Kevin Akkoyun

BPEL-prosessien suunnittelu ActiveVOS Designer -ohjelmalla

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikan tutkinto-ohjelma

Insinöörityö

29.5.2023

Tiivistelmä

Tekijä:	Kevin Akkoyun
Otsikko:	BPEL-prosessien suunnittelu ActiveVOS Designer -ohjelmalla
Sivumäärä:	39 sivua
Aika:	29.5.2023
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Ohjelmistotuotanto
Ohjaajat:	Lehtori Vesa Ollikainen

Tämän työn tarkoituksena on toimia kehitysohjeena uudelle kehittäjälle, joka haluaa oppia ymmärtämään, miten BPEL-prosesseja kehitetään käyttäen ActiveVOS Designer -ohjelmaa.
Työn aikana käydään läpi yleistä historiaa asiaan liittyen. Lisäksi käydään läpi kehitykseen liittyvät standardit.
Ohje on muotoiltu esimerkkiprosessin ympärille, jonka läpikäynnin yhteydessä tutkitaan prosessin toimintaa sekä sen rakentavien elementtien rakennetta ja tarkoitusta prosessikokonaisuudessa.
Lopputuloksena on tiiviisti läpikäyty esimerkki, joka helpottaa kehittäjän ymmärrystä asiassa, jonka oppiminen muuten tapahtuisi monimutkaisesti rakennetun dokumentaation kautta.

Avainsanat: prosessikehitys, ohjelmistotuotanto, prosessihallinta

---

Abstract

Author:	Kevin Akkoyun
Title:	Designing BPEL-Processes with ActiveVOS Designer
Number of Pages:	39 pages
Date:	29 May 2023
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communications Technology
Professional Major:	Software Development
Supervisors:	Vesa Ollikainen, Senior Lecturer

The purpose of this document is to act as a development guide for a new developer, who wants to understand how BPEL-processes are developed using ActiveVOS Designer.
The document goes through general history related to the topic and the standards this type of design requires.
The guide is designed around the example process which is used to display the inner workings of the process and the place of its components in the grand scheme of things.
As a result, this document is an example, that helps a developer to understand a topic, that otherwise requires reading a complex and long set of documentation.

Keywords: process development, software development, process management

Sisällys

Lyhenteet

BPEL: *Business Process Execution Language*. XML-pohjainen kieli, josta prosessit koostuvat.

BPM: *Business Process Management*. Järjestelmien toimintaan liittyvää prosessien ja tiedon hallintaa.

IDE: *Integrated Development Environment.* kehitysympäristö, jossa suunnitellaan ja rakennetaan ohjelmistoja.

JSON: *JavaScript Object Notation*. JavaScript-kielestä yleistynyt tietorakenne, joka koostuu yhdestä tai useammasta objektista ryhmässä.

PSC: *Process Service Consumer*. WSDL-tiedostossa määritellyn web-palvelun vastaanottaja.

PSP: *Partner Service Provider*. WSDL-tiedostossa määritellyn web-palvelun toteuttaja.

W3C: *World Wide Web Consortium.* Kansainvälinen organisaatio, joka ylläpitää/arkistoi internetin standardeja.

WSDL: *Web Service Description Language*. XML-pohjainen kuvauskieli, jolla määritellään web-palveluita.

XML: *Extensible markup language*. merkintäkieli, johon monet muut merkintä- ja kuvauskielet pohjautuvat.

XSD: *XML Schema Definition.* XML-pohjainen määrittelykieli, joka määrittää XML-tiedoston elementit.

1 Johdanto

Tämän työn tarkoituksen on toimia kehitysohjeena ActiveVOS Designer -ohjelmalle, jolla kehitetään BPM:ää varten tietoa käsitteleviä prosesseja. Nämä prosessit voivat hoitaa useampaa eri tehtävää sovellusarkkitehtuurissa, joten tässä työssä keskitytään pääosin yksittäiseen esimerkkiin. Tämän esimerkin kautta demonstroidaan prosessien kulkua, toimintaa ja kehitystä.

On tärkeä huomioida, että ActiveVOS ja ActiveVOS Designer eivät ole sama ohjelma. Designer, eli ohjelma, jota käsittelemme tämän työn aikana, on ActiveVOS-ohjelman kehitysympäristö, ja sillä luodaan ActiveVOS:illa suoritettavia prosesseja.

Tässä työssä käydään kontekstin vuoksi hieman läpi sitä, mikä ActiveVOS on, mutta sen läpikotaisesti tunteminen ei ole mitenkään välttämätöntä. Työtä ymmärtääkseen, ei tule olettaa, että ActiveVOS:ia osaisi käyttää tämän työn perusteella.

Kohdeyleisönä tätä työtä varten toimii tieto- ja viestintätekniikan työntekijä, jonka tulee opetella prosessikehitystä BPEL kielellä, kun hän käyttää ActiveVOS Designer ohjelmaa. Kehitysohjeen ymmärrystä helpottaa, mutta ei ole vaadittua, jos on aikaisemmin käyttänyt Eclipse IDE -kehitysympäristöä.

2 Yleistietoa ja historiaa

Tässä luvussa käsitellään ActiveVOSin historiaa käymällä läpi ohjelman eri vaiheita, ja omistajia. Lisäksi käydään läpi ohjelman yleinen kuvaus siitä, mitä se tekee, ja miten se toimii.

2.1 Informatica, Active-Endpoints ja ActiveVOSin historiaa

ActiveVOS on amerikkalaisen Informatica-yhtiön omistama tuote. Yhtiö perustettiin vuonna 1993. (1.)

Tuotteen on alun perin suunnitellut toinen amerikkalainen yritys: Active-Endpoints. ActiveVOS pohjautuu Active-Endpointsin edelliseen tuotesarjaan: ActiveBPEL:iin. Toukokuussa 2008 julkaistiin ensimmäinen versio Active-VOS:ista, ActiveVOS 5.0.2. Tämä versio oli päivitys edellisestä ActiveBPEL 4.1 -ohjelmasta, johon lisättiin uusia käyttäjäympäristön päivityksiä. (2.)

Vuonna 2013 Informatica osti Active-Endpoints:in, jolloin ActiveVOS siirtyi ostajan hallintaan. Informatica ei ylläpidä vanhempia ActiveVOS-versioita, niiden dokumentaatiota tai julkaisuhistoriaa, joten sovelluksen kehitys-/julkaisuhistoria on liian vaikeaa dokumentoida tähän työhön luotettavasti. Tässä työssä käytetty versio ActiveVOS Designerista on 9.2.4.6. (3.)

2.2 Mikä on ActiveVOS?

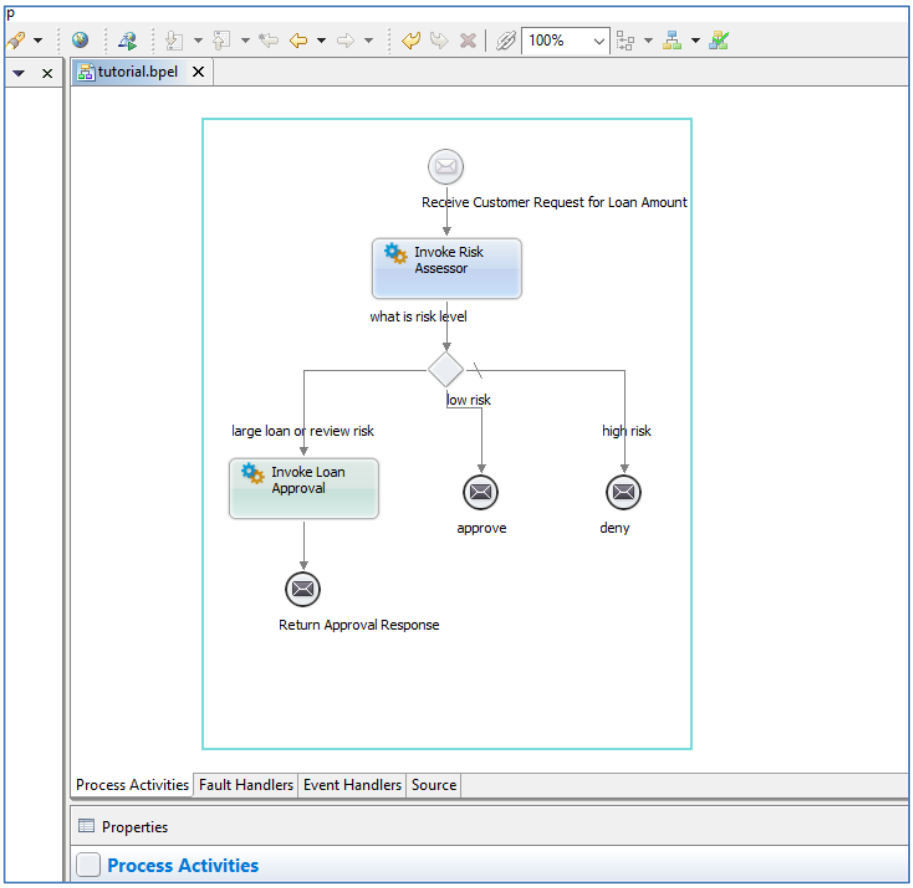
ActiveVOS-nimi tulee sanoista Active **V**isual **O**rchestration **S**ystem (4.). Ohjelmisto on tarkoitettu BPM (Business process management) -työkaluksi, jolla hallitaan järjestelmässä suoritettavia liiketoimintaprosesseja sekä niiden automaatiota.

Liiketoimintaprosessia voi kuvata usealla eri tavalla. Tässä käyttötapauksessa puhutaan BPM:n määrityksestä eli: ”joukko toistuvia toimintoja, joilla yksinkertaistetaan rutiininomaisia tehtäviä”. Hyvä esimerkki liiketoimintaprosessista on esimerkiksi työtiimin hallinnollinen tehtävä: työtehtävien jakaminen niistä vastuussa oleville työntekijöille. (5.)

ActiveVOS usein jaetaan kahteen eri työkaluun:

1. ActiveVOS eli itse ohjelmistoon, joka suorittaa prosesseja ja hallitsee niiden kulkua.
2. ActiveVOS Designeriin, joka on tämän työn kohde eli työkalu, jolla nämä suoritettavat prosessit luodaan.

ActiveVOS Designer on Eclipse IDE:n päälle rakennettu työkalu, jonka tarkoituksena on rakentaa BPEL-tiedostoja, kuvassa 1 nähtävällä visuaalisella käyttöliittymällä, puhtaan kirjoitetun syntaksin sijaan.



Kuva 1. ActiveVOS:n käyttöliittymä.

Käyttöliittymä on Eclipse IDE:n lailla muokattavissa ulkonäöllisesti, mikäli käyttäjä kokee tarpeen tähän.

2.3 BPM ja BPEL

Tätä työtä varten ei ole tarpeellista täysin ymmärtää, mikä BPM:n tarkoitus on. Ainoastaan se, että siihen kuuluu datan saapuminen lähteestä **'a'**, tämän datan käsittely arkkitehtuurissa kuvatulla tavalla, sekä sen mahdollinen syöttö paikkaan **'b'**. Tämän prosessin aikana saatetaan kutsua erilaisten ohjelmien apua, jotta prosessi saavuttaa halutun lopputuloksen. Näin ollen tämä on täysin kiinni arkkitehtuurista, johon prosesseja ollaan luomassa eikä sillä ole mitään väliä tätä työtä ja sen esimerkkiä käsitellessä.

Lyhyesti sanottuna ActiveVOS-ohjelma suorittaa BPM-toimintoja. Tämä työ käsittelee pääosin BPEL-kaavioiden ja -tiedostojen luontia. BPEL-tiedostojen suoritus on prosessien suoritusta, jonka ActiveVOS hoitaa.

BPEL (tunnettu myös nimellä WS-BPEL) on OASIS-nimisen organisaation luoma projekti, jonka tarkoituksena oli jatkaa vuonna 2002 julkaistun BPEL4WS-nimisen standardin kehitystä (6.).

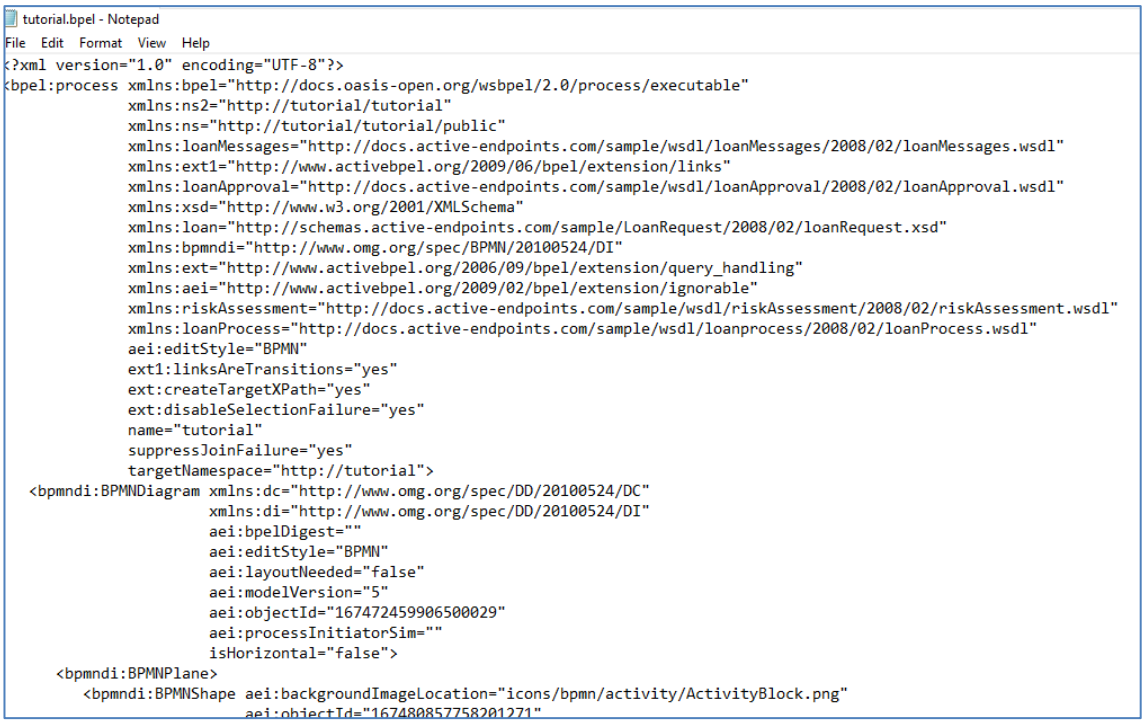
Tämän standardin ovat kehittäneet yhteistyössä IBM, BEA Systems (ostettu vuonna 2008, nykyään Oracle Corporation) ja Microsoft. Sen tarkoitus on tukea web-standardeihin pohjautuvaa kommunikaatiota järjestelmien välillä. BPEL4WS perustuu useampaan XML-spesifikaatioon, joita ovat:

- WSDL 1.1, jonka tarkoitus on rakentaa web-palveluiden käyttämät portit, rakenteet ja kommunikaatiometodit.
- XSD1.0, joka toimii yhteistyössä WSDL:n kanssa, määrittäen tälle käytössä olevat tietorakenteet.
- XPath 1.0, eräänlainen kyselykieli, jonka tarkoitus on tarjota datamanipulaation toimintoja standardissa.

