego elementu (każdy szablon nienazwany ma wzorzec) wybierany jest ten o najwyższym priorytecie (obliczonym na podstawie atrybutu *priority*, postaci wzorca oraz pozycji w dokumencie).
- Znaleziony szablon zostaje zastosowany. Elementy szablonu znajdujące się w przestrzeni nazw XSLT traktowane są jak instrukcje i odpowiednio interpretowane. Pozostała część jest kopiowana do drzewa wynikowego.
- Jeśli w szablonie umieszczona jest instrukcja *xsl:apply-templates*, procesor przechodzi w tym miejscu do rekurencyjnego przetwarzania listy elementów wskazanych atrybutem *select* lub (jeśli go brak) wszystkich potomków aktualnego elementu. Jeśli w szablonie brak jest instrukcji *xsl:apply-templates*, żadne z elementów aktualnego poddrzewa (dzieci i ich następniki) nie są



Rys. 2. Algorytm transformacji XSLT
Fig. 2. XSLT transformation algorithm

w tym miejscu dopasowywane (przetwarzane). Mogą jednak zostać przeznaczone do dopasowania (za pomocą instrukcji *xsl:apply-templates*) z innego poddrzewa. Jeśli w szablonie umieszczona jest instrukcja *xsl:apply-templates*, procesor przechodzi w tym miejscu do rekurencyjnego przetwarzania listy elementów wskazanych atrybutem *select* lub (jeśli go brak) wszystkich następników aktualnego elementu. Jeśli w szablonie brak jest instrukcji *xsl:apply-templates*, żadne z elementów aktualnego poddrzewa (dzieci i ich następniki) nie są w tym miejscu dopasowywane (przetwarzane). Mogą jednak zostać przeznaczone do dopasowania (za pomocą instrukcji *xsl:apply-templates*) z innego szablonu.

2. Przepływ danych

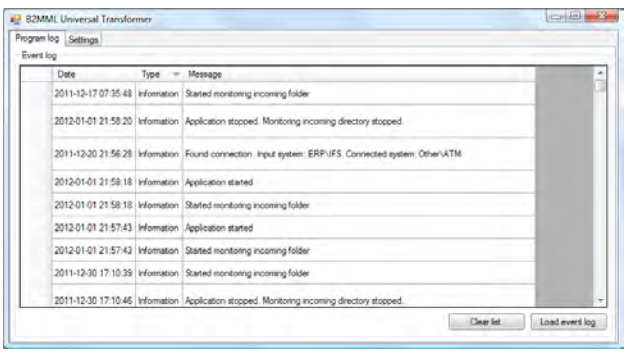
Całkowicie niezależna od problematyki związanej z przetwarzaniem danych pozostaje kwestia przesyłania danych: ich odbiór oraz dostarczenie w określone miejsce. Najistotniejsze w wyznaczaniu ścieżek przepływu danych wydaje się opanowanie zawartości informacji wymienianych, bowiem dla każdej pary systemów wymieniających dane mogą obowiązywać inne reguły [4]. Niosą one za sobą szereg wyzwań. Kluczem do zdefiniowania przepływów danych wydaje się być zrozumienie nie tylko treści przesyłanych wiadomości, ale także ich kontekstu, dlatego celowym wydaje się być utworzenie tzw. oprogramowania pośredniczącego (*middleware*). Warstwa oprogramowania pośredniczącego powinna stanowić spoiwo między systemami, które działać musi na zasadach podobnych do routera, którego zadaniem jest odpowiednia dystrybucja i zarządzanie przychodzącymi informacjami [5]. W omawianym przypadku zadanie to wydaje się być łatwiejsze, ponieważ dane przesyłane do oprogramowania pośredniczącego są odpowiednio ustrukturyzowane, jednoznacznie interpretowane przez procesor XSLT i przekształcane na format B2MML w celu ich dalszego przetworzenia i przesłania.

