STUDIA INFORMATICA 2011
Volume 32 Number 2B (97)

Katarzyna HARĘŻLAK, Łukasz KULISZ
Politechnika Śląska, Instytut Informatyki

# MODEL DANYCH DLA SYSTEMU AGENTOWEGO WYKORZYSTUJĄCEGO REPLIKACJĘ DANYCH

**Streszczenie**. W artykule poruszony został problem replikacji danych z wykorzystaniem systemów agentowych. Połączenie mechanizmów tych dwóch dziedzin informatyki może zaowocować powstaniem rozwiązań, które spełnią rosnące wymagania użytkowników względem funkcjonalności narzędzi informatycznych. Celem niniejszego artykułu jest prezentacja modeli danych, które mogą stanowić podstawę pracy systemów.

**Słowa kluczowe**: systemy agentowe, model danych, replikacja danych

# DATA MODEL FOR REPLICATED DATA MANAGEMENT AGENT SYSTEM

**Summary**. The problem of replicated data management with usage of agent systems was discussed in the paper. Integration of mechanisms of this two branches may result in development of new solutions which meet growing requirements regarding functionality of the computer tools. The aim of the paper is to present data models, which can serve as a base for a system using replicated data managed by independent agents.

**Keywords**: agent system, data model, data replication

## 1. Wstęp

W dzisiejszych czasach wymagania dotyczące funkcjonalności narzędzi informatycznych często przewyższają możliwości, jakie oferują tradycyjne, dwu- lub trójwarstwowe architektury systemów komputerowych. Jedną z dróg, która pozwala na zaspokojenie rosnących wymagań użytkowników jest zastosowanie replikacji danych wraz z funkcjonalnością systemów agentowych. Replikacja danych [1, 2, 3] stosowana jest obecnie we wszystkich systemach

przetwarzających ogromne ilości danych, takich jak systemy bankowe, lotnicze czy wyszukiwarki internetowe. Dzięki niej możliwe jest zwiększenie niezawodności i wydajności narzędzi informatycznych.

Systemy wieloagentowe rozwijane są od około 1980 roku [4, 5, 6]. W latach dziewięćdziesiątych ubiegłego wieku dostrzeżono, że mogą one stanowić model, który pozwoli wykorzystać możliwości ogromnych systemów rozproszonych, takich jak na przykład Internet. Dlatego połączenie mechanizmów replikacji i systemów wieloagentowych może pozwolić na budowę rozwiązań cechujących się rozbudowaną funkcjonalnością oraz na zbadanie możliwości i ewentualnych korzyści, jakie niesie ze sobą połączenie tych dwóch dziedzin informatyki [7, 8].

W systemach o takiej architekturze odpowiedzialność za utrzymanie spójności rozproszonych danych będzie ponosić agent. Z tego powodu należy udostępnić mu odpowiednie struktury danych, które pozwolą na zarządzanie zarówno danymi należącymi do określonej dziedziny przedmiotowej, jak i informacjami stanowiącymi *metadane* procesu replikacji. Celem niniejszego artykułu jest prezentacja modeli danych [9], które mogą stanowić podstawę pracy takich systemów, współpracujących zarówno z relacyjnymi, jak i obiektowymi bazami danych.

