ONTOLOGIA JĘZYKA WZORCÓW PROJEKTOWYCH DLA SYSTEMÓW *SMART CITIES*

Cezary Orłowski[1], ***Tomasz Sitek***[1]
Artur Ziółkowski[1], ***Paweł Kapłański***[1]
Aleksander Orłowski[1], ***Witold Pokrzywnicki***[1]

Streszczenie

W artykule przedstawiono definicję języka wzorów projektowych *Smart Cities* w postaci ontologii. Jako, że wdrażanie rozwiązania *Smart City* jest trudne, drogie i ściśle związane z problematyką dotyczącą danego miasta, wiedza nabywana podczas pojedynczego wdrożenia jest wiedzą niezwykle cenną. Zdefiniowany przez nas język wspiera zarządzanie ww. wiedzą, jako że pozwala na ekspresję rozwiązania, które bazując na najlepszych praktykach zapisanych w postaci wzorców projektowych, jest jednocześnie dostosowane do wymagań miasta dążącego do wdrożenia rozwiązania *Smart City*. Formalna/ontologiczna struktura tego języka pozwala z kolei na automatyczne dowodzenie właściwości zapisanego tak rozwiązania. Ta ostatnia właściwość wprowadzonego języka jest niezwykle istotna w procesie podejmowania decyzji o wyborze danego rozwiązania przez odpowiednie władze. Praca została podzielona na pięć głównych części. W części pierwszej omawiamy problematykę wdrażania szyny integracyjnej na przykładzie IOC. W kolejnej części mówimy o zasadności zastosowania technologii semantycznych w celu rozszerzenia spektrum potencjalnych wdrożeń. Dalej mówimy o stworzonej przez nas, ontologicznej implementacji języka wzorców *Smart-City* – języka, który pozwala zapisywać zarówno wymagania, jak i walidować rozwiązania w nim specyfikowane. Przedstawiamy również przykładowe użycie, które służy nam jednocześnie jako walidacja języka w warunkach rzeczywistych. W ostatniej części dyskutujemy pewne aspekty języka wzorców i możliwe drogi rozwoju związanych z nim badań.

Słowa kluczowe: Smart Cities, ontologies, semantics, Ontology Driven Architecture, Design Patterns, Controlled Natural Language.

https://doi.org/10.34808/reme.2015.02.005

[1] Politechnika Gdańska, Wydział Zarządzania i Ekonomii/ Gdańsk University of Technology, Faculty of Management and Economics

1. Wprowadzenie

Koncepcja Inteligentnego Miasta (*Smart City*) obiecuje uzyskanie nowej jakości w zarządzaniu miastem dzięki zintegrowaniu dostępnych w mieście struktur teleinformatycznych oraz ich ekspansji na inne obszary miejskie. Integracja owych struktur odbywa się poprzez odpowiednie wdrożenie wspólnej, skalowalnej i wysoce wydajnej szyny integracji np. IBM Intelligent Operations Center (Bhowmick, 2012). Szyna taka, integrując się pośrednio z przestrzenią miejską, staje się jej częścią, a co za tym idzie proces projektowania wdrożenia ww. szyny integracji jest wysoce skomplikowany oraz kosztowny. Aby obniżyć koszty kolejnych wdrożeń proces ten powinien cechować się powtarzalnością. Powtarzalność nie może jednak, ze względu na różnorodność przestrzeni miejskich, polegać na kopiowaniu udanych wdrożeń w całości, lecz raczej polegać na doborze najlepszej, dostosowanej do konkretnej specyfiki, kombinacji dobrych praktyk oraz weryfikację (a priori) dopasowania tak powstałego rozwiązania jako całości. Pożądana metoda doboru rozwiązania wdrożeniowego szyny integracyjnej *Smart City* powinna więc pozwalać na:

— katalogowanie wcześniej wypracowanych dobrych praktyk,
— weryfikację spodziewanych efektów wdrożenia.

W tym artykule przedstawiamy wypracowaną przez nas metodę zarządzania doświadczeniem wdrożeniowym szyny integracyjnej *Smart City*, która spełnia oba warunki. Jest ona oparta o wprowadzony przez nas tzw. język wzorców *Smart City*, zapisywany w postaci ontologii informatycznej. Język ten jest wyposażyliśmy we wszystkie warstwy językowe:

— warstwa syntaktyczna (oraz gramatyka) są dane przez sieć możliwych rozwiązań, w które wpasowuje się język wzorców, sławami języka wzorców są wzorce;
— semantyka języka wzorców jest określona przez ograniczenia/możliwości ich wzajemnego przeplatania się (zależności) wzorców;
— warstwa pragmatyczna jest realizowana poprzez odpowiednio dobrane narzędzia pozwalające na pracę z ww. językiem. W tym celu zostało przez nas zaimplementowane kluczowe narzędzie (OntoTransTool) pozwalające na integrację technologii semantycznych z istniejącym rozwiązaniem pozwalającymi na konfigurację *Smart City*.

Opisujemy również wyniki weryfikacji ww. metody na przypadku wdrożenia szyny integracyjnej *Smart City* opartego o szynę integracji IBM IOC w Gdańsku.

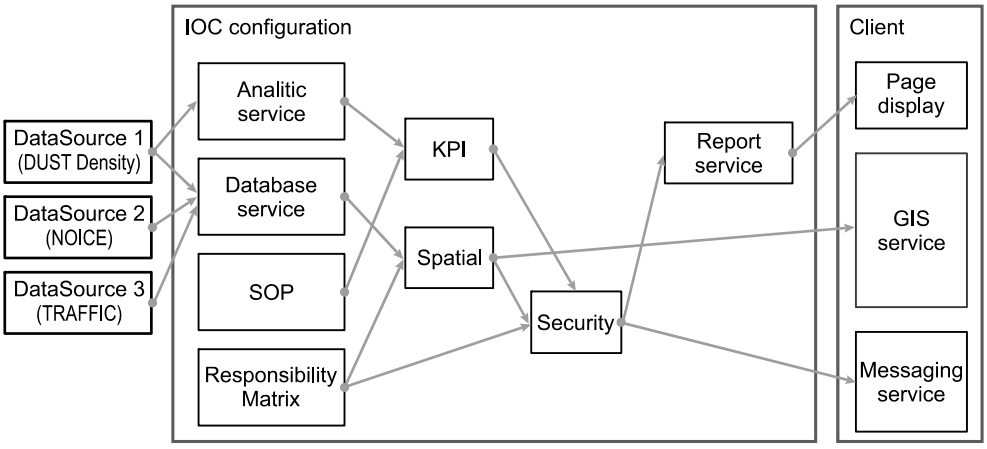
2. Problematyka wdrażania szyny integracyjnej *Smart City* na przykładzie IBM Intelligent Operations Center

W tej części opisujemy proces wdrażania szyny integracyjnej Smart-City na przykładzie IBM Intelligent Operations Center (IBM IOC). Szyna integracyjna IBM IOC pozwala na wysoce wydajne przetwarzanie strumieni danych oraz dostarcza narzędzi pozwalających na ich integrację. Wdrożenie IBM IOC odbywa się na dwóch poziomach. Na poziomie niskim – poziomie szyny integracji (rys. 1)

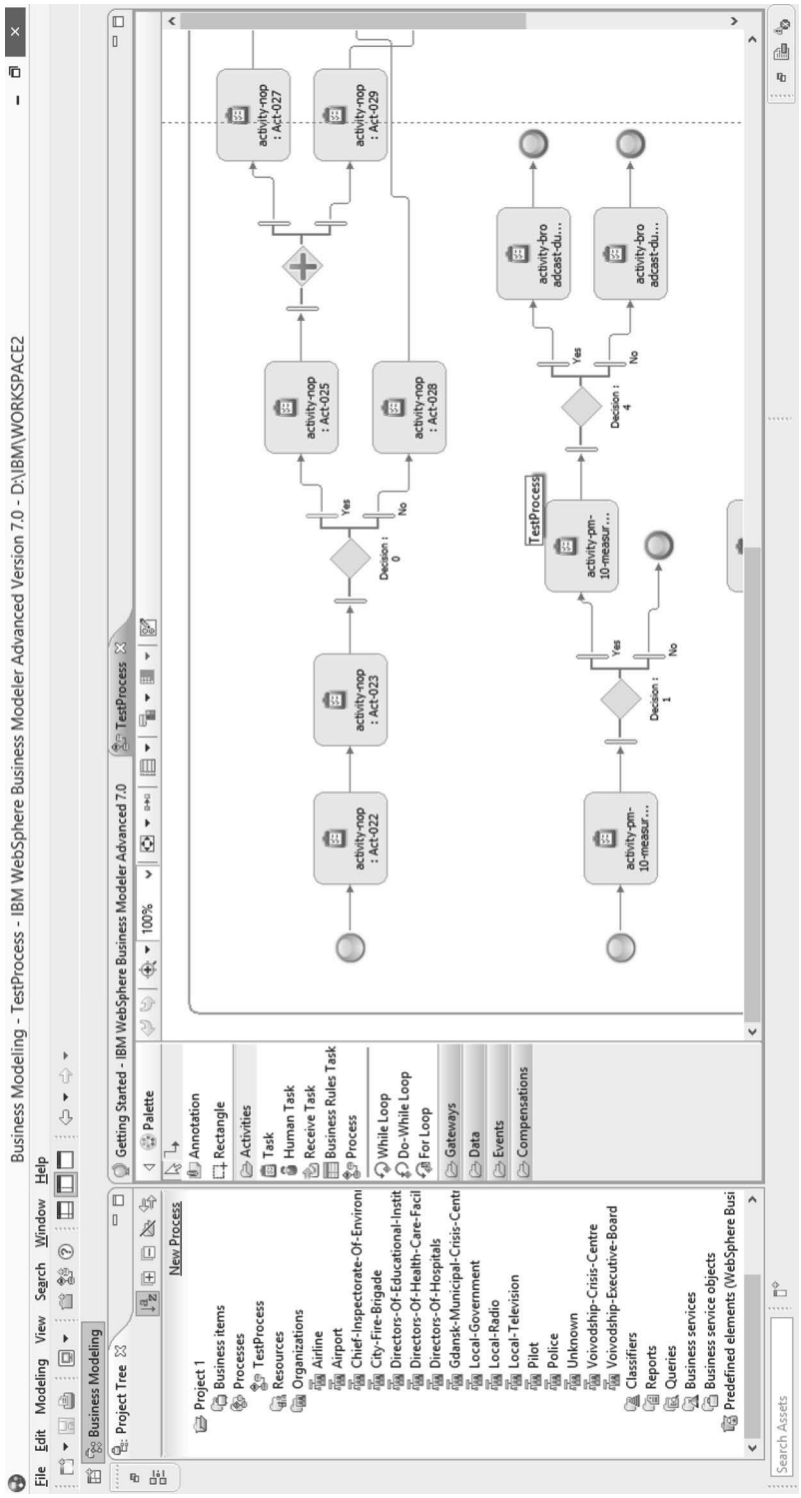
oraz na poziomie wysokim – poziomie modelu (rys. 2). Opisany w tej części, dwupoziomowy proces wdrożenia IBM IOC jest punktem wyjścia dla proponowanego przez nas rozszerzenia procesu wdrożenia o poziom ontologii, który docelowo stanie się środowiskiem do którego odnosi się proponowany przez nas język wzorców *Smart City*.