BPEL4WS-standardi luotiin joustavaksi uudempia XML-pohjaisia standardeja varten, etenkin datamanipulaatioon liittyvien osien kohdalta. (7.)

BPEL kehitettiin siis BPEL4WS-standardin pohjalta, ja vuonna 2007 julkaistiin tähän päivään mennessä viimeisin versio: WS-BPEL 2.0. Tämä uusi versio eroaa aikaisemmasta standardista esimerkiksi tarjoamalla uudistettuja, korvaavia aktiviteettejä, kuten if-, elseif- ja else -ehtomuodot, switch-aktiviteetin sijaan. (8.) (9.)

Lyhyesti sanottuna, BPEL on prosesseja varten luotu XML-pohjainen ohjelmointikieli. Kielenä se on hyvin erilainen tavallisiin skripti tai olio-ohjelmointikieliin verrattaessa. Isoin eroava tekijä on BPEL:n perustuminen XML-syntaksiin. Kuvassa 2 näkyy BPEL-tiedoston rakenne ilman graafista editointiohjelmaa. BPEL:iä siis kirjoitetaan XML-tiedoston muotoon.



Kuva 2. BPEL-kaavion rakenne tiedostossa.

BPEL merkataan tiedostonimessä päätteellä ”**.bpel**”, ei ”**.xml**”. ActiveVOS Designerin tarkoitus on helpottaa BPEL-tiedostojen kirjoittamista tarjoamalla kehittäjälle visuaalisen käyttöympäristön, johon rakennettu prosessikaavio muutetaan ohjelmalle tarkoitettuun XML-muotoon.

Koska BPEL on XML-pohjainen kieli, koostuu se elementeistä, joita kutsutaan nimellä ”tag” (suom. lausunta ”tägi”). Nämä elementit luovat prosessille logiikan, johon kuuluu suoritusjärjestys sekä toiminnot. Otetaan esimerkkinä yksinkertainen elementti ”variable”, eli muuttuja.

```
<bpel:variable name=”V1” type=”xsd:string”/>
```

Tämän elementin tarkoitus on määritellä prosessin aikainen muuttuja. Elementti sisältää attribuutit ”name” eli nimi, ja ”type” eli tietotyyppi. Tässä tapauksessa nimi on ”V1” ja tietotyyppinä on ”xsd:string” eli ”xsd” nimiavaruudessa määritelty tietotyyppi: merkkijono.

”Variable” elementti voi myös sisältää tyyppiattribuutin sijaan elementtiattribuutin. Se tarkoittaa sitä, että tietotyyppinä muuttujalle toimii toinen elementti. Tämä toinen elementti voi olla esimerkiksi listapohjainen itse määritelty tietotyyppi. Nämä tietotyypit ovat määritetty XSD-tiedostossa, joka tarjoaa alustan elementtien ja tietotyyppien määrittämiseksi.

Monesti BPEL:n yhteydessä puhutaan aktiviteeteistä. Nämä ovat teknisesti ottaen vain joukko määritellysti toimivia elementtejä. Hyvä esimerkki on If-aktiviteetti, joka on yksinkertainen ehtorakenne. Mikäli If-aktiviteettiin määrätty ehto toteutuu, siirtyy prosessi if-aktiviteetin alla olevaan haaraan, jos taas ehto ei toteudu, siirtyy prosessi else-haaraan. Elementeillä kirjoitettuna rakenne näyttää tältä:

```
<bpel:if name=”ehto”>
     <bpel:condition>$V1 = “”</bpel:condition>
     <bpel:sequence>
     #tässä lisää elementtejä mikäli ehto toteutuu
     </bpel:sequence>
     <bpel:else>
          <bpel:sequence>
          #tässä elementtejä mikäli ehto ei toteudu
          </bpel:sequence>
     </bpel:else>
</bpel:if>
```

Tästä esimerkistä huomataan elementtien rakenne, jokainen alkava elementti tarvitsee sulkevan elementin (paitsi jos elementti on itsesulkeva). Koodi kirjoitetaan suoraan elementtien väliin. Elementeillä on käytännössä aina nimiavaruus, eli ennen kaksoispistettä (:) oleva osa, josta ne ovat tunnistettavissa, mikäli saman nimisiä elementtejä tulee vastaan.

2.4 WSDL

WSDL eli Web Service Description Language on XML-formaatti, jonka tarkoituksena on kuvata web-palveluita sekä niiden toimintoja. Tämä on hyvin keskeinen

osa BPEL-prosessien toimintaa sillä se mahdollistaa prosessin kommunikaation ulkopuolisten lähteiden kanssa.

Sen olivat alun perin kehitelleet yhteistyössä Ariba (nyk. SAP Ariba), IBM ja Microsoft. Vuonna 2000 julkaistiin ensimmäinen julkinen versio 1.0. Tämän ideana oli yhdistää aikaisemmat standardit, joita kyseiset firmat olivat kehitelleet. Eli NASSL (Network Application Service Specification Language) ja SDL (Service Description Language). (10.)

BPEL on suunniteltu käyttämään WSDL:n versiota 1.1, joka ei eroa käytännössä ollenkaan versiosta 1.0. Versio 1.1 tehtiin WSDL:n virallistamista varten ja siitä lähetettiin versio 1.1 julkaisun yhteydessä pyyntö W3C-organisaatiolle, joka ylläpitää/arkistoi web-pohjaisia standardeja. WSDL:n versio 1.1 ei kuitenkaan saanut W3C:n suositusta. Vasta vuonna 2007 alkoi W3C tukemaan/suosittelemaan WSDL:n versio 2.0 käyttöä web-palveluiden kuvausta varten. (11.)

WSDL-tiedostot ovat siis XML-formaatin mukaisia, eli niitä kirjoitetaan elementtien avulla, joista mainittiin jo aiemmin. WSDL-tiedostossa elementtien tarkoitus on määritellä web-palvelun toimintoja, kuten sisään- ja ulostuloportteja, lähetettävien ja vastaanotettavien viestien sisältöä sekä tietotyyppejä.

Kuvassa 3 voidaan nähdä yksinkertainen WSDL-tiedosto. Tästä kuvasta nähdään, kuinka WSDL-tiedostot aloitetaan usein määritelmillä. Tämä tapahtuu

elementin ”definitions” attribuutteina.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://www.example.org/ruokalista/"
    name="ruokalista" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:tns="http://www.example.org/ruokalista/" xmlns:rka="http://www.ex
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:xsd="http://www.w3.
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.example.org/schema/ruokalistaSchema.xsd
    <wsdl:types>
        <xsd:schema elementFormDefault="qualified">
            <xsd:import namespace="http://www.example.org/schema/ruokalista
                schemaLocation="../schema/ruokalistaSchema.xsd" />
        </xsd:schema>
        <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.example.org/ruokalista/" />


    </wsdl:types>
    <wsdl:message name="pyyntöRequest">
        <wsdl:part name="Document" element="rka:viikkoLista" />
    </wsdl:message>

    <wsdl:message name="validointiRequest">
        <wsdl:part name="parameters" element="rka:viikkoLista" />
    </wsdl:message>
    <wsdl:message name="validointiResponse">
        <wsdl:part name="message" type="xsd:string" />
    </wsdl:message>
    <wsdl:message name="NewMessage">
        <wsdl:part name="parameters" element="rka:viikkoLista" />
    </wsdl:message>
```

Kuva 3. WSDL on XML-pohjainen kuvauskieli.

Kuvassa myös nähdään ”message” elementit. Nämä elementit ovat tärkeitä, sillä ne ovat käytännössä muuttujia, joita käytetään eri sisään-, ja ulostulo -operaatioissa. Esimerkiksi otetaan kuvasta nähty ”message” nimeltä pyyntöRequest.

```
<wsdl:message name=”pyyntöRequest”>
     <wsdl:part name=“Document” element=”rka:viikkoLista”/>
</wsdl:message>
```

Elementille wsdl:message määritetään vain attribuutti ”name”, eli nimi, jolla tämä tunnistetaan muiden elementtien keskellä. Messagen sisällä on elementti ”part”, johon kiinnitetään tietotyyppi ”rka:viikkoLista”. Käytännössä tämä on siis määritelmä sisällölle, jota haluamme tämän message-tyypin sisältävän. Part-elementtejä voi olla useampi, mikäli näin vain määritetään.

Seuraavaksi luodaan portti/operaatio, joka käyttää äsken määriteltyä ”message”-elementtiä.

```
<wsdl:portType name=”lähetäPyyntö”>
     <wsdl:operation name=”pyydä”>
          <input message=”pyyntöRequest”/>
     </wsdl:operation>
</wsdl:portType>
```

Tässä portille annetaan nimeksi lähetäPyyntö. Se sisältää operaation nimeltä ”pyydä”. Tämä operaatio ottaa syötteenä viestin, joka on muotoa pyyntöRequest.

Seuraavaksi käydään läpi kaksi keskeistä teknologiaa WSDL:n kehityksen ajalta: SOAP ja UDDI. Näistä kummatkin ovat kehitetty hieman ennen WSDL:ää. Niiden tarkoitus on tarjota yhtenäinen työkalupakki web-palveluiden kehitykselle liiketoimintajärjestelmissä.

2.4.1 SOAP

Vaikka tässä kehitysohjeessa oletuksena on, että viestintä tapahtuu REST:n kautta, on silti hyvä olla tietoinen siitä, mikä SOAP on.

SOAP eli Simple Object Access Protocol on viestintäprotokolla, joka kehitettiin Microsoftille vuonna 1998 (12.). SOAP:n versio 1.1 lähetettiin vuonna 2000 W3C:n hyväksyttäväksi, mutta se ei mennyt läpi (13.). Vasta versio 1.2 saavutti web-standardin statuksen vuonna 2003 (14.).

SOAP on suunniteltu toimimaan web-palvelujen yhteydessä viestintäprotokollana. Sen tarkoitus on toimia yhdistävänä protokollana, jolla kuljetetaan sille määritettyä dataa eri ohjelmien/järjestelmien välillä. SOAP yleensä käyttää muuta protokollaa datan kuljetukseen, kuten http:tä. Käytännössä se toimii luomalla jokaiseen viestiin kolme osaa:

1. ”Envelope”, joka kuvaa viestin rakennetta ja kuinka se tulee käsitellä.

2. ”Encoding rules”, eli lista koodaussäännöistä, jotka sisältävät ulkoisesti määritellyt datatyypit.

3. ”Communications styles”, viestintä, joka on jaettu kahteen eri kategoriaan: ”remote procedure call (RPC) ja viestipohjainen dokumentti. RPC:n ideana on käynnistää ulkoinen operaatio, joka palauttaa jotain. (15.)

SOAP on hyvin vanha standardi, mutta sitä käytetään edelleen järjestelmissä, joiden uudelleenrakentaminen moderneilla standardeilla tulisi turhan kalliiksi.

2.4.2 UDDI

UDDI eli ”Universal Description Discovery Integration” on vuonna 2000 julkaistu, SOAP-pohjainen XML-protokolla. Sen on kehittänyt yhteistyössä Ariba (nyk. SAP Ariba), IBM ja Microsoft. UDDI:n tarkoitus on olla ns. ”liiketoimintojen puhelinluettelo internetissä”, jonka pohjalta voidaan järjestää kommunikaatio/viestintäjärjestelmiä web-palveluiden kanssa. (16.)

UDDI:n tehtävä BPEL-prosesseissa on toimia web-palveluiden välitysalustana, eli sillä voidaan hakea tai julkaista web-palveluiden kuvauksia. Tässä kehitysohjeessa sitä ei kuitenkaan käytetä.

UDDI siis ylläpitää web-palveluiden rekisteriä, johon yritys voi rekisteröityä mukaan, julkaisten omat web-palvelunsa yleiseen listaan. Nämä palvelut ovat kuvattu XML-muodossa. UDDI myös tarjoaa yksityisiä rekistereitä testausta ja kehitystä varten. (17.) Vuonna 2016 Microsoft poisti UDDI-palvelut uusimmasta BizTalk Server -ohjelmistosta (18).

2.5 XML Schema

XML Schema on yleinen termi kielelle, jolla määritellään XML-dokumentin rakenne. Koska WSDL on XML-pohjainen tiedostomuoto, käytetään XML Schema

-kieliä niiden määrittelyissä. Tässä kehitysohjeessa käytetään W3C:n suosittelemaa XML Schema -kieltä: XSD:tä.

Kun XML-tiedostoja luodaan mihin tahansa haluttuun käyttötarkoitukseen, vaatii se jonkin näköisen tarkastuksen, että dokumentti on varmasti validi käyttötapaukseen. Tätä varten on olemassa XML Schema, jolla voidaan validoida dokumentin sisältö, että siinä asetetut elementit sisältävät niihin tarkoitettua tietoa.

Vanhin näistä XML Schema -kielistä on DTD (”Document Type Definition”), jonka XML peri edeltäjältään: SGML-kieleltä (”Standard Generalized Markup Language”). XSD julkaistiin vuonna 2001 W3C:n suosittelemana standardina, tavoitteena tarjota aikaisempaan DTD:n verrattuna parannuksia. Iso osa XSD:n tarjoamista parannuksista oli nimiavaruuksien tukeminen. (19.)

Kuvassa 4 nähdään XSD-tiedoston sisäinen rakenne, joka muistuttaa hyvin paljon aiemmin nähtyjä WSDL- ja BPEL-tiedostoja.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.example.org/schema/ruokalistaSchema.xsd" xmlns:tns="htt
    attributeFormDefault="unqualified">

    <element name="viikkoLista" type="tns:viikkoLista"/>
    <complexType name="viikkoLista">
        <sequence>
            <element name="maanantai" type="tns:päivanRuoka"></element>
            <element name="tiistai" type="tns:päivanRuoka"></element>
            <element name="keskiviikko" type="tns:päivanRuoka"></element>
            <element name="torstai" type="tns:päivanRuoka"></element>
            <element name="perjantai" type="tns:päivanRuoka"></element>
        </sequence>
    </complexType>
    <complexType name="päivanRuoka"><sequence>
            <element name="liha" type="string"></element>
            <element name="kasvis" type="string"></element>
        </sequence>
    </complexType>
</schema>
```

Kuva 4. XSD-tiedosto sisältää tietotyyppien määritelmiä XML-muodossa.