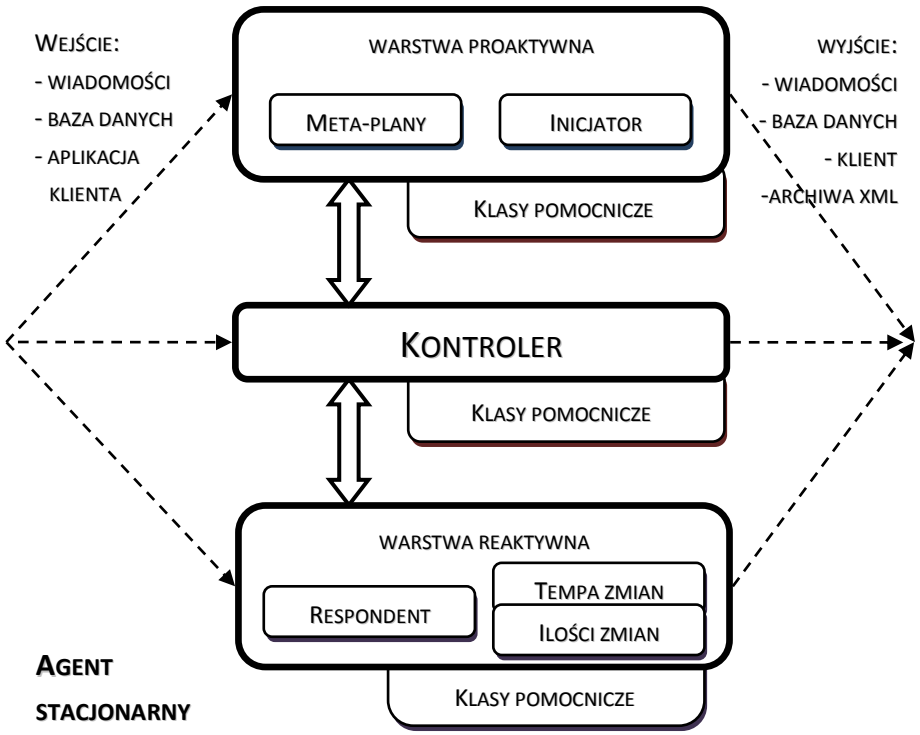
## 2. Opis systemu

Każdy system wieloagentowy jest systemem rozproszonym, którego działanie opiera się na współpracy agentów funkcjonujących w różnych węzłach sieci komputerowej, w zakresie realizacji określonego celu. W przypadku prezentowanych badań celem tym jest zarządzanie danymi uzyskanymi w procesie replikacji. Do zadań stawianych przed agentami pracującymi w takim środowisku zaliczyć należy: synchronizację danych, ich odtwarzanie w węzłach, które uległy awarii, oraz nadzorowanie i raportowanie stanu sieci. W badaniach szczególną uwagę zwrócono na problem rozwiązywania konfliktów wynikających ze zmian tych samych rekordów w różnych węzłach systemu. Znalazło to odzwierciedlenie zarówno w funkcjonalności, w jaką wyposażeni zostali agenci, jak i w modelu danych, który stanowi postawę ich pracy.

Projektując system agentowy, kierowano się sugestiami zawartymi w [4], które, w przypadku budowania systemu agentowego, zalecają rezygnację z tworzenia nowych architektur na korzyść istniejących rozwiązań. Z tego powodu na kształt prezentowanego w artykule rozwiązania wpływ miały wybrane do jego realizacji środowisko pracy oraz architektura agenta.

Na podstawie przeprowadzonej w obu dziedzinach analizy, do badań wykorzystano środowisko JADE [5] oraz typ agenta charakteryzującego się hybrydową architekturą z poziomym układem warstw [4]. Liczba warstw agenta zależna jest od zadań, które ma realizować w systemie. W pracy zaproponowano trzy rodzaje agentów: stacjonarny, mobilny oraz synchronizujący.

- Agent **stacjonarny** uruchomiany jest w każdym węźle zawierającym zestaw replikowanych danych. Jego głównym celem jest synchronizacja danych znajdujących się w lokalnej bazie danych, z rekordami istniejącymi w innych węzłach systemu. W architekturze tego agenta wprowadzono trzy warstwy – *reaktywną*, *proaktywną* i *kontrolera*. Warstwa **reaktywna** odpowiada za działania, które są bezpośrednią reakcją na zmianę środowiska, dla przykładu zaproszenie od jednego z agentów do rozpoczęcia synchronizacji. Warstwa **proaktywna** gromadzi zachowania w postaci *metaplanów*, które powodują, że agent przeprowadza działania z własnej inicjatywy. Przykładem może być zainicjowanie synchronizacji z innymi agentami. Warstwa **kontrolera** dba o to, aby wspominanie akcje nie były wykonywane jednocześnie. Ogólną architekturę agenta stacjonarnego zaprezentowano na rysunku 1.




Rys. 1. Ogólna architektura agenta stacjonarnego
Fig. 1. General architecture of the stationary agent

- Agent **mobilny** jest programem wykorzystującym możliwość przenoszenia do innych węzłów zarówno swojego stanu, jak i kodu, niezbędnego do replikacji i synchronizacji danych. W jego strukturze zaimplementowana została tylko warstwa **proaktywna**.