Poziom szyny integracji pozwala na bezpośredni dostęp do usług dostarczanych przez IBM IOC. Na rysunku (rys. 1) przedstawiono przykładowe wdrożenie IBM IOC widziane z tego poziomu. Możemy tu dostrzec, że przetwarzanie danych w obrębie szyny IBM IOC jest realizowane w czasie rzeczywistym za pośrednictwem strumieni danych (*DataSource*), a wyniki przetwarzania tych strumieni stają się podstawą wyznaczania kluczowych wskaźników wydajności KPI. Wskaźniki te prezentowane są operatorom szyny w postaci intuicyjnego interfejsu użytkownika (Client), zarówno w postaci raportów (Raport), jak i na mapie (Spatial, GIS service), dzięki czemu wspierają ich w procesie podejmowania decyzji. System integruje również pracę operatorów poprzez zintegrowany system zarządzania organizacją typu *workflow* (realizowany przez zbiór procedur *Simple Operational Procedure* – SOP), oraz teleinformatyczne narzędzia pozwalające na bezpośrednią komunikację operatorów (*Messaging service*). Wbudowana w system hierarchia uprawnień (*Responsibility Matrix*) wraz z systemem zabezpieczeń (*Security*) pozwala na odwzorowanie rzeczywistej hierarchii odpowiedzialności.

Poziom szyny integracyjnej udostępnia największe możliwości konfiguracyjne, jednak docelowa konfiguracja z perspektywy tego poziomu jest trudna w utrzymaniu a co za tym idzie wymaga wykwalifikowanej kadry informatycznej. Przykładem może być konfiguracja procesów typu *workflow*: przepływ dokumentów oraz implementacja toru eskalacji odpowiedzialności na tym poziomie wymaga zdefiniowania całego procesu w postaci pojedynczych SOPów, z których jeden może uruchamiać inne generując łańcuchy odpaleń reguł. W gąszczu SOPów umyka nam całościowy obraz procesu.



Rys. 1. Przykładowa konfiguracja *Smart City* na poziomie szyny integracji (w warstwie M0 MOF)



Rys. 2. Przykładowa konfiguracja *Smart City* na poziomie modelu (warstwa M1 MOF)

W przeciwieństwie do poziomu szyny, poziom modelu pozwala na uchwycenie konfiguracji IBM IOC z perspektywy procesów, które ma ona realizować. Jest to metoda naturalna dla analityków i kadry zarządzającej, pozwalająca na pewnym stopniu ogólności uchwycić całościowe aspekty wdrożenia, trudne do dostrzeżenia z poziomu szyny integracyjnej. Modelowanie na tym poziomie odbywa się w narzędziu IBM Business Modeller w języku Business Process Modeling Notation (BPMN) (OMG, 2011). Przykład konfiguracji IBM IOC na tym poziomie przedstawiono na rysunku (rys. 2). Widać tutaj zarówno strukturę organizacyjną (*Organizations*), szablony raportów, definicje zapytań jak i (w przeciwieństwie do poziomu szyny) pełną strukturę procesów typu *workflow* wraz z przypisaniem odpowiednich uczestników biorących w nich udział, z uwzględnieniem przepływających między nimi komunikatów.

Poziom modelu nie udostępnia jednak pełnych możliwości konfiguracyjnych dostępnych z poziomu szyny integracyjnej, takich jak np. możliwość podłączania dowolnych (zewnętrznych źródeł danych), dlatego najczęściej konfiguracja szyny IOC przebiega dwuetapowo (rys. 3). Z poziomu modelu generowana jest perspektywa poziomu szyny integracyjnej, która następnie jest uzupełniana niezbędne elementy i jest docelowo wrażana na poziomie docelowej infrastruktury IOC. Proces ten odbywa się zgodnie z prawidłami transformacji modeli w ujęciu *Model Driven Architecture* (MDA). Architektura MDA o nazwie *Meta-Object Facility* (MOF) została wprowadzona w roku 2001 przez *Object Management Group* (OMG) i jest wynikiem ewolucji metod modelowania obiektowego w szczególności opartych o język UML.

Rys. 3. Proces konfiguracji szyny IOC: przejście z poziomu modelu poprzez poziom szyny integracyjnej

IBM IOC realizuje architekturę MOF. MOF jest zaprojektowana jako czterowarstwowa architektura modeli, z których każdy kolejny jest ogólniejszy od poprzedniego. Warstwy noszą nazwy: M3, M2, M1 oraz M0. Z punktu widzenia IOC poziom szyny integracji odpowiada warstwie M0, natomiast poziom modelu warstwie M1 (rys. 1–2). Modele są w istocie sieciami powiązanych ze sobą encji, które z kolei referują do fenomenów występujący w danej domenie (Fowler, 2003). Jako że modele w MOF są reprezentowane przez grafy, to językami pozwalającymi na przetwarzanie modeli są języki transformacji grafów (Rozenberg, 1997). W MOF, przejście z meta-modelu (modelu z wyższej warstwy) do modelu następuje z użyciem transformacji modelu o nazwie Queries/Views/ Transformations (QVT) zaznaczonej schematycznie na rysunku (rys. 3).

Podsumowując: IBM IOC dostarcza zarówno infrastruktury, jak i wspiera proces wdrażania ww. infrastruktury szyny integracyjnej. Wsparcie odbywa się na dwóch poziomach. Na poziomie modelu (M1 w nomenklaturze MOF) oraz na poziomie szyny integracyjnej (M0 na poziomie MOF). Podczas konfiguracji wdrożenia IBM IOC z perspektywy poziomu modelu zostajemy wyposażeni w potężne narzędzie służące do modelowania procesów biznesowych. Dodatkowo mamy na tym poziomie możliwość modelowania wysokopoziomowej struktury organizacyjnej. Po transformacji z poziomu modelu na poziom szyny integracyjnej otwiera nam się z kolei pełne spektrum możliwości integracyjnych.

W tym miejscu warto wskazać poważne ograniczenie ww. metod wsparcia procesu wdrażania szyny integracyjnej. Brakuje tutaj możliwości zapisania wiedzy o obszarze/domenie, której dotyczy dane wdrożenie IBM IOC, a co za tym idzie modele procesów oraz organizacji pozostają w oderwaniu od specyfiki opisywanego przez nie problemu. Jest to poważne ograniczenie, jako że uniemożliwia ono pełną integrację wiedzy o danym wdrożeniu. Wiedza o obszarze/ domenie/ problemie w ww. ujęciu jest wiedzą, która musi być przechowywana/przetwarzana w systemach zewnętrznych.

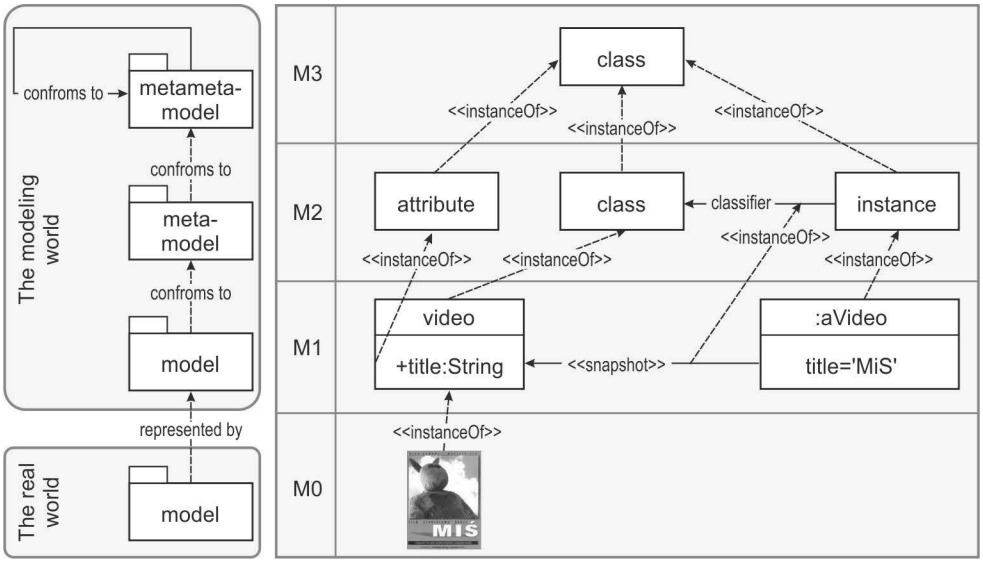
W kolejnej części wskazujemy metodę rozszerzenia reprezentacji wiedzy używanej w IBM IOC o poziom ontologii oraz wskażemy narzędzia pozwalające na jej integrację z istniejącym rozwiązaniem.

3. Ontologie oraz kontrolowany język naturalny w procesie projektowania systemów *Smart Cities*

W poprzedniej części artykułu omówiliśmy stosowane obecnie na rynku podejście do projektowania wdrożenia *Smart City* na przykładzie IBM IOC. Wskazaliśmy również na ważne ograniczenie ww. podejścia, które uniemożliwia pełną integrację wiedzy o rozwiązaniu z wiedzą o obszarze/domenie/problemie, którego dane rozwiązanie dotyczy. W tej części pokażemy, że integracja metod semantycznych z dwupoziomowym procesem konfiguracji IBM IOC pozwala na wypełnienie tej luki. W tym celu stworzymy kolejny poziom konfiguracji szyny integracyjnej – poziom ontologii. Tę część zakończymy wprowadzeniem do nowoczesnych metod ekspresji wiedzy w formie (kontrolowanego) języka naturalnego oraz opiszemy narzędzie pozwalające na integrację poziomu ontologii z pozostałymi dwoma poziomami wdrażania szyny integracji IBM IOC. Powstałe rozszerzenie stanie się podstawą narzędziową dla, opisanego w następnej części, języka wzorców *Smart City* – języka pozwalającego na komponentową dekompozycję przestrzeni kombinatorycznej możliwych rozwiązań wdrożeniowych szyny integracyjnej.

W poprzedniej części mówiliśmy o dostarczanej razem z IBM IOC architekturze MDA o nazwie MOF i określiliśmy ją jako czterowarstwową architekturę metamodeli. Jak przedstawiono to na rysunku (rys. 4). Metameta-model nazywany warstwą M3 to najbardziej ogólny opis świata modeli w MOF, jest to definicja podstawowych pojęć MOF. Warstwa M2 jest modelem języków modelowania takich jak UML czy BPMN. M1 to modele zapisane w ww. językach modelowania,

natomiast M0 jest właściwym modelem, który odnosi się bezpośrednio do modelowanego świata.



