AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI, INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ

KATEDRA INFORMATYKI STOSOWANEJ

Praca dyplomowa inżynierska

Projekt i implementacja oprogramowania generującego modele procesów i decyzji na podstawie diagramów zależności między atrybutami.

Design and Implementation of Software for Process and Decision Models Generation based on Attribute Relationship Diagrams.

Autor: *Karol Grzesiak*
Kierunek studiów: *Informatyka*
Opiekun pracy: *Dr inż. Krzysztof Kluza*

Kraków, 2019

Uprzedzony o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.): „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystycznego wykonania albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie.", a także uprzedzony o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (t.j. Dz. U. z 2012 r. poz. 572, z późn. zm.): „Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godności studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwanym dalej «sądem koleżeńskim».", oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

Chciałbym serdecznie podziękować Doktorowi Krzysztofowi Kluzie za profesjonalizm oraz wsparcie merytoryczne podczas pisania niniejszej pracy. Dodatkowo kieruję podziękowania dla całej rodziny oraz przyjaciół za cierpliwość w tym okresie.

Spis treści

1. Wprowadzenie

BPMN (*Business Process Model and Notation* [1]) oraz DMN (*Decision Model and Notation* [2]) są powszechnie przyjętymi standardami służącymi modelowaniu, opisywaniu oraz zarządzaniu procesami biznesowymi. BPMN dotyczy głównie graficznej reprezentacji procesów, natomiast DMN skupia się na enkapsulacji logiki decyzyjnej (zasad organizacji).

Model procesu zbudowany przy użyciu tych dwóch standardów posiada bardzo potężną siłę ekspresji i może przynieść wiele korzyści dla systemów zarządzania wiedzą. Problem pojawia się jednak przy modelowaniu. Zajmują się tym głównie analitycy biznesowi którzy, korzystając z zasobów biznesowych, używają swojego doświadczenia oraz wiedzy. Niestety sam przebieg działania bywa trudny do dokładnego zdefiniowania, przez co może być odbierany jako sławny efekt *ATAMO*[1]. Innymi słowy, proces prototypowania modelów procesów nie jest łatwym zadaniem.

Jednym z możliwych rozwiązań jest wykorzystanie ARD (*Attribute Relationship Diagrams* [3]) - metody reprezentacji wiedzy dla ustrukturyzowanej specyfikacji systemu - do stworzenia prototypów modelów procesów, a następnie na tej podstawie wygenerowania odpowiednich elementów BPMN oraz DMN[2]

1.1. Motywacja

Mając na uwadze szybki rozwój - w ostatnich latach - zarządzania procesami biznesowymi oraz związany z tym zbiór problemów i proponowanych rozwiązań, wydaję się to być tematem wartym pochylenia, a co za tym idzie, wartym próby implementacji.

Przyjmując, że spośród wielu proponowanych metod prototypowania, metoda *ARD* zyskała na największej popularności w wyniku czego analitycy biznesowi używają jej do stworzenia pierwszych prototypów, a standardy *BPMN* oraz *DMN* nadal królują jako podstawowa oraz najbardziej zrozumiała forma reprezentacji procesów, to łatwo zauważyc, że do pełni szczęścia

[1] "And then a miracle occurs" - fraza spopularyzowana przez kreskówki Sidney Harris, często używana w pracach związanych z BPM aby opisać działania, które występują ale są trudne do zdefiniowania

[2] *From Attribute Relationship Diagrams to Process (BPMN) and Decision (DMN) Models* [4]

potrzebny jest jeszcze jeden element. Element będący pewną luką, którą można łatwo zautomatyzować. Potrzebny jest system przekształcający prototypy *ARD* zapisane w plikach HML[3] (*Hekate Markup Language* [5]) do reprezentacji w standardach *BPMN* oraz *DMN*, który następnie transportuje takie modele do zewnętrznego systemu posiadającego silniki procesowe oraz decyzyjne co umożliwia ich uruchomienie oraz ewaluację.

1.2. Cel pracy

W ramach pracy zaprojektowana oraz zaimplementowana została aplikacja internetowa "DAR", będąca generatorem modeli procesów biznesowych w notacji *BPMN* oraz decyzji w notacji *DMN* na bazie specyfikacji w postaci plików *HML* zawierających diagramy zależności między atrybutami *ARD*. System został zintegrowany z wybranym środowiskiem wspierającym zarządzanie procesami biznesowymi oraz ewaluację decyzji, dzięki czemu umożliwia wygenerowanie modelu, a następnie deployment na opisywane środowisko. Umożliwiona została opcja uzupełnienia niezbędnych informacji, aby następnie możliwe było uruchomienie wybranego obiektu, analiza działania i obserwacja wyników.

1.3. Strukutra pracy

Opis sekcji:

- **Sekcja 2** – zgłębia podłoże teoretyczne, zapewniając informację związane z przetwarzaniem procesów biznesowych. Dokładniej opisuje pojęcia takie jak *BPMN*, *DMN*, *ARD* oraz wprowadza podstawowe idee i koncepty związane z tą domeną.
- **Sekcja 3** – opisuję architekturę systemu w ujęciu abstrakcyjnym. Skupia się na schemacie działania, wydzielając odpowiednie komponenty.
- **Sekcja 4** – pokazuję aplikację od strony implementacyjnej. Omawia wybrane technologię, prezentuję interfejs, stos technologiczny oraz jak system został wykonany.
- **Sekcja 5** – prezentuję działanie całej aplikacji na pewnym przykładzie, omawiając jednocześnie jej przebieg.
- **Sekcja 6** – kończy całą pracę, podsumowując działanie systemu i opisując zebrane spostrzeżenia oraz wnioski. Przedstawia możliwy kierunek rozwoju aplikacji.

[3] Mówiąc o plikach HML, będę miał na myśli wersję *ARDML*, jednak z opcjonalnym fragmentem *TPH*. Więcej pod adresem: *https://ai.ia.agh.edu.pl/wiki/hekate:hml1*

2. Podłoże teoretyczne

2.1. Proces biznesowy

Seria działań lub zadań, które produkują konkretny rezultat. W wielu organizacjach niezdefiniowane lub nieformalne, co często prowadzi do nieefektywności oraz "wąskich gardeł". Innymi słowy proces biznesowy to działanie, które skutkuje osiągnieciem jakiegoś celu. Najlepiej myśleć o nim jako o przepływie zadań, który zawiera początek, środek i koniec. Proces biznesowy często jest przekrojowy – dotyka wielu różnych obszarów.

2.2. Business Process Model and Notation

BPMN (*Business Process Model and Notation*) jest notacją stworzoną przez OMG (*Object Managment Group*[1]) w celu graficznej reprezentacji procesów biznesowych za pomocą diagramów stworzonych przy użyciu schematów blokowych - bardzo zbliżonych do UML (*Unified Modeling Language* [6]). Głównym motorem napędowym do stworzenia tej specyfikacji była potrzeba pewnego medium zrozumienia dla wszystkich użytkowników biznesowych. Dlatego *BPMN* jest zrozumiałe dla wszystkich interesariuszy, poczynając od analityka biznesowego tworzącego pierwsze szkice procesu, aż po deweloperów, którzy następnie implementują technologię odpowiedzialną za wykonywanie tych procesów. Innymi słowy jest to swoisty most łączący świat projektowania i implementacji.