- Agent **synchronizujący** jest to element systemu zapewniający koordynację pracy agentów działających w jednym węźle systemu. W przeciwieństwie do agenta mobilnego, agent synchronizujący posiada jedynie cechy **reaktywne**, które są zgromadzone w jednej warstwie. Jego działanie jest zawsze bezpośrednią reakcją na otrzymane żądania.

## 3. Model danych systemu agentowego

Podstawą pracy systemu agentowego jest baza danych. Znaczący wpływ na jej strukturę ma dziedzina przedmiotowa, w której ma funkcjonować system. W badaniach podjęto próbę zdefiniowania uogólnionego modelu danych, który mógłby gromadzić informacje opisujące wiele takich dziedzin.

Zaproponowane struktury danych (rys. 2) mogą zostać zaimplementowane w dowolnym z dostępnych na rynku serwerze bazy danych. Większość z nich wyposażona jest we własne mechanizmy replikacji danych, na różnym poziomie zaawansowania [10, 11, 12], jednak nie zawsze są one w stanie sprostać wymaganiom konkretnego systemu. Powoduje to konieczność opracowania własnych rozwiązań, co często ma miejsce w przypadku rozwiązywania konfliktów aktualizacji danych. Z tego też powodu w omawianym systemie obsługę replikacji danych powierzono jego agentom, którzy cechują się odpowiednią inteligencją i autonomią w podejmowaniu decyzji, w trakcie przeprowadzenia tego procesu. Taka funkcjonalność szczególnie zyskuje na znaczeniu, jeśli weźmie się pod uwagę bazy danych dedykowane urządzeniom przenośnym [3]. Jej wprowadzenie wymagało jednak uzupełnienia modelu danych również o encje odpowiedzialne za utrzymywanie środowiska replikacyjnego.

### 3.1. Dziedzina przedmiotowa

Podstawowym zadaniem bazy danych w odniesieniu do dziedziny przedmiotowej jest gromadzenie informacji o opisujących ją obiektach. W tym celu do modelu danych wprowadzono encje *Object, ObjectType* oraz *Operation*. W zaproponowanym modelu zakłada się, że zarówno obiekty, jak i typy obiektów posiadają atrybut lub grupę atrybutów będących *naturalnym* kluczem encji. W najprostszym przypadku jest to atrybut *Name*.

Encja *Operation* definiuje dane o fakcie przeprowadzenia operacji pewnego typu. Wraz z encjami *OperationType*, *OperationProperty* i *OperationPropertyValue* pozwala modelować dowolny rodzaj działań na obiektach.

Operacje mogą być definiowane w zależności od dziedziny problemu, który rozwiązuje projektowana aplikacja. To, jakie znaczenie dla użytkownika mają operacje, nie gra roli w procesie replikacji. Typy operacji w konkretnej aplikacji reprezentowane są przez encję

*OperationType*. Oprócz atrybutu służącego do identyfikacji, posiada ona także pole nazwy, które pozwala przedstawiać operacje w formie zrozumiałej dla użytkownika.

Z każdym typem operacji związany jest zbiór właściwości, które są reprezentowane przez encję *OperationProperty*. Jej atrybut *Name* jest czytelną dla użytkownika nazwą właściwości, a *DefaultValue* definiuje domyślną wartość właściwości operacji.

Warto zwrócić uwagę na to, że *DefaultValue* nie ma określonego typu. Wynika to z faktu, że każda właściwość może być innego typu. Użycie takiego rozwiązania wymaga odnotowania w bazie danych także informacji o typie właściwości. Taką funkcję pełni encja *PropertyType*, która oprócz identyfikatora posiada także atrybut *Name*, reprezentujący nazwę typu danych. Nowoczesne serwery bazodanowe pozwalają na definiowanie kolumn, które mogą jednocześnie przechowywać prawie każdy typ danych, zgody ze standardem SQL. Przykładem może być typ *SQL_VARIANT*, który jest dostępny w SZBD SQL Server firmy Microsoft [10].