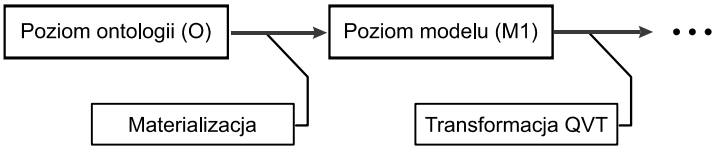
Rys. 4. Ogólna architektura Meta-Object Facility

Warstwę M2 MOF, która definiuje języki modelowania (a więc m.in. pojęcia takie jak klasa, atrybut, relacja czy proces), oraz warstwa M3 definiująca pojęcia najbardziej abstrakcyjne (takie jak pojęcie bycia podklasą, pojęcia bycia relacją czy też bycia konkretnym elementem klasy/zbioru – instancją) okazują się wspólne również dla technologii semantycznych z rodziny W3C Semantic Intiative. OWL jest zintegrowany z MOF poprzez przyporządkowanie elementów UML (warstwa M2) do odpowiednich elementów OWL oraz bezpośrednie wskazanie reprezentacji warstwy M3 w pojęciach odstawowych OWL.

To właśnie dzięki istnieniu tej odpowiedniości pojęć pomiędzy MOF oraz OWL możliwe staje się rozszerzenie dwupoziomowego procesu konfiguracji IBM IOC (bazującego na MOF) o metody semantyczne z rodziny OWL – metody pozwalające na zapis wiedzy w postaci ontologii. To właśnie ontologie są narzędziem pozwalającym na zapis wiedzy o obszarze/domenie/problemie opisywanym przez dane wdrożenie szyny integracyjnej IOC. Co więcej, w ten sposób (dysponując ww. rozszerzeniem) rozszerzamy również spektrum narzędzi (do tej pory były to języki transformacji grafów) o systemy dowodzenia twierdzeń (ang. *reasoner*) operujących w formalizmie logiki opisowej *SROIQ* (Goczyła, 2011; Horrocks i inni, 2006). Urzędnik odpowiedzialny za wdrożenie danego rozwiązania może, posiłkując się systemem dowodzenia twierdzeń, znaleźć potwierdzenie użyteczności danego rozwiązania w kontekście danego obszaru/domeny/problemu.

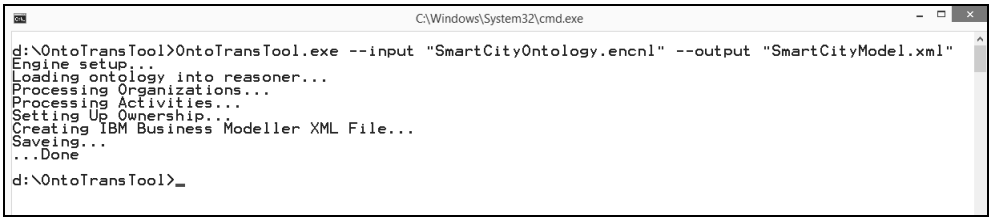
Aplikując kombinację MOF oraz OWL w procesie wdrażania szyny IBM IOC dodajemy kolejny poziom wdrożenia szyny IOC – poziom ontologii (O), natomiast

proces wdrażania rozszerza się o etap materializacji modelu na podstawie ontologii (rys. 5).



Rys. 5. Materializacja poziomu modelu (M1) na podstawie ontologii (O)

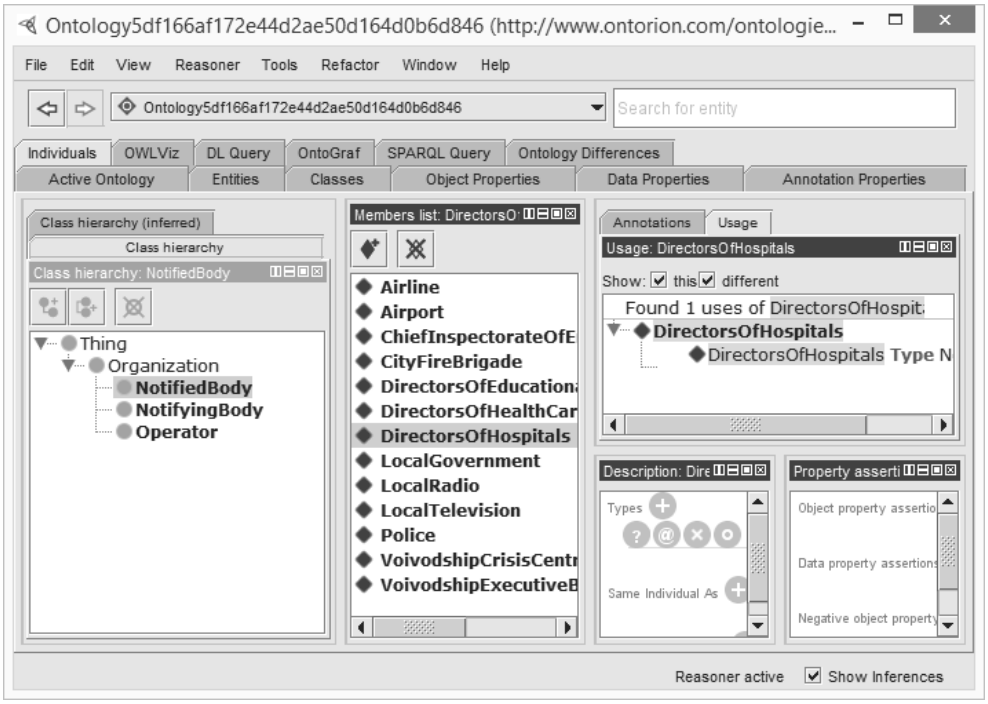
Materializacja jest realizowana w oparciu o system dowodzenia twierdzeń i pozwala na przejście z poziomu ontologii do poziomu modelu analogicznie do sposobu działania transformacji QVT w przypadku przejścia z poziomu modelu do poziomu szyny integracyjnej w standardowym podejściu wspieranym przez IBM IOC. O ile jednak podczas przejścia z poziomu modelu na poziom szyny mamy do czynienia z transformacją grafów, o tyle stworzone przez nas narzędzie przeprowadzające ww. transformację (które nazwaliśmy OntoTransTool) materializuje graf poziomu modelu bazując na teorii (ontologii) zapisanej w logice opisowej (bazy formalnej OWL). OntoTransTool jako wejście przyjmuje ontologię i na jej podstawie generuje plik wejściowy dla IBM Business Modellera (patrz rys. 6) – (IBM Business Modeller jest narzędziem o którym wspominaliśmy w poprzedniej części, służącym do modelowania wdrożenia IBM IOC z poziomu modelu – patrz rys. 2).



Rys. 6. Materializacja modelu M2 z poziomu ontologii (O) przy użyciu narzędzia OntoTransTool

Kolejnym krokiem, po integracji poziomu ontologii z dwupoziomowym procesem wdrażania szyny integracji IOC jest dobór narzędzia służącego na tym poziomie do modelowania ontologii. Najbardziej znanym narzędziem do modelowania ontologii jest Protégé, rozwijane przez uniwersytet Stanford (Musen i inni, 2010). Protégé umożliwia zarówno tworzenie, jak i debugowanie ontologii. Narzędzie jest wyposażone jest w graficzny interfejs użytkownika pozwalający na edycję ontologii w trybie interakcji. Podjęliśmy próby użycia go do modelowania na poziomie ontologii (patrz rys. 7). Niestety narzędzie to wymaga odpowiedniego przeszkolenia użytkownika, który musi poznać metody inżynierii wiedzy w tym narzędziu. W praktyce uniemożliwia to zastosowanie narzędzia w prowadzeniu dialogu

z osobami zainteresowanymi wdrożeniem samego rozwiązania *Smart City* (nie są one zazwyczaj zainteresowane aspektami inżynierii wiedzy).



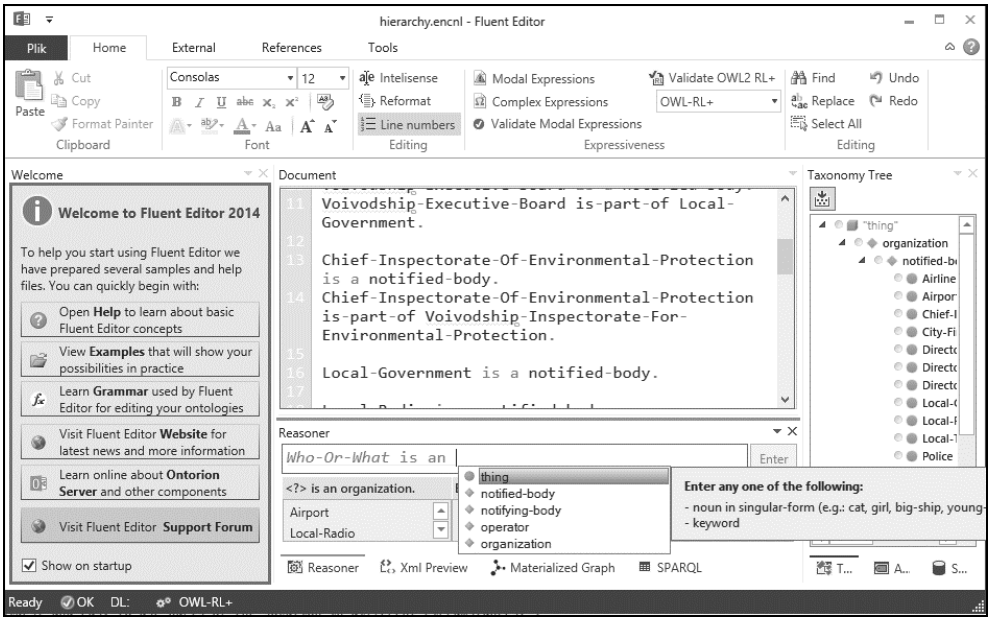
Rys. 7. Przykładowa konfiguracja Smart City modelowana na poziomie ontologii (warstwa O) w narzędziu Protégé

Z drugiej strony możliwość konsultowania ontologii z osobami zainteresowanymi jest ważna z kilku powodów:

— dzięki temu włączamy klienta w proces konfiguracji – ma on realny wpływ nie tylko na wymagania, ale również na kształt rozwiązania – co zwiększa szanse sukcesu;
— elementy ontologii mogą być załącznikiem do kontraktu wiążącego zamawiającego dane wdrożenie IBM IOC z dostawcą platformy;
— klient ma możliwość weryfikacji rozwiązania (a priori), a co za tym idzie dysponuje pewnego rodzaju dowodami na słuszność podejmowanej przez siebie decyzji już na etapie projektowania wdrożenia.

