Софийски университет „Св. Климент Охридски“

Факултет по Математика и Информатика

Катедра „Софтуерни технологии”

Специалност „Информатика”

Специализация „Софтуерни технологии”

Дипломна Работа

Тема: Спецификация и примерна имплементация на XQuery имплементационен тип за компонентна архитектура за услуги (SCA)

Дипломант: Васил Жанов Василев, ФN№ M-21602

Научен ръководител: Доц. Силвия Илиева
катедра „Софтуерни технологии”, ФМИ

Октомври 2007

Съдържание

Използвани термини и съкращения

активност – обмяна на съобщение между услуги

виртуализация (Virtualization) = Enterprise Information Integration

домейн модел – модел, който представя част от понятията в предметната област, за която е изградено дадено приложение

композиране – термин в Service Oriented Architecture, представящ създаването на нова услуга на базата на набор от съществуващи услуги, чрез комбиниране на заявките към тях

контрибуция – форма за пакетиране и доставка на артифакти към среда за изпълнение за Service Component Architecture

медиация - осъвместяване на интерфейсите на извикваната и извикващата услуга, както и на съобщенията, които те си разменят

онтология – модел на данните, представляващ набор от концепции на даден домейн и техните взаимовръзки

свързаност (binding) – задава начина по-който се прави достъп до дадена услуга в SCA

скалируемост – способността на дадена система да обработва нарастващ брой заявки към нея

уеб услуга – услуга достъпна посредством обмяна на съобщения по HTTP протокол

услуга – единица обработваща логика в SOA

федерация – Интеграция на данни, чрез създаване на общ изглед върху множество от източници на данни

BPEL (Business Process Execution Language) – език за описание на бизнес процеси

BPMN (Business Process Model Notation) – графична нотация за описание на бизнес процеси

DOM (Document Object Model) – Дървовиден модел за описание на XML съдържание в паметта

EAI (Enterprise Application Integration) – интеграция на приложения и бизнес процеси

EII (Enterprise Information Integration) – интеграция на данни от различни източници

EJB (Enterprise Java Beans) – компонентен модел в JEE платформата, чиито градивни единици са бийновете

EMF (Eclipse Modeling Framework) – представлява основа за дефиниране на модели и мета-модели, както и за програмен достъп до тях

ESB (Enterprise Service Bus) – архитектура за SOA, дефинираща начин за обмяна на съобщения през едно централно място – шина

ETL (Extract Transform Load) – Интеграция на данни, чрез извличането им в общ склад за данни

JavaBeans – спецификация за структурата на даден Java клас, така че да се работи с него по стандартизиран начин

JBI (Java Business Integration) – спецификация, описваща

интеграцията на различни модули в едно SOA приложение

JCA (JEE Connector Architecture) – архитектура за връзка между приложенията в JEE

JEE (Java Enterprise Edition) – платформа за реализация на бизнес приложения на Java

JMS (Java Message Service) – спецификация за среда за обмяна на съобщения в JEE

JMX (Java Management eXtensions) – спецификация за мониторинг и контрол на Java приложения

MEP (Message Exchange Pattern) – шаблон за обмяна на съобщения между услуги

NMR (Normalized Mesage Router) – медиатор на съобщения в JBI спецификацията

RMI (Remote Method Invokation) – протокол за отдалечено извикване на разпределени Java обекти

SCA (Service Component Architecture) – компонентна архитектура за услуги

SCA асембли модел – компонентен модел за дефиниране на връзки между услугите в SCA

SDO (Service Data Objects) – йерархичен модел за описание на данните

SOA (Service Oriented Architecture) – архитектурата ориентирана към услуги

SOAP (Simple Object Access Protocol) – HTTP-базиран протокол за обмяна на съобщения между уеб услуги

StAX (Streaming API for XML) – програмен интерфейс за

достъп до XML, позволявщ на приложението да извлича информация чрез интерфейс подобен на итератор

STP (SOA Tools Platform) – проект с отворен код, целящ разработката на среда за проектиране на SOA приложения

Tuscany – проект с отворен код, имплементиращ SCA спецификацията

WSDL (Web Service Definition Language) – език за дефиниране на интерфейси на уеб услуги

WS-* – колекция от спецификации, засягащи качествени характеристики на услугите в рамките на SOA. Примери за такива са сигурност, гарантирано изпращане на съобщения и др.

XML (eXtensible Markup Language) – език за описание на данни и документи

XPath – спецификация, дефинираща формата на изрази за адресиране на XML съдържание

XQJ (XQuery API for Java) – програмен интерфейс на Java за изпълнение на XQuery заявки

XQuery – процедурен език за описание на XML трансформации

XSD (XML Schema Definition) – език за описание на схема на данни, които се представят чрез XML

XSLT (Extensible Stylesheet Language Transformations) – декларативен език за описание на XML трансформации

Списък на фигурите

1 Въведение

1.1 Въведение

В своята история, софтуера за разработка на бизнес приложения, ориентирани към предприятията е претърпял значително развитие по отношение на неговата архитектура. В началото се е използвала архитектура, при която един компютър обработва заявки от множество терминали, като за всеки от тези терминали се отделя част от процесорното време. С нарастването на броя на потребителите на такава система, обаче, се е установило, че тази архитектура не е удачна, т.е. не е скалируема (трудно осигурява поддръжка на нарастващия брой потребители). За да се реши този проблем се преминава към т.нар. клиент-сървър архитектура, при която отговорността за взаимодействието с потребителя се делегира към отделен компютър, с който той работи. Има централен сървър, който обработва потребителските заявки и връща отговор. Приложенията, разработени с използването на клиент-сървър архитектурата имат съществения недостатък - те се разширяват трудно, т.е. много трудно е да се накара едно приложение да комуникира с друго приложение, което съществува и по този начин да се използва вече написана логика. Освен това, при клиент-сървър архитектурата, всяко приложение отговаря само за достъпа до източниците на информация, т.е. няма единен модел на данните.

За да се решат описаните проблеми се преминава към разработката на приложения, базирани на разпределени архитектури. Примери за такива са CORBA, EJB, .NET и др. Основното предимство на тези архитектури е, че при тях

обектите живеят самостоятелно и техните методи могат да се извикват отдалечено. Това позволява използването на вече съществуваща логика, написана от едно приложение в друго, а също така осигурява един допълнителен слой, който отделя източниците на информация (като бази данни и др.) от логиката на приложението.

Въпреки значителните предимства на разпределените архитектури спрямо останалите, въпреки това че те дават големи възможности за разработка на скалируеми приложения, които да се използват в рамките на дадено предприятие, през последните години все повече нараства необходимостта от това различни предприятия да комуникират по между си по електронен път. Начина за тази комуникация трябва лесно да се адаптира към променящите се условия на бизнеса. Нека си представим, че се налага да се осигури комуникацията на приложение написано с използването на EJB с такова написано на .NET. Освен това тази комуникация трябва да се извърши по защитен начин, който да гарантира сигурността на комуникиращите си страни. Очевидно, това изискване не може да се осигури от съществуващите разпределени системи. Необходима е нова архитектура, която да е базирана на отворени стандарти и да е широко приета от големите софтуерни компании по света. Тя трябва да дава възможност за безпрепятствената комуникация между модули написани на различни езици, като връзката между тези модули трябва да се осъществява по възможно най-необвързващ начин (loose coupling). Тази нова архитектура трябва да осигурява лесното композиране на отделните

модули по начин, който е най-подходящ за съществуващите в момента бизнес процеси, както и да позволява лесната адаптация към промените в бизнеса. Друг важен елемент от архитектурата е това, че тя трябва да позволява лесната откриваемост на съществуващи модули, по същия начин както Интернет потребителя използва търсачка, за да открие необходимата му информация.

За да се адресират всички тези изисквания на съвременния бизнес, преди няколко години се появи *архитектурата ориентирана към услуги* (Service Oriented Architecture – SOA), като в момента тя все повече придобива облик и се развива и стандартизира по такъв начин, че да е от полза за всички предприятия, независимо на какъв програмен език са стъпили техните приложения и с какъв доставчик на технология са обвързани (Microsoft, SUN, BEA, ...). Голяма стъпка в това развитие представлява създаването на спецификация за компонентен модел на SOA, която дефинира понятия като услуги, компоненти, връзки между компонентите, изисквания към сигурността и др. Тази спецификация се нарича *компонентна архитектура за услуги* (Service Component Architecture – SCA), като през месец март тази година беше създаден финалния вариант на първата и версия [1].

Чрез SCA се дефинира SOA в хоризонтален план, като тя е достатъчно разширяема, за да предостави интегрирането на различни вертикални технологии. Тук понятията хоризонтален и вертикален са избрани условно, за да се разграничи модела на дадено SOA приложение,

който се дефинира посредством SCA от конкретната реализация на отделните слоеве в SOA. В SCA хоризонталния модел е специфициран посредством т.нар. *SCA асембли модел* (SCA assembly model), докато разширяемостта се постига, чрез допълнителни спецификации – по една за всяка от технологиите, които трябва да се впишат в модела. Примери за съществуващи допълнителни спецификации са тези за Java, C и BPEL имплементация на услуги, тези за връзки между услуги посредством уеб услуги, JMS, JCA и др.

1.2 Цел на дипломната работа

Ако се разгледат съществуващите вертикални спецификации, може да се забележи, че липсва един важен елемент, който способства за реализацията на два от основните принципи на SOA – възможността за използване на вече съществуваща логика в контекста на друго приложение (reusability) и възможността за интегриране на стари системи (не SOA-базирани системи или още – legacy systems) в контекста на SOA приложението (Encapsulation) [2]. За да реализира принципа reusability, едно SOA приложение се нуждае от технология, позволяваща му да *адаптира* контракта на използваната услуга, към контракта, който очаква използващата услуга. От друга страна, за да се реализира принципа encapsulation, е необходима технология, която да позволява *интеграцията на данни* от различни източници и представянето на тези данни под формата на един общ модел, който да се използва от приложението (т.нар. домейн модел).

Решение за запълване на тази липса в SCA спецификацията е представено в настоящата дипломна работа, като за базова технология е избран езика XQuery, който представлява мощно XML-базирано средство за адаптиране на интерфейси и интеграция на данни.

1.3 Полза от реализацията

Решението за интеграция на данни и адаптация на интерфейси, представено в дипломната работа далеч не представлява завършена реализация. То представя прототип, чрез който да се даде начало на една дискусия и развитие в областта на приложимостта на средствата за интеграция в рамките на SCA. Целта е да се предизвика интереса на SOA общността и да се доразвие идеята в завършена спецификация, която да е приложима продуктивно. Именно за това разработената реализация ще се предостави на проекта с отворен код Tuscany, където тя ще може да продължи своето развитие.

1.4 Структура на дипломната работа

Глава 2 представлява уводната част на дипломната работа. В нея се прави разширен обзор на използваните технологии, както и се аргументира избора на съответна технология.

Същинската част на дипломната работа се състои от пет глави.

В глава 3 се направи анализ на това как световните фирми производители на бизнес софтуер правят интеграция на данни, обосновава се важността на това да се предостави средство за интеграция и адаптиране за SCA и се

аргументира избора на XQuery, като език за реализация на това средство.

В глава 4 се представя спецификация за интеграция на XQuery в SCA. Тази спецификация описва разширения в дефиницията на XQuery скрипт, подпомагащи използването му като SCA компонент.

В глава 5 се представя разработената примерна имплементация на тази спецификация. Тази имплементация е създадена за проекта с отворен код Tuscany.

В глава 6 се разглежда примерно приложение, аргументиращо приложимостта на имплементацията за реализацията на крайната цел на дипломната работа.

В глава 7, която е последна за дипломната работа, се прави предложение за потребителски интерфейс за разработка на приложения съгласно новата спецификация.