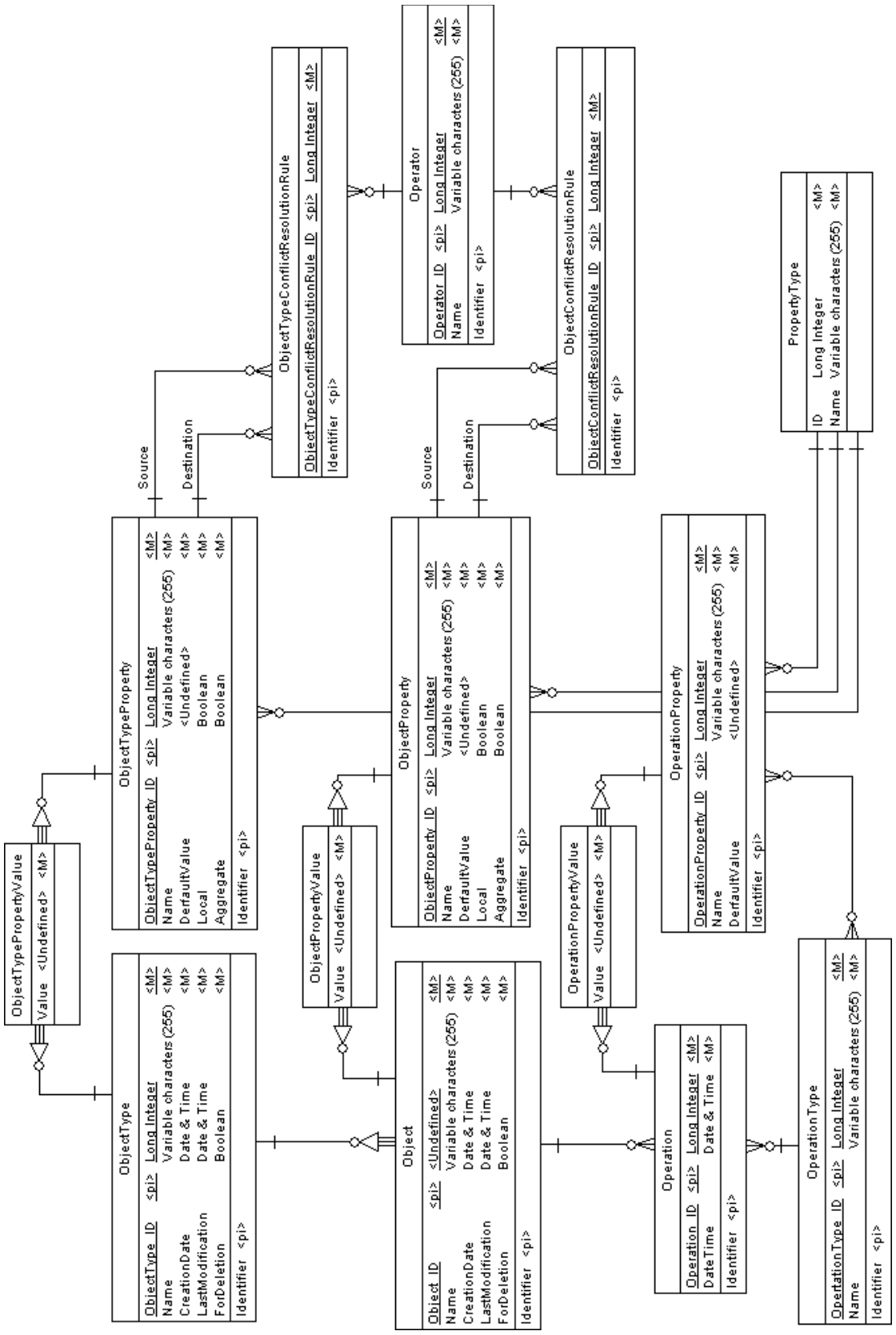
Encje *ObjectTypeProperty* i *ObjectProperty* są odpowiednikami *OperationProperty*, ale związanymi z encjami odpowiednio *ObjectType* i *Object*. Reprezentują właściwości, jakie posiadają każdy typ obiektu i obiekt. Wartości właściwości obiektów i ich kategorie stanowią przedmiot replikacji. Z tego powodu wprowadzony został dodatkowy atrybut – *Local*. Jest to flaga, która oznacza, że wartości danej właściwości nie powinny być replikowane. Korzystając z tej flagi, można wprowadzać właściwości obiektów i typów obiektów, które mają znaczenie tylko w lokalnej bazie danych, np. wartości wyliczane w procesie replikacji danych. Pobieranie tej właściwości w czasie synchronizacji nie ma sensu, gdyż będzie ona nadpisana przez niezależnie obliczoną wartość.