BPMN 2.0 [1] jest w tym momencie najnowszą wersją notacji i zapewnia ona cztery różne typy diagramów aby w pełni opisać różne aspekty procesów biznesowych:

- **Conversation Diagram**
- **Collaboration Diagram**
- **Choreography Diagram**

[1] *https://www.omg.org/*

– **Process Diagram**

Z perspektywy tej pracy, najbardziej interesujący jest ostatni typ diagramów - *Business Process Diagram*. Jest on najbardziej podstawowym typem. Reprezentuje całe flow danego procesu - w jakiej kolejności będą wykonywane zadania, podział procesu na osobne flow lub eventy i unikatowe sytuacje, które mogą się wydarzyć podczas wykonywania procesu. Składa się z elementów, które należą do jednej z podstawowych kategorii:

– **Flow Objects** – określają zachowanie procesu biznesowego.

  – **Events** – reprezentują zdarzenia podczas procesu. Przeważnie występują trzy główne typy: Początkowe, Środkowe oraz Finalne.

  – **Activities** – zwane również jako Tasks, czyli praca wykonywana podczas procesu. Może to być przykładowo Task odpowiedzialny za podjęcie decyzji (połączony z tabelą decyzyjną w notacji *DMN*). Jest to element wykonywalny w procesie.

  – **Gateways** – determinują zbieżność lub rozbieżność ścieżki (flow) procesu. Ich reprezentacja graficzna posiada dodatkowe znaczniki, mówiące o tym w jaki sposób flow jest kontrolowane:

    * **Parallel Gateways** - odpowiednik funkcji “AND”.
    * **Inclusive Gateways** - odpowiednik funkcji “OR”.
    * **Exclusive Gateways** - odpowiednik funkcji “XOR”.

– **Swimlanes** – Swimlane określa odpowiedzialność danego procesu, natomiast Pool określa jego uczestnika. Generalnie prezentują dany scope.

  – **Flow Elements** – ich połączenie różni się w zależności czy są połączone wewnątrz jednego uczestnika czy pomiędzy paroma.

  – **Message Flows** – tylko one mogą być użyte do komunikacji między uczestnikami procesu.

  – **Pool** – nie może posiadać więcej niż jednego procesu.

  – **Sequence Flows** – nie powinny być używane pomiędzy uczestnikami. Lanes są bardziej odpowiednie do użycia Sequence Flows.

– **Data** – określają informacje o Activities. Informacje są albo przechowywane albo dostarczane do Activity.

  – **Data Objects**

  - **Data Inputs**
  - **Data Outputs**
  - **Data Stores** – zasób z którego proces może odczytywać lub do którego może zapisywać informacje.

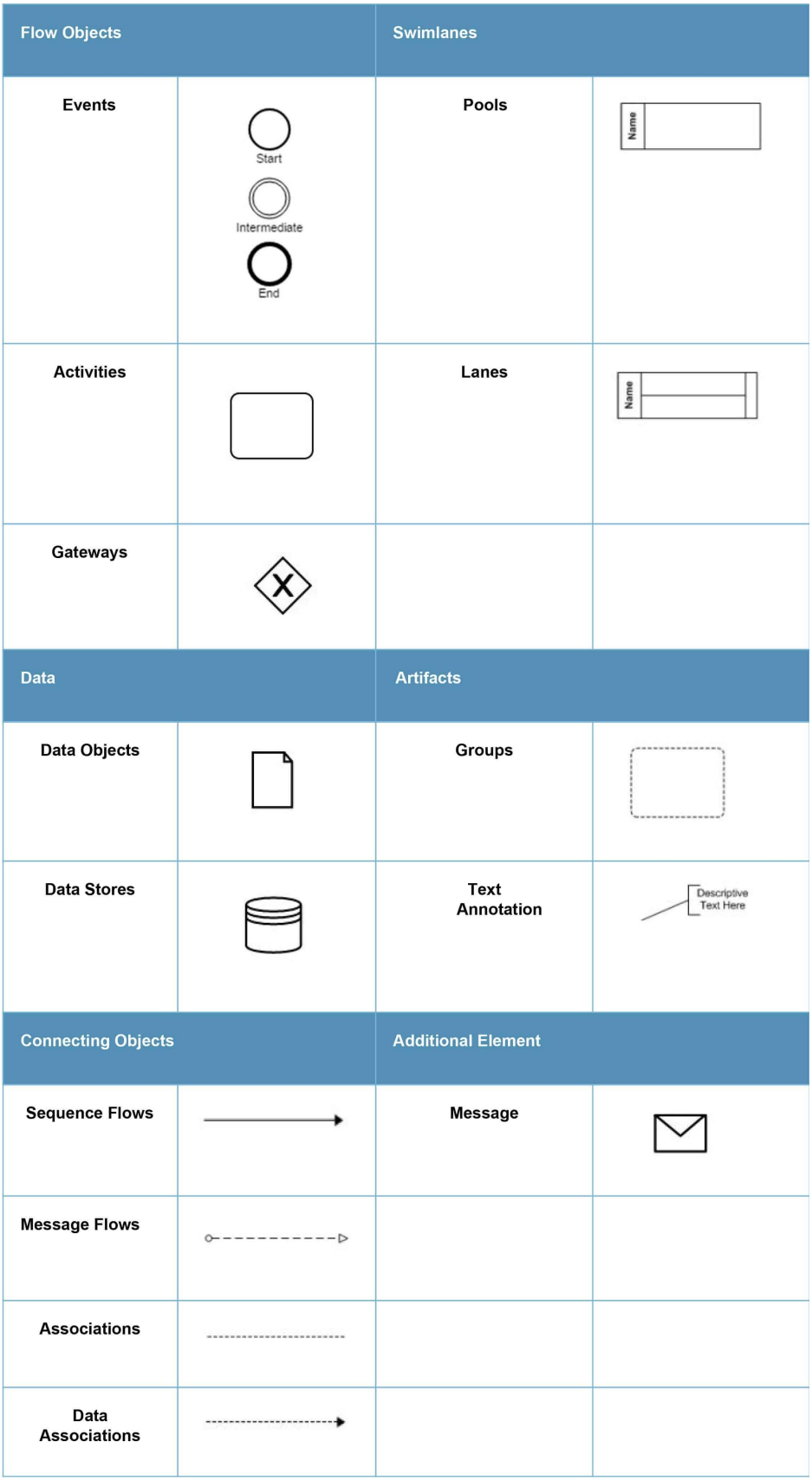
- **Artifacts** – dodają dodatkowe informacje na temat procesu.
  - **Groups** – grupują elementy aby pokazać ich pokrewność/relację.
  - **Text Annotations** – dodatkowy tekst, dodający pewną dodatkową informację - rodzaj komentarza.

- **Connecting Objects** – obiekty odpowiedzialne za łączenie elementów z kategorii Flow Objects innych informacji.
  - **Sequence Flows** – pokazują kolejność w której Activites będą wykonywane.
  - **Message Flows** – prezentują kolejność informacji między elementami biorącymi udział.
  - **Associations** – łączą ze sobą informację oraz Artifacts.
  - **Data Associations** – posiadają końcówkę w formie strzałki i wskazują kolejność flow w Association