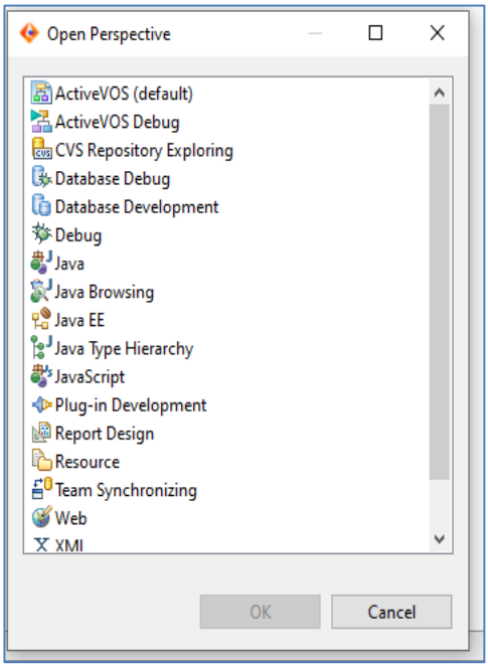
DTD ei myöskään tue elementtien arvojen tyypitystä. Tämä on ongelma, mikäli yritetään rakentaa isoa järjestelmää, jossa voi olla kymmeniä, ellei satoja eri palveluita, joista jokainen lähettää ja vastaanottaa suuria määriä dataa. Ongelmien korjaaminen, saati sitten löytäminen virheen sattuessa voi olla hyvin hankala urakka. Tämä ja nimiavaruuksien puute, ovat isoimmat syyt, minkä takia

BPEL-prosessien yhteydessä XSD on järkevämpi valinta WSDL-tiedostojen tietotyyppien määrittämiseksi. (20.)

3 Käyttöliittymä ja editorin käyttö

3.1 Yleiskuva ja tärkeimmät kohdat

Kuten aikaisemmin mainittiin, ActiveVOS Designer pohjautuu Eclipse-sovelluskehittimeen, eli Eclipsen käyttäjä voi huomata samankaltaisuuksia sovellusten välillä kuten ”Perspective”-toiminto, eli perspektiivi, joka on muokattu näyttämään halutun tehtävän toimintoja sekä ominaisuuksia. Nämä perspektiivit ovat listattu ”Perspective”-valikkoon, kuten kuvassa 5 näkyy.

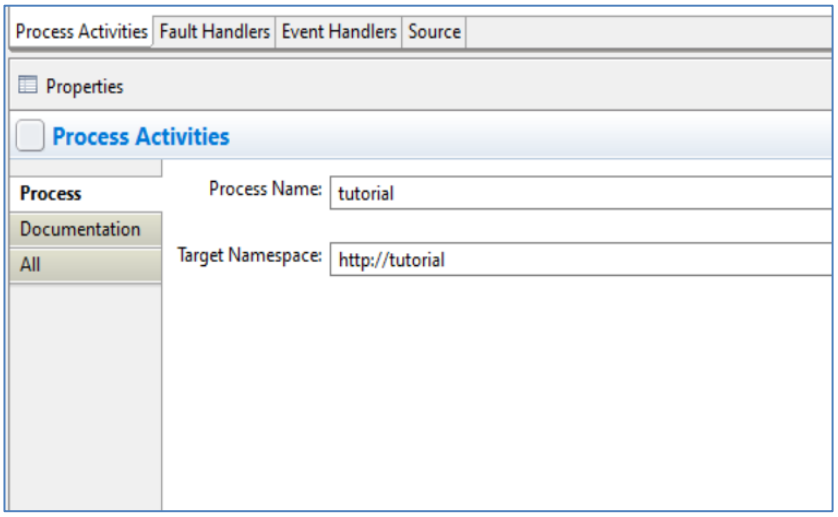


Kuva 5. Perspective–valikko.

Näistä yleisin sekä tärkein on ”ActiveVOS(default)” eli oletusperspektiivi. Tämä on keskeisin perspektiivi, sillä se sisältää kaikki halutut työkalut, joita tarvitaan BPEL-kehitystä varten.

Perspektiiveihin voi myös avata näkymiä (eng. ”View”), jotka sisältävät hyödillisiä työkaluja ja tietoja. Nämä näkymät ovat monesti kontekstipohjaisia, eli ne vaativat valitun objektin editorista toimiakseen.

Hyvä esimerkki tällaisesta kontekstin vaativasta näkymästä on ”Properties” eli ominaisuudet-näkymä. Tämän näkymän tarkoitus on näyttää valitun objektin tarpeellisia tietoja. Kuvassa 6 näkyy, millainen properties-näkymä on, kun kontekstiksi on valittuna koko prosessi.

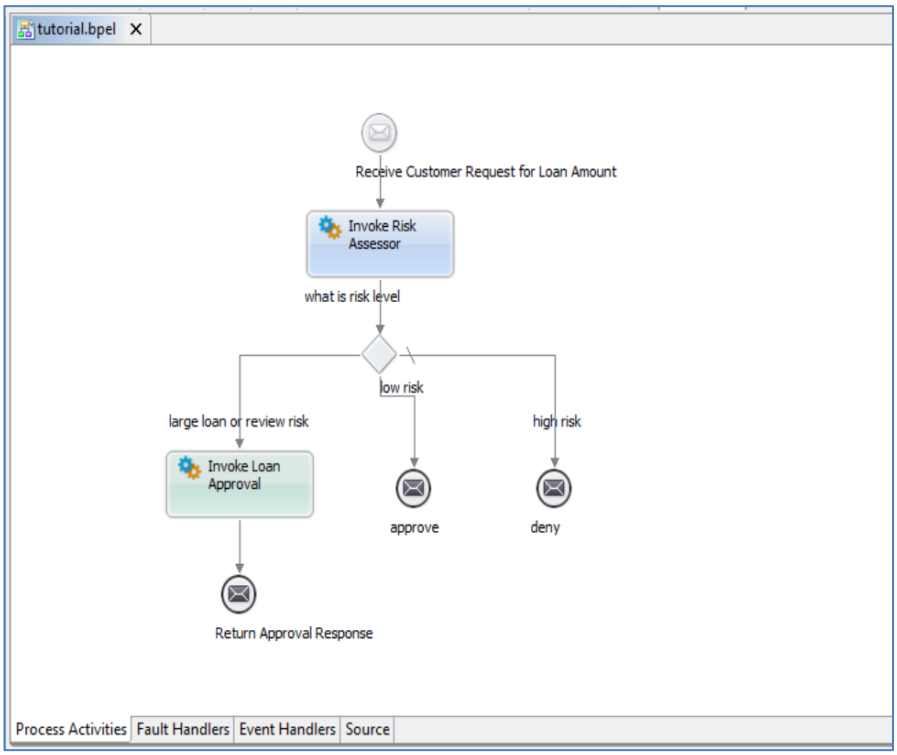


Kuva 6. Properties-näkymä.

Tämä näkymä muuttuu kontekstin mukaan, ja voi sisältää useita syöttö- ja valintakenttiä kontekstin vaatimuksien mukaan. Esimerkiksi, jos kyseinen konteksti olisi aktiviteetti, joka pystyy ottamaan syötteen kautta dataa, olisi properties-näkymässä kenttä tälle syötteelle.

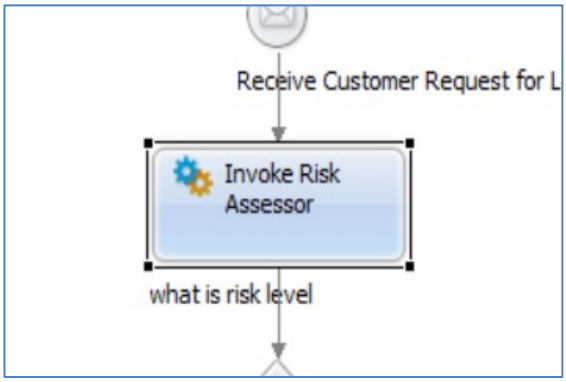
3.2 Visuaalinen editori-ikkuna

Designerin toiminnot ovat hyvin riippuvaisia kontekstista kuten aikasemmin mainittiin. Tämä tulee hyvin esiin ohjelman keskeisimmässä toiminossa: kuvassa 7 näkyvässä visuaalisessa editointi-ikkunassa.



Kuva 7. Editor-ikkuna, tunnettu myös nimellä "Canvas".

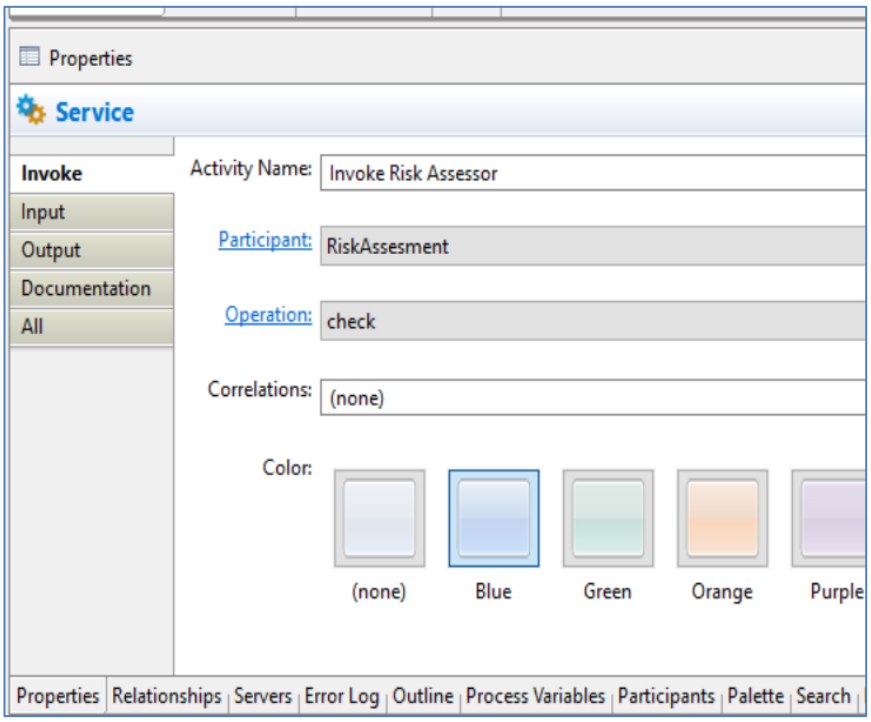
Tämä ikkuna tarjoaa mahdollisuuden valita ja siirrellä objekteja sekä järjestellä objekteja, joilla luodaan yhteyksiä niiden välille. Objektien välille piirretyt nuolet ilmaisevat hyvin prosessin suoritusjärjestyksen kuten kuvassa 8 näkyy.



Kuva 8. "Invoke Risk Assessor" -objekti valittuna editorissa.

Objekteja ikkunassa voi muokata klikkaamalla, mikä tarjoaa tarvittavan kontekstin muille muokkaustyökaluille kuten aiemmin mainittu ”Properties”.

Kuvassa 9 näkyvän “Properties”-näkymän avulla voidaan muokata objektin ominaisuuksia, mitä se tekee ja siihen liittyvää metadataa (väri kaaviossa, dokumentaatio, yms).