Flaga *Aggregate* oznacza, że wartość właściwości ma znaczenie po replikacji, tylko jeżeli jest ona przeprowadzana ze wszystkimi węzłami systemu.

### 3.2. Organizacja replikacji danych

Uniezależnienie procesu replikacji danych od konkretnego serwera baz danych pociągnęło za sobą umieszczenie w opracowanym modelu elementów, które pozwolą agentom na samodzielne podejmowanie decyzji w czasie jego realizacji. Decyzje te obejmą między innymi określenie czasu rozpoczęcia tego procesu, zbiorów replikowanych rekordów oraz sposobu rozwiązywania zaistniałych sytuacji konfliktowych.

Przy założeniu, że dane reprezentowane przez encję *Operation* są istotne tylko z punktu widzenia węzła, w którym przeprowadzane są operacje, replikacji podlegać będą dane zdefiniowane przez encje *Object* i *ObjectType*. Z tego powodu wprowadzono do nich dodatkowe atrybuty.




Rys. 2. Diagram przedstawiający uogólniony model danych
Fig. 2. Diagram of the general data model

- *CreationDate* – data i czas utworzenia obiektu lub jego typu. Wartość ta może się zmienić tylko wtedy, kiedy identyfikator krotki zostanie zmieniony w wyniku replikacji, co pozwala na ustalenie zbioru rekordów dodanych po ostatniej synchronizacji stanu baz danych.
- *LastModification* – data i czas ostatniej aktualizacji krotki, która nie może być mniejsza od *CreationDate*. Jest ona również wykorzystywana do określenia zbioru danych zaktualizowanych po ostatniej synchronizacji.
- *ForDeletion* – flaga oznaczająca, że krotka, z punktu widzenia użytkownika, została usunięta. W czasie replikacji stan flagi jest aktualizowany w węzłach, z którymi przeprowadzana jest synchronizacja. Dzięki temu użytkownicy wszystkich baz danych zauważają usunięcie obiektu lub jego typu. Rekord ten jest trwale usuwany, kiedy wszystkie węzły mają ten sam stan flagi, co oznacza, że synchronizacja danych objęła wszystkie lokalne bazy danych.

Bardzo ważnym elementem modelu, z punktu wiedzenia zadań synchronizacji danych, są encje *ObjectConflictResolutionRule* i *ObjectTypeConflictResolutionRul*e. Wraz z encją *Operator* opisują one sposób rozwiązywania konfliktów w wartościach poszczególnych właściwości. Obie posiadają po trzy zależności. *Source* wskazuje właściwość, która ma posłużyć jako źródło danych do rozwiązywania konfliktów. *Destination* definiuje docelowe miejsce dla wyników operacji. *Operator* to encja, która określa, jakie działania mają być przeprowadzone, by rozwiązać konflikt.

Ponadto, do modelu wprowadzono trzy encje pomocnicze, które, ze względu na jego czytelność, pominięto na diagramie. Atrybuty pierwszej z nich, *SpeedOfChanges*, reprezentują tempa zmian obiektów o określonym typie, w danym węźle systemu. Atrybuty te opisują różnorodne statystyki, które mają posłużyć agentom w wyborze momentu uruchomienia kolejnej synchronizacji pomiędzy wybranymi węzłami. Obejmują one średnią liczbę nowych *(InsertedPerSec*) lub zaktualizowanych (*UpdatedPerSec*) obiektów określonego typu na sekundę. Średnie te obliczane są między ostatnią synchronizacją kategorii obiektu w węźle a aktualnym czasem rzeczywistym. Ponadto, atrybut *Date* definiuje datę i czas ostatniej aktualizacji statystyk. Jeżeli nie są one zbyt odległe od aktualnego rzeczywistego czasu, można twierdzić, że wykorzystanie statystyk pozwala na w miarę dokładne prognozowanie aktualnej liczby zmian obiektów danej kategorii.

Zadaniem kolejnej z wymienionych encji, *SynchronizationLog*, jest opis jednej synchronizacji określonego typu obiektu z określonym zbiorem węzłów. Jej atrybuty opisują dane wykorzystywane do określenia zbioru obiektów, które zmieniły się w czasie między ostatnią synchronizacją a aktualnym czasem rzeczywistym. Są to:

- *SyncDateTime* – data i czas ostatniej replikacji.
- *Successful* – flaga, która określa, czy replikacja została zakończona sukcesem.