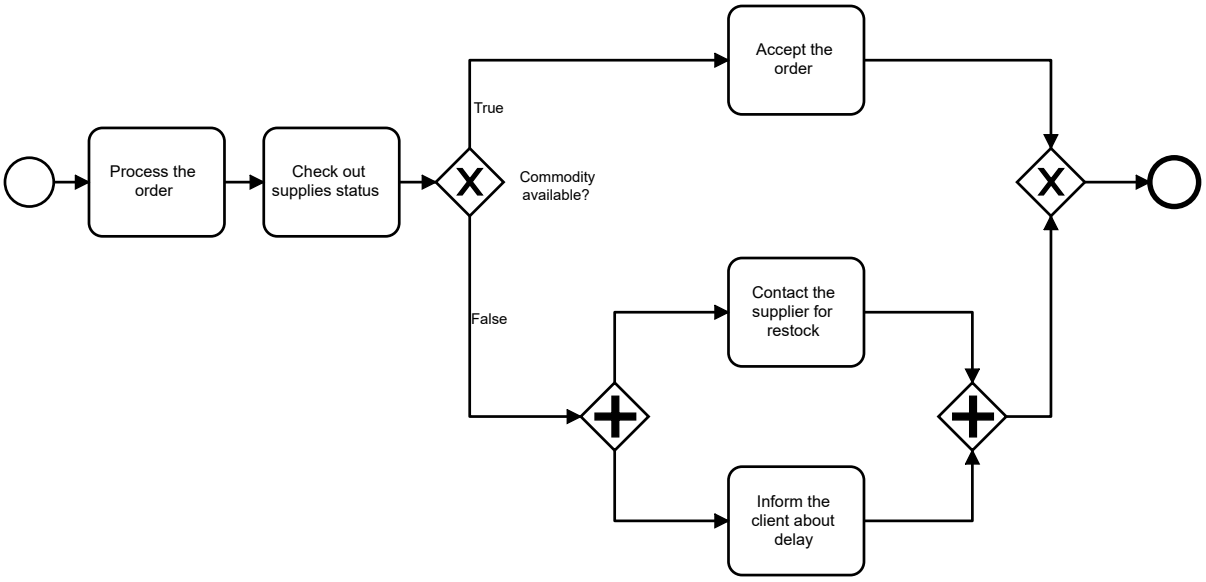
- **Message** – element nie należący do żadnej z konkretnych kategorii. Reprezentuje komunikację pomiędzy uczestnikami.

Grafika 2.1 przedstawia graficzną reprezentację opisanych elementów *BPMN*.

Grafika 2.2 pokazuję przykładowy model procesu w notacji *BPMN*. Łatwo zauważyć, że już na pierwszy rzut oka jest on generalnie zrozumiały i łatwy do analizowania. Ilustruje mały wycinek procesu związanego obsługą złożonego zamówienia przez klienta. Zaczynając uruchamiany jest Event startowy, który rozpoczyna cały proces. Następnie wykonywane są po kolei dwa elementy wykonywalne (Tasks), a dokładniej ”Process the order” oraz ”Check supplies status”. W zależności od danych, które dostarczą te elementy, Inclusive Gateway rozdziela ścieżkę na dwie możliwe. Ścieżka górna posiada jeden Task wykonywalny, natomiast na ścieżce dolnej znowu dochodzi do podziału tym razem za pomocą Parallel Gateway, gdzie wszystkie następujące elementy są wykonywane równolegle. Finalnie ścieżki łączą się i proces jest zakończony przez Event końcowy.



Rys. 2.1. Graficzna reprezentacja podstawowych elementów *BPMN*. Grafika za pośrednictwem artykułu [7].



Rys. 2.2. Przykładowy proces *BPMN*.

2.3. Decision Model and Notation

DMN (*Decision Model and Notation*) jest notacją stworzoną przez OMG (*Object Managment Group*) w celu prostego opisu oraz modelowania zasad występujących w procesach/organizacjach. Tak samo jak w przypadku *BPMN* główną zaletą oraz założeniem tego standardu było umożliwienie prostej drogi komunikacji dla użytkowników z dziedzin biznesu oraz IT. Dzięki niemu bezproblemowa staje się kolaboracja pomiędzy ludźmi biznesu monitorującymi i zarządzającymi decyzjami, analitkami biznesowymi, którzy szkicują wstępne wymagania decyzyjne wraz z logiką decyzyjną oraz technicznymi deweloperami odpowiedzialnymi za implementację systemów, które te decyzje podejmują. Standard ten bezproblemowo może być używany jako samodzielny, jednak najczęściej towarzyszy on notacji *BPMN*. *BPMN* definuje specjalny Business Rule Task, który za kulisami połączony jest do odpowiedniego obiektu w notacji *DMN*. Dzięki temu zasady biznesowe mogą być wcielone do wykonywalnych procesów i wraz z działaniem otrzymywać dane oraz odpowiednio ewaluować decyzję aby zwracać wyniki.

DMN 1.2 [2] jest w tym momencie najnowszą wersją notacji. Definiuje cztery podstawowe elementy:

– **Decision Requirements Diagram** – diagram pokazujący zależności między elementami w notacji *DMN*. Określa wymagania dla logiki decyzyjnej.

– **Business Context** – kontekst decyzji, przykładowo jak duży wpływ mają decyzję w kontekście wydajności.

– **FEEL (*Friendly Enough Expression Language*)** – język służący do określania zasad biznesowych w tabelach decyzyjych lub innych logicznych formatach.

– **Decision logic** – logika będąca wewnątrz obiektów występujących w *Decision Requirements Diagram*. W kontekście tej pracy głównie będzie się to odnosić do tabel decyzyjnych.

Decision Requirements Diagram jest zbudowany bardzo podobnie do diagramu w notacji *BPMN*. Podstawowe elementy z których się składa to:

– **Decision** – determinuje wynik na podstawie pewnej ilości danych poprzez aplikowanie logiki decyzyjnej.

– **Input Data** – zewnętrzne dane, nie pochodzące z Business Context.

– **Knowledge Source** – źródła określające pewne zasady co do podejmowania decyzji. Przykładem może być tutaj pewna polityka firmy lub regulcje.

– **Business Knowledge** – pewne funkcje które enkapsulują logikę decyzyjną (zasady biznesowe, tablice decyzyjne). Łatwe do wielokrotnego użycia.

– **Information Requirements** – łączą dane z decyzjami.

– **Knowledge Requirements** – łączą wiedzę biznesową z decyzjami.

– **Authority Requriements** – łączą źródła wiedzy z decyzjami.

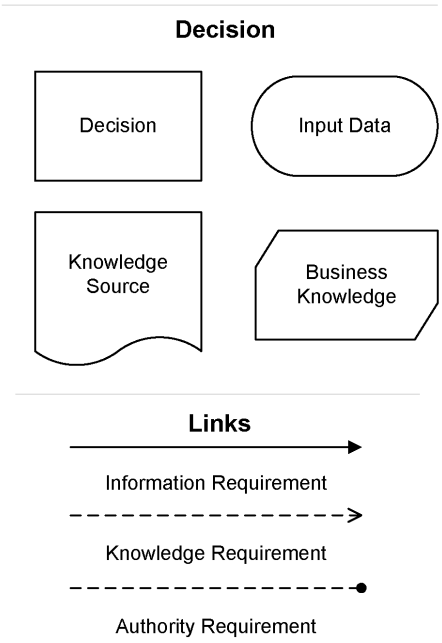
Grafika 2.3 przedstawia graficzną reprezentację opisanych elementów.

Grafika 2.4 opisuję przykładową tablicę decyzyjną. Warto zauważyc użycie *FEEL* - przykładowo w kolumnie OrderSize, można w prosty sposób uzależnić zwracaną wartość od tego czy dostarczone dane będą mniejsze lub większe od liczby dziesięć.