2 Глава 2. Увод в предметната област

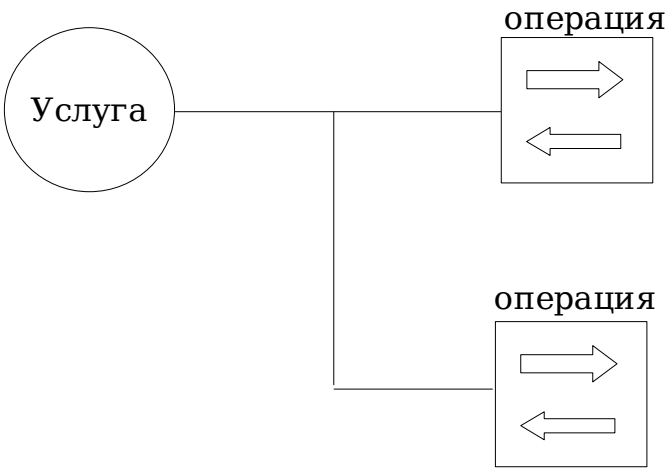
В тази глава ще бъдат разгледани основните технологии, свързани с предмета на дипломната работа. Ще се започне с преглед на това какво е SOA, какви са нейните основни концепции, характеристики и принципи. След това ще се направи обзор на двете съществуващи спецификации за SOA – SCA и JBI, както и на имплементациите, които ги поддържат. В изложението основно ще се наблегне на SCA и на двете имплементации с отворен код – Tuscany и STP, на които е базирана разработката.

Ще се отдели внимание и на технологията за достъп до данни – SDO, която е много популярна в момента и е използвана в дипломната работа за доставка и получаване на данни в примерното приложение за използване на XQuery имплементационния тип.

Изложението в тази въвеждаща глава ще приключи с преглед на самия език XQuery и на един от най-добрите интерпретатори за него – Saxon, който е използван от дипломанта.

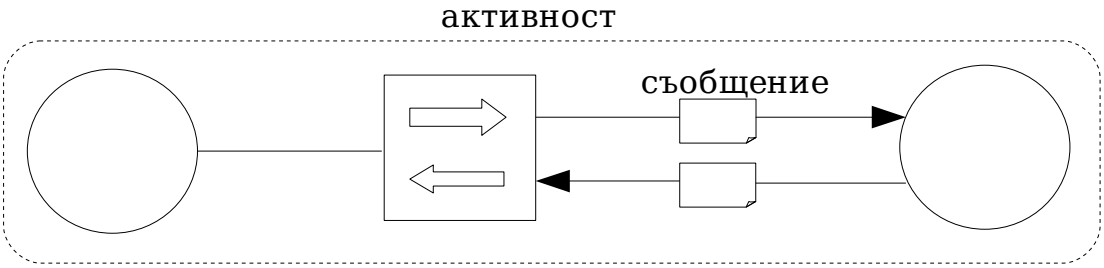
2.1 Архитектура ориентирана към услуги (SOA)

Архитектурата ориентирана към услуги [3] е създадена в отговор на променящите се изисквания към съвременния бизнес софтуер. В основата на тази архитектура са услугите. Услугата представлява единица обработваща логика (unit of processing logic), която притежава набор от операции, които от своя страна са единици за извършвана работа (unit of work) – виж фигура 1.



Фигура 1: Услуга – общ изглед

Всяка от операциите може да обменя съобщения с друга услуга, като тази обмяна се нарича активност (activity) или процес (process). В SOA, съобщението представлява единица за комуникация (unit of communication), а активността – единица за автоматизационна логика (unit of automation). На фигура 2 е показано как се извършва комуникацията между две услуги.



Фигура 2: Активност между две услуги

Всяка услуга има свое описание (contract), което определя какви операции се предлагат от нея и какви съобщения се обменят с нея.

Освен дефиницията на това какво е услуга, SOA определя и набор от основни принципи, на които се подчинява едно приложение изградено от услуги. Тези

принципи са:

- развързаност (loose coupling) – връзките между услугите трябва да се осъществяват така, че те да не зависят една от друга;
- услугите имат описание (Service contract) – което определя как се извършва комуникацията с тях;
- автономност (autonomy) – услугите имат пълен контрол върху логиката, която представят;
- абстракция (abstraction) – услугите показват на външен клиент само това, което е дефинирано в интерфейса им. Останалата част от логиката им е скрита;
- възможност за повторно използване (reusability) – услугите могат да се използват в различен контекст от този, за който са били създадени;
- композируемост (composability) – набор от услуги могат да бъдат координирани и асемблирани, така че да изпълняват една обща услуга, наречена композитна услуга (composite service);
- услугите не пазят състояние сами по себе си (statelessness) – минимизира се запазването на информация, свързана с определена активност;
- откриваемост (discoverability) – благодарение на техните описания, услугите са откриваеми на базата на определени критерии за търсене.

При обмяна на съобщения между услугите се дефинират т.нар. шаблони на размяна на съобщения

(message exchange patterns – MEP). Тези шаблони се разделят условно на примитивни и комплексни. Всеки от тези видове MEP ще бъдат разгледани по-подробно.

Примитивните MEP се два вида:

- заявка-отговор (request-response) – както показва името – това означава, че ако дадена услуга изпрати съобщение до друга услуга, втората трябва да отговори със съответно съобщение;
- еднопосочно изпращане (fire-and-forget) – при този MEP дадена услуга изпраща съобщение а друга го получава без да му отговаря. В зависимост от получателите на съобщението се различават следните разновидности на шаблона:
  - един адресат (single-destination) – има само една услуга, която получава съобщението;
  - набор от адресати (multi-cast) – няколко, предварително известни услуги получават съобщението;
  - множество от адресати (broadcast) – съобщението се изпраща и всяка услуга може да го получи ако се интересува от него.

Комплексните MEP са съставени от набор от примитивни MEP. Пример за комплексен шаблон е публикуване-абониране (publish-subscribe), при който се извършва абониране с помощта на шаблон заявка-отговор, а след това абонатите се уведомяват за дадено събитие посредством еднопосочно изпращане (multicast).

Услугите могат да участват в различни видове активности. Въпреки това съществува набор от основни шаблони за такива активности, които са в съответствие с принципите на SOA. Това са:

- координация (coordination) представлява композиция от услуги, при която една от услугите играе ролята на координатор при обмяна на съобщенията. Целта е този координатор да пази състоянието на активността, която се провежда в момента и по този начин да даде възможност на останалите услуги да не пазят състояние (stateless);
- атомарна транзакция (atomic transaction) – базира се на координацията и осигурява транзакционност на операциите извършвани от услугите, участващи в шаблона, съблюдавайки ACID принципите;
- бизнес активност (business activity) – подобен на атомарната транзакция, но при него се разчита на компенсация при неуспех. Т.е. операциите изпълнявани от услугите се извършват, а при неуспех се извикват други операции, които да компенсират действието на първите;
- оркестрация (orchestration) – представлява композиция от услуги, при която едната услуга изпълнява някакъв бизнес процес и в зависимост от стъпките в този процес извиква съответна услуга от композицията;
- хореография (choreography) – използва се за осъществяване на взаимодействие между процесите и

приложенията на бизнес партньори.

В допълнение към основните принципи на SOA се прилагат и разширения, дефинирани от WS-* спецификациите. По долу са разгледани някои от тях.

- адресиране (дефинирано от WS-Addressing спецификацията) – позволява точно определяне на адресата на дадено съобщение, като в допълнение могат да се задават специфични свойства, характерни за активността;
- сигурно предаване на съобщения (reliable messaging, дефинирано от WS-ReliableMessaging спецификацията) – чрез него се дефинират нива на сигурност (delivery assurance) за предаване на съобщенията. Такива нива са:
  - Най-много веднъж (At Most Once) – то гарантира, че изпратеното съобщение ще пристигне при получателя веднъж или въобще няма да пристигне;
  - Най-малко веднъж (At Least Once) – то гарантира, че изпратеното съобщение ще пристигне при получателя поне веднъж (може и няколко пъти);
  - Точно веднъж (Exactly Once) – то гарантира, че изпратеното съобщение ще пристигне при получателя веднъж и само веднъж;
- корелация (correlation) – задава връзка между две съобщения (например изпратено и пристигащо). Използва се в шаблоните координация и оркестрация;

- политика (policy, дефинирано от WS-PolicyFramework спецификацията) – чрез него се дефинират изисквания, които дадена услуга има към тези, които я извикват. Такива изисквания могат да представляват някакви ограничения по отношение на сигурността, или че дадена операция изисква изпълнение в транзакция и др. Политиките представляват колекция от алтернативи (policy alternative), като е достатъчно една от алтернативите да е удовлетворена, за да е удовлетворена цялата политика. Всяка алтернатива представлява колекция от твърдения (assertions), като е необходимо всички твърдения да са удовлетворени, за да е удовлетворена алтернативата;

- обмяна на мета-данни (metadata exchange, дефинирано от WS-MetadataExchange спецификацията) – специфицира как да се опише допълнителна информация за дадена услуга, така че тя да стане по-лесна за откриване и разбиране от този, който я ползва;

- сигурност (security, дефинирано от WS-Security спецификацията) – чрез него дадена услуга може да дефинира изисквания по отношение на сигурността в следните аспекти: идентификация (identification – определя необходимостта извикващия услугата да се идентифицира), аутентикация (authentication – определя кой е извикващия в съответствие с неговия идентификационен номер), оторизация (authorization – дефинира какви операции е разрешено извикващия да

изпълнява), конфиденциалност (confidentiality – определя, че е необходимо съобщенията между извикващия и услугата да се предават в криптиран вид), непроменост (integrity – изисква гаранция за това, че съобщението не е променяно през неговия път от извикващия към услугата – посредством цифров подпис, например);

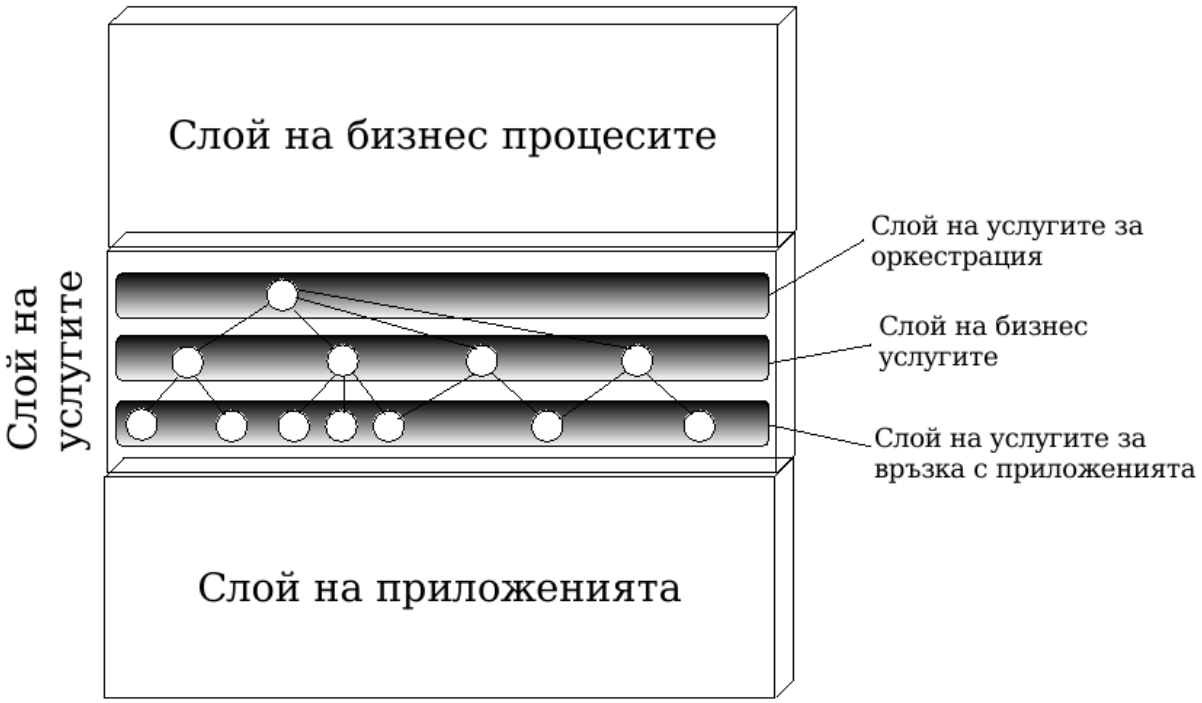
- събитийност (notification and eventing, дефинирано от WS-NotificationFramework и WS-Eventing спецификациите) – определя механизъм за разпространение на информация за настъпили събития в рамките на дадено SOA приложение.

След като вече е изяснено какво е SOA, какви са нейните изграждащи елементи, какви са нейните основни принципи и какви разширения съществуват към тях, ще бъде разгледана архитектурата и основните елементи на примерно SOA-базирано приложение. На фигура 3 са оказани слоевете, изграждащи SOA приложението. Вижда се, че слоя на услугите е междинен и разделя приложната логика от бизнес-логиката. По този начин се постига гъвкавост, позволяваща на бизнес слоя и приложния слой да се развиват самостоятелно в зависимост от променящите се изисквания съответно на бизнеса и на технологията.