Ostatnią encją, która została dodana do modelu danych, a której nie umieszczono na diagramie, jest encja *Settings*. Reprezentuje ona ustawienia agenta i klienta działających w jednym węźle. Encja opisuje tylko jedną krotkę, która zawierać będzie: ścieżkę do folderu, w którym będą tworzone archiwa XML, zawierające usunięte, zsynchronizowane obiekty wraz z lokalnymi operacjami; rodzaj algorytmu rozwiązywania kolizji; okres (w sekundach) uruchamiania przez agenta procesu podejmowania decyzji o kolejnej replikacji; czas, po którym statystyki tempa zmian w danych tracą ważność oraz liczbę agentów mobilnych, uruchamianych przy starcie węzła.

## 4. Obiektowy model danych

Relacyjny model danych jest niewątpliwie najpopularniejszym modelem wykorzystywanym w systemach informatycznych działających na podstawie baz danych. Nie jest to jednak jedyny z dostępnych modeli. Równolegle z nim funkcjonuje model obiektowy, choć nie posiada takiego zasięgu jak model relacyjny.

Dla takiego sposobu reprezentowania danych opracowany został model rozszerzony o encje, które posiadają relacje typu *rodzic – dziecko* z więcej niż jednym rodzicem. Model ten, w postaci diagramu klas, na którym można dokładniej zaprezentować skomplikowane związki, zaprezentowano na rysunku 3.

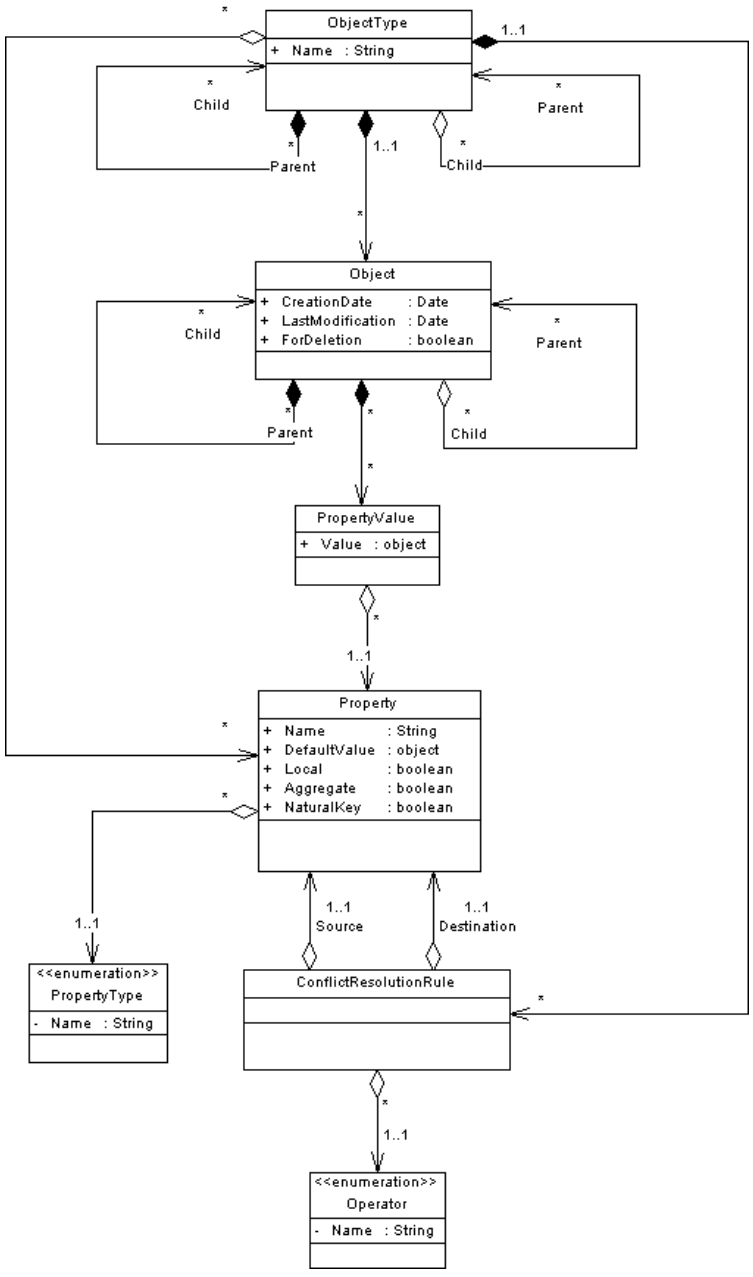
Klasa *ObjectType* reprezentuje typ obiektu. Służy tylko do opisu danych reprezentowanych przez klasę *Object*. Dwa związki, w które wchodzi klasa *ObjectType – Has parents* i *Has children,* pozwalają organizować typy obiektów w struktury hierarchiczne. Dla każdego typu określone są również zbiór zasad rozwiązywania konfliktów (*ConcflictResolutionRule)* oraz właściwości (*Property*).