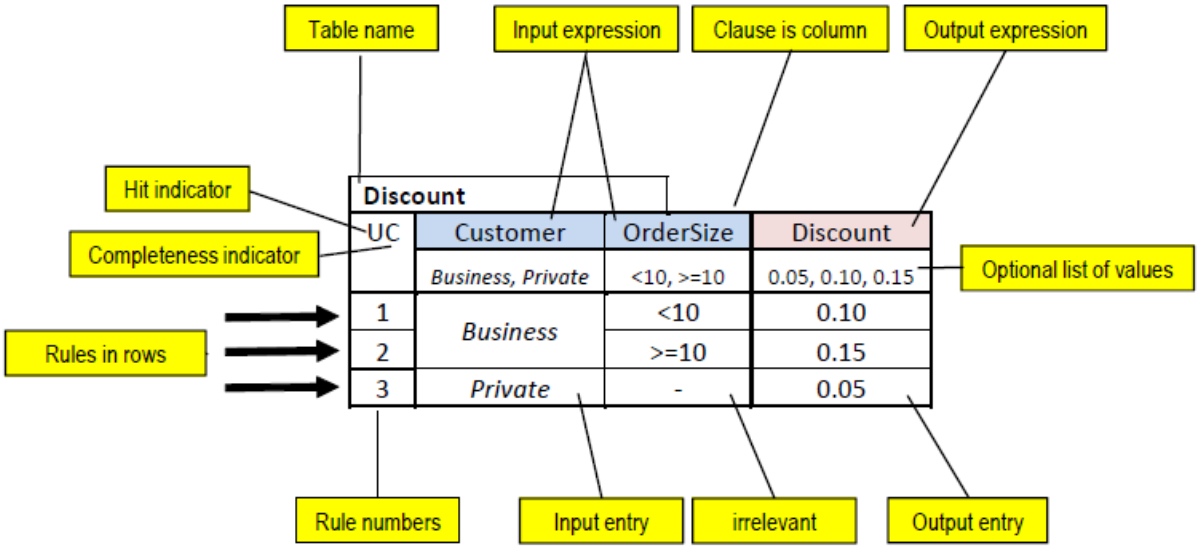
Grafika 2.5 prezentuje jak w praktyce wygląda integracja opisanych standardów. Task w procesie zbudowanym za pomocą notacji *BPMN*, jest połączony z decyzją w notacji *DMN*. Ta z kolei posiada pewną logikę - zasady biznesowe wyrażone za pomocą tabeli decyzjnych i na tej podstawie, podczas działania procesu, ewaluuje decyzję.

2.4. Attribute Relationship Diagrams

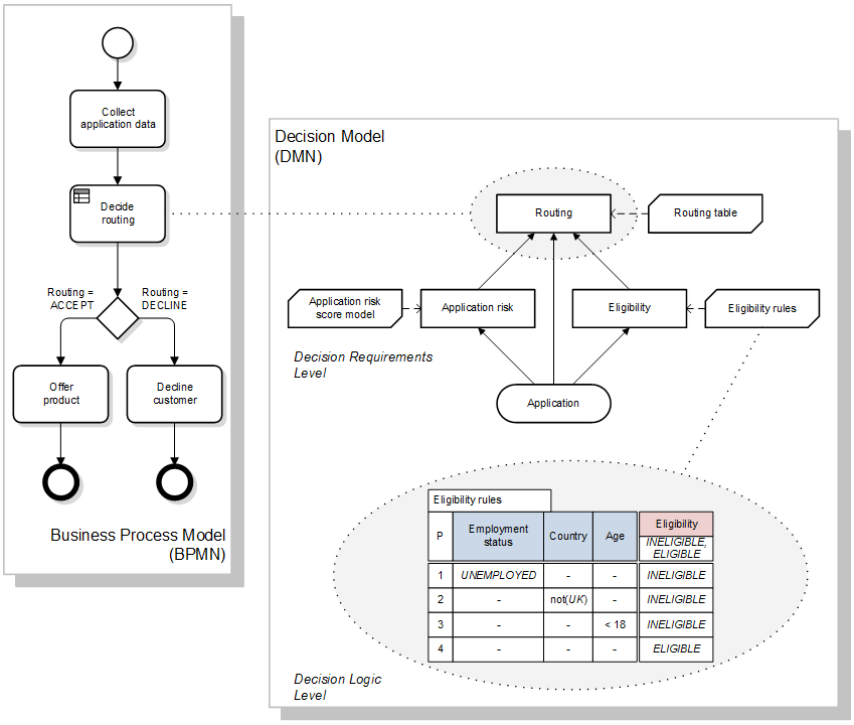
ARD (*Attribute Relationship Diagrams*) to metoda, której celem jest przedstawienie relacji pomiędzy pewnymi “atrybutami“ danego systemu. Wspominane atrybuty są tak naprawdę wartościami branymi pod uwagę w przypadku rozważania logiki biznesowej. Z początku *ARD*



Rys. 2.3. Graficzna reprezentacja elementów *Decision Requirements Diagram*. Grafika za pośrednictwem [8].



Rys. 2.4. Graficzna reprezentacja elementów *Decision Requirements Diagram*. Grafika za pośrednictwem [9].



Rys. 2.5. Integracja modelu *BPMN* oraz *DMN*. Grafika za pośrednictwem [2].

było zaproponowane jako pomysł na prototypowanie baz wiedzy, głównie wykorzystywanych w silnikach regułowych [10]. Miało to działać na podobnej zasadzie jak generowanie struktur relacyjnych baz danych na podstawie diagramów ER[2] (*Entity-Relationship Diagrams*). W kontekście tej pracy *ARD* jest odpowiedzią na problem związany z prototypowaniem modelów procesów biznesowych. Oferuje ona prosty sposób na szybkie tworzenie szkiców z których następnie system potrafi stworzyć bardziej skomplikowane struktury w notacjach *BPMN* oraz *DMN*. Proces tworzenia modelu *ARD* jest prosty i iteracyjny co przyczynia się do zwiększania szczegółowości modelu z każdym kolejnym krokiem. Innymi słowy całość opiera się na przechodzeniu od generalnej koncepcji do bardzo szczegółowego modelu[3], jednocześnie zachowując funkcjonalne zależności pomiędzy poszczególnymi elementami.

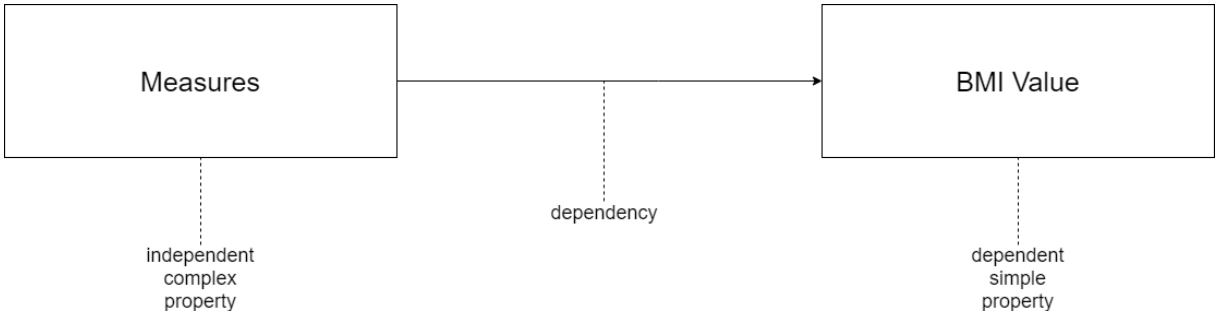
Diagram *ARD* jest zbiorem pewnych własności i zależności między nimi. Elementy z których jest zbudowany to:

– **Property** – własność, która reprezentuje pewne atrybuty. Własności można podzielić na proste i kompleksowe.

  – **Simple** – własność prosta, reprezentuje jeden atrybut. Innymi słowy takiej własności nie da się podzielić na pewien zbiór własności, jest atomiczna. Przykładem takiej

[2]Więcej na temat *ERD*: *https://bit.ly/2ZJymvv*

[3]Rozumowanie dedukcyjne - "od ogółu do szczegółu".



Rys. 2.6. Przykład prostego diagramu *ARD*.

własności może być wiek lub wzrost, są to własności reprezentujące dokładnie jeden atrybut.