Самият слой на услугите е разделен на три под-слоя:

- слой на услугите за връзка с приложенията – той съдържа услуги изпълняващи интеграция на данни и процеси от различни източници и приложения, като представя един общ изглед за останалата част от SOA

приложението;



Фигура 3: Слоеве на SOA приложение

- слой на бизнес услугите – в него съществуват услуги, предоставящи функционалност разработена специално според принципите на SOA и за нуждите на приложението. Съществуват две разновидности на бизнес услуги:
  - насочени към домейн модела (entity-centric) – тяхна цел е да изпълняват ролята на елементите от домейна на приложението;
  - насочени към изпълнението на определени задачи (task-centric) – в тях е съсредоточена бизнес логиката на приложението;
- слой на услугите за оркестрация – съдържа услуги, изпълняващи конкретни бизнес процеси. Тези услуги реализират шаблона оркестрация.

Съществуват различни комбинации от подслоевете на слоя на услугите, тъй като не винаги и трите подслоя са нужни. Например може да има само един подслой изграден от хибридни услуги (т.е. услуги, които комбинират достъпа до съществуващите приложения с бизнес логиката).

2.2 Спецификации за SOA и техните реализации

В момента съществуват две спецификации, целящи стандартизиране на приложенията, базирани на архитектурата ориентирана към услуги: едната е компонентната архитектура за услуги (SCA), която е изготвена с общите усилия на големите софтуерни компании като IBM, IONA, SAP, Oracle, BEA и др., но без участието на SUN. Другата спецификация се нарича Java бизнес интеграция (Java Business Integration – JBI) [4]. В нейното изготвяне също участват много от големите софтуерни компании, но водеща роля заема SUN. Дори самата спецификация се разработва по стандартния за SUN Java community process.

До скоро се смяташе, че SCA и JBI са две конкуриращи се спецификации, изразяващи виждането на техните основни поддръжници – съответно IBM и SUN. В последствие, обаче се оказва, че те взаимно се допълват, а не се изключват. Например среда за изпълнение на SCA, може да се реализира като JBI контейнер. Също така голяма част от производителите на платформи обещават да поддържат както едната, така и другата спецификация.

Повече подробности по тази тема ще бъдат

предоставени в следващите точки.

2.2.1 Спецификация на компонентна архитектура за услуги (SCA)

В началото разработката на SCA спецификацията започва като общо усилие на множество фирми да създадат документ, който да стандартизира SOA и да е общоприет от всички. Като резултат от тази разработка се изготвя първа версия на тази спецификация (SCA 1.0), която е поверена на организацията за развитие на отворени стандарти за информационното общество (Organization for Advancing Open Standards for Information Society – OASIS), за да я доразвива. В момента тя се развива в проект с кодово название OSOA (Open SOA) [5].

Понастоящем SCA включва следните спецификации:

- SCA Assembly Model – дефинира понятията от света на SOA, които са необходими за изграждане на SOA-базиран домейн-модел. Тази спецификация предоставя възможност за разширяване с допълнителни документи, специфициращи различните технологии, които могат да се използват при разработката на едно SOA приложение. Следващите подточки описват именно такива разширения;
- SCA Policy Framework – дефинира начина, по който се описват правила и изисквания, които трябва да бъдат изпълнени от клиента на дадена услуга, за да може той да я използва. Тук се включват дефиниции на ограничения свързани със сигурността, изисквания по отношение на начина на комуникация с услугата (дали

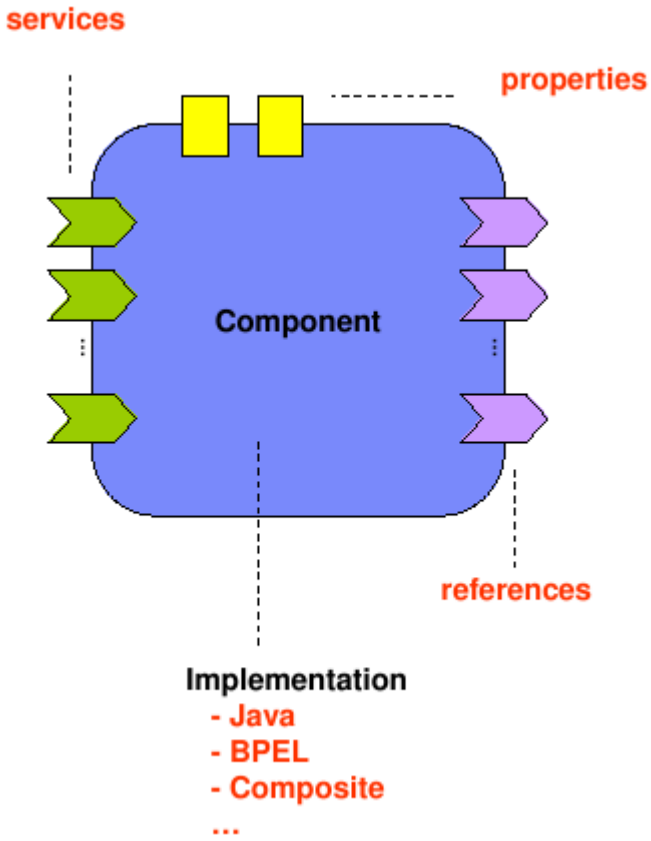
е транзакционен) и др.;

- SCA Java Common Annotations and APIs и SCA Java Component Implementation – специфицират Java имплементационен тип за SCA;
- SCA C++ Client and Implementation – специфицира C++ имплементационен тип за SCA;
- SCA Spring Component Implementation – дефинира как компоненти писани на Spring, могат да се интегрират със SCA;
- SCA BPEL Client and Implementation – специфицира използването на компоненти написани на BPEL в SCA. Тази спецификация е много важен компонент от SCA, тъй като чрез нея се дава възможност за интеграция на бизнес процеси, което е от голямо значение за потребителите на SOA приложения;
- SCA Web Services Binding – осигурява възможността за връзката на SCA компоненти с уеб услуги (web services) и обратното – експонирането на SCA компоненти като уеб услуги;
- SCA JMS Binding – осигурява връзката на SCA компоненти с инфраструктура за обмяна на съобщения, базирана на JMS стандарта;
- SCA EJB Session Bean Binding - осигурява възможността за връзката на SCA компоненти със сесийни бийнове и обратното – експонирането на SCA компоненти като сесийни бийнове.

Следва по-подробно разглеждане на асембли модела,

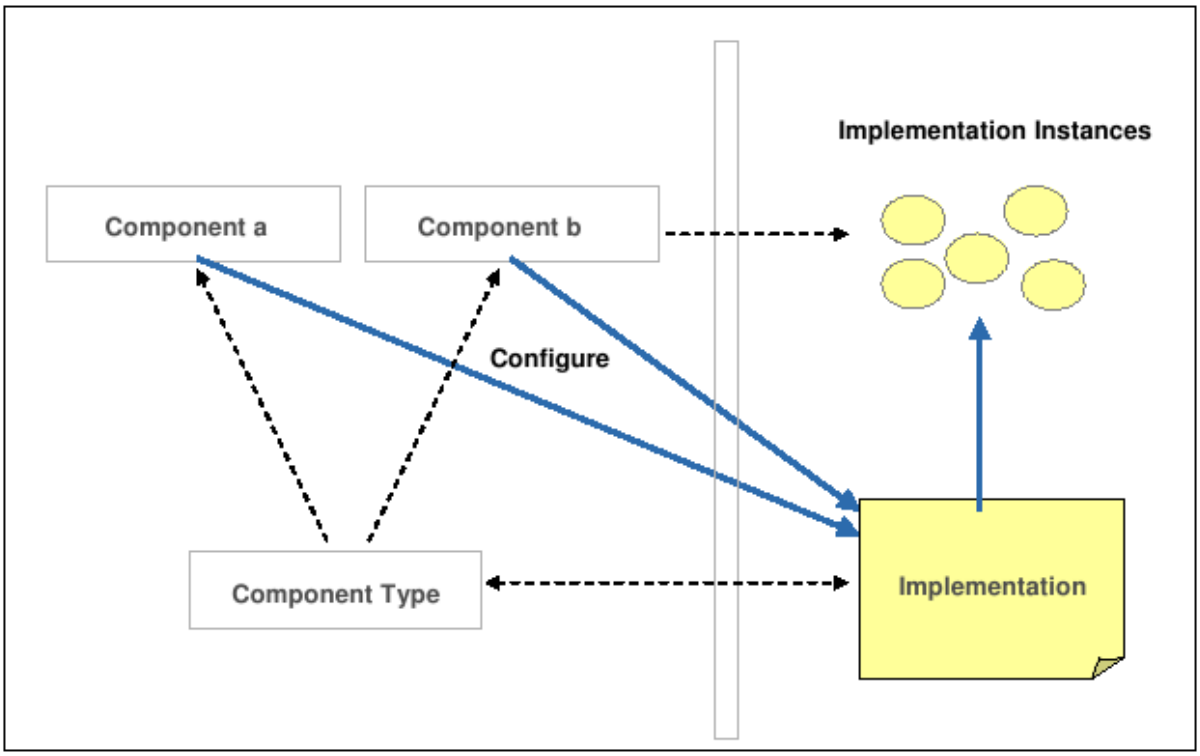
който е в основата на SCA.

Основна градивна единица на SCA модела е компонента (Component) – виж фигура 4. Той представлява черна кутия, която предоставя възможност за връзка с външния свят посредством неговите услуги (services), референции (references) и входните точки, позволяващи неговото конфигуриране и параметризиране (properties).



Фигура 4: SCA Компонент

Всеки компонент има свой тип, наречен компонентен тип (Component Type), (виж фигура 5). Този тип дефинира какви услуги, референции и входни точки за конфигуриране предлага даден компонент.



Фигура 5: Компонентен тип и имплементация

От фигурата лесно може да се види, че компонентния тип представлява дефиницията на даден компонент, тъй както класа от обектно ориентираното програмиране представлява дефиницията на даден обект. За разлика от типа, компонента представлява конкретна инстанция на даден компонентен тип с конфигурирани референции и входни точки, по същия начин както обекта представлява конкретна инстанция на даден клас.

Всеки компонент има и своя инстанция на имплементация (implementation instance), като конфигурируемите аспекти на имплементацията (implementation) се дефинират от компонентния му тип. Например, ако имплементацията е написана на Java, то всички услуги предлагани от компонента трябва да са описани като имплементирани интерфейси на Java класа;

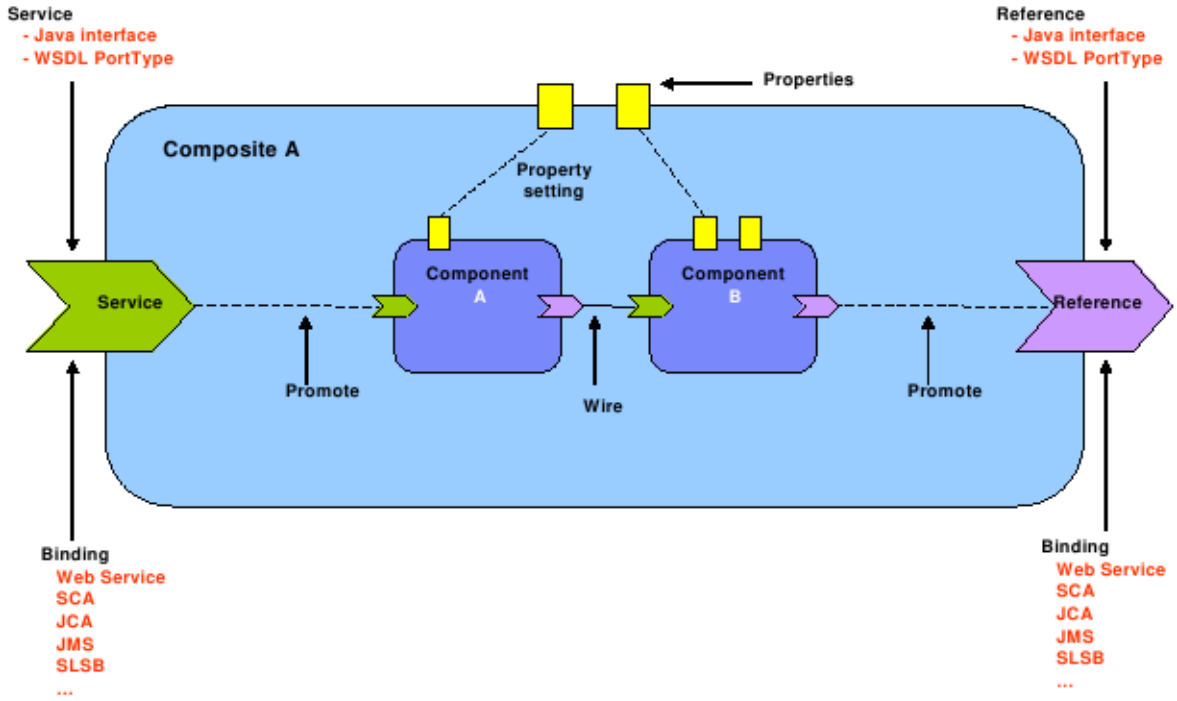
всички референции и входни точки трябва да са описани като полета в класа със съответстващи им методи за установяване (по JavaBeans схемата).