W klasie *Object* zgromadzone są informacje na temat obiektów znajdujących się w bazie danych. Posiada ona takie same atrybuty, jak jej odpowiednik z uogólnionego modelu danych. Różnica występuje w jej związkach z innymi encjami. Każdy obiekt związany jest z dokładnie jednym typem obiektu, może być rodzicem dowolnej liczby dzieci i sam może posiadać ich dowolną liczbę. Zbiór rodziców pełni tutaj funkcję kategorii, która w modelu opisywanym w poprzednim podrozdziale, zdefiniowana jest przez encję *ObjectType*. Klasa *Object* związana jest z klasą *PropertyValue*, która opisuje zbiór wartości jej właściwości. Posiada ona jedno pole typu *Object*, które reprezentuje wartość dowolnego typu.

W językach programowania, w których hierarchia dziedziczenia oparta jest na wspólnym korzeniu, *Object* może reprezentować dowolny typ danych.

Informacje dotyczące właściwości obiektów, zdefiniowane są przez klasę *Property*. Jej pola odpowiadają atrybutom encji *ObjectProperty*, zaprezentowanej wcześniej na uogólnio-

nym modelu danych. Dodatkowo została ona uzupełniona o atrybut *NaturalKey*, stanowiący flagę, którą oznaczone są właściwości należące do naturalnego klucza obiektu. Dzięki temu obiekt może posiadać złożony klucz naturalny.




Rys. 3. Diagram przedstawiający obiektowy model danych
Fig. 3. Diagram of the object data model

W celu uproszczenia, z diagramu opisywanego modelu usunięte zostały klasy *SynchronizationLog*, *SpeedOfChanges* i *Settings*. Ich znaczenie w obiektowym modelu danych jest takie samo, jak analogicznych encji z poprzedniego modelu danych.

Obiektowy model danych pozwala na replikowanie skomplikowanych struktur, a agent operujący na tym modelu może replikować dane bez znajomości ich znaczenia. Dzięki temu replikacja może być również zastosowana bez zmian w dużej gamie aplikacji.

Należy jednak pamiętać, że ze względu na potrzebę tworzenia dużej liczby złączeń w celu wybrania interesujących użytkownika danych oraz skomplikowania operacji aktualizacji i wstawiania rekordów, nie nadaje się on do zastosowania w relacyjnej bazie danych.

## 5. Podsumowanie

Celem zaprezentowanych w artykule badań było opracowanie modelu danych, który mógłby posłużyć jako podstawa pracy systemu wykorzystującego replikację danych, zarządzaną przez autonomicznie działających agentów. W badaniach uwzględniono możliwość replikowania danych pomiędzy bazami danych zbudowanymi zarówno na podstawie relacyjnego, jak i obiektowego modelu danych. Rozważania dotyczyły przykładowej dziedziny przedmiotowej, reprezentującej pewną ich klasę, jednak zaproponowane mechanizmy zarządzania replikacją nie nakładają żadnych ograniczeń na dziedzinowy model danych.

W pracy wskazano cechy (atrybuty, encje), o jakie powinien zostać rozszerzony podstawowy model danych, aby mógł stanowić podstawę pracy systemu agentowego. Ważnym w tym zakresie elementem opracowanego modelu są struktury stanowiące *metadane* procesu replikacji. Na podstawie ich zawartości agent może samodzielnie podejmować decyzje zarówno o momencie, jak i sposobie jego realizacji. Taka możliwość nabiera szczególnego znaczenia w przypadku agentów mobilnych, które większość czasu pracują bez kontaktu z innymi elementami systemu i w dokonywaniu niezbędnych wyborów muszą polegać na swojej inteligencji oraz na informacjach zawartych w bazie danych.