Wyżej wymieniona bariera komunikacyjna dotycząca wsparcia narzędziowego jest w znacznym stopniu eliminowana przez tzw. naturalne interfejsy użytkownika – w szczególności interfejsy mające postać zbliżoną do języka naturalnego. Wydaje się to oczywiste (wszyscy ludzie niezależnie od dziedziny komunikują się w języku naturalnym) i w kontekście ekspresji ontologii poprzez zastosowanie kontrolowanych języków naturalnych (CNL) zostało to wykazane m.in. przez Kuhna (2009), jednak – co pokażemy w części dotyczącej wdrożenia – często nad

językiem naturalnym przewagę zdobywają graficzne języki reprezentacji wiedzy. W szczególności graficzne języki zapisu procesów biznesowych, takie jak BPMN – pozwalające na przedstawienie procesu w postaci grafu – są łatwiej przyswajalne dla człowieka niż ich reprezentacja w języku naturalnym. Ta właściwość graficznych języków zapisu procesów biznesowych uświadomiła nam, że warto reużyć artefakty, które naturalnie powstają na poziomie modelu w trakcie modelowania ontologii . Zaistniała potrzeba wypracowania transformacji dwustronnej – zarówno z poziomu ontologii do poziomu modelu, jak i z poziomu modelu do poziomu ontologii która również jest zapewniona przez zaimplementowane przez nas narzędzie narzędzie OntoTransTool.



Rys. 8. Przykładowa konfiguracja Smart City modelowania na poziomie ontologii (warstwa O) w narzędziu Ontorion Fluent Editor

Narzędziem pozwalającym na modelowanie ontologii w języku naturalnym jest Fluent Editor (Cognitum, 2014). Ontologie zapisane w narzędziu Fluent Editor są równoważne z ontologiami zapisanymi w narzędziu Protégé, jako że oba narzędzia operują w technologii OWL, jednak o ile w Protégé mamy do czynienia z graficznym interfejsem użytkownika, o tyle w Fluent Edytorze mamy ontologię przedstawioną w postaci dokumentu w języku naturalnym. Włączając narzędzie Fluent Editor do zbioru narzędzi wspierających konfigurację IOC otrzymaliśmy możliwość zapisu wiedzy o obszarze/domenie/problemie związanej z wdrożeniem w języku naturalnym (patrz rys. 8). Na rysunku 8 przedstawiono narzędzie Fluent Editor z załadowaną ontologią przykładu, o którym będziemy mówić w dalszej części artykułu.

Podsumowując, IBM IOC dostarcza narzędzi pozwalających na dwupoziomowe wdrażanie szyny integracyjnej, lecz pomija aspekt integracji wiedzy o obszarze/domenie, której dotyczy dane wdrożenie, a co za tym idzie często (ze względu na brak wsparcia narzędziowego) modele procesów oraz organizacji pozostają w oderwaniu od specyfiki opisywanego przez nie problemu. Wprowadzony przez nas trzeci poziom – poziom ontologii – wypełnia tę lukę. Modelowanie na poziomie ontologii jest realizowane z użyciem kontrolowanego języka naturalnego, umożliwiającego na bezpośrednie używanie zapisanych w nim ontologii do komunikacji z klientem systemu. Integracja poziomu ontologii jest zrealizowana za pomocą stworzonego przez nas narzędzia o nazwie OntoTransTool, które zapewnia możliwość materializacji ontologii do poziomu modelu.

W następnej części wprowadzamy oparty na stworzonej bazie narzędziowej, ontologiczny język wzorców *Smart City*, który pozwala na dekompozycję przestrzeni kombinatorycznej możliwych rozwiązań wdrożeniowych szyny integracyjnej a co za tym idzie umożliwia efektywne reużywanie dobrych praktyk projektowych wyłaniających się w trakcie pracy ekspertów.

4. Implementacja języka wzorców dla Smart City

W poprzedniej części wprowadziliśmy poziom ontologii do dwupoziomowego procesu wdrażania szyny integracyjnej IBM IOC. Poziom ontologii umożliwia ekspresję wiedzy na temat obszaru/domeny/problemu którego dotyczy dane wdrożenie szyny integracyjnej IBM IOC w sposób spójny z pozostałymi dwoma poziomami. Powstaje pytanie – jak konstruować ontologie na tym poziomie w sposób optymalny. Założyliśmy już we wprowadzeniu, że dążymy do stworzenia narzędzia, które będzie umożliwiało reużywanie cennej wiedzy eksperckiej na temat wdrożeń. Najlepszym do tego narzędziem, co dowodzi narastająca ich popularność, są języki wzorców.

Christopher Alexander (1977) zauważył, że miasta powstają na podstawie wzorców. Obecnie, wzorce projektowe są powszechnie używaną strukturą wymiany rozwiązań powtarzalnych problemów. Są one obecne w literaturze dotyczącej urbanistyki (skąd się wywodzą), informatyki, pedagogiki i wielu innych. Alexander zauważa również, że skatalogowanie wzorców tworzy nową wartość w postaci języka wzorców. Języka, którego słowami stają się nazwy poszczególnych wzorców. Jednocześnie, język wzorców pozwalając na rozwiązywanie problemów będących kombinacjami prostszych problemów opisanych przez wzorce-słowa, sam staje się wzorcem: wzorcem-językiem. Wzorzec definiujemy współcześnie, jako metodę dokumentowania rozwiązania pewnego powtarzalnego problemu projektowego w ramach pewnej domeny. W roku 1995, Gamma, Helm, Johnson and Vlissides opublikowali pierwszy katalog wzorców projektowania oprogramowania, aplikując idee Alexandra na polu inżynierii oprogramowania.

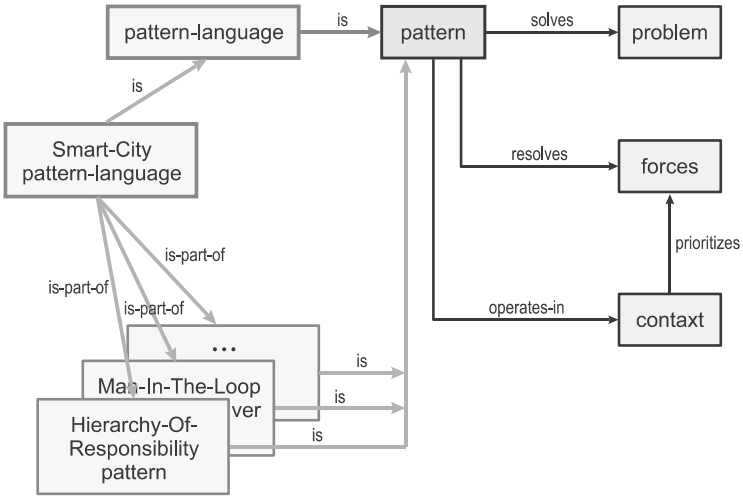
Jedną z pierwszych prób sformalizowania języków wzorców można odnaleźć w publikacji Meszarosa (1997), w której można znaleźć autodefinicję języka wzorców jako wzorca służącego tworzeniu bardziej skomplikowanych wzorców. Publi-

kacja ta stała się dla nas inspiracją i dała nam silną podstawę do stworzenia formalnego metajęzyka wzorców. Naszym dążeniem było przedstawienie „wzorców pisania wzorców" opisanych w pracy Meszarosa na formalizm zapisu ontologii. Wehikułem, który nam to umożliwił jest kontrolowany język naturalny.

Warto w tym miejscu pochylić się nad językiem zapisu ontologii – językiem kontrolowanym, który w narzędziu Fluent Editor ma postać pliku tekstowego. Poszczególne twierdzenia (aksjomaty) oddzielone są kropką (.) i możemy je podzielić na kilka podstawowych grup:

1. Concept subsumption represents all cases where there is a need to specify (or constrain) the fact about a specific concept or instance (or expressions that evaluate the concept or instance) in the form of subsumption (e.g.: Every cat is a mammal, Pawel has two legs or One cat that is brown has red eyes).
2. Role (possibly complex) inclusion specifies the properties and relationships between roles in terms of the expressiveness of SROIQ(D) (e.g.: If X loves something that covers Y then X loves-cover-of Y).
3. Complex rules; If [body] then [head] expressions that are restricted to the DL-Safe SWRL subset (Glimm i inni, 2008) of rules (e.g.: If a scrum-master is-mapped-to a provider and the scrum-master has-streamlining-assessment-processes-sprints-level equal-to 2 then the provider has-service-delivery-level equal-to 1 and the provider has-support-services-level equal-to 2).
4. Complex OWL expressions; the grammar allows the use of parentheses that can be nested if needed in the form of (that) e.g.: Every human is something (that is a man or a woman or a hermaphrodite).
5. Modal Expressions allows to express "knowledge about the knowledge" with CNL. This is to enforce fulfilling several properties of the knowledge. There are six modal words that can be used: must, should, can, must-not, should-not, cannot. All instances that are subject to validation against modal expressions are highlighted in Fluent Editor. Green means all requirements are fulfilled. Red means there is some requirement not met. Yellow means warning. It appears when requirements with "should" expression is not fulfilled. Light yellow means there is nothing wrong but it is marked just for informing the users. For example, regarding the statement "Every man can have a wife." – there is nothing wrong if there is some man that does not have a wife. For each modal expression different colors will appear as below:
   a) *X* must *Y* (e.g. Every *X* must also *Y*) – if not, it will be marked in red.
   b) *X* should *Y* (e.g. Every *X* should also *Y*) – if not, it will be marked in yellow.
   c) *X* can *Y* (e.g. Every *X* can also *Y*) – if not, it will be marked in light yellow.
   d) *X* can-not *Y* (e.g. Every *X* cannot also *Y*) – if it is, then it will be marked in red.
   e) *X* should-not *Y* (e.g. Every *X* should not also *Y*) – if it is, then it will be marked in yellow.
   f) *X* must-not *Y* (e.g. Every *X* must not also *Y*) – if it is, then it will be marked in light yellow.

Język wzorców mający postać ontologii staje się narzędziem pozwalającym na zarządzanie zaufaniem do rozważanego, zapisanego w nim, rozwiązania m.in. pozwala zastosować metody komputerowe jako ważnego uczestnika, który eliminuje wiele błędów generowanych przez „czynnik ludzki". Ontologia języka wzorców projektowych powinna być samoopisująca się, rozpoczniemy więc od definicji podstawowych pojęć. Poniższy przykład obrazuje użycie języka kontrolowanego na przykładzie definicji w postaci ontologii wzorca „wzorzec" oraz wzorca „język wzorców". Docelowa struktura ontologii języka wzorców projektowych jest schematycznie narysowana na rysunku rys. 9. Wzorzec jest tutaj zdefiniowany jako sposób reprezentacji wiedzy o rozwiązaniu problemu, działa w pewnym kontekście i rozwiązuje pewne siły priorytetyzowane przez ww. kontekst. Język wzorców spełnia definicję wzorca (jest wzorcem), rozwiązującym skomplikowany problem za pomocą sieci powiązanych ze sobą wzorców.



Rys. 9. Struktura ontologii języka wzorców projektowych Smart-City

Przejdźmy teraz do szczegółowego zapisu struktury ontologii języka wzorców, która jest schematycznie przedstawiona na rysunku rys. 9. Jako, że bazujemy tu na taksonomii wprowadzonej przez Meszarosa, pomiędzy apostrofami zawieramy bezpośrednie cytaty z (Meszaros i Doble, 1997). Proszę zwrócić uwagę na samopsujący charakter opisu struktury w kontrolowanym języku naturalnym.