Компонентният тип може да бъде зададен по два начина: първият е експлицитно – в специален файл в xml формат; вторият е чрез интроспекция на имплементацията (както в примера по-горе за Java, на базата на имплементирани интерфейси и дефинирани полета).

Могат да съществуват различни видове имплементации за SCA компоненти. Примери за такива са Java, Spring, BPEL. За всяка от тези имплементации съществува отделна спецификация, като броя на различните видове имплементации не е ограничен, т.е. има пълна разширяемост по отношение на имплементационния тип. Наред с изброените имплементации за компоненти съществува един специален имплементационен тип, наречен композит (composite). Всеки композит може да съдържа един или повече компоненти, заедно с връзките между тях (wires) (виж фигура 6). По този начин се осигурява йерархичност на SCA модела, т.е. всеки композит може да съдържа компоненти, но всеки от тези компоненти може да има като свой имплементационен тип композит, който от своя страна съдържа под-компоненти и т.н.

Освен връзки между компонентите в даден композит могат да се дефинират и връзки между услугите, референциите и входните точки на компонента и услугите, референциите и входните токи на композита. Тези връзки се наричат промоциране (promote) съответно на услуга,

референция или входна точка.



Фигура 6: SCA Композит

При конфигурирането на даден компонентен тип, т.е. при превръщането му в компонент в рамките на даден композит, за всяка услуга и за всяка референция могат да се зададат т.нар. свързаности (binding). Тези свързаности определят по какъв начин може да се осъществи връзката (wire) с компонента. Има един тип свързаност, който е по подразбиране ако не е зададено друго и той се нарича SCA свързаност (SCA binding). Връзките между компонентите в рамките на един композит са винаги от тип SCA свързаност. Само промоциращите връзки могат да имат друг тип свързаност.

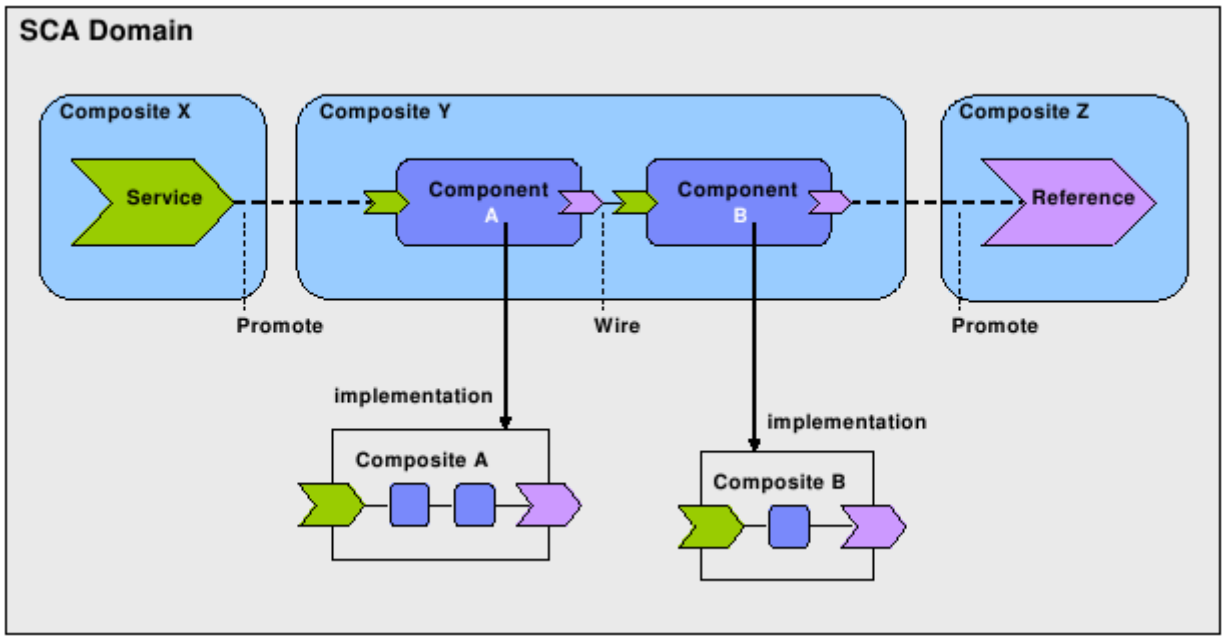
Характерното за SCA свързаността е, че тя позволява връзки между компоненти на локално ниво, т.е. в рамките на една Java виртуална машина, като всички параметри, които не са от прост тип се предават по референция. За да може

да се осъществи връзка, обаче с компонент от друга виртуална машина, или пък SCA компонент да се свърже с уеб услуга, или пък с EJB, или пък с даден код чрез RMI е необходима съответна свързаност, която да позволява това. Примери за такива свързаности са: Web Service, JMS, RMI, EJB и др. като, подобно на имплементационните типове съществува пълна разширяемост по отношение на броя им.

Всяка услуга и всяка референция предлагана от даден компонент се характеризира със своя интерфейс, като връзка между референцията на даден компонент и услугата на друг компонент може да се осъществи само ако двете имат съвместими интерфейси. В SCA спецификацията са описани принципите за съвместимост и те няма да бъдат разглеждани в настоящото изложение. В момента съществуват два начина за описание на интерфейси (два типа интерфейси): Java и WSDL, но подобно на свързаностите и имплементационните типове, това множество е разширяемо.

В SCA всеки компонент може да съществува само в рамките на композит, т.е. не може да има самостоятелен компонент. Това правило се обяснява от факта, че за да се опише той е необходимо да се настроят конфигурируемите аспекти на неговия компонентен тип, а такова описание може да се извърши само в рамките на композит. За това съществува т.нар. композит на ниво домейн (SCA Domain), който е контейнер за всички останали компоненти и композити на дадено приложение. Този домейн композит е най-малката единица (unit of deployment), която може да се

инсталира на дадена среда за изпълнение за SCA, по същия начин както архива за бизнес приложения (enterprise archive) е най-малката единица, която може да се инсталира на среда за изпълнение за JEE (виж фигура 7).



Фигура 7: SCA Домейн

В момента няма много фирми производители на софтуер, които да предлагат средства за разработка и изпълнение на SCA-базирани приложения поради факта, че самата спецификация е все още доста млада. Въпреки това в плановете на повечето големи компании е залегнала поддръжката на SCA в бъдеще. IBM са едни от пионерите в тази област, като те имат пълна собствена имплементация на SCA, залегнала в проекта IBM WebSphere Process Server [6]. Освен това има два проекта с отворен код, спонсорирани основно от IBM и IONA, които имат за цел да създадат среда за изпълнение на SCA приложения, и среда за разработката им. Тези проекти са съответно Apache Tuscany, предлагащ SCA средара за изпълнение [7] и Eclipse SOA Tools Platform

(платформа от средства за SOA), предлагащ средства за разработка на SOA приложения [8]. Както подсказва и самото име на втория проект, той не е ориентиран само към SCA. Чрез него се осигуряват средства за разработка както на SCA приложения, така и на JBI приложения, а също и на обикновени уеб услуги и др.

Съществува и трети проект с отворен код, който има за цел разработката на среда за изпълнение за SCA. Този проект се нарича Fabric3 [9] и се спонсорира основно от BEA. Тъй като той, обаче е в много ранен стадий от своята разработка, не представлява интерес в дипломната работа.

В следващите точки ще бъдат разгледани двата основни проекта свързани с SCA – Apache Tuscany и Eclipse STP.

2.2.2 Apache Tuscany

Основната цел на проекта Apache Tuscany е да предостави среда за изпълнение, отговаряща на спецификациите на OSOA (Open SOA), т.е. на SCA и SDO спецификациите.

SDO (Service Data Objects) дефинира йерархичен начин за представяне на данните от различни източници, като се дава възможност за тяхното четене, модифициране и запис. Повече информация за тази технология ще бъде предоставена в специално посветената на темата точка 2.3 от дипломната работа. По-долу е разгледан в подробности проекта за създаване на среда за изпълнение за SCA.

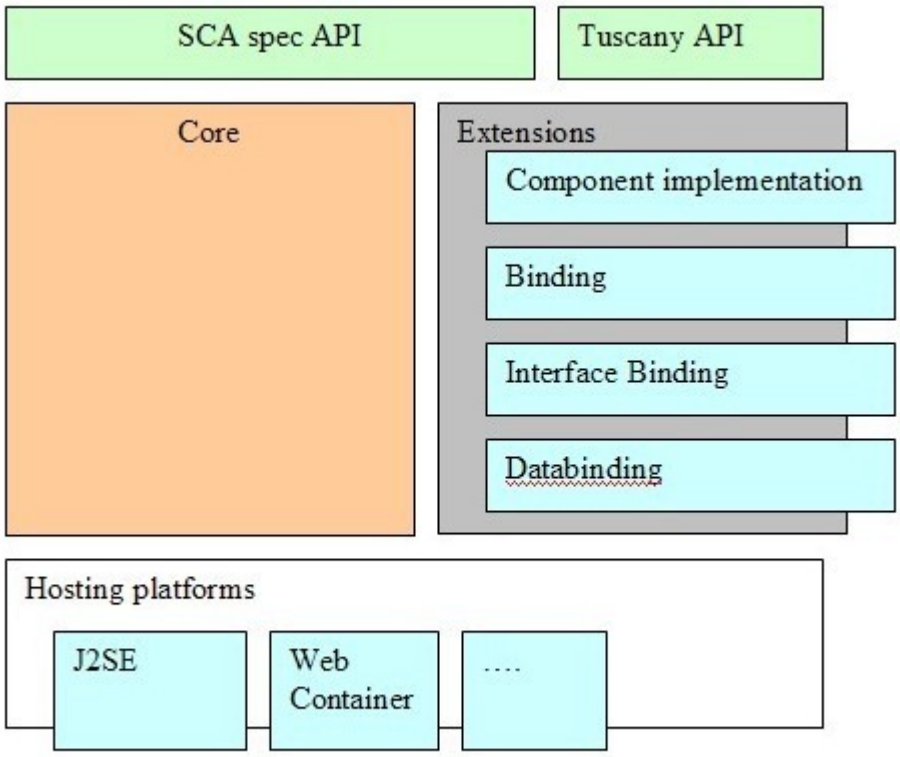
В Apache Tuscany съществуват три проекта за разработка на среда за изпълнение за SCA като всеки един от тях представлява разработка на платформа за поддръжка

на SCA на определен програмен език. Тези проекти са предназначени за Java, C++ и PHP, като най развития от тях е този за Java. Поради това той беше избран като основа за разработката в настоящата дипломна работа. По-долу се разглежда архитектурата и основните принципи на работата му [10].