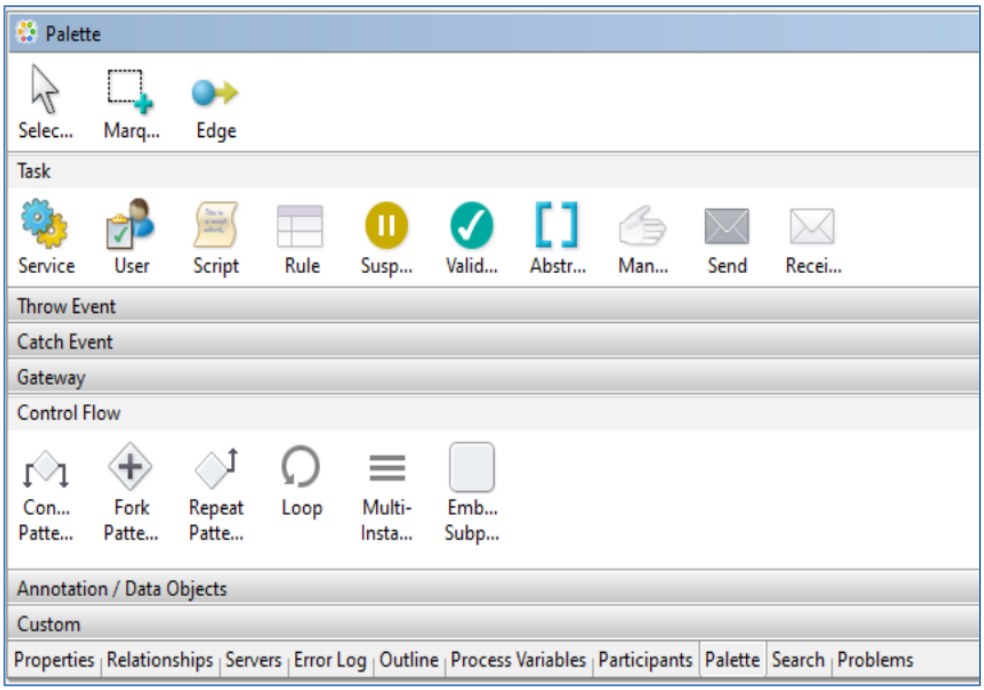


Kuva 9. "Invoke Risk Assessor" Properties-ikkunassa.

Nämä muokattavat kentät sekä valinnat ovat jaettu eri kategorioihin, jotka ovat listattu properties-näkymän vasempaan laitaan.

3.3 ”Palette”-näkymä

”Palette” eli paletti on hyvin tärkeä näkymä kehitykseen liittyen, sillä se sisältää kaikki Designeriin kuuluvat objektit. Paletissa listatut objektit on jaettu niitä kuvaaviin kategorioihin kuten kuvassa 10 näkyy.



Kuva 10. Paletti-näkymä.

Nämä kategoriat kuvaavat haluttua toimintoa, jonka siihen valitut objektit pystyvät tavalla tai toisella suorittamaan. Objekteja voi vapaasti vetää editori-ikkunaan paletista hiirellä.

4 Esimerkkitilanne ja läpikäynti

Tässä luvussa käydään läpi esimerkki vaihe vaiheelta, jonka kautta voidaan selittää objektien sekä niihin liittyvien ominaisuuksien ja tiedostojen toimintaa sekä merkitystä prosessissa.

4.1 Esimerkin alustus

Prosessin tarkoitus tässä esimerkissä on tarkastaa sekä hyväksyttää ruokalista, jonka käyttäjä on jotain reittiä syöttänyt järjestelmään. Käyttäjän syöttötavalla ei ole merkitystä, sillä tämä kuvitteellinen arkkitehtuuri on järjestelty niin, että ruokalista saapuu prosessiin JSON-muodossa.

Ruokalista hyväksytetään vain, jos se täyttää seuraavat ominaisuudet.

1. Se sisältää pääruuan jokaiselle viidelle arkipäivälle.

2. Jokaisena päivänä on olemassa kasvisvaihtoehto.

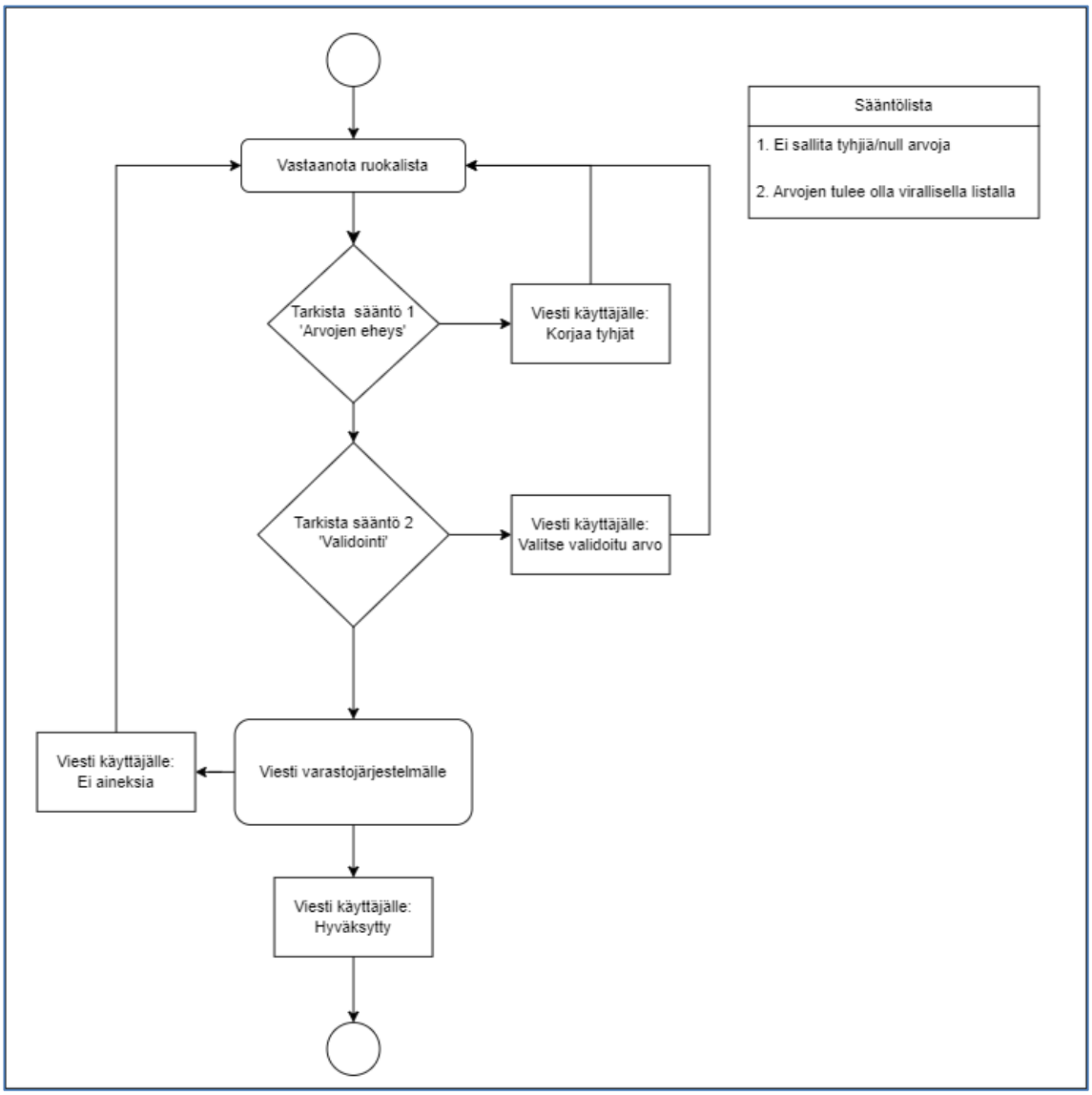
3. Varastossa on aineksia valmistukseen jokaiselle päivälle.

Ideana on, että ruokalistassa on jokaiselle arkipäivälle kaksi lounasvaihtoehtoa: kasvis ja sekaruoka. Se, miten nämä ehdot tarkastetaan, selvitetään läpikäynnin aikana.

4.2 Prosessin läpikäynti

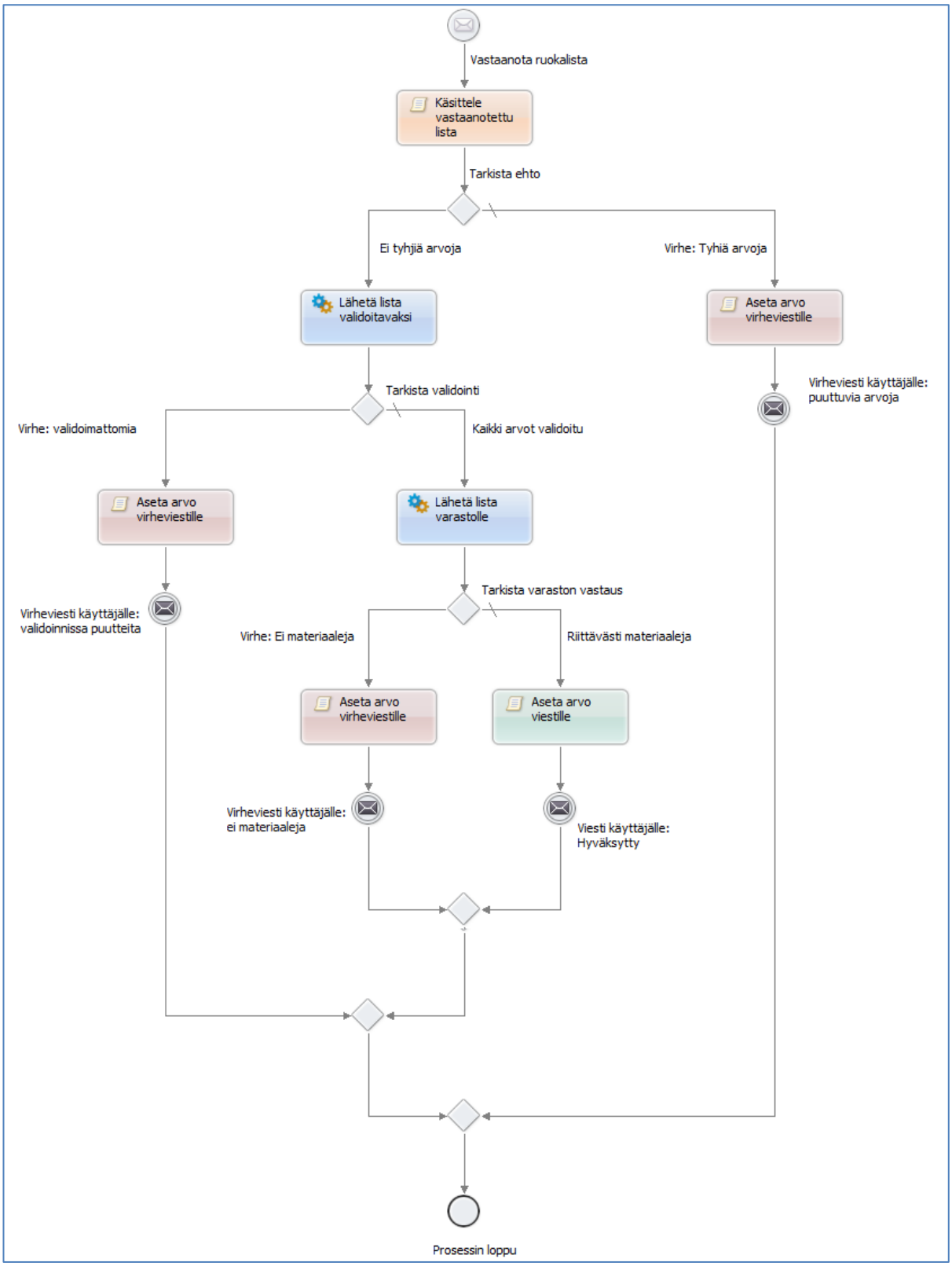
Prosessi, jota käydään läpi, on logiikaltaan yksinkertainen, mutta toteutus voi ensisilmäykseltä vaikuttaa turhan monimutkaiselta. Tämä johtuu siitä, että esimerkin tarkoitus on havainnollistaa mahdollisia toteutustapoja. Mikäli jokin tapa ei ole käytännöllinen, mainitaan siitä läpikäynnin aikana.

Prosessin sisäinen logiikka voidaan tiivistää seuraavanlaisesti: ensimmäisenä tarkistetaan vastaanotetun datan eheys, eli katsotaan, että jokaisena päivänä on kaksi ruokalajia. Kummatkaan kentät eivät saa olla tyhjiä. Kuvassa 11 näkyy esimerkki, josta on piirretty yksinkertainen vuokakaavio.



Kuva 11. Prosessin logiikka kaaviomuodossa.

Seuraavaksi tarkistetaan, että annetut arvot ovat ns. ’valideja’ eli ne käytetään validointiohjelman läpi. Mikäli nämä ehdot täyttyvät, lähetetään kysely varasto-ohjelmalle. Mikäli vastaus on myöntävä, hyväksytään vastaanotettu lista ja lähetetään siitä viesti käyttäjälle. Jos missään kohtaa prosessia ehto ei toteudu, lähetetään siitä käyttäjälle viesti, jossa kerrotaan, minkälainen ongelma on kyseessä. Kuvassa 12 tämä kokonaisuus näkyy ActiveVOS-toteutuksena ohjelman omassa editor-ikkunassa.



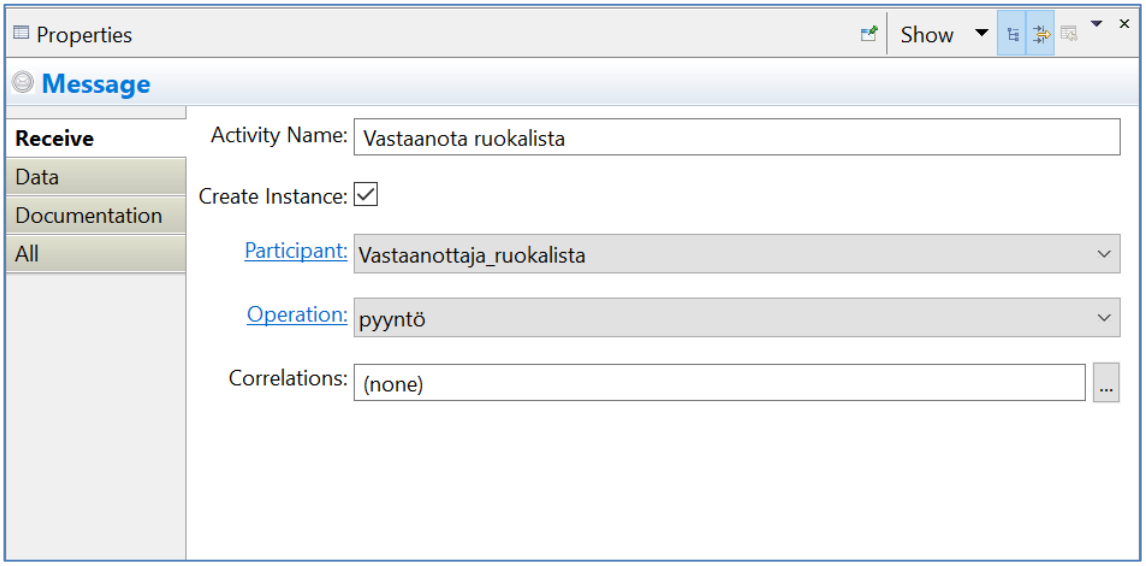
Kuva 12. Prosessin toteutus ActiveVOS-kaaviona.