§1 **The** problem **that** has-description **equal-to** "How do you share a recurring solution to a problem with others so that it may be reused?" **is** solved **by** Pattern-Pattern.

§2 **Every** pattern solves **at-most one** problem.

Warto tu zwrócić uwagę na to, że element opisujący rozwiązanie wzorca-wzorca wymaga zastosowania (za Meszarosem, 1997) innego wzorca: wzorca elementów obligatoryjnych – tu również pomiędzy apostrofami zawieramy bezpośrednie cytaty z (Meszaros i Doble, 1997).

§3 **The** problem **that** has-description **equal-to** "*How do you make sure **that** all necessary information **is** covered in **a** pattern?*" **is** solved **by** Mandatory-Elements-Present-Pattern.

Wymaganie dotyczące formatu wzorca (w skład którego wchodzi: kontekst, siły, rozwiązanie oraz problem) manifestuje się w poniższej linii, w której wskazujemy za pomocą wyrażenia modalnego, że każdy wzorzec musi realizować elementy obligatoryjne.

§4 **Every** pattern **must** realize Mandatory-Elements.

Z kolei wskazujemy na rozwiązanie wzorca elementów obligatoryjnych – wskazując explicite które elementy muszą być zawarte we wzorcu projektowym aby spełniał wzorzec elementów obligatoryjnych.

§5 **If a** pattern solves **a** problem **and the** pattern resolves **a** force **and the** pattern has-name **(some string value) and** the pattern operates-in **a** context **and the** context prioritizes **the** force **then the** pattern realizes Mandatory-Elements.

Zapiszmy rozwiązanie oferowane przez wzorzec wzorca używając konstrukcji modalnej „should". Nakłada ona na wiedzę niejednoznaczne (sic!) ograniczenie, inaczej mówiąc: dopuszczamy jako twórcy specyfikacji możliwość złamania kontraktu pozostawiając wolność wyboru użytkownikowi języka. Użytkownik języka łamiąc naszą rekomendację pozostanie jednak ostrzeżony o ww. naruszeniu:

§6 **Every** pattern **should** solve **a** problem.
§7 **Every** pattern **should** resolve **a** force.
§8 **Every** pattern **should** operate-in **a** context.
§9 Every pattern should have-name (some string value).
§10 **Every** context **should** prioritize **a** force.

Działanie ww. rekomendacji zademonstrujemy podczas definiowania języka wzorców Smart-City, wcześniej jednak zdefiniujmy abstrakcyjny język wzorców (który również jest wzorcem) – pomiędzy apostrofami zawieramy bezpośrednie cytaty z (Meszaros i Doble, 1997).

§11 Pattern-Language-Pattern solves **the** problem **that** has-description **equal-to** "*How do you describe the solution such that it is easy to digest and easy to use parts of the solution in different circumstances*?".

Wzorzec „język-wzorców" zdefiniowany powyżej generuję klasę języków wzorców zdefiniowaną poniżej, jednocześnie wymagamy (obligatoryjnie) aby implementował wzorzec „język wzorców":

§12 **Every** pattern-language **is a** pattern.
§13 **Every** pattern-language **must** implement Pattern-Language-Pattern.

Wymagamy/prosimy (nieobligatoryjnie), aby język wzorców był wyposażony w warstwę syntaktyczną manifestującą się za pomocą słownika:

§14 **Every** pattern-language **should** use **a** pattern-dictionary.

Wymagamy, aby język wzorców rozwiązywał (tym razem również nieobligatoryjnie) złożony problem. Ponadto definiujemy, co rozumiemy pod pojęciem „złożonego problemu":

§15 **Every** pattern-language **must** solve **a** complex-problem.

§16 **If a** problem**(**1**)** is-part-of **a** problem**(**2**) then the** problem**(**2**) is a** complex-problem.

Wskazujemy, że zastosowanie się do naszych próśb zaowocuje spełnieniem wymagania obligatoryjnego.

§17 **If a** pattern-language uses **a** pattern-dictionary **and the** pattern-language solves **a** complex-problem **then the** pattern-language implements Pattern-Language-Pattern.

Mając zdefiniowane pojęcia abstrakcyjne, stanowiące bazę pojęciową wysokiego poziomu możemy w końcu przejść do definicji konkretnego języka wzorców który będzie dostosowany do potrzeb poziomu ontologii procesu projektowania i wdrażania szyny integracyjnej IBM IOC. I tak definiujemy w języku kontrolowanym:

§18 Smart-City-Pattern-Language **is a** pattern-language **and** operates-in Context-Of-Smart-City-Pattern-Language **and** has-name **equal-to** "*Smart City Pattern Language*".

§19 Context-Of-Smart-City-Pattern-Language **is a** context **and** has-description **equal-to** "*A single pattern is insufficient to deal with all Smart City problems at hand.*"

§20 Smart-City-Pattern-Language solves **the** problem **that** has-description **equal-to** "*How do you describe the solution for Smart City such that it is easy to digest and easy to use parts of the solution in areas of*: a) *People-First*; b) *Business-Attractive*; c) *Green*; d) *Cheap* (*PBGC*)."

§21 **The** force **that** has-description **equal-to** "*A single large Smart City solution may be too specific to the circumstance and impossible to reuse in other circumstances.*" **is** resolved **by** Smart-City-Pattern-Language **and is** prioritized **by** Context-Of-Smart-City-Pattern-Language.

§22 **The** force **that** has-description **equal-to** "*A complex Smart City solution may be hard to describe in a single pattern. A* '*divide and conquer*' *approach may be necessary to make the solution tractable.*" **is** resolved **by** Smart-City-Pattern-Language **and is** prioritized **by** Context-Of-Smart-City-Pattern-Language.

§23 **The** force **that** has-description **equal-to** "*Factoring the Smart City solution into a set of reusable steps can be very difficult. Once factored, the resulting pieces may depend on one another to make any sense*." **is** resolved **by** Smart-City-Pattern-Language **and is** prioritized **by** Context-Of-Smart-City-Pattern-Language.

§24 **The** force **that** has-description **equal-to** "*Other pattern languages may want to refer to parts of the Smart City solution*; *they require some sort of* '*handle*'

for each of the parts to be referenced." **is** resolved **by** Smart-City-Pattern-Language **and is** prioritized **by** Context-Of-Smart-City-Pattern-Language.
§25 Smart-City-Pattern-Dictionary **is a** pattern-dictionary.
§26 Smart-City-Pattern-Language uses Smart-City-Pattern-Dictionary.

Powyżej zdefiniowaliśmy pojęcie języka wzorców Smart-City. Język wzorców Smart-City jest wzorcem pozwalającym na rozwiązywanie skomplikowanych problemów, których nie potrafimy rozwiązać pojedynczym wzorcem. Język wzorców rozwiązuje problem wdrażania skomplikowanego rozwiązania Smart-City. Inżynier wiedzy operując w warstwie ontologii ma możliwość używania języka wzorców Smart-City, z pojęciami reprezentującymi poszczególne wzorce. Jest „chroniony" przez wyrażenia modalne (obligatoryjne nakazy, prośby czy sugestie) zawarte w „metajęzyku wzorców" przed jego niewłaściwym użyciem.

Aby zilustrować działanie języka wzorców w praktyce zdefiniujmy dwa wzorce oraz użyjemy ich w przykładowym wdrożeniu. Będą to wzorce powszechnie i naturalnie występujące we wszystkich miastach, odpowiadające na realne, powszechne problemy.
— hierarchia odpowiedzialności,
— Man In The Loop-Resource conflict solver.

Oczywiście są to jedynie dwa spośród całego spektrum wzorców projektowych. Udało nam się do tej pory skatalogować dziesięć wzorców, jednak ich szczegółowy opis wykracza swoją obszernością poza ramy tego artykułu, wymienimy je jedynie z nazwy:
1. Agregowany KPI,
2. Mapa rozkładu KPI,
3. Prognoza KPI,
4. SOP Administracyjny,
5. Smart-Monitoring,
6. Wektoryzacja strumieni,
7. Social monitoring channel,
8. Dynamiczny znak drogowy.

Dwa wzorce, które opiszemy szczegółowo zostały przez nas wyselekcjonowane ze względu na ich późniejsze użycie podczas walidacji języka wzorców w przykładowym rzeczywistym wdrożeniu szyny konfiguracyjnej IBM IOC.

5. Wzorzec Smart-City: Hierarchia odpowiedzialności

Jako pierwszy opiszemy wzorzec rozwiązujący kluczowy z punktu widzenia miasta problem – problem odpowiedzialności za podejmowane decyzje. W tej części pokażemy również sposób, w jaki należy stosować język wzorców Smart-City na potrzeby ekspresji wiedzy o znanym, powszechnie akceptowalnym rozwiązaniu danego problemu, w postaci wzorca. Operujemy na poziomie ontologii rozszerzonego procesu konfiguracyjnego szyny integracyjnej IBM IOC.

Miasto jest zarządzane przez odpowiednie władze. Najbardziej rozpowszechnionym podejściem do zarządzania miastem jest hierarchia odpowiedzialności. Hierarchia, dzięki ustaleniu reguł przepływu komunikacji, usprawnia delegowanie zadań. Poniżej zapisany jest wzorzec „hierarchia odpowiedzialności". Jest on częścią słownika Smart-City i jednocześnie jest problemem, który rozwiązuje Wzorzec Języka Wzorców Smart-City:

§27 Context-Of-Hierarchy-Of-Responsibility-Pattern **is a** context **and** has-description **equal-to** „*Bez hierarchii przepływu oraz możliwości eskalacji problemu jest trudno zarządzać miastem*".

§28 Hierarchy-Of-Responsibility-Pattern **is a** pattern **and** operates-in Context-Of-Hierarchy-Of-Responsibility-Pattern **and** has-name **equal-to** "*Hierarchy Of Responsibility*" **and** solves **the** problem **that** has-description **equal-to** "*Jak ułożyć hierarchię przepływu decyzji oraz eskalacji problemów*?" **and** is-part-of Smart-City-Pattern-Dictionary.

§29 **The** problem **that is** solved **by** Hierarchy-Of-Responsibility-Pattern is-part-of **the** problem **that is** solved **by** Smart-City-Pattern-Language.

§30 **The** force **that** has-description **equal-to** "*często kompetencje urzędow są zduplikowane lub niejednoznacznie określone*" **is** resolved **by** Hierarchy-Of-Responsibility-Pattern **and is** prioritized **by** Context-Of-Hierarchy-Of-Responsibility-Pattern.

Rozwiązanie tego wzorca wymaga zdefiniowaniu kilku prawd. I tak:

§31 **Every-single-thing that** manages **is a** management.

§32 **Every** city **must be** managed **by**.

§33 **Every** smart-city **is a** city.

§34 **Every** smart-management **is a** management.

Wymagamy, aby Smart-City było zarządzane przez smart-management.