На фигура 8 е представен блок-схема на основните компоненти, изграждащи Tuscany. Това са:

- SCA spec API – включва програмен интерфейс за четене и обработка на SCA модели така както са дефинирани в SCA Assembly Model спецификацията. Включва, също така и програмен интерфейс спрямо SCA Java Common Annotations and APIs и SCA Java Component Implementation спецификациите;
- Tuscany API – включва програмен интерфейс за допълнителни функции, дефинирани от Tuscany;
- Core – съдържа имплементацията на средата за изпълнение за Tuscany;
- Extensions – предлага възможности за разширения така, както са дефинирани в SCA спецификацията (на имплементационни типове, свързаности и интерфейси) заедно с още една възможност наречена свързаност за данни (databinding), която е специфична за Tuscany и ще бъде разгледана по-долу;
- Hosting platforms – е показано за илюстрация на средите, в които може да се използва Tuscany. Примери за такива среди са: самостоятелно приложение

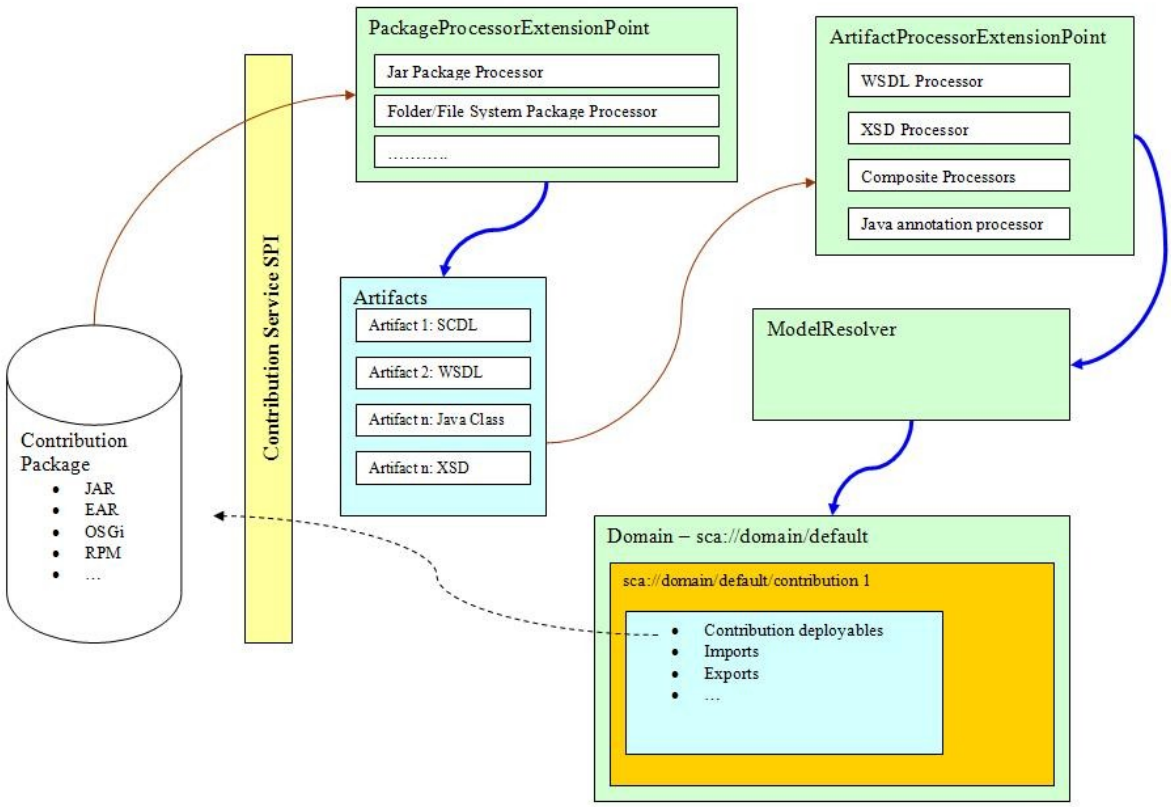
работещо на Java, уеб контейнер и др.



Фигура 8: Архитектура на Tuscany

Зареждането на едно SCA базирано приложение от средата за изпълнение – Tuscany се извършва на два етапа: първият се състои в зареждането на артифактите на това приложение (java класове, wsdl дефиниции, описания на композити и компонентни типове и др.) от даден източник, а при втория се осъществяват връзките между заредените компоненти.

Фигура 9 илюстрира действията, които се извършват при първия етап на зареждането.



Фигура 9: Зареждане на артифакти

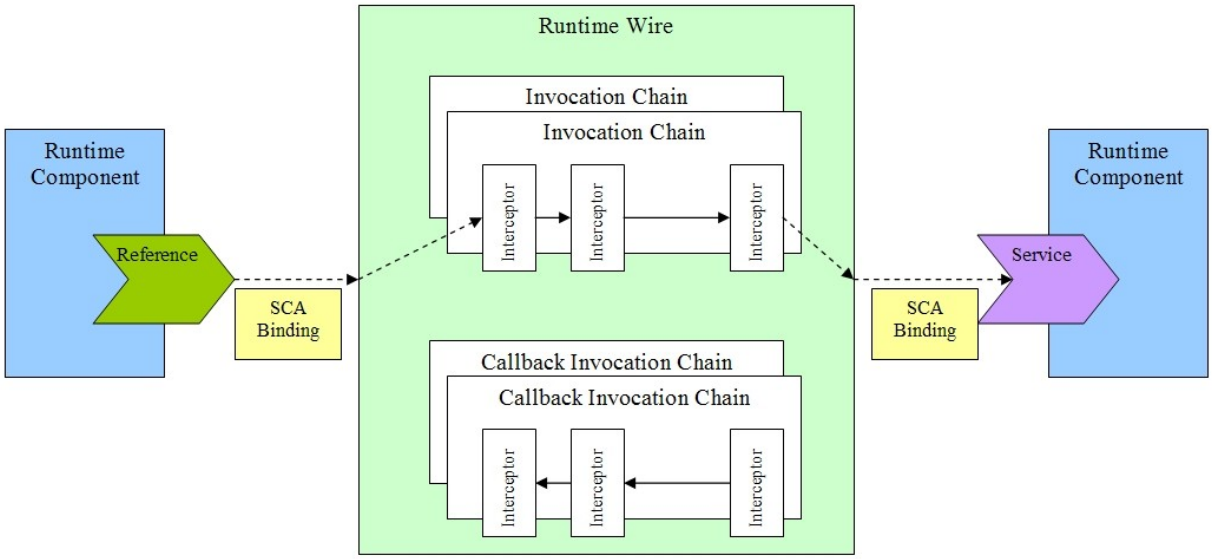
Първоначално се прочита приложението от съответния източник. Това приложение се нарича контрибуция (contribution) и може да е в произволен формат, който се поддържа от средата за изпълнение – Tuscany. Примери за формати са: файлове и директории; jar, ear или zip архив; OSGi пакети и др. Броят на поддържаните формати е разширяем, което се осъществява от точката за разширение *PackageProcessorExtensionPoint*, т.е. ако някой иска да предостави свой формат за съхранение на контрибуции е достатъчно да напише клас, който имплементира *PackageProcessorExtensionPoint* интерфейса и да го регистрира в Tuscany.

След като се прочетат артифактите, налични в контрибуцията се изгражда SCA модел в паметта на тяхна

база. За прочитането на различните типове артифакти отговарят различни процесори. Тъй като точката за разширение *ArtifactProcessorExtensionPoint* предлага възможност за регистриране на произволен брой процесори, то и броя на типовете артифакти, които могат да се прочетат от средата за изпълнение – Tuscany е неограничен. Например ако трябва за артифакт от тип java файл да се създаде компонент с имплементационен тип java е необходимо да се регистрира процесор, четящ java файлове. Именно този процесор ще е отговорен, на базата на предоставената java имплементация, да образува, чрез интроспекция, компонентния тип, който и отговаря.

ModelResolver представлява делегат, който осигурява йерархичното обработване на артифактите. Например ако трябва да се обработи артифакт, дефиниращ композит, този артифакт се подава на *ModelResolver*-а. Той от своя страна делегира заявката към *CompositeProcessor*-а. *CompositeProcessor*-а вижда, че в него са дефинирани набор от компоненти. За всеки от тези компоненти той извиква *ModelResolver*-а, който в зависимост от имплементационния тип на компонента преценя кой *ArtifactProcessor* да извика.

При втория етап от зареждането се създават връзки между референциите на даден компонент, които сочат към услуги предлагани от друг компонент. Тези връзки в Tuscany имат специфична архитектура, която е показана на фигура 10.



Фигура 10: Връзки между компонентите

На фигурата е показан сценарий, при който се използва свързаност по подразбиране – SCA binding.

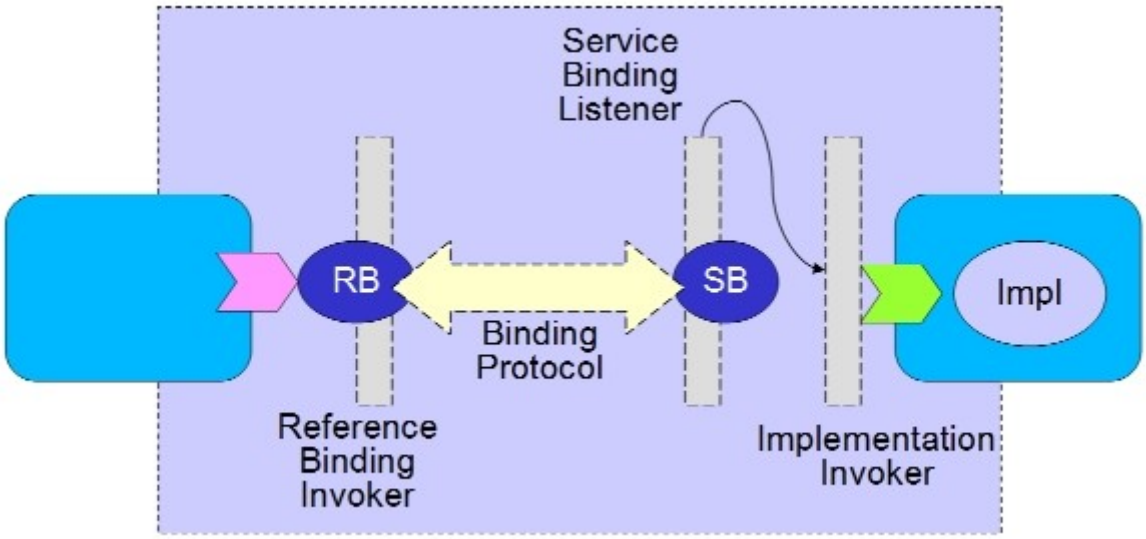
Всяка връзка се състои от множество вериги на извикване (Invocation Chains), по една за всеки метод от интерфейсите на услугата и референцията, които участват във връзката (тъй като двата участващи интерфейса са съвместими тези методи са еднакви по имена и брой). Във всеки от краищата на една такава верига стои по един метод съответно от референция и от услуга като е необходимо всеки един от параметрите на метода от референцията да се съпостави на параметър от метода на услугата. Допълнително трябва да се спомене, че тази двойка параметри трябва да са от един и същ логически тип, но не и от един и същи физически тип. За да се изяснят понятията физически и логически тип може да се даде следния пример: Даден компонент предлага услуга, имаща, метод приемащ XML съдържание като входен параметър. Това означава, че съответната референция, която иска да ползва

услугата трябва да има метод също с параметър представляващ XML съдържание. Въпреки това тези два съответстващи си параметъра не е необходимо да са от един и същи физически тип, т.е. в референцията XML съдържанието може да представлява StAX поток (StAX stream), например, докато в услугата да е представено чрез DOM възел (DOM Node). При това двата параметъра биха имали еднакъв логически тип, но различни физически представяния.

За да работи гореописания сценарий, Tuscany въвеждат понятието свързаност за данни (databinding). В продължение на горния пример, свързаността за данни на параметъра на референцията би бил *StAXStream*, докато свързаността за данни на услугата би бил *Node*. За да може да се осъществи извикване на метода от услугата по тази схема е необходимо веригата на извикване между двата метода да предлага набор от т.нар. интерцептори, които поемат отговорността по трансформиране на единия физически тип към друг. Интерцепторите са подредени верижно като резултата от всеки един интерцептор служи като входни данни за следващия интерцептор. Освен за трансформация на данни интерцепторите могат да предлагат и други услуги, които да се извършат по време на извикване като контрол на транзакциите и др.

Всяка верига на извикване завършва с т.нар. целеви извикващ (Target Invoker). Този извикващ е специфичен за конкретния SCA имплементационен тип или тип на свързаност, който ще се извиква. На фигура 11 са показани

извикващите при свързаност различна от SCA.