Kuvassa 12 näkyvät aktiviteetit ovat väritetty selvyyden vuoksi, eikä niillä ole mitään toiminnallista virkaa.

4.2.1 Prosessin aloitus – Message Receive

Prosessi aloitetaan ”Receivellä” eli viestin vastaanottavalla aktiviteetilla. Kaaviossa se on prosessin ylin kohta nimellä ”Vastaanota ruokalista”.

Receive-aktiviteetin tarkoitus on vastaanottaa viesti sille määritetyllä tavalla ja muodolla. Kuvassa 13 nähdään aktiviteetin ”properties”-osuus, jossa nämä asetukset on valittu.



Kuva 13. Receive-aktiviteetin "properties"-näkymä.

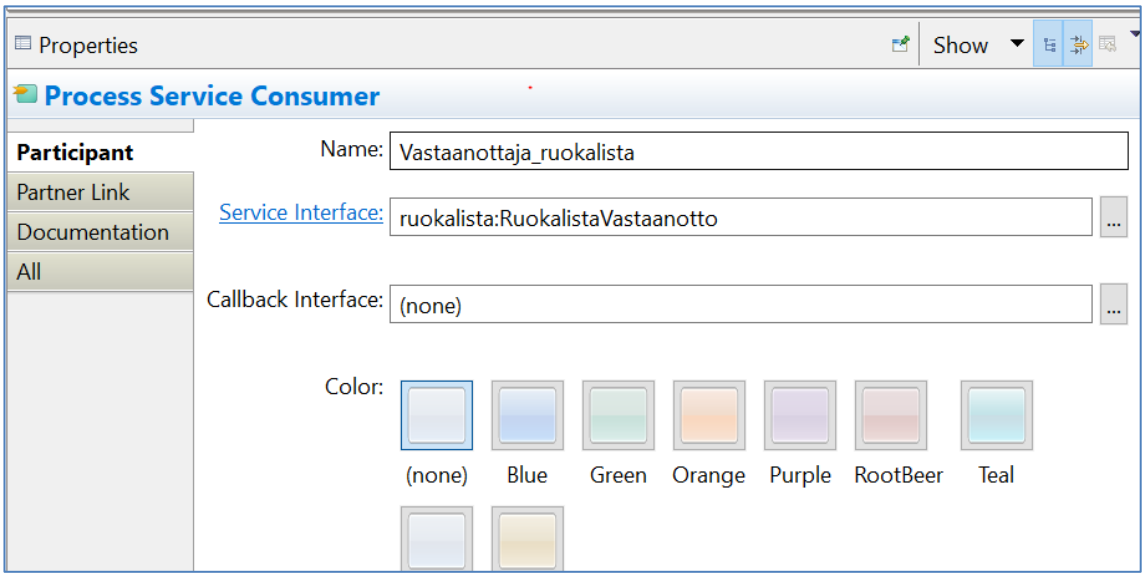
Kuvasta 13 nähdään aktiviteetin properties-näkymän ”receive”-osuus. Tässä kohdassa on määritelty aktiviteetin keskeisimmät asetukset, näitä ovat.

- ”Activity name” eli aktiviteetin nimi on myös näkyvillä kaaviossa.
- ”Create instance”, luoko aktiviteetti instanssin, eli alustaa prosessin. Tämä valinta tulee olla päällä, jos aktiviteetti aloittaa prosessin.
- ”Participant” Määritelty rajapinta, joka tarjoaa aktiviteetille suoritettavia operaatioita. Sisältää myös tietorakenteet, joita aktiviteetti käyttää.
- ”Operation”, valittu operaatio, jonka aktiviteetti suorittaa.

- ”Correlations” eli korrelaatiot, joiden tarkoitus on tehdä viesteistä uniikkeja. Esim. käyttäjän tunnusluku.

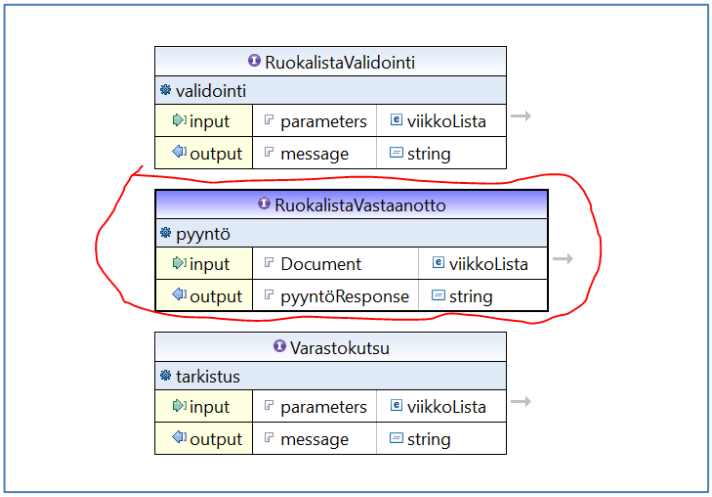
Tässä aktiviteetissä tärkeä osa on ”Vastaanottaja_ruokalista”. Tämä PSC on tarkoitettu vastaanottamaan prosessin keskeisimmän tietotyypin: viikkoLista-rakenteen.

Vastaanottaja_ruokalista on konfiguroitu käyttämään ruokalista-palvelurajapinnassa määriteltyä porttia: RuokalistaVastaanotto. Tämä näkyy käytännön toteutuksena kuvassa 14.



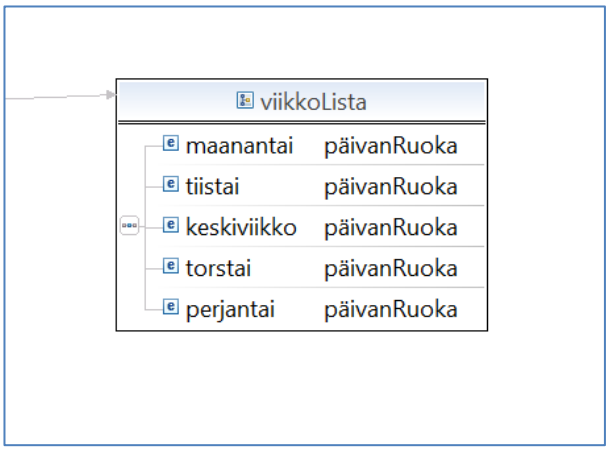
Kuva 14. Vastaanottaja_ruokalistan properties–näkymä.

Kuvassa 15 näkyy, kuinka ”RuokalistaVastaanotto” sisältää määritelmän, jossa kerrotaan, mitä portin on tarkoitus vastaanottaa ja missä muodossa.



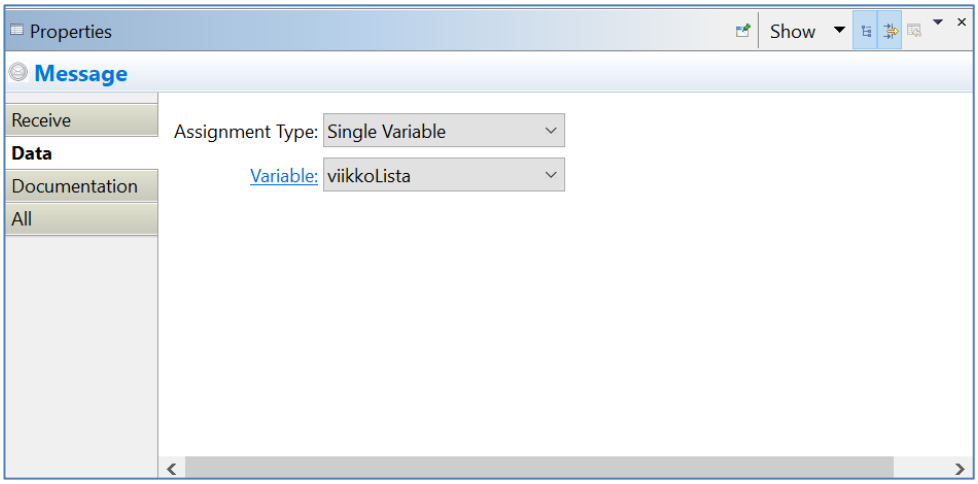
Kuva 15. ruokalista.wsdl kuvattuna.

RuokalistaVastaanotto on määritelty vastaanottamaan viikkoLista-tyyppisen rakenteen, joka näkyy graafisessa muodossa kuvassa 16. Tämä rakenne on käytännössä ruokalista, johon koko prosessi perustuu. Lista sisältää alkion jokaiselle arkipäivälle maanantaista perjantaihin. Jokainen päivä on tyyppiä päivänRuoka, joka sisältää kaksi alkiota: liha ja kasvis. Näihin alkioihin on tarkoitus tallentaa nimen viittaama ruokalaji.



Kuva 16. viikkoLista-rakenne kuvattuna.

Vaikka tämä rakenne voi vaikuttaa ensisilmäykseltä turhankin monimutkaiselta, on se käytännössä hyvin yksinkertainen. ”Receive”-aktiviteetti käyttää rajapintaa pyytääkseen viikkoLista-tyyppisen rakenteen. Tämä rakenne siirretään muuttujaan ”viikkoLista”, jota prosessi voi käyttää ja muokata. Kuvassa 17 näkyy, miten tämä tapahtuu käytännössä.

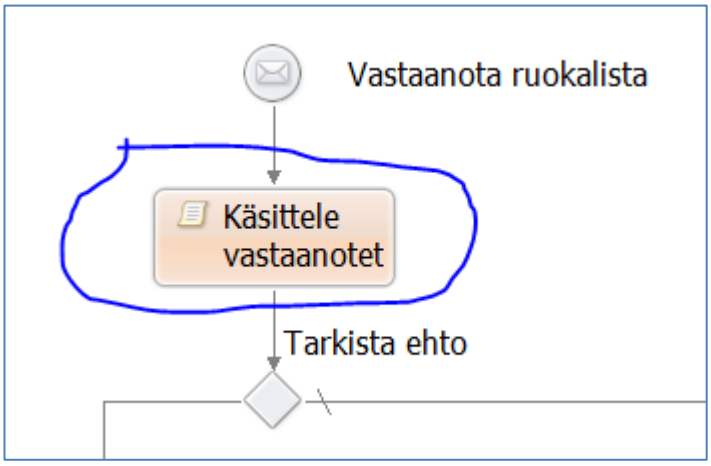


Kuva 17. "Receive" siirtää tiedon muuttujaan "viikkoLista".

Tämä muuttuja saa olla minkä tahansa niminen, mutta selvyyden vuoksi nimetään se tietotyypin mukaan ”viikkoLista”. Kun tämä operaatio on suoritettu ilman virheitä, siirrytään seuraavaan kaavion aktiviteettiin: Script-aktiviteettiin.

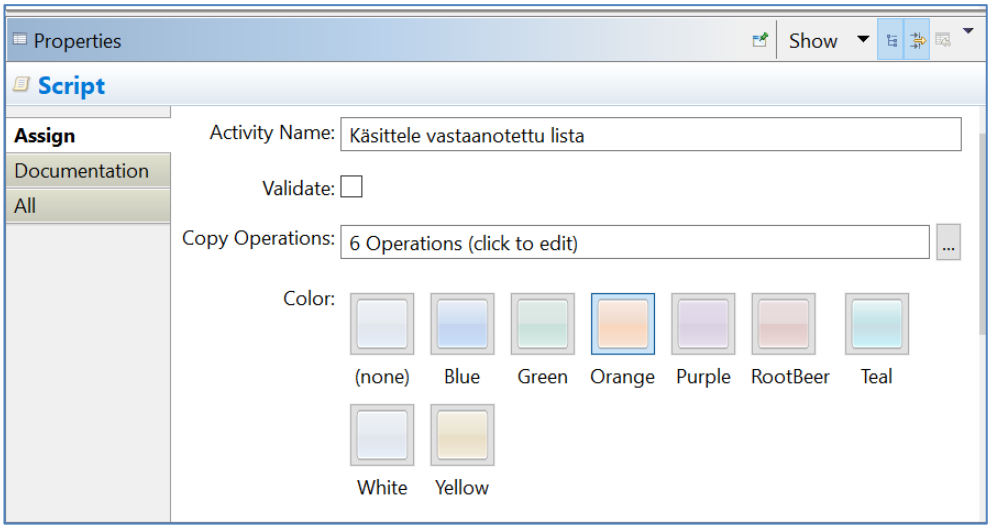
4.2.2 Script-aktiviteetti

Kuvassa 18 näkyvä script-aktiviteetti on tarkoitettu tiedon muotoiluun, formatointiin sekä yleisiin logiikkaa vaativiin operaatioihin.