Podczas wysyłania przez system wejściowy wiadomości dedykowanej innemu systemowi, trafia ona najpierw do warstwy oprogramowania pośredniczącego, która tłumaczy metadane i strukturę wiadomości na język B2MML (*schema conversion*), a następnie przetwarza ją ponownie do postaci natywnej dla systemu docelowego. Transformacja metadanych i struktury informacji na język właściwy dla systemu wyjściowego wymaga nie tylko przekształcenia metadanych, ale także jej części semantycznej, jeśli zachodzi taka konieczność (*data conversion*). Sytuacja taka może mieć miejsce, np. przy zmianie formatu zapisu daty, kiedy jeden z systemów określa ją w porządku dzień-miesiąc-rok, a drugi miesiąc-dzień-rok. Dopiero po tym zabiegu i uzyskaniu gwarancji, że właściwy system go otrzyma, dokument może zostać przekazany dalej.

3. B2MML Universal Transformer (BUT)

3.1. Koncepcja

B2MML Universal Transformer (BUT) to opracowana przez autorów na potrzeby realizacji celu niniejszej pracy aplikacja, której głównym zadaniem jest usprawnienie procesu wymiany danych między systemami klasy ERP i MES, a także wszystkimi innymi, w przypadku których strukturę ich danych wyj-

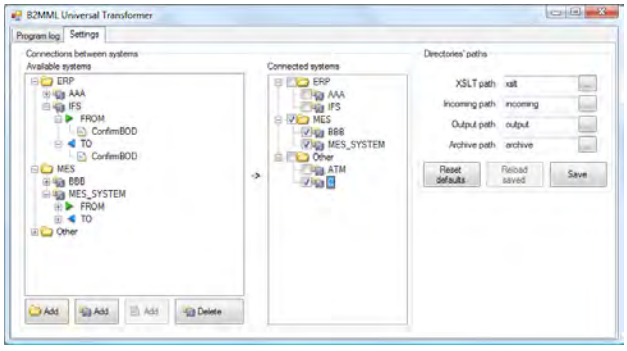


Rys. 3. Zakładka *Program log*
Fig. 3. *Program log* tab

ściowych bądź wejściowych można mapować na odpowiednią strukturę zgodną ze standardem B2MML. Aby zapewnić maksymalną uniwersalność i rozszerzalność, transformacje pomiędzy poszczególnymi formatami zostały oparte na arkuszach XSLT, które użytkownik może własnoręcznie opracować i dodać do programu, aby zwiększyć jego możliwości dotyczące przekształceń.

Podstawowym celem i istotą działania programu jest przekształcanie danych. Program jednak ma z założenia nie tylko służyć do ułatwienia konwersji danych, ale także do zautomatyzowania całego procesu wymiany informacji między poszczególnymi systemami. Założone funkcjonalności determinują budowę i sposób działania programu. Dlatego też po uruchomieniu aplikacji automatycznie wczytuje ona swoje ustawienia i rozpoczyna monitorowanie zawartości odpowiedniego katalogu, by móc natychmiast zareagować, gdy tylko pojawi się w nim plik wejściowy. Taki plik wejściowy jest następnie przekształcany na format wyjściowy i umieszczany w odpowiednim katalogu, do którego dostęp winien mieć system docelowy. Interfejs użytkownika w tym rozwiązaniu jest znikomy. Podejście do problemu zakłada ograniczenie udziału człowieka w procesie wymiany danych i konieczność jedynie minimalnej konfiguracji związanej z obsługą katalogów dedykowanych danym systemom i połączeń między nimi oraz zarządzaniem plikami przekształceniowymi.

W warstwie wizualnej program składa się z dwóch zakładek: *Program log* i *Settings*. Pierwsza zakładka (rys. 3.) zawiera listę wszystkich zdarzeń występujących podczas działania programu. Nie ma ona rozbudowanych możliwości konfiguracyjnych. Służy jedynie do prezentacji zdarzeń, których listę można sortować, wyczyścić, albo wczytać z rejestru systemu operacyjnego. Zakładka *Settings* (rys. 4.) pełni funkcje konfiguracyjne. Podzielona została na dwie sekcje. Sekcja *Connections between systems* zapewnia możliwość zarządzania klasami i należącymi do nich instancjami systemowymi, między którymi wymieniane są dane, definiowania połączeń między systemami oraz obsługi plików przekształceniowych. W sekcji *Directories' paths* możliwa jest konfiguracja ścieżek dostępu katalogów, w których przechowywane są pliki wejściowe, wyjściowe, przekształceniowe oraz archiwum.