Фигура 11: Верига на извикване

В този случай има два извикващи, тъй като има две вериги на извикване: първата – между референцията на единия компонент и доставчика на свързаност и втората между доставчика на свързаност и услугата на втория компонент.

Tuscany предлага пълна разширяемост по отношение на извикващите, свързаностите за данни и трансформационните интерцептори. Всъщност, за да се добави нов имплементационен тип за SCA, например е необходимо да се направи следното:

- Да се предостави artifact processor, който да разпознава артифактите на новия тип;
- Да се предостави фабрика за извиквачи на компоненти, имплементиращи новия тип;
- Ако е необходимо да се предоставят свързаности за данни и трансформатори в съответствие с типа на

данните, който очакват целевите извиквачи за дадения компонент.

На базата на направеното разглеждане може да се види, че Apache Tuscany представлява една добре развита основа, предлагаща множество от възможности за разширяване, като превъзхожда с пъти другите съществуващи в момента имплементации на среди за изпълнение за SCA. Освен това той има съществено предимство, че е с отворен код, което го прави много лесен за разбиране и адаптиране. На базата на изброените причини, той беше избран като средство за реализация на XQuery разширението, в настоящата дипломна работа.

2.2.3 Eclipse STP

Проекта Eclipse STP [11][12] има за цел създаването на среда за проектиране и разработка на SOA базирани приложения, като се използва за основа Eclipse среда. Идеята на проекта е да се даде пълна свобода за разработка, като готовите приложения могат да се изпълняват на най-разнообразни среди за изпълнение. Например, чрез STP разработчика може да създава обикновени уеб услуги, които да инсталира на Apache Tomcat сървър, също така може да проектира SCA композити, които да инсталира на среда за изпълнение, поддържаща SCA. С помощта на BPMN редактор могат да се дефинират бизнес процеси, които да се инсталират като услуги за оркестрация.

За да се посрещне тази нужда от разнообразие, свързана с разработката на SOA приложения са дефинирани следните под-проекти на STP:

- STP Core Frameworks (CF) – основната му цел е да дефинира EMF модел за SCA. В допълнение има модули за интроспекция на Java класове и точки за разширение на модела;
- STP SOA System (SOAS) – предоставя средство за пакетиране, построяване и инсталиране на SOA приложения. Архитектурата на SOAS е такава, че дава възможност за дефиниране на разширения, които да определят как да се образуват архивите, които ще се инсталират, както и разширения за различни типове среди за изпълнение (обикновен JEE сървър, JBI среда, SCA среда и др.);
- STP Service Creation (SC) – представлява основа, позволяваща интегрирането на различни средства за създаване на услуги. В момента съществуват две разширения – едното позволява създаването на услуги на базата на JAX-WS спецификацията, а второто – Java услуги за SCA контейнер;
- STP BPEL 2 Java (B2J) – представлява средство за генериране на java код от BPEL описание на бизнес процес, както и среда, в която този код да се изпълнява;
- STP BPMN (BPMN) – представлява графичен редактор на бизнес процеси съгласно с BPM нотацията. Дава възможност, също така, за генериране на BPEL на базата на графично моделирания процес.

В допълнение към описаните основни под-проекти съществуват и няколко предложения за под-проекти, които в момента са в статус на IP (intellectual property)

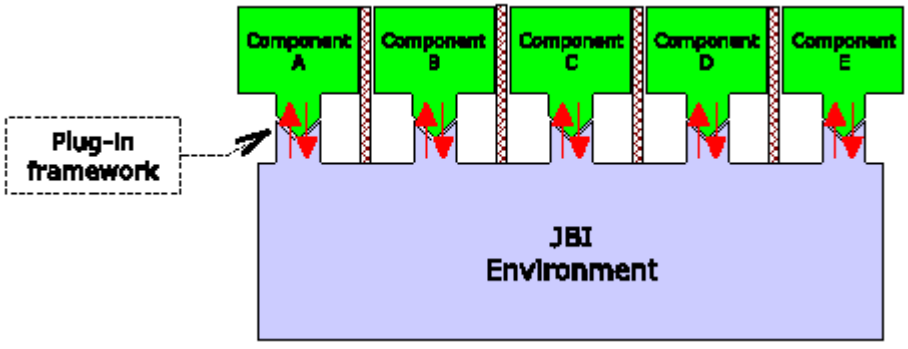
верификация. Това са:

- STP Intermediate Metamodel – представлява модел на най-важните компоненти на едно SOA приложение, като целта е той да е основа за всички SOA редактори. За разлика от модела в STP Core, който е специфичен за SCA, този модел може да се използва за дефиниране на едно място, както на бизнес процеси, така и услуги и техните взаимовръзки (необходими за SCA имплементациите) и артифакти, необходими за създаването на JBI приложения. С помощта на този модел ще е възможна лесната интеграция на артифакти създадени с BPMN редактор, например, в редактор за SCA;
- SCA Composite Editor – представлява редактор за SCA, чрез който могат да се създават SOA приложения на базата на top-down начина (т.е. първо се описват услугите и техните взаимовръзки и след това се прави имплементация). Този редактор е предложен от компанията Obeo и скоро ще е достъпен като код. Съществува и друг редактор, предложен от SAP, който реализира bottom-up начин на създаване на SCA композити. При този редактор се прави интроспекция на съществуващите на файловата система артифакти и се генерира дефиниция на композит. За съжаление, той няма да е наличен скоро;
- Cimero 2 – графичен редактор за реализиране на SOA приложения съгласно с JBI спецификацията.

2.2.4 Java бизнес интеграция (JBI) и Open ESB

Ако SCA е спецификация, дефинираща компонентен модел за описание на SOA приложения, то JBI спецификацията е насочена главно към т.нар. интеграционни сценарии. Тя има за цел да стандартизира решенията в областта на шините за комуникация между услуги (Enterprise Service Bus – ESB). В нейната основа лежи използването на шаблона *медиатор* (mediator pattern) [13], чрез който се осигурява независимост между комуникиращите си страни.

На фигура 12 е показана принципната схема на една JBI среда. В нея участват множество от компоненти, наречени плъгини (plug-ins), които си комуникират с медиатора (JBI Envrionment), но не и директно по между си. Всъщност това е и една от основните разлики между JBI и SCA, където връзките между отделните компоненти се задават експлицитно.



Фигура 12: Принципна схема на JBI

Два типа компоненти могат да се свързват към JBI средата: единия се нарича двигател за услуги (Service Engine – SE), а другия – компонент осигуряващ връзките (binding component – BC).

Двигателите за услуги, всъщност не са компоненти сами по себе си, но представляват контейнери за специфичен тип бизнес логика като предоставят достъп до тази логика посредством услуги. Примери за SE са EJB контейнер, XSLT процесор и др.

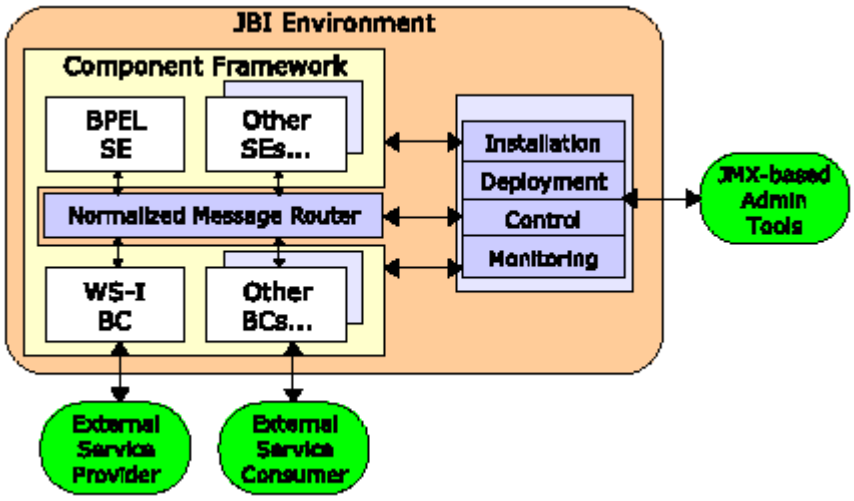
Компонентите осигуряващи връзки предоставят функционалност за връзка с външния свят посредством даден специфичен протокол (connectivity). Например те могат да предоставят възможност за извикване на външни услуги, чрез SOAP, RMI, или JMS. В тази си функция тези компоненти са много подобни на SCA свързаностите.

В JBI всяка услуга, предоставяна от SE или BC се представя посредством WSDL-базиран интерфейс. Характерното за този интерфейс е това, че за описанието му се използва главно абстрактната част на WSDL дефиницията, докато конкретната част е само про-форма, т.е. не съдържа реална крайна точка (endpoint) на услугата, а само нейното име. Също така ако дадена услуга изпрати съобщение на друга услуга, като адрес се поставя само името на целевата услуга, а не пълният и адрес. Този тип съобщение се нарича *нормализирано съобщение* (normalize message) и неговата цел е да делегира задачата по избиране на конкретния му адресат на медиатора. Освен целевия адрес, нормализираното съобщение може да съдържа и допълнителни метаданни, които са необходими за правилното насочване на съобщението.

В термините на JBI медиатора, разпределящ съобщения се нарича *маршрутизатор на нормализирани съобщения*

(normalized message router – NMR). Той е отговорен да изпрати дадено съобщение до неговия адресат, като преди това, по свое усмотрение или базирайки се на метаданните в съобщението, той може да го разпредели към различни междинни получатели. Например ако една услуга иска да извика друга услуга, но формата на съобщението очакван от целевата услуга не съвпада с формата на изпатеното съобщение, тогава NMR може да предаде първо съобщението към междинно звено (например за XSLT или XQuery трансформация), което да приведе съобщението в желания за целта вид.

На фигура 13 е показана архитектурата на една JBI среда.



Фигура 13: Архитектура на JBI

JBI средата предоставя:

- Маршрутизатор на нормализирани съобщения (Normalized Message Router);
- Компонентна основа (Component Framework) – нейната цел е да предостави стандартни интерфейси,

които даден SE или BC трябва да имплементират, за да участват в JBI (това са т.нар service provider interfaces – SPIs), а също така и стандартни интерфейси, чрез които JBI средата предоставя услуги на регистрираните SE и BC (това са т.нар. application provider interfaces – APIs);

- JMX – базирани средства за: инсталиране и деинсталиране на SE и BC (installation); добавяне на нови артифакти към даден SE (deployment); контрол на цикъла на живот на инсталираните компоненти (пускане и спиране на даден компонент); мониторинг на системата.

В JBI едно SOA приложение представлява колекция от артифакти, наречени единици за услуги (serivice units), които се разпознават от съществуващите SE и BC. Тази колекция се нарича *асембли от услуги* (service assembly) или още *композитно приложение* (composite application). Примери за единици за услуги са EJB-та, XSLT трансформации, JSP файлове и др.

За да се разграничи условно добавянето на нов компонент към JBI средата от добавянето на нови единици за услуги към даден компонент, за първото се използва термина инсталация (installation), а за второто – термина добавяне (deployment).

От направеното разглеждане може да се заключи, че в терминологията на JBI, JBI средата представлява контейнер от контейнери, а пък SE и BC са контейнери за артифакти. Това всъщност обуславя още една разлика между SCA и JBI – в SCA понятието услуга е равносилно на понятието

компонент, докато в JBI понятието контейнер за услуги е равносилно на понятието компонент.

Въпреки отбелязаните различия между JBI и SCA, обаче, лесно може да се забележи, че двете технологии не си противоречат, а се допълват. Например, лесно би могла да се постигне интеграция на SCA в рамките на JBI като се предостави SE, който да е контейнер за SCA артифакти. По подобен начин лесно би могла да се постигне интеграция на JBI в рамките на SCA като се въведе нов имплементационен тип или свързаност, които да се изпълнява от JBI среда.

От съществуващите имплементации на JBI, най-развита е имплементацията с отворен код на SUN, наречена Open ESB [14]. Нейната среда за изпълнение е базиран на JEE сървъра на SUN – Glassfish, като е предоставен голям набор от SE и BC, които дават възможност за реализация на комерсиални SOA приложения. Като среда за разработка се използва SUN NetBeans, чрез която се разработват композитни приложения, които могат да се добавят към JBI средата за изпълнение.

В повече подробности ще бъдат разгледани тези SE на Open ESB, които са ориентирани към интеграция на данните в главата за интеграция на данните и адаптация на услуги.