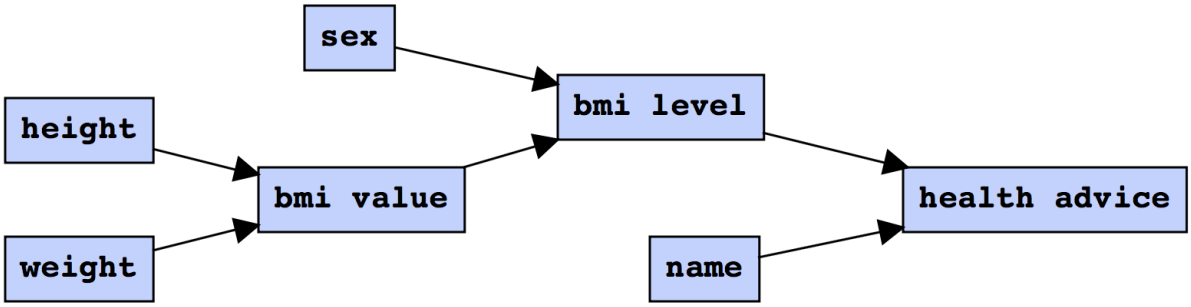
– **Complex** – własność kompleksowa, reprezentuje zbiór atrybutów. Da się ją podzielić na zbiór prostych własności. Przykładem takiej własności mogą być "pomiary" - jest to własność, która pod sobą może kryć wiele atrybutów takich jak np. wzrost, waga.

– **Dependency** – zależność, łączy ze sobą własności i pokazuje relację między nimi.

Grafika 2.6 prezentuję prosty diagram *ARD*. Występują na nim dwie własności. "Measures" jest własnością kompleksową. Przyładowo, kompletnie poprawnym byłoby zastąpienie jej wartościami prostymi typu waga oraz wzrost. "BMI Value" to własność prosta, jest atomiczna i reprezentuje jeden atrybut. Jest to dodatkowo własność zależna więc nie jest możliwe ewaluowanie jej bez poprzedniego określenia własności źródłowych.

Jasno widać, że nietrudnym zadaniem byłoby rozbudowanie takiego diagramu o kolejne własności i to właśnie jest główną zaletą tej metody - prostota i iteracyjna natura. Aby to zobrazować warto pochylić się nad konkretnym przykładem, wykorzystując jednocześnie diagram z grafiki 2.6. Niech przykładową sytuacją będzie potrzeba naszkicowania prototypu modelu związanego z poradami zdrowotnymi przez analityka biznesowego. Model ma opierać się na BMI *Body Mass Index*[4]. Wykorzystując wcześniej wspomniany iteracyjny proces (logikę dedukcyjną), mogłoby to wyglądać tak, że finalną własnością byłaby porada zdrowotna i z każdą kolejną iteracją, analityk wprowadzałby coraz to nowsze i bardziej szczegółowe własności, budując w taki sposób diagram zależności. Porada potrzebowałaby imienia pacjenta oraz jego poziomu *BMI*. Do obliczenia poziomu *BMI* potrzebna by była płeć pacjenta oraz wartość *BMI*. Natomiast sama wartość byłaby obliczana na podstawie wagi oraz wzrostu. Finalny wykres i opisane rozumowanie obrazuje grafika 2.7.

[4]Więcej na temat *BMI*: *https://www.researchgate.net/publication/276444598_Body_Mass_Index*



Rys. 2.7. Diagram *ARD* dla modelu związanego z poradą zdrowotną. Grafika za pośrednictwem [4].

Wykorzystując dalej przykład z grafiki 2.7 oraz mając na uwadze wcześniej opisane standardy *BPMN* oraz *DMN*, można pokusić się o przeniesienie tego prototypu do wspomnianych notacji. Grafika 2.8 prezentuję koncept takiej operacji. Wystarczyłoby odpowiednio pogrupować własności i ich zależności, przedstawić każdą własność prostą jako User Task w którym użytkownik musi podać dane, a każdą własność kompleksową jako Business Rule Task czyli Task połączony z Decision (gdzie wykorzystywana jest tablica decyzyjna). W wspomnianej grafice przykładem User Task jest "Enter Measures", które dostarcza "height" oraz "width" do tablic decyzyjnych, a przykładem Business Rule Task jest "Determine bmi level".

2.5. Hekate Markup Langauge

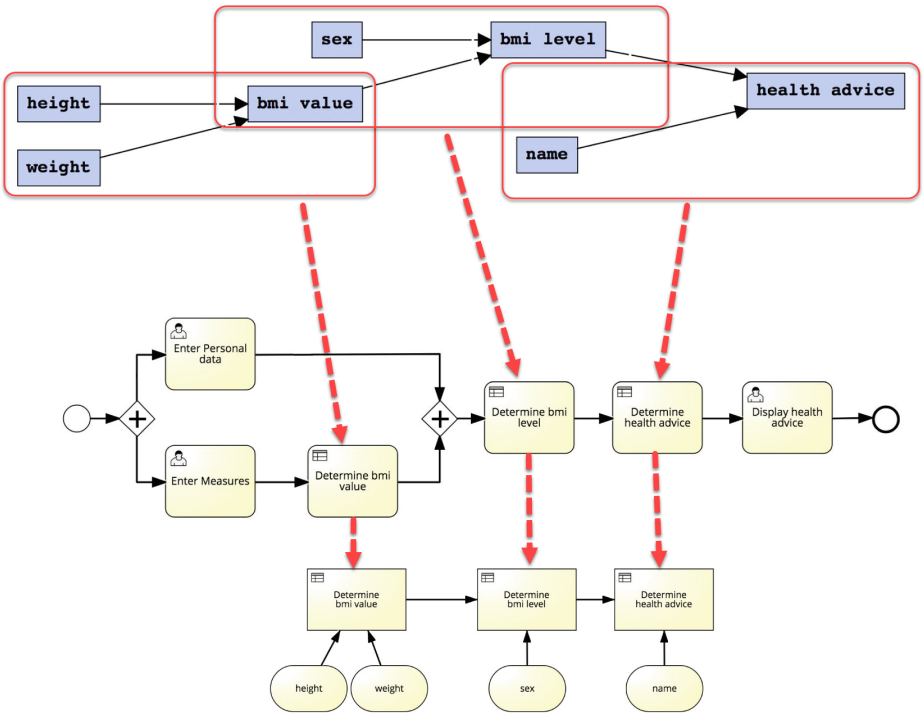
HML (*Hekate Markup Language*) to język stworzony do reprezentacji bazy zasad *HeKatE*[5], zapisanej w czytelnym dla człowieka formacie HMR (*Hekate Meta Representation*[6]), w formacie *XML*. Język ten posiada trzy podzbiory:

- **attml** – Attribute Markup Language opisujący atrybuty zasad.
- **ardml** – Attribute Relationship Markup Language służący do ekspresji prototypów w ARDplus (rozszerzeniu *ARD*)
- **xttml** – XTT2 Rule Markup Language służący do reprezentacji ustrukturyzowanych zasad XTT2[7]

W zależności od potrzeb, możliwe jest różnorakie wykorzystanie opisanych podjęzyków:

- **attml** – tylko definicje atrybutów, minimalistczny przypadek.

[5]Więcej na temat *HeKatE*: *https://ai.ia.agh.edu.pl/hekate:start*
[6]Więcej na temat *HMR*: *https://ai.ia.agh.edu.pl/hekate:hmr*
[7]Więcej na temat *XTT2*: *https://ai.ia.agh.edu.pl/hekate:xtt2*



Rys. 2.8. Koncept transofrmacji diagramu *ARD* do modelu w notacji *BPMN* oraz *DMN*. Grafika za pośrednictwem [4].

– **attml + ardml** – atrybuty oraz zależności *ARD*.

– **attml + ardml + xttml** – atrybuty, zależności oraz zasady *XTT2*.