Rys. 4. Zakładka *Settings*
Fig. 4. *Settings* tab

Kod źródłowy programu napisany został w języku C#, a do jego uruchomienia konieczna jest zainstalowana platforma .NET Framework 4.0.

3.2. Charakterystyka programu BUT

3.2.1. Struktura katalogów

Kluczowym elementem związanym z konfiguracją programu jest struktura katalogów, która musi zostać zachowana, aby program działał zgodnie z oczekiwaniami. Założono bowiem, iż poszczególne katalogi pełnić będą określone funkcje w procesie wymiany danych. Nadanie odpowiednich ról katalogom ma na celu zachowanie spójności w procesie przetwarzanych danych, co z kolei pozwala uniknąć błędów natury organizacyjnej. W zakładce z ustawieniami użytkownik może zdefiniować ścieżki dostępu do czterech katalogów, z których korzysta aplikacja:

- *XSLT* – folder, w którym umieszczane są arkusze przekształceń,
- *Incoming* – folder, który monitorowany jest przez aplikację w oczekiwaniu na pliki wejściowe,
- *Output* – folder, w którym umieszczane są pliki wyjściowe,
- *Archive* – folder, do którego przenoszone są niezmodyfikowane pliki wejściowe po dokonaniu już wstępnej transformacji oraz pliki przetworzone na format pośredni B2MML.

W każdym z tych katalogów została przyjęta pewna konwencja tworzenia podkatalogów, na której oparte jest działanie programu. Ideą jest utworzenie takiej struktury, która ułatwić ma zarządzanie większą liczbą obsługiwanych systemów, jeżeli użytkownik postanowi rozszerzyć podstawowy wachlarz możliwości transformacyjnych aplikacji. Najprostszym sposobem na dokonanie tego jest pogrupowanie systemów według ich typu, zamiast umieszczania ich bezpośrednio w każdym z czterech wymienionych wyżej folderów programu. Tak też zamiast ścieżki *[podkatalog programu]/[nazwa systemu]* przyjęte zostało stosowanie ścieżki typu *[podkatalog programu]/[typ systemu]/[nazwa systemu]*. Ułatwia to nie tylko zarządzanie samymi plikami związanymi z danym systemem, ale i definiowanie połączeń między różnymi systemami.

3.2.2. Struktura katalogu arkuszy transformacji

Poza wymienioną uprzednio wymaganą strukturą *[xslt]/[typ systemu]/[nazwa systemu],* w przypadku arkuszy transformacji przyjęta została dodatkowa konwencja segregacji plików, na której bazuje obsługa transformacji. Mianowicie, każdy folder odpowiadający pewnemu systemowi musi mieć dwa podkatalogi: *FROM* oraz *TO*. Pliki XSLT, które służą do przekształcenia danych z tego systemu do formatu B2MML muszą zostać umieszczone w katalogu *[xslt]/[typ systemu]/[nazwa systemu]/FROM*. Arkusze XSLT odpowiedzialne za przekształcenia z formatu B2MML do formatu natywnego danego systemu muszą natomiast znajdować się w folderze *[xslt]/[typ systemu]/[nazwa systemu]/TO*.

Z poziomu programu, w zakładce ustawień użytkownik może dodawać i usuwać zarówno grupy, jak i konkretne systemy oraz importować pliki przekształceń. Jest to możliwe oczywiście także spoza aplikacji – by dodać grupę wystarczy w folderze *[xslt]* utworzyć folder o nazwie takiej, jaką chcemy dodać grupę; w przypadku dodawania systemu do danej grupy postępowanie jest analogiczne. Pierwotne ustalenia zakładały udostępnienie jedynie trzech grup (*ERP, MES, Other*), jednak ujednolicenie grup dostępnych w programie

ze strukturą katalogów pozwoliło na umożliwienie rozszerzalności także w tej sferze.