Имплементационният тип за XQuery, който е предмет на настоящата дипломна работа, би могъл да се създаде и за JBI съвместими имплементации под формата на SE, което би могло да бъде посока за доразвиване на идеята. Въпреки това в JBI по подразбиране се залага на трансформацията на съобщения, като също така вече има изготвени

трансформатори за XSLT. Поради това нуждата от такъв компонент, който да прави адаптиране на интерфейси е много по-голяма в рамките на SCA отколкото в рамките на JBI и всъщност това е причината за избора на SCA като цел на разработката.

2.3 Достъп до данните чрез SDO

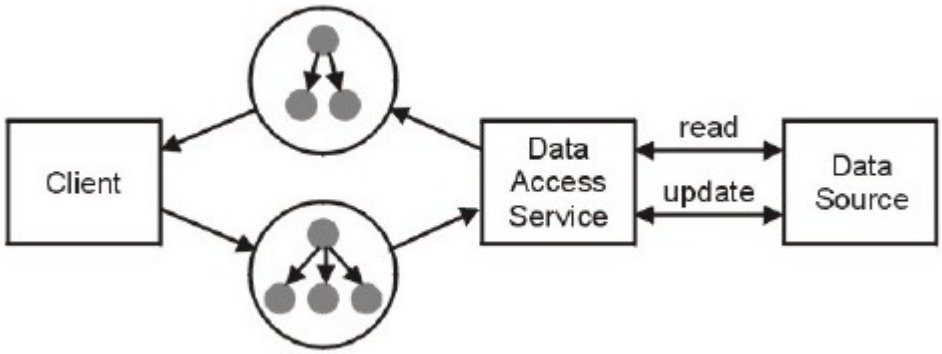
Спецификацията за SDO (Service Data Objects) върви ръка за ръка със спецификацията за SCA и също е поверена на OASIS за по-нататъшна разработка и доразвиване.

SDO представлява развързан (disconnected), йерархичен модел за достъп до данни от различни източници.

Когато даден клиент иска да получи достъп до някаква информация, той се свързва с услуга за достъп до данните (Data Access Service – DAS), виж фигура 14. Тази услуга знае начина, по който да си комуникира със съответния източник на данни. Това може да бъде база от данни, XML файл и др. Услугата прави заявка към източника, който връща резултат. След това DAS превежда резултата в модела на SDO и го подава към клиента. Клиента от своя страна може да модифицира получения модел и след това да изпрати резултата към DAS, който от своя страна ще приложи промените върху съответния източник на данни.

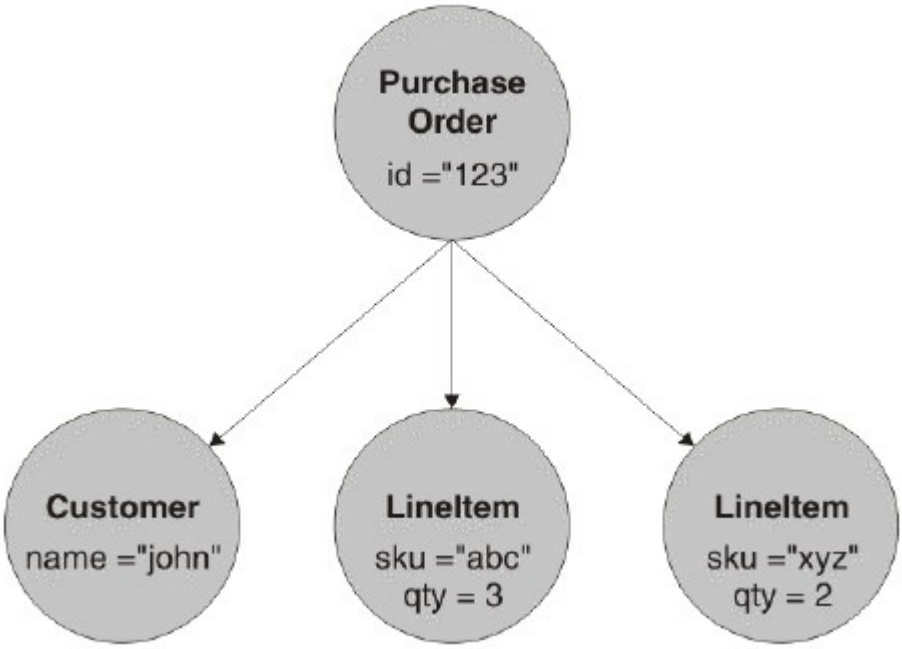
В основата на модела за представяне на данни в SDO стои т.нар. обект на данните (Data Object). Всеки обект на данните може да има определен брой свойства (properties) и референции към други обекти на данни. Именно с помощта на тези референции се осъществява граф от обекти, който

представят данните в йерархичен вид.



Фигура 14: Достъп до данните в SDO

Примерен граф от обекти и техните свойства е показан на фигура 15. Вижда се, че има един основен обект, който представлява корен на графа (root data object), представящ заявка за покупка на продукти. Той притежава свойството “id”, представящо уникален идентификатор на покупката. Всяка заявка за покупки се прави от определен клиент и има определен брой закупени продукти (line items).



Фигура 15: Граф на SDO обектите

От фигурата може да се види, че графа представя

конкретните данни, а не техния модел (т.е. не метаданните). Т.е. този граф може да се оприличи на графа от java обектите на едно приложение, което работи в рамките на java виртуалната машина. SDO, обаче, дава възможност и за достъп до метаданните на графа ако са налични такива (метаданни са налични например тогава когато даден XML, който е прочетен от DAS е имал и XSD схема, която също е била прочетена), по същия начин по който java обекта дава достъп до неговия клас, чрез getClass метода.

Друга услуга, предоставяна от SDO е история на промените, които даден клиент е извършил по графа. Тази история е необходима в последствие на DAS, за да извърши съответните промени върху източника на данни.

Съществуват два начина за достъп до графа: първият е с използване на динамичния интерфейс (dynamic API), който е аналогичен на това да се работи с java обекти посредством reflection. При втория начин се използват генерирани класове (Java, C++, или други в зависимост от имплеметацията на SDO), които предоставят т.нар. статичен интерфейс (static API). Генерацията на тези класове се извършва от специални генератори на базата на метаданни. Т.е. за да може да се извърши такава генерация е необходимо да са налични метаданни на данните които се очакват (например XSD схемата на XML-ите, които се очакват като входни данни).

Като заключение може да се каже, че SDO е гъвкава технология за достъп до данни от различни източници, която постепенно се превръща в стандарт поради широката

поддръжка, която се осигурява от световните производители на софтуер.

2.4 Преглед на езика XQuery

XQuery е език, предназначен за правене на заявки към данни представени в XML формат. По това че той трансформира входните данни от един формат в друг, той прилича на XSLT. От друга страна, поради специфичния си синтаксис и това че е ориентиран повече към обработка на данни отколкото на документи, той представлява за XML-базираните източници на данни това, което е SQL за релационните бази от данни.

XQuery е специфициран от W3C консорциума [15] и се базира на XPath за достъп до данните.

Едно от основните предимства на XQuery се състои в това, че той не изисква данните да са представени в XML формат, а само да може да се прави достъп до тях, като че ли са XML. Това означава, че е необходим само преводач на заявка написана на XQuery към заявка поддържана от съответния източник на данни (например SQL) за да се осъществи достъп до тези данни. Това предимство, заедно с факта, че XQuery се поддържа от основните производители на бази от данни като Microsoft, IBM и Oracle, го превръща постепенно в стандарт за достъп и интеграция на данните.

Основна синтактична конструкция в езика XQuery представлява т.нар. флауър шаблон (FLWOR). Името му произхожда от основните елементи на шаблона:

- **F**or – дефинира началото на цикъл;

- **L**et – представлява декларация на променливи в рамките на тялото на цикъла;
- **W**here – предоставя възможност за задаване на условие, при което би се изпълнило тялото на цикъла (абсолютно аналогично на *where* клаузата в SQL);
- **O**rder by – дефинира начина, по който ще са подредени резултатите от цикъла (абсолютно аналогично на *order by* клаузата в SQL);
- **R**eturn – Дефинира тялото на цикъла – т.е. това което ще се върне от дадена итерация на този цикъл (аналогично на *select* в SQL).

По-долу е показан пример за FLOWR клауза написана на XQuery.

```
for $x in doc("books.xml")/bookstore/book
let $thePrice := $x/price
let $theTitle := $x/title
where $thePrice>30
order by $theTitle
return <title>{ $theTitle }</title>
```

От примера може да се види как се използва XPath за селекция на елементи от входния XML. Също така прави впечатление начина, по който се смесва текста предназначен за изходния документ със елементите от синтаксиса на XQuery: <title>**{** $theTitle **}**</title> - посредством фигурни скоби.

Други средства на езика XQuery, които го правят особено полезен са:

- Възможността да се организира съдържанието под формата на функции, които могат да се извикват една –

друга;

- Възможността да се декларират и използват т.нар. външни функции и променливи, което дава възможност да се извика например функция написана на Java или на C++ от рамките на XQuery кода;

- Възможността за организация на XQuery кода под формата на модули, като всеки модул се намира в отделен файл. В последствие даден модул може да декларира използването на друг модул (module import). По този начин се осигурява гъвкавост по отношение на организацията на кода;

- Резултатът от изпълнението на XQuery може да е в произволен формат, т.е. не е задължително да е XML.

По-долу е представено сравнение между езиците XQuery и XSLT, чрез което се аргументира избора на XQuery пред XSLT, за изпълнение на задачата по интеграция на данни в настоящата дипломна работа [16]:

- FLOWR шаблона е много по-разбираем и експресивен за хора свикнали с използването на SQL, отколкото XSLT шаблоните;

- XQuery поддържа независима компилация на модули;

- XQuery много по лесно се интегрира в приложения написани на даден програмен език, тъй като резултата от дадена XQuery заявка може да е от произволен тип на данните, а не само XML, както при XSLT;

- XQuery е много по-подходящ за обработка на

големи обеми от данни, докато XSLT е по-подходящ за обработка на малки обеми от документи. Това прави XQuery незаменим за задачите по интеграция на данни;

- Синтаксиса на XQuery е по-процедурно ориентиран, за разлика от XSLT, който е по декларативен. Това прави XQuery по-удобен за използване от потребители свикнали да използват процедурни езици за програмиране (каквито са повечето съвременни езици).

Като средство за интерпретация и изпълнение на XQuery заявки в настоящата дипломна работа е използван процесора с отворен код Saxon [17], който в момента е една от най-добрите съществуващи некомерсиални имплементации.

Saxon се предлага в две разновидности:

- Saxon B – разработка с отворен код;
- Saxon SA – комерсиална разработка.

Разбира се, Saxon SA предлага много повече възможности, които са от значение при използването на XQuery в продуктивна среда, но тъй като целта на дипломната работа е повече ориентирана към демонстрация на концепция, отколкото към създаване на приложение, което да се използва от крайни клиенти, то Saxon B, с неговото базово множество от функции, е напълно достатъчен.

По долу са представени характеристиките на Saxon B процесора:

- Поддръжка на XSLT 2.0 спецификацията;
- Поддръжка на XQuery 1.0 спецификацията;
- Поддръжка на XPath 2.0 спецификацията.

В допълнение Saxon SA предлага функции като поддръжка на схема при обработка на входния XML; възможност за генерация на Java код от XQuery, което би повишило бързината на изпълнение; Много по-добре оптимизиран е по отношение на използваната памет, което дава възможност да се обработват по-големи обеми от информация за по-кратко време.

3 Глава 3. Интеграция на данни и адаптация на услуги (медиация)

Един от най-важните елементи за успешното реализиране на едно SOA приложение е да се създаде възможност то да черпи информация от различни източници по еднакъв начин, а също така да се осигури възможност, услуги с различни интерфейси, обменящи си различни по тип съобщения да комуникират една с друга. Благодарение на тези елементи от SOA инфраструктурата, приложението би могло успешно да се интегрира с различни източници на данни, част от които вече съществуващи (legacy), а също така би могло лесно да се разширява без да се променят вече дефинираните интерфейси на услугите, които го изграждат.