– **attml + xttml** – atrybuty i zasady (bez prototypu *ARD*).

Wykaz 2.1 prezentuje plik *HML* wykorzystujący wszystkie wspomniane podjęzyki.

Wykaz 2.1. Przykład struktury pełnego pliku *HML*.

```
<hml>
    <!-- attml zaczyna się tutaj -->
    <types>
      ...
    </types>

    <attributes>
      ...
    </attributes>
    <!-- attml kończy się tutaj -->
```

```
    <!-- ardml zaczyna się tutaj -->
    <properties>
      ...
    </properties>

    <tph>
      ...
    </tph>

    <ard>
      ...
    </ard>
    <!-- ardml kończy się tutaj -->

    <!-- xttml zaczyna się tutaj -->
    <xtt>
      ...
    </xtt>
    <!-- xttml kończy się tutaj -->

    <!-- dodatkowe specyfikacje systemu -->
    <states>
    </states>
</hml>
```

Składnia wymaga aby wszystkie ważne elementy posiadały atrybut “id”, będący unikatowym identyfikatorem. Dodatkowo w zależności od elementu powinny zawierać odpowiedni przedrostek:

- **types** – identyfikator powinien zaczynać się od “tpe”
- **attributes** – identyfikator powinien zaczynać się od “att”
- **properties** – identyfikator powinien zaczynać się od “prp”
- **type groups** – identyfikator powinien zaczynać się od “tgr”
- **attribute groups** – identyfikator powinien zaczynać się od “agr”
- **dependencies** – identyfikator powinien zaczynać się od “dep”

– **ARD history** – identyfikator powinien zaczynać się od “hst”

W kontekście niniejszej pracy pliki *HML*, które będą brane pod uwagę, powinny zawierać w sobie pozdbiór językowy *ardml*, jednak z opcjonalnym fragmentem *TPH*, gdyż nie jest on wykorzystywany przy tworzeniu modelu procesu finalnego. Przykład wymaganego pliku prezentuje wykaz 2.2

Wykaz 2.2. Przykład struktury pliku *HML* z podzbiorem *ardml*.

```
<hml>
    <types>
        <type id="tpe_1" name="Temperature" base="numeric" length="5">
            <desc>Reprezentuje temperaturę</desc>
            <domain>
                <value from="1" to="5"/>
            </domain>
        </type>
        <type id="tpe_2" name="Integer" base="numeric" length="5">
            <desc>Reprezentuje numer</desc>
            <domain>
                <value from="-10000" to="10000"/>
            </domain>
        </type>
    </types>
    <attributes>
        <attr name="Thermostat" id="att_0" type="tpe_1"/>
        <attr name="Something" id="att_1" type="tpe_2"/>
    </attributes>
    <properties>
        <property id="prp_1">
            <attref ref="att_0"/>
        </property>
        <property id="prp_2">
            <attref ref="att_0"/>
            <attref ref="att_1"/>
        </property>
    </properties>
    <ard>
```

```
        <dep id="dep_01" independent="prp_1" dependent="prp_2"/>
        <dep id="dep_02" independent="prp_2" dependent="prp_3"/>
        <dep id="dep_03" independent="prp_1" dependent="prp_4"/>
        <dep id="dep_04" independent="prp_2" dependent="prp_4"/>
    </ard>
</hml>
```

3. Projekt aplikacji

3.1. Podstawowe przypadki użycia

Aplikacja “DAR” jest z punktu widzenia funkcjonalności dość prostym systemem. Grafika 3.1 prezentuje podstawowe przypadki użycia - warto mieć jednak na uwadze fakt, że aplikacja została zintegrowna z zewnętrznym systemem posiądającym silniki decyzyjne i procesowe, więc wiele funkcjonalności nie pokazanych tutaj, takich jak uruchamianie procesu, ewaluacja decyzji czy monitorowanie wydajności procesów, leżą po jego stronie.

Funkcjonalnościom warto przyjrzeć się bliżej:

- **Deployment procesu** – aplikacja przesyła dwa pliki do zewnętrznego systemu. Pierwszy z nich to plik reprezentujący proces w notacji *BPMN*, natomiast drugi to plik reprezentujący diagram decyzji w notacji *DMN*.

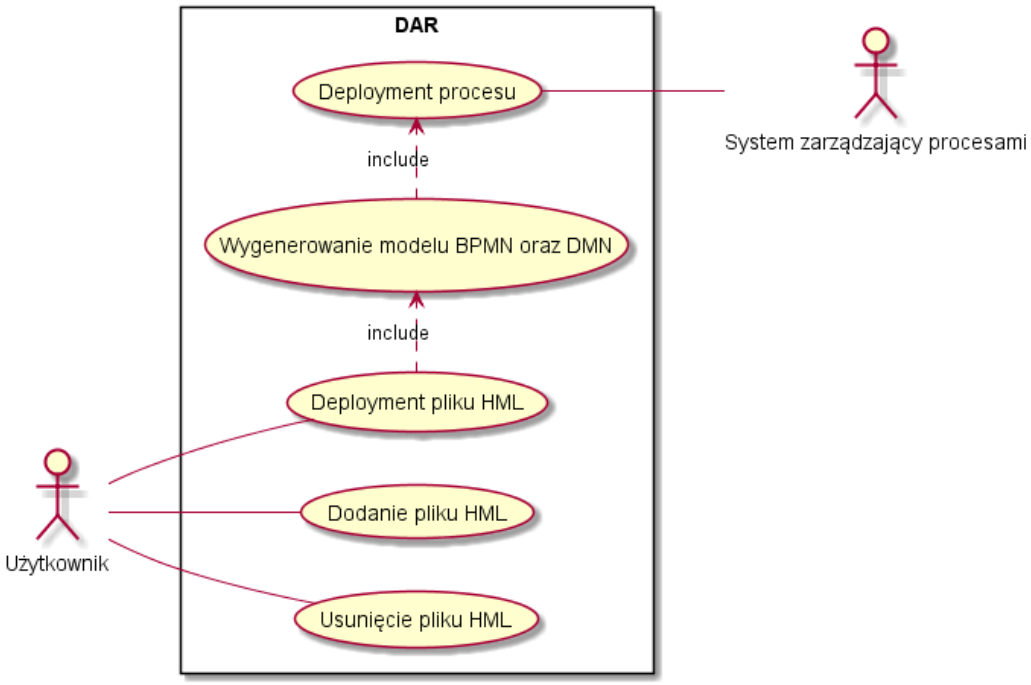
- **Wygenerowanie modelu BPMN oraz DMN** – aplikacja na podstawie dostarczonych danych generuje dwa modele - model w notacji *BPMN* oraz model w notacji *DMN*. Jednocześnie spaja te dwa modele poprzez odpowiednie ustawianie atrybutów.

- **Deployment pliku HML** – aplikacja przesyła wybrany plik *HML* i rozpoczyna proces jego przetwarzania aby finalnie przesłać go na odrębny system procesowy.

- **Dodanie pliku HML** – aplikacja zapisuje informację z plików *HML*.

- **Usunięcie pliku HML** – aplikacja usuwa wszystkie zapisane informację z danego pliku *HML*.

- **Uruchamianie procesu, ewaluacja decyzji, monitorowanie procesu...** – jak już zostało wspomniane, wiele funkcjonalności nie należy stricte do ram aplikacji “DAR”, jednak poprzez integrację z zewnętrznym systemem, aplikcja pośredniczy w umożliwianiu takich działań jak uruchamianie i śledzenie procesu, edycję tabeli decyzyjnych, uzupełnianie danych czy ewaluację decyzji.



Rys. 3.1. Graficzna reprezentacja podstawowych przypadków użycia.