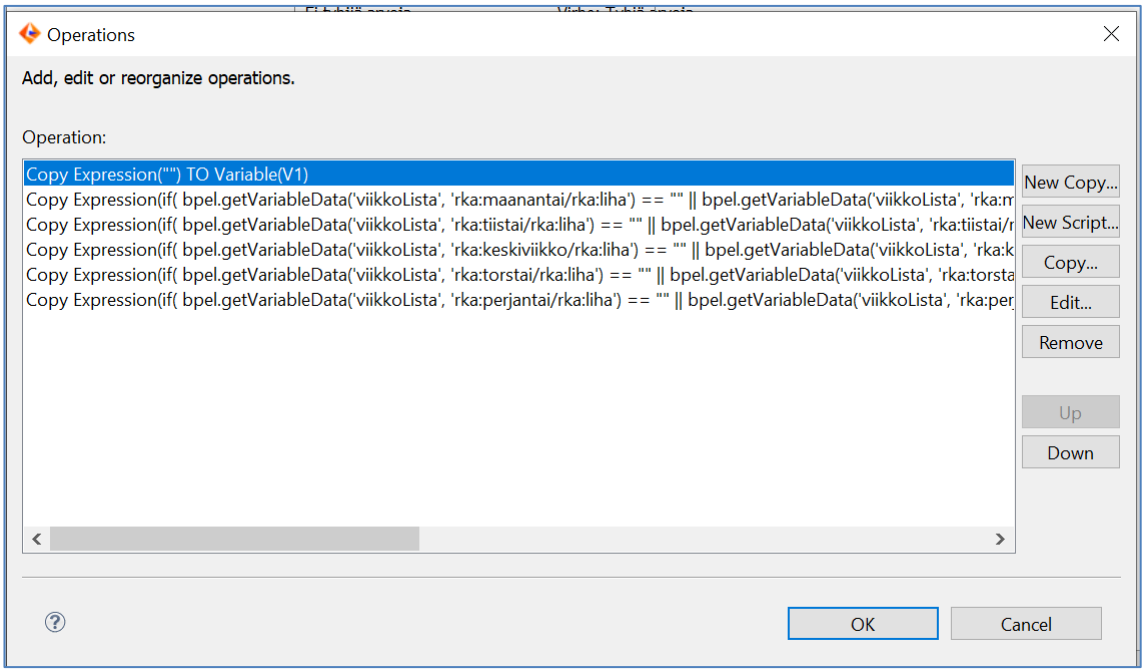
Kuva 18. Script-aktiviteetti kaaviossa.

Hyvä esimerkki script-aktiviteetin käyttötarkoituksesta on tyhjien kenttien tarkastus. Tietorakenne, jonka prosessi vastaanottaa, saattaa sisältää kohtia, joihin käyttäjä ei ole syöttänyt dataa. Nämä kohdat ovat hyvä tarkastaa itse kirjoitetulla koodipalalla. Kirjoitettu koodi löytyy kuvan 19 properties-näkymän ”Copy Operations” -kohdasta.



Kuva 19. Script-aktiviteetin properties-näkymä, jossa näkyy ”Copy Operations” -kenttä.

Klikkaamalla ”Copy operations”-kohdan oikeasta laidasta olevaa kuvaketta (kolme pistettä) avautuu kuvassa 20 näkyvä ”Operations”-ikkuna, jossa suoritettavat koodipalat ovat suoritusjärjestykseen pohjautuvassa listassa.



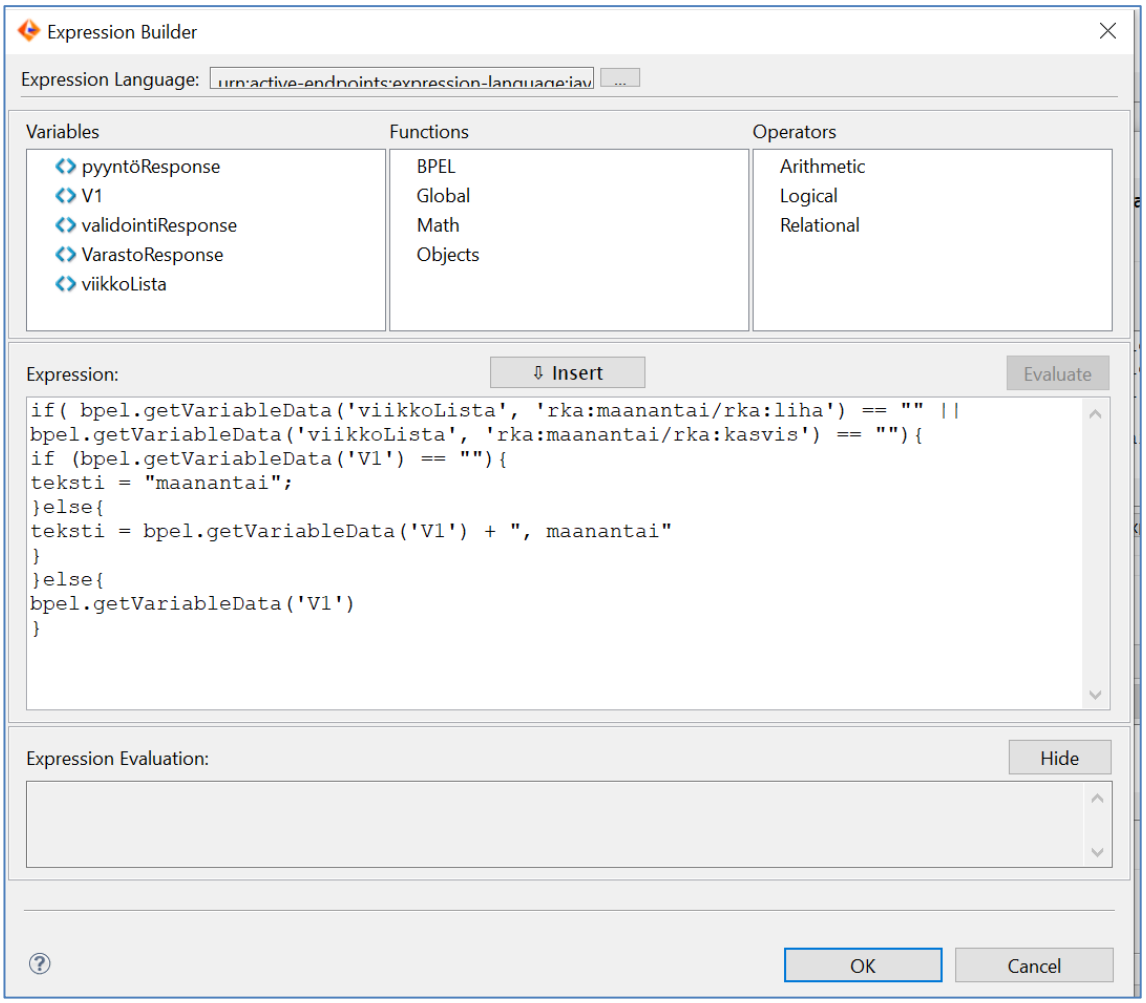
Kuva 20. Operations-ikkuna.

Operations toimii skriptipohjaisella expression-systeemillä, jonka idea on suorittaa sille syötetty koodi, ja palauttaa sen tulos valittuun päätteeseen. Tämä pääte on usein muuttuja tai muuttujan attribuutti, mutta voi myös olla toinen skripti.

Koodikieli skripteille on valittavissa kolmesta eri vaihtoehdosta: XPath 1.0, XQuery 1.0, ja JavaScript 1.5. Nämä kielet ovat vanhoja, eli mahdolliset ratkaisut, jotka löytyvät netistä, eivät välttämättä toimi näillä versioilla. Projekteihin voi lisätä ylimääräisiä kirjastoja, mikäli tarve vaatii lisätoiminnallisuuksia.

Vaikka kieltä voi käyttää operaatioiden yhteydessä, ei se tarkoita, että se toimisi kuten natiivissa ympäristössään, eli esim. JavaScript ei sisällä sen kaikkia luontaisia kirjastoja, joita web-ympäristössä on totuttu käyttämään. Sen sijaan käytössä on ActiveVOS:in oma kirjastofunktioita, jotka ovat editorissa listattuna.

Kuvassa 21. nähdään ”Expression builder” eli editorin toimintoja. Näihin kuuluu lista prosessin muuttujista vasemmalla, joista voi valita halutun muuttujan tai muuttujan osan (mikäli muuttuja on taulukkopohjainen tietotyyppi). Tämä skripti on kirjoitettu JavaScript-kielellä.



Kuva 21. Expression builder -ikkuna.

Valittu muuttuja ilmestyy editoriin muodossa:

```
bpel.getVariableData(“muuttujan_nimi”,”attribuutin polku”)
```

Muuttujat ovat usein kompleksisia tietotyyppejä (eli taulukko, matriisi), joten funktio, jolla muuttuja haetaan, vaatii ylimääräisenä parametrina polun haluttuun attribuuttiin. Esimerkiksi jos halutaan muuttujasta ”viikkoLista” päivän maanantai liharuoka, haetaan se seuraavanlaisesti:

```
bpel.getVariableData('viikkoLista','rka:maanantai/rka:liha')
```

Editorissa muuttujan sekä attribuutin valittua täyttää ohjelma yllä olevan syntaksin automaattisesti, jolloin ei käyttäjän tarvitse muistaa ulkoa tietotyyppien nimiavaruuksia (namespace) skriptiä kirjoittaessa. Tämä usein näkyy kehitystyössä, jossa .wsdl-tiedostot, jotka sisältävät nämä tietotyypit, tulevat kehitystiimin ulkopuolelta (asiakas, yleinen rajapinta yms.).

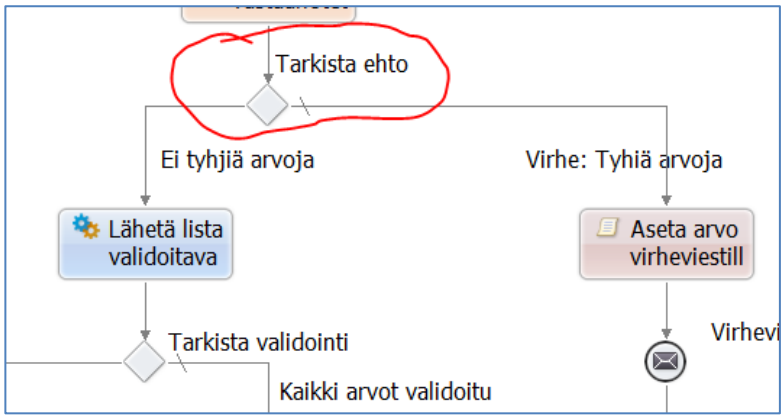
Esimerkkiprosessissa script-aktiviteetin logiikka on seuraavanlainen: ensimmäinen operaatio alustaa muuttujan ”V1” kopioimalla siihen tyhjän merkkijonon. Tämän jälkeen käydään jokaisen päivän ruokalajit läpi. Jokainen operaatio on omalle päivälleen. Tämä on toteutettu näin siksi, ettei aktiviteetti sisällä yksittäistä isoa skriptiä, jota on hankala lukea.

Näin on myös jatkossa helpompi etsiä ongelmia, mikäli prosessi sattuu keskeytymään skript-aktiviteetin aikana. Jokaisen päivän kohdalla tarkistetaan, onko liha- tai kasviskenttä tyhjä. Mikäli näin on, palautetaan muuttuja ”teksti”, johon lisätään ”V1” sisältämä tieto + päivän nimi (maanantaina ”maanantai”, tiistaina ”tiistai” yms.).

Jos taas kummatkin kentät sisältävät jotain, palautetaan vain ”V1”-arvo, joka kopioidaan itseensä takaisin. Tämän tuloksena, kun kaikki päivät on käyty läpi, sisältää ”V1” pilkuilla eritellyn merkkijonon, johon on listattu päivät, jotka sisältävät tyhjiä kenttiä. Pidetään mielessä ”V1”-muuttuja ja siirrytään seuraavaan kohtaan prosessissa: if-valintarakenne.

4.2.3 If-valintarakenne/conditional-pattern

Seuraavaksi prosessi saapuu ensimmäiseen ehdolliseen valintarakenteeseen. Kyseessä on kuvassa 22 näkyvä yksinkertainen if-ehto, joka löytyy paletista nimellä ”conditional pattern”.

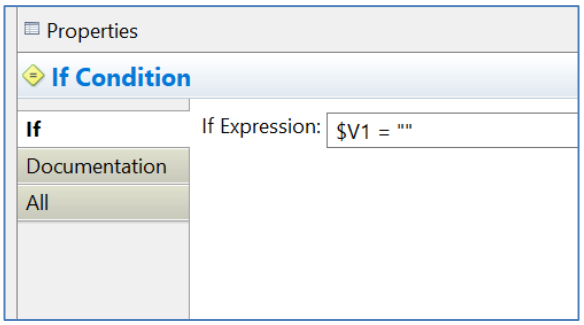


Kuva 22. If-ehto kaaviossa.

Kaaviossa esiintyvä rakenne on hyvin yksinkertainen. Minimissään se koostuu kahdesta haarasta, jotka ovat ”if-ehtohaara” sekä ”else-haara”. If-ehtohaaraan on määritelty tutulla expression-tyylisellä skriptillä lauseke, joka palauttaa joko ”true”- tai ”false”-arvon, jos arvo on ”true”, jatkaa prosessi if-ehtohaaraa pitkin. Jos taas arvo on ”false”, jatketaan else-haaraan.

Rakenteeseen voi lisätä useamman ”if-haaran”, joiden määritellyt ehdot tarkistetaan vasemmalta oikealle, mikäli edeltävä ehto ei mene läpi. Tämä jatkuu, kunnes päädytään ”else”-haaraan.

Kuvassa 23 näkyy if-ehdon properties-näkymä, jossa on yksittäinen syöttökenttä. Tähän kenttään lisätään ehto, jota if-haara testaa tullakseen päätökseen, mikä haara valitaan.