Aplikacja w czasie swojego działania nasłuchuje katalog *[xslt]* i w przypadku stworzenia w niej w jakikolwiek sposób nowej grupy, czy systemu tworzy ona automatycznie odpowiednią strukturę w folderach *[incoming]*, *[output]* oraz *[archive]*. To samo tyczy się dodawania klasy systemów czy systemu z poziomu samego programu.

3.2.3. Połączenia między systemami

W zakładce ustawień aplikacji, poza sekcją dotyczącą definiowania ścieżek do katalogów programowych, znajdują się także dwa drzewa służące do tworzenia połączeń transformacyjnych między konkretnymi systemami.

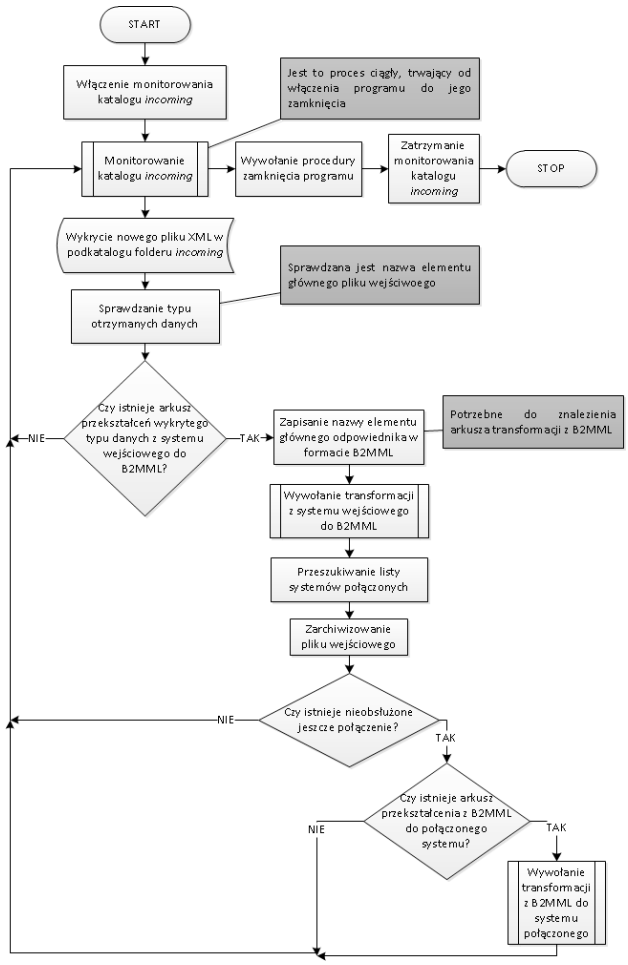
Rozwiązanie to zostało zaimplementowane, aby program, po dokonaniu przekształcenia danych z systemu wejściowego do formatu B2MML mógł dokonywać tylko koniecznych operacji transformacji z B2MML, do systemu wyjściowego, czyli takich, które wynikają z narzuconych więzów między systemami. Plik z listą połączeń danego systemu znajduje się pod ścieżką *[xslt]/[typ]/[system wej*ściowy]/connections.txt. Zawartość pliku jest aktualizowana natychmiast po dokonaniu zmian na liście połączeń międzysystemowych, czyli po zaznaczeniu lub odznaczeniu na liście systemu połączonego.

3.2.4. Przebieg transformacji

Od momentu uruchomienia program monitoruje katalog *[incoming]* i wszystkie jego podkatalogi w celu natychmiastowego wychwycenia pliku z danymi z któregoś z obsługiwanych systemów. Po znalezieniu nowego pliku sprawdzany jest typ przechowywanych w nim danych. Jest to realizowane przez sprawdzenie nazwy elementu głównego w wejściowym pliku XML. Gdy ta nazwa jest już znana, przeszukiwany jest katalog *[xslt]/[typ]/[system wejściowy]/FROM* w celu znalezienia arkusza przekształceń, który jest w stanie obsłużyć dane wejściowe tego typu. Jeżeli plik pozwalający na transformację do formatu B2MML zostanie znaleziony, to zostaje uruchomiony proces przekształcenia danych wejściowych na jego podstawie. Zapisywana jest nazwa elementu głównego pliku pośredniego w formacie B2MML, ponieważ będzie ona konieczna w dalszym etapie. Po pomyślnej transformacji plik wejściowy zostaje zarchiwizowany, czyli przeniesiony do katalogu *[archive]/[typ]/[system wejściowy],* z dopisaną do nazwy aktualną datą i godziną, aby uniknąć problemu nadpisania.