Zaproponowany w pracy uogólniony model danych zaimplementowany został w opracowanym systemie agentowym, zarządzającym danymi przykładowej dziedziny przedmiotowej (sieci aptek). Uzyskane wyniki potwierdziły słuszność przyjętych w pracy rozwiązań oraz stanowią podstawę do dalszych badań dotyczących rozwoju algorytmów replikacji danych oraz efektywności przeprowadzania takiego procesu.

Praca naukowa finansowana ze środków na naukę w latach 2010-2012 jako projekt rozwojowy nr O R00 0113 12

**BIBLIOGRAFIA**

1. Bernstein Ph.A., Hadzilacos V., Goodman N.: Concurrency Control and Recovery in Database Systems. Addison-Wesley, 1987.
2. Harężlak K: Asynchroniczna metoda aktualizacji kopii danych. Współczesne problemy sieci komputerowych. Nowe technologie. WNT, 2004.
3. Harężlak K.: Replikacja danych ma urządzenia przenośne. Studia Informatica, ZN Pol. Śl., s. Informatyka, Vol. 30, No. 2A (83), Wyd. Pol. Śląskiej, Gliwice 2009.
4. Wooldridge M.: An Introduction to MultiAgent Systems. Second Edition. John Wiley & Sons Ltd, 2009.
5. Bellifemine F.C., Greenwood D.: Developing Mulit-Agent Systems with JADE. John Wiley & Sons Ltd, 2007.
6. Foundation for Intelligent Physical Agents. FIPA ACL Message Structure Specification. *FIPA Specifications*, http://www.fipa.org/specs/fipa00061/SC00061G.html.
7. Harężlak K., Josiński H.: Dostęp do rozproszonych baz danych z wykorzystaniem agentów przenośnych. Studia Informatica, ZN Pol. Śl., s. Informatyka, Vol. 23, No. 4 (51), Wyd. Pol. Śląskiej, Gliwice 2002.
8. Harężlak K.: Aktualizacja kopii danych z wykorzystaniem agentów przenośnych. Studia Informatica, ZN Pol. Śl., s. Informatyka, Vol. 24, No. 4 (56), Wyd. Pol. Śl., Gliwice 2003.
9. Ullman J.D., Widom J.: Podstawowy wykład z systemów baz danych. Wydawnictwa Naukowo-Techniczne, 2001.
10. Sujoy P.: SQL Server 2008 Replication. Apress New York, Springer-Verlag, Berkeley, CA 2009.
11. Introduction to DB2 replication and publishing, http://publib.boulder.ibm.com/infocenter/db2luw/v8/index.jsp.
12. Oracle® Database. Advanced Replication, http://download.oracle.com/docs/cd/B19306_01/server.102/b14226.pdf.

Recenzenci: Prof. dr hab. inż. Henryk Rybiński
Dr inż. Tomasz Traczyk

Wpłynęło do Redakcji 16 stycznia 2011 r.

**Abstract**

The aim of the research presented in the paper was to prepare data model which can serve as a base for a system using replicated data managed by independent agents. Both technologies enable to build tools characterized by high scalability and dedicated distributed environments. All the more integration of their mechanisms may result in development of new solutions which meet growing requirements regarding functionality of the computer tools.

Studies concerning these issues started with designing data model providing structures for storing records of given domain and objects for replication process management. As a result there were two data model prepared. First of them, dedicated to relational data, model have been implemented in a sample agent system. The second one requires usage of structures of object data model. Generality of solutions achieved in both cases allows to use them in many fields.

The obtained results confirmed the rightness of preliminary assumptions. They constitute a basis for further research regarding development of algorithms for data replication and effectiveness of conducting such process.

**Adresy**

Katarzyna HARĘŻLAK: Politechnika Śląska, Instytut Informatyki, ul. Akademicka 16, 44-100 Gliwice, Polska, katarzyna.harezlak@polsl.pl.
Łukasz KULISZ: Politechnika Śląska, Instytut Informatyki, ul. Akademicka 16, 44-100 Gliwice, Polska, lukasz.kulisz@gmail.com.