§35 **Every** smart-city **must be** managed **by a** smart-management.

§36 **Every** management **that** manages **a** smart-city **must be a** smart-management.

Dodatkowo wskazujemy, że użycie tego wzorca powoduje zrealizowanie wymagania obligatoryjnego.

§37 **If a** management implements Hierarchy-Of-Responsibility-Pattern **then the** management **is a** smart-management.

Powyższe zdanie (patrz §37) wydaje się być zbytnim uproszczeniem. Aby walidacja części obligatoryjnej przebiegała poprawnie wystarczy zadeklarować jego implementację (np. stwierdzając: Management-X implements Hierarchy-Of-Responisbility-Pattern). Jednak osoba, która takie zdanie wprowadzi staje się jednocześnie odpowiedzialna za wytłumaczenie, w jaki sposób to się jej udało. Wymagamy wobec tego dodatkowo, aby:

§38 **Every** management **that** implements Hierarchy-Of-Responsibility-Pattern **must** provide-implementation-details (**some** ontology-reference **value**).

To wymusza na użytkowniku języka wskazania uzasadnienia faktu zaimplementowania danego wzorca. W dalszej kolejności możemy rozszerzać nasze wymagania co do warunków, przy których zgadzamy się na to, aby wzorzec został uznany za spełniony: np. weryfikowanie pewnych właściwości wzorca, które musi on spełniać itp. W tym artykule pozostajemy jednak przy obecnym, najprostszym podejściu.

Dodajmy również, że miasto zarządzane za pomocą wzorca „Hierarchia-odpowiedzialności" jest miastem, w którym dochodzi do eskalacji problemów. Będzie to istotne z punktu widzenia kolejnego wzorca, który wymaga, aby taka eskalacja była możliwa:

§39 **If a** management implements Hierarchy-Of-Responsibility-Pattern **then the** management **is a** smart-management **and the** management **is an** escalator-based-management.

Na powyższym przykładzie widzimy, w jaki sposób dochodzi do przeplotu wzorców w języku wzorców. Wzorce nie zależą od siebie bezpośrednio, lecz pojęciowo – splatają się na poziomie pojęć. Dzięki temu jedne wzorce „pasują" do innych, a inne nie, natomiast używanie języka wzorców (jego pragmatyka) przypomina dobieranie odpowiednich klocków ze zbioru.

6. Wzorzec Smart-City: Man-In-The-Loop-Resource Resolver

Kolejnym i naszym zdaniem najważniejszym problemem, z którym boryka się każde miasto jest problem ograniczonych zasobów. W tej części zaprezentujemy rozwiązanie tego powtarzalnego problemu, które jest najczęściej wdrażane. Operujemy tutaj również na poziomie ontologii rozszerzonego procesu konfiguracyjnego szyny integracyjnej IBM IOC.

Konflikty pojawiające się w czasie sytuacji kryzysowych są związane z próbami zagospodarowania ograniczonych zasobów w kontekście koincydencji czasowej wielu sytuacji kryzysowych. W szczególności przeplatanie się tych samych zasobów pomiędzy równymi domenami, może (w przypadku pojawienia się konfliktów) doprowadzić do sytuacji wymagającej szybkiego wypracowania „najlepszej możliwej" decyzji. Wzorzec Man-In-The-Loop-Resource-Resolver rozwiązuje ten problem wstawiając w proces decyzyjny człowieka, na którym spoczywa odpowiedzialność podjęcia decyzji. Nie jest on jednak osamotniony, bowiem wzorzec obligatoryjnie wymaga istnienia środowiska do podnoszenia eskalacji – takie środowisko tworzy np. omawiany wcześniej wzorzec Hierarchii Odpowiedzialności. Wzorzec ten jest częścią tworzonego przez nas słownika Smart-City i jednocześnie jest problemem, który rozwiązuje Wzorzec Języka Wzorców Smart-City:

§40 Context-Of-Man-In-The-Loop-Resource-Resolver-Pattern **is a** context **and** has-description **equal-to** "*Smart-City wymaga zamodelowania kilku domen, wszystkie ww. domeny współdzielą ograniczone zasoby*".

§41 Man-In-The-Loop-Resource-Resolver-Pattern **is a** pattern **and** operates-in Context-Of-Man-In-The-Loop-Resource-Resolver-Pattern **and** has-name **equal-to**

"*Man In **The** Loop Resource Resolver*" **and** solves **the** problem **that** has-description **equal-to** "*W jaki sposób współdzielić ograniczone zasoby między różne domeny?*" **and** is-part-of Smart-City-Pattern-Dictionary.

§42 **The** problem **that is** solved **by** Man-In-The-Loop-Resource-Resolver-Pattern is-part-of **the** problem **that is** solved **by** Smart-City-Pattern-Language.

§43 **The** force **that** has-description **equal-to** "*Efektywne zarządzenie zasobami wymaga wiedzy eksperckiej*" **is** resolved **by** Man-In-The-Loop-Resource-Resolver-Pattern **and is** prioritized **by** Context-Of-Man-In-The-Loop-Resource-Resolver-Pattern.

§44 **The** force **that** has-description **equal-to** "*Niekiedy decyzje o przydziale zasobów wymagają zaangażowania wielu osób*" **is** resolved **by** Man-In-The-Loop-Resource-Resolver-Pattern **and is** prioritized **by** Context-Of-Man-In-The-Loop-Resource-Resolver-Pattern.

Poniżej prezentujemy definicję miasta wrażliwego na współdzielone zasoby:

§45 **Every-single-thing that is** dealt-with **by something is a** domain.

§46 **Every-single-thing that is** used-resource **by something is a** resource.

§47 **If a** smart-city deals-with **a** domain**(**1**) and the** smart-city deals-with **a** domain**(**2**) and the** domain**(**1**) is-not-the-same-as the** domain**(**2**) and the** domain**(**1**)** uses-resource **a** resource**(**1**) and the** domain**(**2**)** uses-resource **a** resource**(**2**) and the** resource**(**1**) is-the-same-as the** resource**(**2**) then the** smart-city **is a** resource-fragile-smart-city.

Wymagamy, aby management miasta (które decyduje się wdrożyć powyższy wzorzec) miał możliwość eskalacji problemów:

§48 **Every** management **that** manages **a** smart-city **that** implements Man-In-The-Loop-Resource-Resolver-Pattern **must be an** escalator-based-management.

§49 **Every** resource-fragile-smart-city **can** implement Man-In-The-Loop-Resource-Resolver-Pattern.

Określamy warunek, którego spełnienie spowoduje zbalansowanie współdzielonych zasobów miejskich.

§50 **Every** resource-fragile-smart-city **must be a** resource-balanced-smart-city.

Oraz zapewniamy o słuszności rozwiązania głoszonego przez ww. wzorzec. Miasto implementując ww. wzorzec będzie miastem wywarzonym.

§51 **Every** smart-city **that** implements Man-In-The-Loop-Resource-Resolver-Pattern **is a** resource-balanced-smart-city.

Oraz podobnie jak dla wzorca „Hierarchia Odpowiedzialności" wymagamy dostarczenia szczegółów implementacji rozwiązania:

§52 **Every** smart-city **that** implements Man-In-The-Loop-Resource-Resolver-Pattern **must** provide-implementation-details (**some** ontology-reference **value**).

Podsumowując: przedstawiliśmy powyżej język wzorców Smart-City oraz szczegółowo przedstawiliśmy dwa wyselekcjonowane przez nas wzorce. Wzorce te

zostały przez nas wyselekcjonowane ze względu na ich późniejsze użycie podczas walidacji języka wzorców. Walidację przeprowadzamy w następnej części.

7. Przykładowe zastosowanie języka wzorców Smart-City

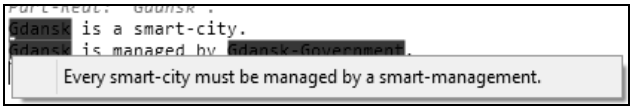
Przedstawiony wyżej formalizm można traktować, jako narzędzie pozwalające na dobór najlepszych praktyk-wzorców spośród pewnego katalogu. Elementem kluczowym wzorca (warunkującym jego istnienie) jest rozwiązanie, które ma do zaoferowania. Wzorce wskazują na te rozwiązania oraz przedstawiają ich specyfikację. Konkretna (rzeczywista) implementacja wzorca, aby być z nim zgodna, musi tę rekomendację spełniać. Inaczej mówiąc: „sztuką" w używaniu wzorców projektowych jest umiejętność ich doboru, oraz – co ważniejsze – umiejętność ich materializowania, urzeczywistniania. Rozpatrzymy poniżej przykład urzeczywistniania wyżej wymienionych wzorców.

Rozważmy przykładowe wdrożenie Smart-City w Gdańsku. Deklarujemy, co następuje:

§53 Gdansk **is** managed **by** Gdansk-Government.

§54 Gdansk **is a** smart-city.

Weryfikując tak otrzymaną wiedzę – Gdańsk mieni się na czerwono (rys. 10):



Rys. 10. Błędy weryfikacji zadeklarowanej potrzeby wdrożenia Smart-City dla miasta Gdańska

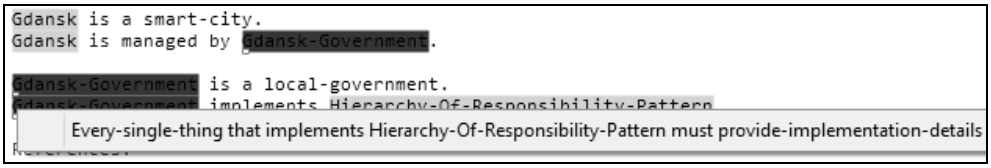
Okazuje się, zgodnie z wcześniejszymi ustaleniami, że: „***Every** smart-city **must be** managed **by a** smart-management*."

Wiemy, że:

§55 Gdansk-Government **is a** local-government.

§56 Gdansk-Government implements Hierarchy-Of-Responsibility-Pattern.

Ponowna walidacja wskazuje na potrzebę uzasadnienia implementacji wzorca (rys. 11).



Rys. 11. Deklaracja wdrożenia Smart-City dla miasta Gdańska po uzupełnieniu informacji o rodzaju wzorca przyjętego dla zarządzania miastem

Uzasadnienie przeprowadzimy w osobnym pliku opisującym hierarchię oraz wskażemy ten plik jako miejsce w którym znajduje się ww. uzasadnienie. Pierwszym krokiem jest właściwa lokalizacja zarządu Smart-City – w tym przypadku jest to Gdansk-Municipal-Crisis-Centre.

§57 Gdansk-Government **is** Gdansk-Municipal-Crisis-Centre[hier].

Idąc dalej:

§58 Voivodship-Executive-Board is-part-of Local-Government.
§59 Chief-Inspectorate-Of-Environmental-Protection is-part-of Voivodship.
§60 Inspectorate-For-Environmental-Protection.
§61 Directors-Of-Health-Care-Facilities is-part-of The-"NZOZ".
§62 Directors-Of-Health-Care-Facilities is-part-of Local-Government.
§63 Directors-Of-Educational-Institutions is-part-of Local-Government.
§64 Directors-Of-Care-Facilities is-part-of Local-Government.
§65 Airport-Fire-Brigade is-part-of Airport.
§66 City-Fire-Brigade is-part-of Provincial-Office.
§67 Airport-Medical-Care is-part-of Airport.