Drugim etapem jest dokonanie transformacji danych z B2MML do formatu systemu wyjściowego. Ponieważ do wykonania tej operacji wymagana jest znajomość nazwy systemu docelowego, faza ta rozpoczyna się wczytaniem listy połączeń systemu wejściowego. Dla każdego systemu, który się na tej liście znajduje, sprawdzane jest, czy w katalogu *[xslt]/[typ]/[system połączony]/TO*, znajduje się odpowiedni dla danej konwersji arkusz transformacji. Sprawdzanie to jest dokonywane analogicznie do poprzedniego etapu – poszukiwany jest plik XSLT, który obsługuje transformację z formatu danych o nazwie elementu głównego, takiej samej, jak zapisana uprzednio nazwa elementu głównego pośredniego pliku B2MML. Jeżeli transformacja dla danego systemu przebiegnie bezbłędnie, to w katalogu *[output]/[typ]/[system połączony]* zostanie umieszczony plik wynikowy w odpowiednim formacie.

Każda faza procesu transformacji zapisuje do logu odpowiednie informacje, wraz ze ścieżkami plików, które są wykorzystywane. Po sprawdzeniu wszystkich połączeń zapisywana jest liczba dostępnych połączeń wraz z liczbą pomyślnie wykonanych transformacji. Algorytm przebiegu transformacji został przedstawiony na rys. 5. Należy nadmienić, iż w programie nie utworzono nowego procesora XSLT, a wykorzystano funkcje wbudowanego w platformę *.NET Framework* procesora XSLT.



Rys. 5. Algorytm monitorowania i przetwarzania danych w aplikacji BUT

Fig. 5. Monitoring and data transformation algorithm applied in BUT application

3.2.5. Logowanie zdarzeń

Zgodnie z założeniami koncepcyjnymi, ze względu na konieczność monitorowania każdej pojedynczej operacji mającej miejsce w trakcie działania programu, została zaimplementowana funkcjonalność logowania zachowania programu do logu systemowego oraz pliku tekstowego. W oknie aplikacji wyświetlane są zdarzenia zarejestrowane w danej sesji uruchomieniowej programu, jednak naciśnięcie przycisku *Load event log* pozwala załadować wszystko, co system operacyjny zarejestrował w związku z działaniem programu. Logowanie do pliku pozwala na łatwe przesłanie informacji o zapisanych akcjach w razie problemów. Plik jednak można łatwo przypadkowo usunąć, dlatego też poza logiem plikowym została wykorzystana odpowiednia funkcjonalność systemu operacyjnego.

Operator programu może przeglądać zalogowane zdarzenia w głównej zakładce, gdzie domyślnie znajduje się lista zdarzeń z bieżącej sesji uruchomieniowej. Możliwe jest sortowanie tej listy, dzięki czemu można łatwo sprawdzić chronologiczne następstwo zdarzeń, bądź posortować je na podstawie typu i wyświetlić obok siebie wszystkie ewentualne błędy.

3.2.6. Komplementarność I kompatybilność

W ramach prezentowanego rozwiązania dokonano analizy działania programu pod kątem obsługi danych generowanych przez system IFS należący do klasy systemów ERP oraz system Wonderware MES należący do klasy systemów MES. Dokonana została również analiza dostępności narzędzi potrzebnych do wymiany danych po stronie obydwu systemów. System IFS zapewnia agregację i obsługę danych potrzebnych do wymiany z systemami klasy MES. Ponadto, system IFS posiada moduł *Data Migration* odpowiedzialny za migrację danych. Rozwiązanie to zapewnia obsługę wiadomości przychodzących i wychodzących. Podobnie jest z systemem Wonderware MES, który odpowiada przede wszystkim za gromadzenie i przetwarzanie danych generowanych na poziomie produkcyjnym, a przy tym ma wbudowaną aplikację o nazwie *Supply Chain Connector*, dzięki której możliwy jest import i eksport danych. Aplikacja BUT wydaje się być zatem narzędziem kompatybilnym i komplementarnym w kontekście integracji tych systemów, które spełniają określone warunki, podobnie jak wyżej wymienione przykłady.