Kuva 23. If-ehdon properties-näkymä.

Tämä esimerkkiprosessi on suunniteltu niin, että mikäli ehto, jonka haluamme toteutuvan (eli ei tyhjiä arvoja, annetut arvot validoituja yms.), ei toteudu, menee prosessi tilaan, jossa se ilmoittaa käyttäjälle virheellisestä syötteestä, jonka käymme läpi seuraavaksi.

4.2.4 Virhetila, arvonasetusskripti ja reply-aktiviteetti

Oletetaan, että aikaisempi script-aktiviteetti löysi tyhjiä kenttiä, ja lisäsi niitä sisältävät päivät muuttujaan ”V1”. Tämä muuttuja jäi edellisessä if-ehdossa kiinni ja ohjasi prosessin virheenilmoitusosioon. Tämä ei ole niin sanotusti ”oikeaoppinen” virhetila, sillä prosessin suoritus on sujunut suunnitellun mukaisesti. Tällaista virhetilaa varten on olemassa ”Fault Handler”-toiminnallisuus, mutta sitä ei käytetä tässä esimerkissä. Vaikka viittaamme tähän osioon ”virhetilana”, on se oikeasti vain vaihtoehtoinen suoritusreitti, eli ”else-haara”.

Kuvassa 12 nähdään tämä osio oikeassa laidassa. Se koostuu kahdesta aktiviteetistä: script-aktiviteetista, jonka tarkoitus on asettaa virheviestille arvo, sekä reply-aktiviteetista, joka lähettää tämän viestin käyttäjälle.

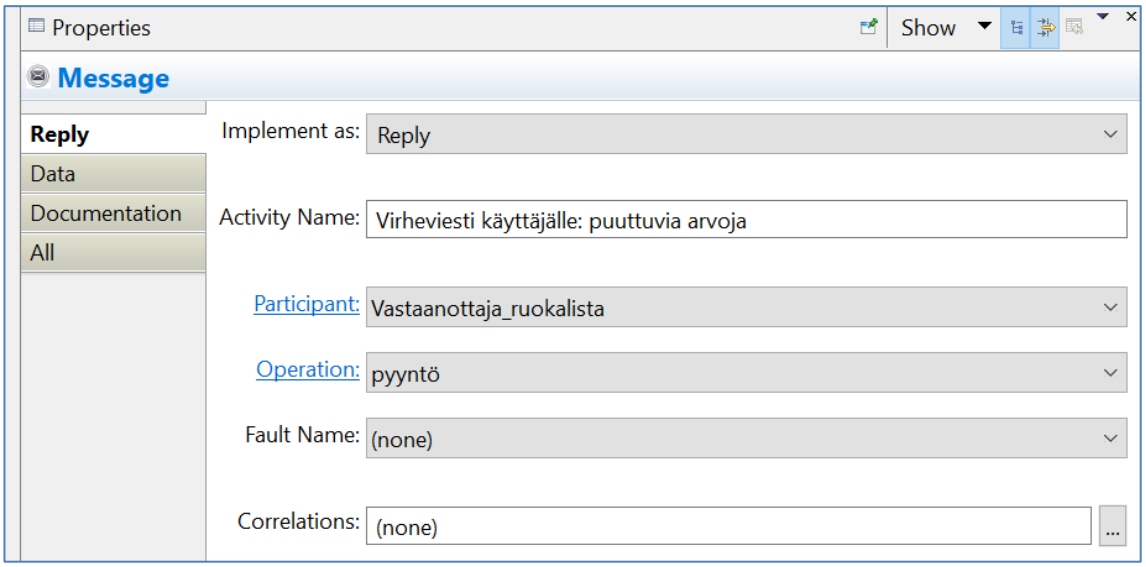
Kuvasta 12 voidaan huomata, että tämä osio ohjaa prosessin suoraan loppuun ohittaen muut aktiviteetit. Kaikki muut ”virhetilat” seuraavat tätä samaa kaavaa lähettäen käyttäjälle virheviestin ja ohjaten prosessin lopetukseen. Ainoana erona on käyttäjälle lähetettävän virheviestin sisältö.

Script-aktiviteetti ”Aseta arvo virheviestille”, nimensä mukaisesti, asettaa arvon virheviestille. Se sisältää vain yhden kopiointioperaation, jonka ehto on seuraavanlainen:

```
concat("Puuttuvia kenttiä kohdissa: ",$V1)
```

Tämä koodipala yhdistää ”V1”-muuttujassa olevat päivät edellä määriteltyyn viestiin. Ideana on tehdä tästä virheviestistä mahdollisimman selkolukuinen käyttäjälle. Tämä arvo kopioidaan muuttujaan ”pyyntöResponse”, joka jakaa nimensä ”RuokalistaVastaanotto”-portin ”output”-parametrin kanssa. Tämä parametri on tyyppiä ”string” eli yksinkertainen merkkijono.

Seuraavaksi siirrytään ”Reply”-aktiviteettiin. Tämä aktiviteetti on hyvin samantyyppinen kuin ”Receive”, joka käynnisti koko prosessin. Kuvassa 24 voidaan huomata, että aktiviteetin ”Participant”-osio on sama kuin alussa olevalla ”Receive”-aktiviteetilla.

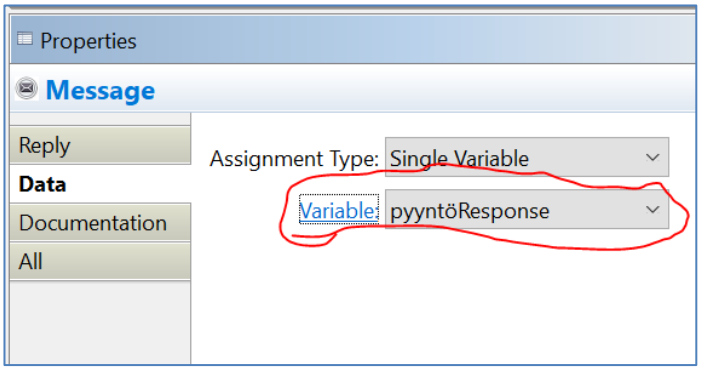


Kuva 24. "Reply"-aktiviteetin properties-näkymä.

Tämä on sen takia, että ne ovat kummatkin ruokalista.wsdl-tiedostossa määritetyn web-palvelun toteuttajia. ”Receive” vastaanotti määritellyn tiedon, ”Reply” palauttaa prosessin vastauksen. ”Reply”-aktiviteettiä voi käyttää muuhunkin kuin virheviestien lähettämiseen kuten myöhemmin prosessissa nähdään, mutta tässä tapauksessa se hoitaa virheviestin toimittajan virkaa.

Kuvasta 25 voimme nähdä, että ”RuokalistaVastaanotto”-palvelulle on määritelty parametri tyyppiä ”string”. Näin ollen ”Reply” vaatii merkkijonon

syötteekseen, jotta se voi lähettää sen eteenpäin.

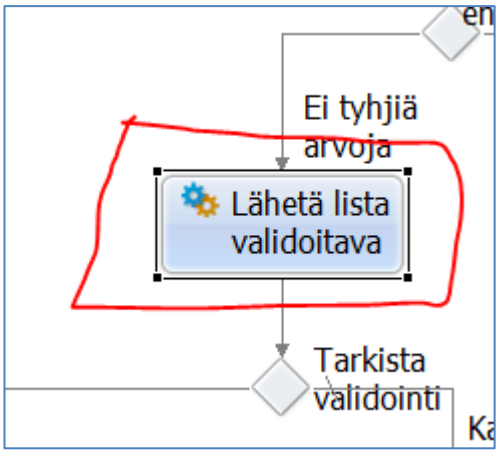


Kuva 25. "Reply"-aktiviteettiin syötetään muuttuja.

Se, miten tämä merkkijono otetaan vastaan, on ulkoisen sovelluksen hoidettavana. Tästä vaiheesta prosessi ohjautuu päätöspisteeseen, ja suoritus lopetetaan.

4.2.5 Arvojen validointi service-aktiviteetilla

Seuraavaksi palataan edelliseen if-ehtoon, josta oletamme, että muuttuja ”V1” on tyhjä. Tästä seuraa se, että jatkamme if-haaraa eteenpäin kuvassa 26 näkyvään service-aktiviteettiin.

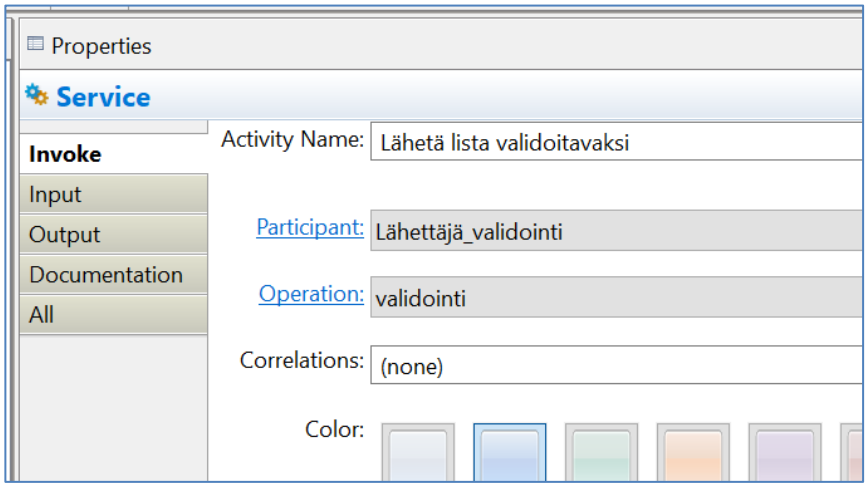


Kuva 26. Validoinnin service-aktiviteetti kaaviossa.

Service-aktiviteetti seuraa hyvin samanlaista kaavaa kuin aikaisemmin nähdyt ”receive”- ja ”reply”-aktiviteetit. Sen tarkoitus on lähettää sekä tarvittaessa vastaanottaa viestejä tai dataa. Toisin kuin aiemmat viestintäaktiviteetit, service on ennemmin tarkoitettu kutsumaan ohjelmistokokonaisuuteen kuuluvia Active-VOS:in ulkopuolisia sovelluksia ja lähettää niille dataa tai viestejä tarpeen vaatiessa.

Aktiviteetilla voi myös tehdä yksisuuntaisia operaatioita, kuten ”receive” ja ”reply” -aktiviteeteillä. Service-aktiviteetillä yksisuuntaiset operaatiot ovat pääosin tarkoitettu tilanteisiin, jossa halutaan kutsua tai käynnistää ulkoinen palvelu, eikä tältä palvelulta haluta tai odoteta vastausta.

Service konfiguroidaan samalla tavalla kuin ”receive” ja ”reply”. Kuva 27 sisältää service-aktiviteetin properties-näkymä, josta nähdään aktiviteetin vaatimat asetukset. Pieniä eroja ovat tiettyjen vaihtoehtojen puute, kuten ”create instance”, joka määritellään ”receive”-aktiviteetille, ja ”fault name”, joka on taas määritelty ”reply”-aktiviteetille.

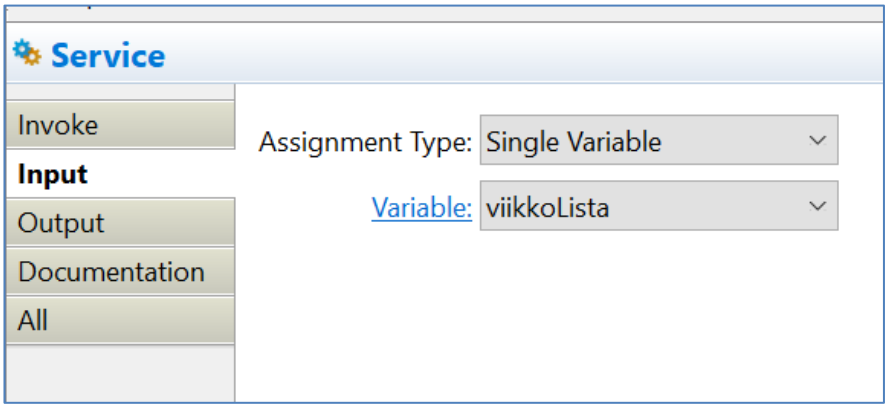


Kuva 27. Service-aktiviteetin konfiguraatioikkuna.

Isoin eroavaisuus service-aktiviteetilla on, että aikaisempien ”PSC”:iden sijaan määritellään ”PSP”. Ero näiden välillä on hieman epäselvä, mutta voidaan tiivistää seuraavanlaisesti: ”PSC”:n tarkoitus on, että ulkoinen sovellus ottaa yhteyden prosessiin, ja mahdollisesti saa vastauksen takaisin. ”PSP”:n tarkoitus on,

että prosessi ottaa yhteyden ulkopuoliseen sovellukseen ja mahdollisesti vastauksen takaisin.

Esimerkkiprosessissa tämän ”Lähetä lista validoitavaksi” -servicen tarkoitus on kutsua ulkoisen sovelluksen apua validoidakseen annettuja arvoja. Aktiviteetti lähettää dataa ohjelmalle ”viikkoLista” rakenteen muodossa. Kuvasta 28 nähdään, kuinka muuttujan syöttö tapahtuu aktiviteetille käytännössä.