3.2. Architektura systemu

3.2.1. Architektura N-tier

Aplikacja “DAR” została oparta na architekturze wielowarstwowej zwanej również architekturą N-tier. Jest to pewna metoda tworzenia oprogramowania, opierająca się na wydzieleniu warstw aplikacji - w sensie fizycznym oraz logicznym. Warstwy są narzędziem do odseparowania obowiązków[1] danych rejonów aplikacji oraz do zarządzania zależnościami. Każda warstwa posiada konkretną odpowiedzialność. Warstwy wyższe mogą używać serwisów z warstw niższych, jednak nie działa to w drugą stronę. Architektura wielowarstwowa może być:

– **Zamknięta** – tak jak jest to w przypadku opiswanego systemu. Dana warstwa może “wołać“ jedynie warstwę pod sobą.

– **Otwarta** – dana warstwa może “wołać“ dowolną warstwą pod sobą.

“DAR” posiada trzy logiczne warstwy:

[1] Jest to ściśle związane z zasadą SoC (*Separation of Concerns*), czyli podstawową zasadą inżynierii oprogramowania, stanowiącą o tym, że każda część systemu powinna mieć silnie określone granice i adresować dokładnie odseparowaną kwestie.

– **Warstwa prezentacji** – najbardziej górna warstwa. Jej zadaniem jest wyświetlanie danych i komunikacja z warstwą niżej. Umożliwia użytkownikowi interakcję z systemem. Reprezentuje interfejs użytkownika.

– **Warstwa logiki** – zwana również logiką biznesową, jest to główna część aplikacji w której zaimplementowane zostały wszystkie funkcjonalności. W przypadku "DAR" do dyspozycji są tutaj dwa serwisy, jeden będacy stricte serwerem "DAR", natomiast drugi to zintegrowany zewnętrzny system do obsługi procesów. Warstwa ta przetwarza dane z warstwy niżej i przekazuje je do warstwy prezentacji.

– **Warstwa danych** – warstwa związana z dostępem do danych, w przypadku "DAR" jest to dostęp do relacyjnej bazy danych, gdzie przechowywane są informację na temat dodanych plików *HML*. Udostępnia ona dane dla warstwy logiki biznesowej.

Dzięki takiemu podziałowi aplikacja zyskuje wiele benefitów:

– **Skalowalność** – dzięki oddzieleniu odpowiedzialności bardzo łatwo zdiagnozować gdzie aplikacja potrzebuję usprawnień i co najważniejsze, można to robić tylko w jednym konkretnym obszarze, co daje o wiele więcej możliwości skalowania.

– **Luźne powiązania** – architektura wymusza na deweloperze ograniczenie zależności, dzięki czemu zmiany w danej warstwie są bezbolesne z punktu widzenia innych warstw.

– **Bezpieczeństwo** – podział na warstwy niesie też ze sobą więcej możliwości ochrony przed atakami, ponieważ w każdej warstwie może występować inna metoda ochrony. Dodatkowo tak samo jak w przypadku skalowania, można odpowiednio poświęcić środki na ochronę tylko newralgicznych punktów aplikacji (przykładem może być zwiększona ochrona warstwy logiki biznesowej oraz danych niż warstwy prezentacji).

– **Łatwiejszy proces tworzenia aplikacji** – każdą warstwą może zajmować się inna osoba lub zespół, ze względu na luźne powiązania.

– **Większy nacisk na reużywalność** – dzięki modularności tego podejścia, komponenty są raczej projektowane w sposób generyczny co ułatwia ich późniejsze, ponowne użycie.

3.2.2. Architektura klient-serwer

"DAR" jest aplikacją internetową działająca na zasadzie klient-serwer. Jest to metoda działania, która opiera się na podzieleniu aplikacji na dwie części i na komunikacji tych modułów:

– **Front-end** – część kliencka, działająca na oprogramowaniu klienckim - w tym przypadku jest to przeglądarka użytkownika - i to z tą częścią ma on do czynienia. W aplikacji "DAR" została zrealizowana w formie SPA (*Single Page Application*[2]). Do tej części należy *warstwa prezentacji* opisywana wcześniej. Termin *front-end* bierze się stąd, że jest to ta część aplikacji o której konsument wie i przez którą komunikuję się z pozostałymi elementami aplikacji. Innymi słowy reprezentuje ona interfejs użytkownika.

– **Back-end** – część serwerowa, działająca na osobnym serwerze, przeważnie użytkownik nie jest świadomy jej istnienia, ponieważ po prostu jej nie widzi - stąd termin *back-end*. Do niej należą *warstwy logiki biznesowej oraz danych* opisane wcześniej. W przypadku systemu "DAR" została zrealizowana za pomocą internetowego *REST*[3] API (*Application Programming Interface*).

W opisywanej aplikacji klient wysyła zapytanie *HTTP*[4] przez sieć internetową do serwera, a ten przetwarza żądanie i zwraca odpowiedź, często z towarzyszącymi temu danymi - również za pomocą protokołu *HTTP*. Komunikacja jest asynchroniczna, dzięki temu aplikacja jest bardziej responsywna i szybsza.

3.2.3. Protokół HTTP

Protokół HTTP (*Hypertext Transfer Protocol*) to bezstanowy protokół do przesyłania dokumentów typu hypermedia, przeważnie bazowany na warstwie *TCP/IP*, jednak może być bezproblemowo używany w oparciu o każdą inną solidną warstwę transportu. Określa on zasady wymiany informacji, normalizuje i ujednolica sposoby komunikacji pomiędzy urządzeniami. Jego głównym zastosowaniem jest umożliweinie komunikacji pomiędzy aplikacjami internetowymi (przeglądarkami), a serwerami (komputerami/chmurą). Bardzo często towarzyszy architekturze klient-serwer. Wymiana informacji przebiega w następujący sposób:

– Klient (przeglądarka) wysyła zapytanie *HTTP* do sieci.

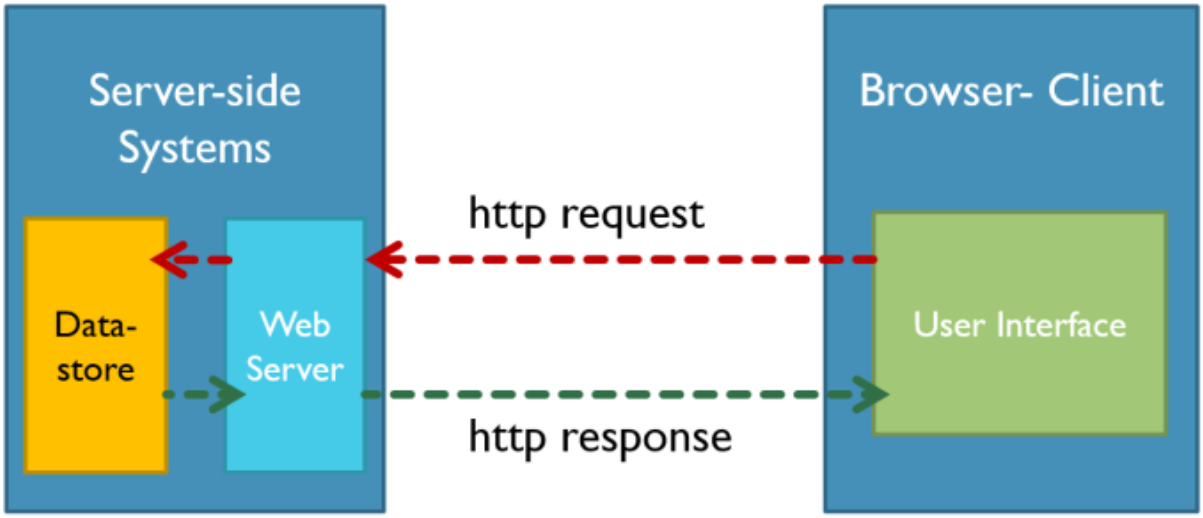
– Serwer internetowy otrzymuje zapytanie.