4. Podsumowanie

Prezentowane rozwiązanie zostało przetestowane na przykładowo utworzonych arkuszach XSLT oraz dokumentach XML wygenerowanych przez system IFS. Wyniki działania programu są na obecnym etapie jego rozwoju zgodne z oczekiwaniami. Uniwersalność, elastyczność i operatywność w definiowaniu nowych powiązań aplikacji BUT w pełni wpisuje się w teorię związaną z wymianą danych między systemami klasy ERP i MES, opisaną w standardzie ISA-95 jako rozwiązanie stanowiące warstwę pośredniczącą (*middleware*) między systemami. Pozwala ona związać dane pochodzące z różnych środowisk ze sobą i, wraz z modelami transakcyjnymi dotyczącymi m.in. różnych strategii biznesowych, modeli produkcyjnych itd., stanowi punkt wyjścia dla nadrzędnego celu, jakim jest integracja systemów ERP i MES. To właśnie ta dumnie brzmiąca integracja, a nie jedynie wymieszanie dwóch oddzielnych systemów jest kluczem do prężnej i zyskowniejszej przedsiębiorczości.

Bibliografia

1. Chwajoł G.” *The Evolution of Middleware used in distributed manufacturing control systems*. III Ukraińsko-Polska Konferencja Młodych Naukowców „Mechanika i informatyka”, 2005, 18–20.
2. Gould L.: *B2MML Explained*. Automotive Design & Production, 2007, [www.autofieldguide.com/articles/b2mml-explained].
1. Pękala J.: *Wykorzystanie języka B2MML jako narzędzia integracji informacji w przedsiębiorstwie produkcyjnym w ramach standardu ISA-95*. Research Reports Project CII-SK-0030-03-0708, 2008, 304–309.
2. Pękala J.: *Wyznaczanie ścieżek przepływu informacji między systemami w przedsiębiorstwie produkcyjnym z wykorzystaniem języka B2MML*, Zeszyty Naukowe Politechniki Poznańskiej, seria Budowa Maszyn i Zarządzanie Produkcją. Wydawnictwo Politechniki Poznańskiej, Poznań, Nr 2 (16), 2011, 187–197.
3. Scholten B.: *The road to integration: A guide to applying the ISA-95 Standards in Manufacturing*. ISA – Instrumentation, Systems and Automation Society, 2007.
4. Skura K., Smalec Z.: *Integracja systemów informatycznych w automatyzacji procesów produkcyjnych*. „Pomiary Automatyka Robotyka” 7–8/2005, 6–11.
3. Wilson G.: *Przetwarzanie danych dla programistów*. Helion, 2006.
4. *XSL Transformations*. [pl.wikipedia.org/wiki/XSL_Transformations].
5. Zając J., Chwajoł G.: *Integracja informacji w systemach sterowania wytwarzaniem*. AUTOMATION 2005, 96–105.

Data transformation from native structure to B2MML format as part of systems integration

Abstract: This paper is an attempt to bring closer the issues of data exchange between different subjects of enterprises IT structure which is the essence of their integration. The paper focuses primarily on transformation of data swapped between systems. The article presents a design concept of solution for data transformation and its structure. Its operability was tested on sample data generated by the IFS – exemplary ERP system.

Keywords: ERP, transformation, XSLT, B2MML, ISA-95

mgr inż. Jacek Pękala
Asystent naukowo-dydaktyczny w Instytucie Technologii Maszyn i Automatyzacji Produkcji Politechniki Krakowskiej.
e-mail: pekala@m6.mech.pk.edu.pl

Konrad Gadzina
Student Informatyki na Wydziale Mechanicznym Politechniki Krakowskiej
e-mail: fenix.b3@gmail.com