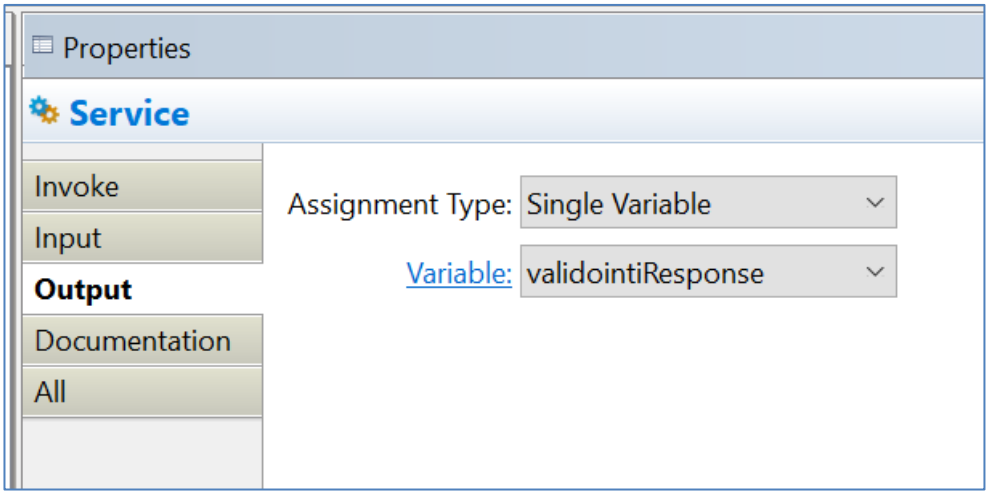


Kuva 28. Service lähettää dataa muuttujasta "viikkoLista".

Käytännössä tämä tarkoittaa sitä, että annetut ruoka-annokset ovat oikeita ruokia. Voidaan esittää, että datan vastaanottavassa päässä on sovellus, johon on listattu kaikki ruuat, mitä voidaan kuvitella, lihapullista pestokalaan.

Tämä sovellus, vastaanotettuaan listan, käy jokaisen päivän yksitellen läpi, tarkistaa, että liha- ja kasvisruuan nimet löytyvät tältä listalta. Mikäli näin ei ole, listaa ohjelma kyseisen päivän palautteena tulevaan merkkijonoon, samalla tavalla kuin aikaisemmassa script-aktiviteetissä, jossa tarkastelimme puuttuvia arvoja.

Kun ohjelma on suorittanut validoinnin, lähettää se vastauksen takaisin prosessille, jonka vastaanotamme samalla service-aktiviteetilla. Kuvasta 29 nähdään, kuinka muuttujan vastaanotto tapahtuu täysin samalla tavalla kuin sen syöttö.



Kuva 29. Ohjelman lähettämä data vastaanotetaan service-aktiviteetissä.

Vastaanotettu data talletetaan muuttujaan ”validointiResponse” myöhempää käyttöä varten. Tätä muuttujaa käytetään seuraavassa if-ehdon tarkistuksessa.

4.2.6 Prosessin seuraavat vaiheet ja lopetus

Seuraavaksi tullaan uuteen if-ehtoon, jossa tarkistetaan validoinnin tulos. Tästä pisteestä eteenpäin prosessi käyttää aikaisemmin esiteltyjä aktiviteettejä, joita ei ole tarpeen tutkia yhtä yksityiskohtaisesti, vaan käydään loppuosa prosessista hieman tiiviimmin läpi.

Ensimmäisenä prosessi suorittaa if-ehdon:

```
$validointiResponse.message != ""
```

$validointiResponse on ulkoisen ohjelman palauttama vastaus, jonka on tarkoitus sisältää merkkijonomuodossa ne viikonpäivät, joiden ruokalistassa oli validoimattomia arvoja.

Jos ehto toteutuu, eli validoimattomia arvoja on olemassa, menee prosessi aikaisemmin kuvattuun virheviestihaaraan, ja prosessi lopetetaan.

Olettaen, että kaikki arvot on validoitu, siirtyy prosessi else-haaraan, jossa lista lähetetään ulkoisen varastosovelluksen käsiteltäväksi. Tämä suoritetaan kuten validointi, service-aktiviteetin avulla. Service ottaa sisään muuttujan ”viikkoLista” ja lähettää tämän ulkoiselle ohjelmalle. Ohjelma palauttaa vastauksen samanlaisessa muodossa kuin validointi, eli merkkijonon sisältäen virheelliset kohdat.

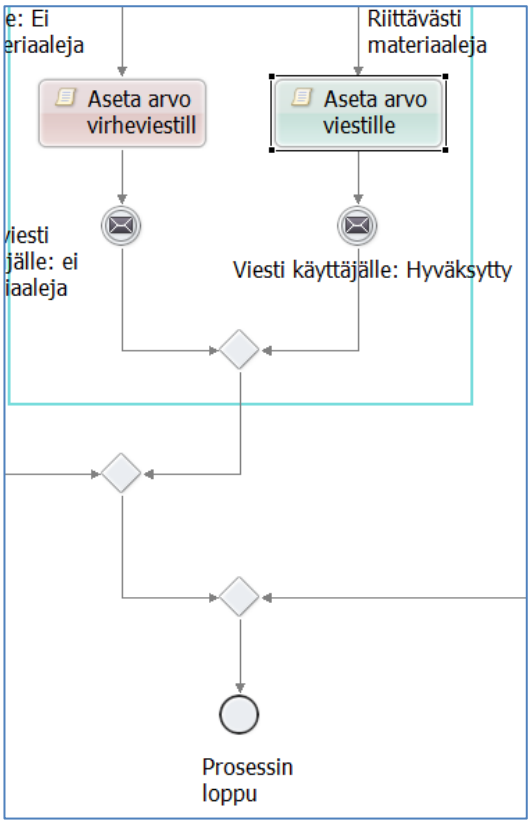
Seuraavana prosessi jatkaa viimeiseen if-ehtoon, jossa tarkistetaan, sisältääkö ohjelman vastaus arvoja.

```
$VarastoResponse.message != ""
```

Mikäli ehto täyttyy, eli virheellisiä arvoja on löytynyt, ohjataan prosessi taas samankaltaiseen virheviestihaaraan kuten aiemmin.

Olettaen, että virheellisiä arvoja ei löytynyt, siirtyy prosessi script-aktiviteettiin, jossa käyttäjälle lähetettävän viestin arvo asetetaan. Seuraavaksi viesti lähetetään käyttäjälle reply-aktiviteetilla. Tämän jälkeen prosessi lopetetaan. Kuvasta 30 nähdään prosessin rakenne, joka ohjaa jokaisen haaran samaan lopputulokseen eli lopetukseen.

Vaihtoehtoisesti prosessin voisi myös suunnitella siten, että jokainen erkaneva haara ohjautuu eri lopetuskohtaan. Toiminnan kannalta tämä ei vaikuta mihinkään, mutta tämän kaltaisen rakenteen voi ymmärtää paremmin.



Kuva 30. Viestin lähetys ja prosessin lopetus.

Prosessin loppuessa kaikkien muuttujien arvot pyyhitään muistista.

5 Yhteenveto

Työssä esiteltiin, kuinka BPEL-prosessien kehitys tapahtuu ActiveVOS Designer -ohjelmalla ja minkälaisia ominaisuuksia ja vaatimuksia kehitystyöhön liittyy.

Työtä varten oli luotu esimerkkiprosessi, joka sisältää WSDL-määritelmät kaikille sen web-palveluille. Mikäli näille web-palveluille luotaisiin vastaanottavat pisteet, eli ulkoiset ohjelmat, toimisi tämä prosessi normaalisti kuten mikä tahansa muu viralliseen käyttöön tarkoitettu prosessi.

Tällaiset prosessit ovat vain pieni osa isompaa kokonaisuuta, joka muodostaa järjestelmälle suunnitellun toiminnallisen logiikan.

Standardit, joita tässä työssä käytiin läpi, ovat vanhoja, mutta ne ovat silti nykypäivänä käytössä luotettavuuden ja suhteellisen yksinkertaisuutensa takia. Kyseisiä standardeja ei nykypäivänä enää mainosteta alan huippuna uusien kehitystapojen julkaisujen saatossa, mutta ne ovat silti nykypäivänäkin yhtä hyödyllisiä.

Lisäksi monet yritykset, joiden sovelluskokonaisuus on suunniteltu kauan aikaa sitten, eivät enää halua vaihtaa uudempiin toteutuksiin, sillä nämä vanhat hoitavat saman roolin lähes yhtä hyvin, ellei paremmin. Näin ollen olisi turhan vaivalloista sekä kallista lähteä suunnittelemaan täysin uutta toteutusta, jos vanha toimii silti yhtä hyvin.

Isoin haaste työssä oli kehitellä esimerkkiprosessi, joka demonstroisi tarpeeksi kattavasti BPEL-prosessien toiminnallisuutta ja tarkoitusta laajemmissa toteutuksissa, mutta samaan aikaan olisi tarpeeksi yksinkertainen ja helposti ymmärrettävä uudelle kehittäjälle, jolla ei välttämättä ole minkäänlaista käsitystä aiheesta.

Työ toimi hyvin mahdollisuutena kehittää omaa osaamista XML-pohjaisten standardien ja BPEL-kehityksen parissa, ohjaten etsimään tietoa vanhoilta, arkistoiduilta sivustoilta, joita ei voi enää löytää tavallisella hakukoneella.

Koska kyseessä on pohjimmiltaan kehitysohje, voi tätä työtä kehittää jatkossa pidemmälle, kuvailla yksityiskohtaisemmin siihen liittyviä standardeja, listata niihin liittyviä avainsanoja ja elementtejä sekä lisätä hyviä vinkkejä, jotka tulevat kehitystyön tilanteissa vastaan.

Lähteet

1 Informatica. Corporate fact sheet. Verkkoaineisto https://www.informatica.com/content/dam/informatica-com/en/collateral/other/informatica-corporate-fact-sheet.pdf Luettu 7.5.2023

2 Active-Endpoints. 2008. Release Notes. Verkkoaineisto https://web.archive.org/web/20090412124444/http:/www.activebpel.org/infocenter/ActiveVOS/v50/topic/com.activee.bpel.doc/ReleaseNotes.html Luettu 7.5.2023

3 Holloway, Simon. 2013. Informatica acquire Active Endpoints. Verkkoaineisto https://www.bloorresearch.com/2013/05/informatica-acquire-active-endpoints/ Luettu 7.5.2023

4 Active-Endpoints. 2009. Product About section. Verkkoaineisto https://web.archive.org/web/20090816180839/http:/www.activevos.com/products-activevos.php - :~:text=ActiveVOS%20is%20the%20first%20of%20a%20new%20generation%20of%20development%20systems%20%2D%2D%20the%20visual%20orchestration%20system Luettu 7.5.2023

5 Villanova University. 2022. What is a business process? | Definition, importance and examples. Verkkoaineisto https://www.villanovau.com/resources/bpm/what-is-a-business-process/ Luettu 7.5.2023

6 OASIS. 2003. OASIS Web Services Business Process Execution Language TC. Verkkoaineisto https://www.oasis-open.org/committees/wsbpel/charter.php Luettu 7.5.2023

7 Francisco Curbera, Yaron Goland, Johannes Klein, Frank Leymann, Dieter Roller, Satish Thatte, Sanjiva Weerawarana. 2002. Specification: Business Process Execution Language for Web Services, Version 1.0. Verkkoaineisto https://web.archive.org/web/20030528111624/http:/www-106.ibm.com/developerworks/webservices/library/ws-bpel1/ Luettu 7.5.2023

8 OASIS Web Services Business Process Execution Language (WSBPEL) TC. 2007. Web Services Business Process Execution Language Version 2.0. Verkkoaineisto http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html - _Toc164738477 Luettu 7.5.2023

9 Geyer, Carol. 2007. WS-BPEL 2.0 versus BPEL4WS 1.1. Verkkoaineisto http://bpel.xml.org/changes Luettu 7.5.2023

10 Erik Christensen, Francisco Curbera, Greg Meredith, Sanjiva Weerawwarana. 2000. Web Service Description Language (WSDL) 1.0. Verkkoaineisto http://xml.coverpages.org/wsdl20000929.html Luettu 7.5.2023

11 Roberto Chinnici, Jean-Jacques Moreau, Arthur Ryman, Sanjiva Weerawarana. 2007. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. Verkkoaineisto https://www.w3.org/TR/wsdl20/ Luettu 7.5.2023

12 Ferguson Derek. 2004. Exclusive .NET Developer's journal "Indigo" Interview with Microsoft's Don Box. Verkkoaineisto https://web.archive.org/web/20110716181223/http:/dotnet.sys-con.com/node/45908 Luettu 7.5.2023

13 W3C. 2000. Simple Object Access Protocol (SOAP) 1.1. Verkkoaineisto https://www.w3.org/TR/2000/NOTE-SOAP-20000508/ Luettu 7.5.2023

14 W3C. 2003. SOAP Version 1.2 Part 1: Messaging Framework. Verkkoaineisto https://web.archive.org/web/20030811220612/http:/www.w3.org/TR/soap12-part1/ Luettu 7.5.2023

15 IBM. 2021. SOAP. Verkkoaineisto https://www.ibm.com/docs/en/radfws/9.6.1?topic=standards-soap Luettu 7.5.2023

16 uddi.org. 2000. FAQs. Verkkoaineisto https://web.archive.org/web/20001202140900/http:/www.uddi.org/faqs.html Luettu 7.5.2023

17 IBM. 2021. Universal Description, Discovery And Integration (UDDI). Verkkoaineisto https://www.ibm.com/docs/en/radfws/9.6.1?topic=standards-universal-description-discovery-integration-uddi Luettu 7.5.2023

18 Microsoft. 2016. What's New In BizTalk Server 2016. Verkkoaineisto https://learn.microsoft.com/en-us/biztalk/install-and-config-guides/what-s-new-in-biztalk-server-2016 Luettu 7.5.2023

19 W3C. 2001. XML Schema Part 0: Primer. Verkkoaineisto https://web.archive.org/web/20010627215824/http:/www.w3.org/TR/xmlschema-0/ - Intro Luettu 7.5.2023

20 Oracle. 2012. Application Server Integration InterConnect User's Guide. Verkkoaineisto https://web.archive.org/web/20120123045044/https:/docs.oracle.com/cd/B14099_19/integrate.1012/b14069/xsd.htm Luettu 7.5.2023