Tak zdefiniowana hierarchia wymaga również zdefiniowania sposobów eskalacji problemów. W tym celu powołamy dwa nowe byty:

§68 **Every** notified-body **is an** organization.
§69 **Every** notifying-body **is an** organization.

Oraz przypiszemy je do ww. hierarchii oraz do innych bytów biorących udział w pewnym scenariuszu:

§70 Directors-Of-Hospitals **is a** notified-body.
§71 Pilot **is a** notifying-body.
§72 Airport **is a** notified-body.
§73 Airline **is a** notified-body.
§74 Police **is a** notified-body.
§75 Directors-Of-Health-Care-Facilities **is a** notified-body.
§76 Directors-Of-Educational-Institutions **is a** notified-body.
§77 City-Fire-Brigade **is a** notified-body.
§78 Voivodship-Crisis-Centre **is a** notified-body.
§79 Voivodship-Executive-Board **is a** notified-body.
§80 Chief-Inspectorate-Of-Environmental-Protection **is a** notified-body.
§81 Local-Government **is a** notified-body.
§82 Local-Radio **is a** notified-body.
§83 Local-Television **is a** notified-body.

Powyższa implementacja wzorca zawarta jest w referowanej ontologii „hierarchy.encnl". Wspisując ww. fakt i poddając ontologię ponownej walidacji otrzymujemy zielone światło. Obecnie, wszystkie obligatoryjne wymagania nałożone na miasto oraz jego management są spełnione skoro tylko miasto implementuje wzorzec „Hierarchy of Responsibility" (rys. 12).

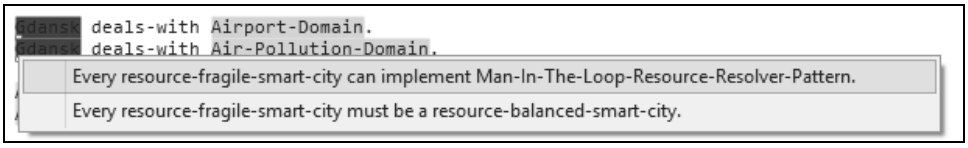
```
Gdansk-Government is a local-government.
Gdansk-Government implements Hierarchy-Of-Responsibility-Pattern.

Gdansk-Government provides-implementation-details equal-to 'hierarchy.encnl'.
```

Rys. 12. Deklaracja wdrożenia Smart-City dla miasta Gdańska po uzupełnieniu informacji o rodzaju wzorca przyjętego dla zarządzania miastem i wskazaniu sposobu jego implementacji

Następnie okazuje się, że w Gdańsku mamy do czynienia m.in. z lotniskiem oraz z problemem zanieczyszczeniem powietrza, a obie ww. domeny współdzielą zapotrzebowanie na zasób Helthcare-Facilities:

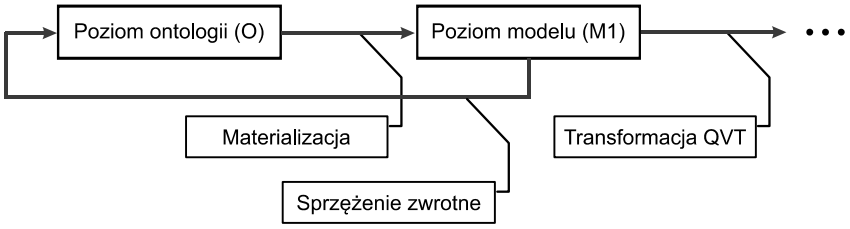
§84 Gdansk deals-with Airport-Domain.
§85 Gdansk deals-with Air-Pollution-Domain.
§86 Airport-Domain has-name **equal-to** "*Gdansk Airport Domain*".
§87 Air-Pollution-Domain has-name **equal-to** "*Gdansk Air Pollution Domain*".
§88 Airport-Domain uses-resource Healthcare-Facilities.
§89 Air-Pollution-Domain uses-resource Healthcare-Facilities.

A co za tym idzie Gdańsk staje on się *resource-fragile* (rys. 13):



Rys. 13. Dalsza weryfikacja wskazuje na występowanie konfliktu zasobów

Rozwiązaniem – sugerowanym przez walidator jest implementacja wzorca Man-In-The-Loop-Resource-Resolver-Pattern. Protokoły postępowania wynikają z odpowiednich ustaw i są związane z odpowiednimi domenami. Protokoły opisują procesy, które najlepiej przedstawić w języku modelowania procesów. W tym miejscu wracamy do poziomu modelu IBM IOC, jako, że procesy modelowane są na poziomie M1 MOF – a więc najlepiej, jeśli jest wspierany przez narzędzie pozwalające na modelowanie w BPMN, jakim jest IBM Business Modeller. Okazuje się, w tym miejscu, że proces powstawania konfiguracji IOC nie jest procesem jednokierunkowym, lecz procesem działającym w pętli sprzężenia zwrotnego (rys. 14**Rys.**).



Rys. 14. Sprzężenie zwrotne w procesie modelowania powstałe ze względu na potrzebę modelowania procesów

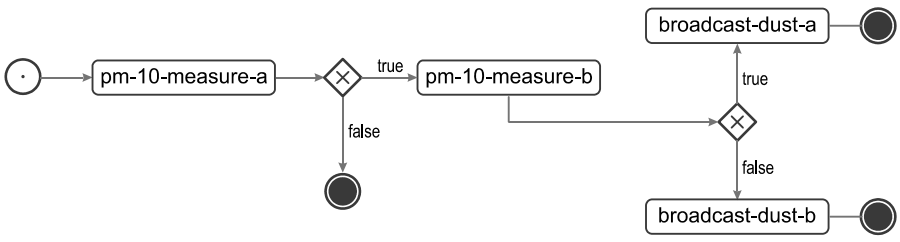
Oczywiście protokół ten jest równoważny z następującym (całkowicie nieczytelnym) zapisem w CNL:

§90 Ev-000 **is a** start-event.
Act-001 **is a** activity-pm-10-measure-a.
Act-003 **is a** activity-pm-10-measure-b.
…
Act-019 **is a** activity-send-ambulances.
Ev-020 **is a** end-event.
Ev-021 **is a** start-event.
…
Ev-007 follows-if-false Act-001.
Act-005 follows-if-true Act-003.
Act-006 follows-if-false Act-003.
…
Ev-020 follows Act-019.
Act-022 follows Ev-021.
Act-023 follows Act-022.
…

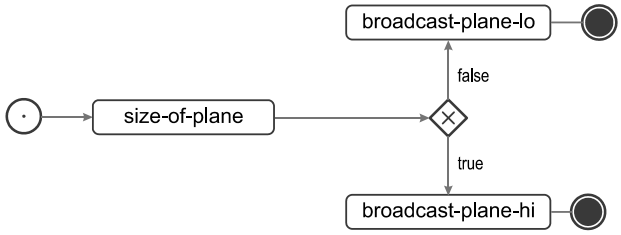
Język naturalny okazuje tu swoją słabość stereopizowaną przysłowiem „jeden obraz więcej jest wart niż tysiąc słów". W tym przypadku przysłowie to wydaje się prawdziwe. Dowodzimy tutaj istnienia granicy interfejsu bazującego na języku naturalnym – są to granice użyteczności, granice odnajdywane na nowo. Richard Riehle (2006) stwierdza wręcz: *„look back on the fascination we all had with our flowcharting as a gigantic waste of time. Yet, there was a kernel of a linguistic idea in those templates that survives today.*". Niestety diagram mają zasadniczą wadę – reprezentują sieci – grafy – związki między bytami, podczas gdy zdania w języku naturalnym reprezentują prawa rządzące światem – reprezentują prawdę, których realizacją (materializacją) są grafy.

Poniżej przedstawiamy diagramy:

— protokołu postępowania w przypadku przekroczenia zawartości pyłu PM10 (rys. 15),
— protokołu postępowania w razie lądowania awaryjnego samolotu w zależności od jego rozmiaru (rys. 16)



Rys. 15. Protokół postępowania w przypadku przekroczenia zawartości pyłu PM10 w notacji BPMN



Rys. 16. Protokół postępowania w przypadku lądowania awaryjnego w zależności od wielkości samolotu

Konstruując "hierarchię odpowiedzialności" musieliśmy wyznaczyć pewien "szlak" eskalacji problemów. Ten szlak jest podstawą do umiejscowienia osoby, która może podejmować decyzje w przypadku konfliktu zasobów. Zakładamy, że osoba ta postępuje racjonalnie. Skoro jednak podejmuje ona racjonalne decyzje, to mogą one zostać podmienione przez decyzje automatyczne. Dowodzenie tej tezy wymaga użycia systemów zarządzania wiedzą niemonotoniczną (pozwalające na usuwanie wniosków w zależności od przesłanek), a co za tym idzie wykraczających poza zakres tego artykułu, warto jednak przytoczyć przykłady reguł eksperymentalnych zapisanych w formalizmie niemonotonistycznym. I tak dla naszego przykładu:

§91 **If an** agent has-started-activity **an** activity **and the** activity **is an** activity-pm-10-measure-a **and the** agent has-pm-10-measurement **equal-to the value**(1) **then for the** agent **and the** activity **and the value**(1) **execute**.

```
KnowledgeInsert("Comment:\""+agent + " is executing " + activity+"\'.");
KnowledgeDelete(agent +" has-started-activity " + activity+".");
KnowledgeInsert(agent +" has-condition equal-to
"+(value[1]>=50?"true":"false")+".");
KnowledgeInsert(agent +" has-finished-activity " + activity+".");
```

§92 **Every** activity-pm-10-measure-b **is an** activity.

§93 **If an** agent has-started-activity **an** activity **and the** activity **is an** activity-pm-10-measure-b **and the** agent has-pm-10-measurement **equal-to the value**(1) **then for the** agent **and the** activity **and the value**(1) **execute**

```
KnowledgeInsert("Comment:\""+agent + " is executing " + activity+"\'.");
KnowledgeDelete(agent +" has-started-activity " + activity+".");
KnowledgeInsert(agent +" has-condition equal-to
"+(value[1]>=300?"true":"false")+".");
KnowledgeInsert(agent +" has-finished-activity " + activity+".");
```

§94 **Every** activity-size-of-plane **is an** activity.