– Serwer uruchamia aplikację aby przetworzyć zapytanie.

– Serwer zwraca odpowiedź *HTTP* to przeglarki.

– Klient otrzymuje odpowiedź.

[2]Więcej na temat *SPA* w dalszej części pracy.
[3]Więcej na temat protokołu *REST* w dalszej części pracy.
[4]Więcej na temat protokołu *HTTP* w dalszej części pracy.



Rys. 3.2. Przykład komunikacji za pomocą protokołu *HTTP*. Grafika za pośrednictwem [11].

Grafika 3.2 obrazuje wymianę informacji pomiędzy klientem a serwerem używając protokołu *HTTP*.

Klient aby otrzymać odpowiedź od serwera musi określić jego adres oraz specjalny czasownik określający jakiego typu jest jego zapytanie - co chce nim osiągnąć. Specyfikacja *HTTP* [12] określa w tym momencie dziewięć metod:

- **GET** – pobranie zasobu wskazanego za pomocą adresu.
- **PUT** – wysłanie danych, najczęściej aby zaktualizować pewne istniejące już dane.
- **POST** – wysłanie danych, najczęściej aby stworzyć kompletnie nowy zasób.
- **DELETE** – usunięcie danych.
- **OPTIONS** – pobranie informacji o opcjach (najczęściej komunikacji).
- **TRACE** – testowanie kanału komunikacyjnego.
- **CONNECT** – tworzenie połączenia do serwera docelowego.
- **HEAD** – pobranie informacji o zasobie, stosowane do sprawdzania dostępności zasobu.
- **PATCH** – aktualizacja części danych.

Serwer zwracając odpowiedź, niezależnie od tego czy towarzyszą jej jakieś dane czy nie, wysyła wiele dodatkowych informacji. Jednym z podstawowych elementów odpowiedzi jest status. Informują one o tym czy żądanie zostało spełnione. Statusy zostały podzielone na pięć grup:

- **Statusy 1xx** – statusy informacyjne, przekazują informację o przetwarzaniu zapytania.
- **Statusy 2xx** – statusy z tej grupy oznaczają powodzenie żądania. Przykładami najczęściej używanych statusów z tej grupy są:
  - **200 OK** – zapytanie zostało przetworzone poprawnie.
  - **201 Created** – używane w kontekście tworzenia pewnych danych, informuje o sukcesie takiego żądania.
- **Statusy 3xx** – statusy przekierowujące, informują o potrzebie zapytania na inny adres.
- **Statusy 4xx** – statusy związane z błędami po stronie klienta. Przykładami najczęściej używanych statusów z tej grupy są:
  - **400 Bad Request** – błędne zapytanie.
  - **403 Forbidden** – zasób wymaga uwierzytelnienia.
  - **404 Not Found** – brak zasobu.
- **Statusy 5xx** - statusy związane z błędami po stronie serwera. Przykładami najczęściej używanych statusów z tej grupy są:
  - **500 Internal Server Error** – serwer napotkał niespodziewany przypadek przez co nie potrafi wypełnić zapytania.
  - **502 Bad Gateway** – serwer będący pewnym węzłem otrzymał informację o błędzie od źródła.

3.2.4. REST API

REST (*Representational State Transfer*) jest to styl architektury oprogramowania, bazujący na zbiorze określonych reguł, które określają jak definiowane są zasoby, a co za tym idzie jak można otrzymać do nich dostęp. API (*Application Programming Interface*) jest to zestaw pewnych zasad określających komunikację pomiędzy oprogramowaniem komputerowym. Innymi słowy wiążąc te dwa pojęcia, *API* to reguły określające jakie zasoby są dostępne i jak interesariusz może je uzyskać, natomiast *REST* to styl określający schemat oraz budowę *API* (jak będzie wyglądać). Aby interfejs spełniał w pełni styl *REST* czyli był *RESTful*, musi spełnić sześć podstawowych reguł z nim związanych:

- **Client-server** – architektura klient-serwer. Mocno związane z podziałem odpowiedzialności. Odseparowując interfejs użytkownika od przechowywania danych zwiększona zostaje modularność, przez co zyskuje na tym skalowalność.

– **Stateless** – każda komunikacja na kanale klient-serwer jest w pełni samowystarczalna, posiada wszelkie niezbędne informacje. Żaden kontekst klienta nie jest przechowywany na serwerze pomiędzy zapytaniami.

– **Cacheability** – odpowiedź z *REST API* musi być jasno określona jako cacheable albo non-cacheable.

– **Uniform interface** – endpoint czyli adres danego zasobu powinien być jedoznaczny. Użytkownik zawsze powieniene wiedzieć do jakiego zasobu się odwołuje. Polega to głównie na odpowiedniej budowie adresów.

– **Layered system** – architektura wielowarstwowa. Najważnieszym czynnikiem jest tutaj separacja warstw.

– **Code on demand** – zasada opcjonalna i polega na udostępnianiu klientowy apletów i skryptów.

Główną kwestią na którą kładzie nacisk *REST* są zasoby. Każda informacja, która może być nazwana może być zasobem. Stan danego zasobu w dowolnym czasie jest znany jako reprezentacja zasobu. Zawiera ona dane, metadane opisujące dane oraz linki typu hypermedia, które pomagają w przechodzeniu do kolejnego wymaganego stanu. Format danych jest znany jako

4. Implementacja

5. Ewaluacja

6. Podsumowanie

Bibliografia

[1] *Business Process Model and Notation (BPMN) Version 2.0*. Spraw. tech. Object Management Group (OMG), 2011.

[2] *Decision Model and Notation (DMN) Version 1.2*. Spraw. tech. Object Management Group (OMG), 2019.

[3] Grzegorz J. Nalepa i Igor Wojnicki. *Towards Formalization of ARD+ Conceptual Design and Refinement Method*. Red. David C. Wilson i H. Chad Lane. Menlo Park, California, 2008.

[4] Krzysztof Kluza, Piotr Wiśniewski i Antoni Ligęza. „From Attribute Relationship Diagrams to Process (BPMN) and Decision (DMN) Models". W: sierp. 2019. ISBN: 978-3-030-29550-9. DOI: *10.1007/978-3-030-29551-6_55*.

[5] *Hekate Markup Language*. *https://ai.ia.agh.edu.pl/wiki/hekate:hml1*. AGH University of Science and Technology. 2008.

[6] Stephen A. White. *Process Modeling Notations and Workflow Patterns*. Spraw. tech. IBM Corp., 2010.

[7] *Beginners Guide to Business Process Modeling and Notation (BPMN)*. *https://www.smartsheet.com/beginners-guide-business-process-modeling-and-notation-bpmn*.

[8] *Integrating BPMN and DMN*. *https://modernanalyst.com/Resources/Articles/tabid/115/ID/3189/categoryId/90/Integrating-BPMN-and-DMN.aspx*.

[9] *Filip Drools Decision Tables in Spreadsheets*. *https://training-course-material.com/training/Filip_Drools_Decision_Tables_in_Spreadsheets*.

[10] Grzegorz Nalepa i Antoni Ligęza. „Conceptual Modelling and Automated Implementation of Rule-Based Systems". W: (maj 2005).

[11] Kereshmeh Afsari, Charles Eastman i Dennis Shelden. „Data Transmission Opportunities for Collaborative Cloud-Based Building Information Modeling". W: list. 2016. DOI: *10.5151/despro-sigradi2016-448*.

[12] *Hypertext Transfer Protocol – HTTP/1.1*. Spraw. tech. The Internet Society, 1999.