§95 **If an** agent has-started-activity **an** activity **and the** activity **is an** activity-size-of-plane **and the** agent has-size-of-plane **equal-to the value**(1) **then for the** agent **and the** activity **and the value**(1) **execute**

```
KnowledgeInsert("Comment:\""+agent + " is executing " + activity+"\'.");
```

```
KnowledgeDelete(agent +" has-started-activity " + activity+".");
KnowledgeInsert(agent +" has-condition equal-to
"+(value[1]>=150?"true":"false")+".");
KnowledgeInsert(agent +" has-finished-activity " + activity+".").
```

Kończąc nasze rozważania oraz zapisując powyższe rozwiązanie w pliku 'man-in-the-loop.encnl' oraz deklarując w naszej ontologii ww. rozwiązanie, jako wyjaśnienie implementacji wzorca, walidator znowu daje nam zielone światło (rys. 17).

```
Gdansk implements Man-In-The-Loop-Resource-Resolver-Pattern.
Gdansk provides-explanation equal-to 'man-in-the-loop.encnl'.
```

Rys. 17. Deklaracja wdrożenia Smart-City dla miasta Gdańska po uzupełnieniu informacji o rodzaju wzorca przyjętego dla rozwiązywania konfliktu zasobów i wskazaniu sposobu jego implementacji

Podsumowując, w tej części przedstawiliśmy walidację wcześniej wprowadzonego przez nas języka wzorców. Polegała stworzeniu ontologii konfiguracji szyny integracyjnej IBM-IOC na podstawie ww. języka wzorców. Pokazano ona na pełny cykl pracy z językiem wzorców. Okazało się, że model kaskadowego przejścia z poziomu ontologii (O) do poziomu modelu (M1) warto zmienić dodając pętlę sprzężenia zwrotnego, co pozwoliło uzupełnić spektrum narzędzi o odpowiednią transformację.

8. Krytyka wzorców projektowych

Krytykę wzorców projektowych prowadzona m.in. przez Petera Norwiga[2], mówi o wzorcach projektowych jako o rozwiązaniach problemów generowanych przez sam język (w którym są zapisane lub o którym traktują). Inaczej mówiąc – język wzorców stara się zniwelować problemy języka do którego dociera – czyli rozwiązuje niedoskonałości, które sam język powinien wyeliminować. Norwig operuje w świecie oprogramowania komputerowego, lecz ma tutaj dużo racji przedstawiając większość wzorców GOF jako „problemów sztucznych" w językach takich jak LISP czy Dylan, które tego typu problemów nie generują. Wydaje się jednak, że Norwig przez to, że jego krytyka dotyczyła wzorców dotyczących oprogramowania, zaniedbuje atrybuty wzorca i utożsamia wzorzec z rozwiązaniem. Wzorzec natomiast pozostawał i pozostaje jedynie sposobem wymiany doświadczeń na temat rozwiązań powtarzalnych problemów. Z tego punktu widzenia język programowania jest istotną częścią kontekstu wzorca, a co za tym idzie krytyka prowadzona przez Norwiga może być odparta stwierdzeniem – kontekst wzorców GOF nie zawiera języków takich jak LISP czy Dylan.

W naszym rozumieniu wzorzec jest sposobem dystrybucji doświadczenia, jego specyficzna forma oraz liczba wdrożeń idei wzorców projektowych każe nam są-

[2] http://norvig.com/design-patterns/

dzić, że pod atrybutami wzorca kryje się uniwersalna metoda. Co więcej, wprowadzony przez nas język wzorców *Smart City* ma postać formalnych ontologii, w których można dowodzić, mimo to pozostaje jedynie sposobem wymiany doświadczeń o rozwiązaniach najczęściej występujących problemów. Sposobem – co staramy się wykazać – wartym dalszych badań.

9. Podsumowanie

Projektowanie Smart-Cities jest zagadnieniem urbanistycznym w swojej istocie, ponieważ struktury miejskie, przedmiot badań urbanistyki pozwalają na opracowanie koncepcji planistycznych. Projekt wdrażania *Smart City* jest realizacją koncepcji planistycznej więc wpisuje się w ww. definicję. Proponowane przez nas wzorce projektowe *Smart City* są analogicznie do wzorców projektowych w urbanistyce. Inaczej mówiąc: architektura danego wdrożenia szyny integracyjnej *Smart Cities*, podążając za architekturą miasta, w którym jest wdrażana, pozostaje architekturą systemu informatycznego. Jest ona zatem wirtualnym rozszerzeniem danego miasta. Stosując odkrycie Christofera Aleksandra w kontekście wdrażania szyny integracyjnej *Smart City*, zataczamy więc koło – wzorce znowu dotyczą miast, tym razem operują jednak na wirtualnym modelu/rozszerzeniu miasta.

W tym artykule przedstawiliśmy rozwiązanie problemów związanych z ponownym użyciem, niezwykle cennej wiedzy, która powstaje podczas wdrażania szyny integracyjnej *Smart City*. Proponujemy w tym miejscu zastosowanie naturalnego (w języku naturalnym), dopasowanego do problematyki, formalnie zdefiniowanego oraz wyposażonego w odpowiednie narzędzia języka wzorców *Smart City*. Co ważne, język ten daje bardzo silne narzędzie organom odpowiedzialnym za decyzje – narzędziem tym jest możliwość weryfikacji rozwiązania przez systemy dowodzenia twierdzeń w logice. Metody formalne, bazujące na systemach dowodzenia twierdzeń są najsilniejszym narzędziem pomocnym w budowaniu zaufania do proponowanego rozwiązania.

Z drugiej strony, skoro podejmowanie decyzji (dotyczącej wyboru konfiguracji szyny integracyjnej *Smart City*) może być wspierane komputerowo, stawiamy pytanie: czy racjonalne decyzje może w ww. przypadkach podejmować automat? Niestety dla nas ludzi odpowiedź wydaje się pozytywna. Maksimum rozumienia „na raz" wiedzy przez istotę ludzką wydaje być się wskazywane przez problem „prowadzenia samochodu". Prowadzenie samochodu wymaga wiedzy z zakresu kodeksu drogowego, umiejętności stosowani tej wiedzy oraz umiejętności szybkiego podejmowania decyzji operacyjnych. Z drugiej strony – co wykazuje noblista Daniel Kahneman (2011) myślenie analityczne jest dla ludzi bardzo kosztowne energetycznie. Przyswajanie przez człowieka „całej" wiedzy w danej domenie dociera do pewnej granicy określonej przez czas wymagany na jej zrozumienie, a pełna głębia rozumienia staje się w pewien sposób nieosiągalna. Pewnym dowodem na to mogą być matematycy, rozumują oni głęboko, lecz sama matematyka (choćby w porównaniu z naukami miękkimi) rozwija się wolno. Teoretycznie, dla komputerów czas osiągnięcia głębi rozumienia jest wielokrotnie krótszy, a przetwarza-

nie wiedzy wydaje się ograniczone jedynie przez prawa logiki. Tutaj komputery uzyskują nad nami przewagę i pomysł wyposażenia miast inteligentnych *Smart City* w rozwiązania typu IBM Watson wydaje się interesujący.

Źródła

1. Alexander C., Ishikawa S., Silverstein M.: (1977). *A Pattern Language: Towns*, *Buildings*, *Construction* (*Center for Environmental Structure Series*). Oxford University Press.
2. Bhowmick A.: (2012). *IBM Intelligent Operations Center for Smarter Cities Administration Guide.* IBM Corporation, International Technical Support Organization.
3. Cognitum (2014). *Fluent Editor 2014 – Ontology Editor*. Pobrano z lokalizacji: http://www.cognitum.eu/semantics/FluentEditor/ [data dostępu: 5.09.2015].
4. Fowler M.: (2003). *Patterns of Enterprise Application Architecture*. Addison-Wesley.
5. Gamma E., Helm R., Johnson R.I., Vlissides J.: (1995). *Design patterns: elements of reusable object-oriented software*. Boston: Addison-Wesley Longman Publishing Co., Inc.
6. Glimm B., Horridge M., Parsia B., Patel-Schneider P.F.: (2008). A syntax for rules in OWL 2. R. Hoekstra I P.F. Patel-Schneider (red.), OWLED, volume 529 of *CEUR Workshop Proceedings*. CEUR-WS.org.
7. Goczyła K.: (2011). *Ontologie w systemach informatycznych*. Akademicka Oficyna Wydawnicza EXIT.
8. Hitzler P., Krötzsch M., Parsia B., Patel-Schneider P.F., Rudolph S.: (2009). *OWL 2 Web Ontology Language Primer. W3C Recommendation.* World Wide Web Consortium.
9. Horrocks I., Kutz O., Sattler U.: (2006). The even more irresistible sroiq. P. Doherty, J. Mylopoulos i C.A. Welty (red.), KR (57–67). AAAI Press.
10. Kahneman D.: (2011). *Thinking, fast and slow*. New York: Farrar, Straus and Giroux.
11. Kuhn T.: (2009). How to Evaluate Controlled Natural Languages. N.E. Fuchs (red.), *Pre-Proceedings of the Workshop on Controlled Natural Language* (*CNL 2009*), volume 448 of *CEUR Workshop Proceedings*. CEUR-WS.
12. Meszaros G., Doble J.: (1997). Pattern languages of program design 3. Chapter A. *Pattern Language for Pattern Writing* (529–574). Boston: Addison-Wesley Longman Publishing Co., Inc.
13. Musen M., Noy N., Nyulas C., O'Connor M., Redmond T., Tu S., Tudorache T., Vendetti J.: (2010). *Protégé*. Stanford School of Medicine. Pobrano z lokalizacji: http://protege.stanford.edu [data dostępu: 5.09.2015].
14. OMG (2011). *Business Process Model and Notation* (*BPMN*), Version 2.0.
15. Riehle R.: (2006). Linguistic continuity in software engineering. *ACM SIGSOFT Software Engineering Notes*, 31(1) (1–5).
16. Rozenberg G.: (1997). *Handbook of graph grammars and computing by graph transformation: volume I. Foundations*. New York: World Scientific Publishing Co. Inc., River Edge.

ONTOLOGY OF THE DESIGN PATTERN LANGUAGE FOR SMART CITIES SYSTEMS

The paper presents the definition of the design pattern language of Smart Cities in the form of an ontology. Since the implementation of a Smart City system is difficult, expensive and closely linked with the problems concerning a given city, the knowledge acquired during a single implementation is extremely valuable. The language we defined supports the management of such knowledge as it allows for the expression of a solution which, based on best practices recorded in the form of design patterns, is also tailored to the requirements of the city seeking to implement the Smart City solution. The formal/ontological structure of the language in turn allows the automatic management of the properties of a solution recorded in this way. This final feature of the introduced language is extremely important in the decision-making process regarding the choice of a particular solution by the relevant authorities. The work is divided into five main parts. In the first part we discuss the implementation issue of the integration bus using the example of the IOC. In the next part we talk about the validity of using semantic technologies in order to expand the spectrum of potential implementations. Then we discuss the ontological implementation of the Smart City pattern language which we created, a language which allows for both the saving of requirements and the validation of solutions specified in it. We also present an example of usage, which at the same time serves as a validation of the language in real-life conditions. In the last part we discuss certain aspects of the pattern language and the possible ways to develop research related to it..

Key words: Smart Cities, ontologies, semantics, Ontology Driven Architecture, Design Patterns, Controlled Natural Language.