Основата на интеграцията на данни и адаптацията на услуги се състои в това от данни с един физически формат и логическа структура да се получат данни с друг формат и друга логическа структура. Съществуват два сценария, в

които се налага да се извърши това [18]:

- при интеграция на данни от различни източници като бази от данни, XML файлове, уеб услуги и др. (data integration). Съществуват три подхода към извършването на тази интеграция:
  - чрез създаване на виртуален образ на данните от различни източници, така че те да изглеждат все едно са един източник (т.нар. virtualization или още Enterprise Information Integration - EII). Това се постига, чрез процес наречен федерация (federation);
  - чрез извличане, трансформиране и записване на данните от различни източници в една обща база от данни наречена склад за данни (data warehouse). Процеса се нарича ETL (Extract Transform Load) и може да се стартира от различни тригери: на регулярна основа, при промяна на входните данни и др;
  - чрез поддържане на репликация между различните източници на данни. По този начин всеки от източниците съхранява всички необходими данни;
- интеграция при изпълнение на даден процес (т.нар. process integration или Enterprise Application Integration – EAI), когато се налага осъвместяване на интерфейсите на извикваната и извикващата услуга, както и на съобщенията, които те си разменят. Това осъвместяване се нарича още медиация (mediation). Съществуват два типа медиация:

- на данни – тогава, когато дадено съобщение се превежда от един формат в друг. Много често този тип медиация се използва в ESB решенията;
- на интерфейси – тогава, когато трябва да се осигури комуникацията между компоненти с различни интерфейси. Много често този тип медиация се използва при оркестрацията и композицията на услуги.

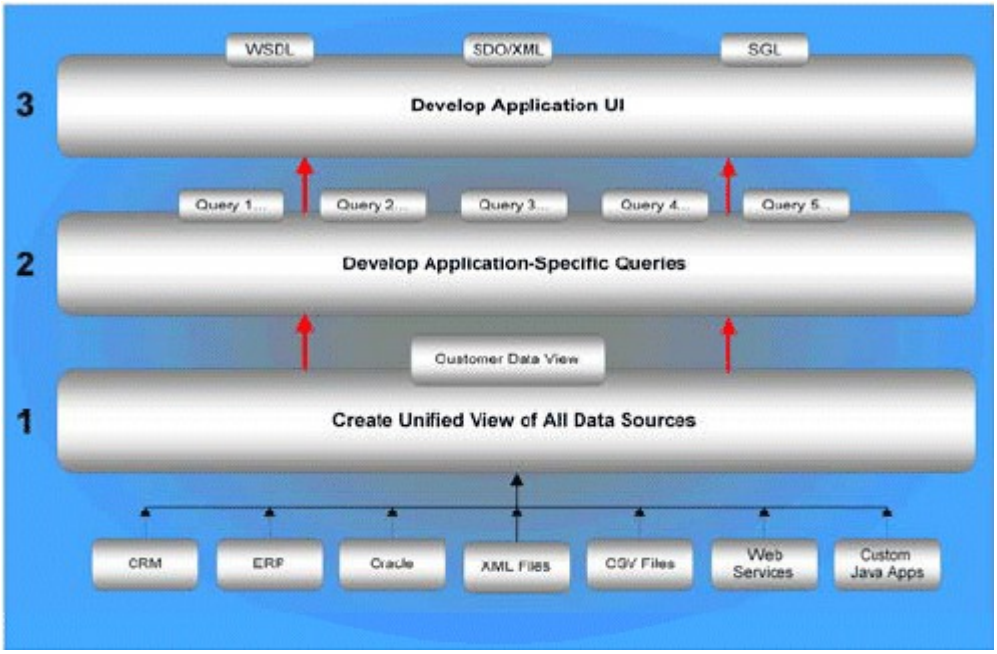
Това са сценариите, за които е направено и проучването в тази глава между различните фирми производители на софтуер, като целта е да се установят и съпоставят начините, по които те се справят с проблемите поставени при всеки един от тези сценарии.

3.1 BEA AquaLogic платформа

На фигура 16 са показани слоевете залегнали в архитектурата за интеграция на данни в BEA AquaLogic платформата. [19]

Първият слой съдържа т.нар. физически услуги за данни (physical data services), които представят данните от различни източници в един физически формат, но запазват тяхната семантика, т.е. не променят тяхната логическа структура.

Вторият слой съдържа т.нар. логически услуги за данни (logical data services), които използват информацията предоставена им от физическите услуги, за да променят логическата структура, така че тя да се осъвмести с домейн модела на конкретното приложение, което се разработва.



Фигура 16: Слоеве на BEA AquaLogic платформата

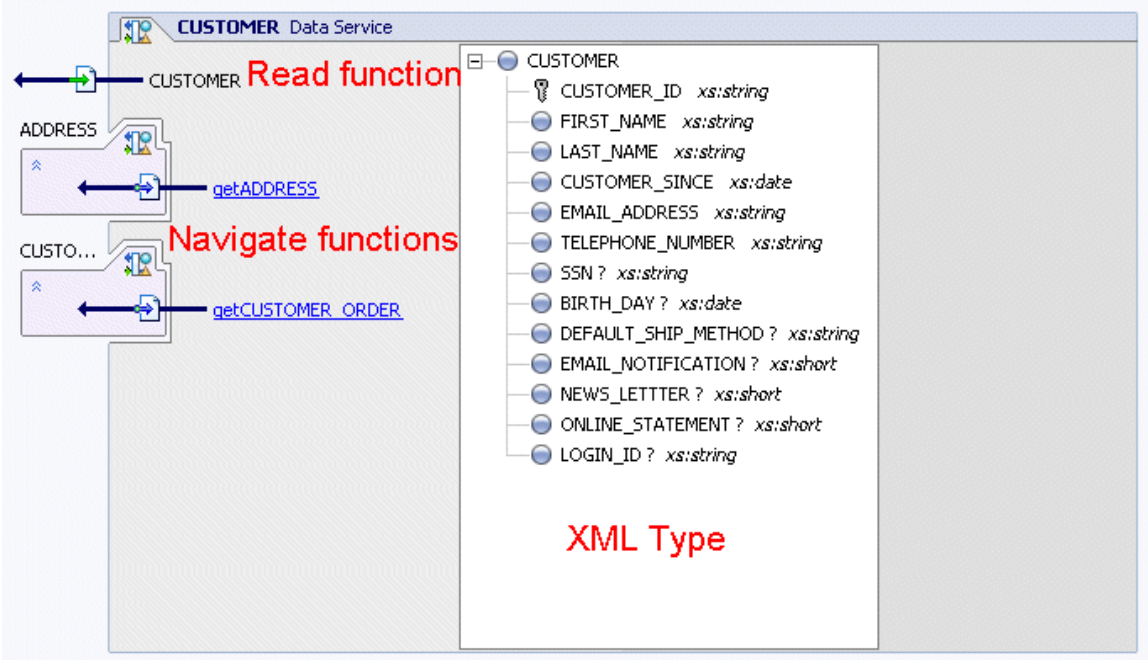
Третият слой представя клиента на AquaLogic платформата, който би могъл да представлява потребителския интерфейс на приложението например, или пък услуги имплементиращи дадена бизнес логика или процес.

Следните принципи са залегнали в AquaLogic:

- имлементацията на услугите за данни (физически и логически) е базирана на езика XQuery, като за неговото разширение се използват анотации във формата на коментари;
- за достъп до данните от страна на клиента се използва SDO;
- типовете на данните се описват с помощта на XSD;
- всички връзки между услугите за данни се осъществяват, чрез използването на декларации на

пространства от имена (namespace) в съответните услуги.

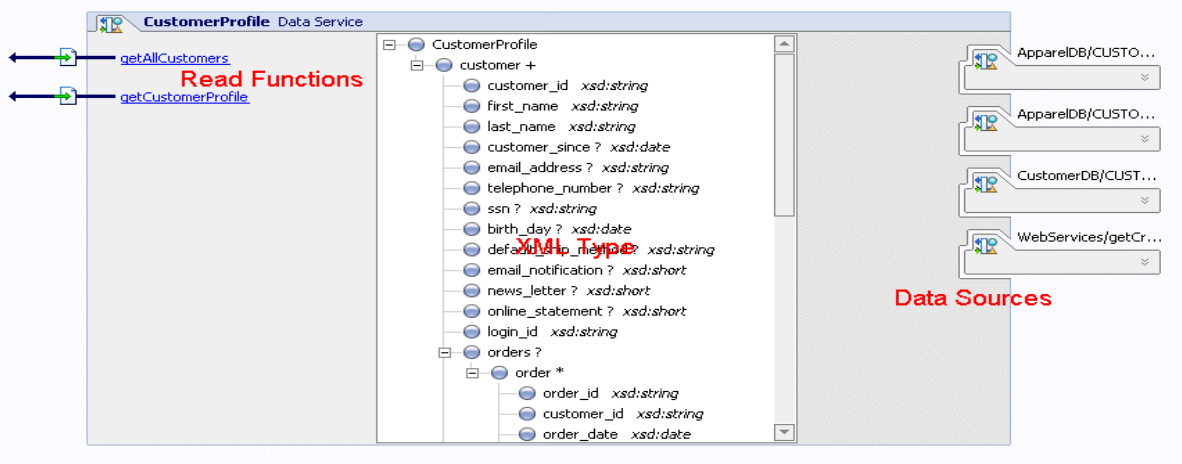
Физически услуги за данни могат да се създават от различни източници. Например може да се генерират на базата на схема на релационна база от данни или да се генерират на базата на WSDL на дадена уеб услуга и др. Като резултат се получава физическа услуга за данни имаща свой XML тип (отговарящ на съответния тип от източника на данните), операция за четене на данните от този тип (read function), операции за навигиране към услуги за данни, които са реферирани от тази услуга (navigate functions), а също така операции за запис на промените, изтриване и добавяне на нови записи (т.нар. CRUD операции – create, read, update, delete). Трябва да се отбележи че физически налични са само операциите за четене на данни и навигиране, докато останалите се изпълняват автоматично при записване на модифицирания SDO граф (виж фигура 17).



Фигура 17: Физическа услуга за данни

За разлика от физическите услуги за данни, логическите не се генерират, а се създават от разработчика. Те, заедно с взаимовръзките си, представят домейн модела на SOA приложението.

Логическите услуги за данни също имат свой XML тип, функция за четене и функции за навигация. В допълнение, обаче, те могат да имат референции към 0 или повече други физически или логически услуги. На фигура 18 може да се види пример за това как една логическа услуга може да комбинира информацията постъпваща от няколко физически услуги, за да създаде профил на даден потребител.



Фигура 18: Логическа услуга за данни

В AquaLogic платформата има графичен редактор за създаване на логически услуги, като по този начин, дори потребител, неразбиращ езика XQuery може да работи успешно. Друга положителна характеристика от гледна точка на потребителския интерфейс е редактор позволяващ визуализирането и модифицирането на целия домейн модел под формата да диаграма подобна на UML.

BEA AquaLogic платформата предлага възможност за адаптиране на интерфейси, което е много полезно особено при създаването на бизнес процеси, когато се налага трансформиране на данните, получени от дадена услуга във формата, който е необходим на процеса. Това става посредством използването на контроли. Контролът представлява елемент обхващащ даден компонент от архитектурата на приложение написано с AquaLogic, с цел този компонент да може да се използва на различни места. Например една услуга за данни, една уеб услуга или една XQuery трансформация могат да се представят чрез контрол, който да се използва от бизнес процес или друга

услуга. В случая на адаптирането на интерфейси, съществува контрол, който обхваща дадена XQuery трансформация и този контрол може да се използва от всеки, който се нуждае от него. Тази архитектура донякъде наподобява SCA, като контролите са аналогични на SCA компоненти.

От графична гледна точка е направен редактор, с който могат да се правят много сложни трансформации и който записва направената трансформация директно в XQuery формат. Обратното също е валидно: с този редактор може да се отвори произволна XQuery трансформация и тя да се представи графично.

Освен за целите на интеграцията на данни и адаптацията на интерфейси за бизнес процеси XQuery е залегнал и в основата на трансформацията и маршрутизацията на съобщения в продукта на BEA, реализиращ Enterprice Service Bus (ESB). При него с помощта на уеб-базиран потребителски интерфейс се дефинират таблици, в които се описват условията за маршрутизиране на дадено съобщение. В рамите на дефиницията на условието могат да се използват данни от входящото съобщение и от други източници и на тяхна база да се определи какво трябва да се извърши със съобщението. Една от възможностите е то да се нуждае от трансформация. Ако това е така се прилага XQuery-дефинирана трансформация върху входното съобщение и тогава то се предава за по-нататъшна обработка.

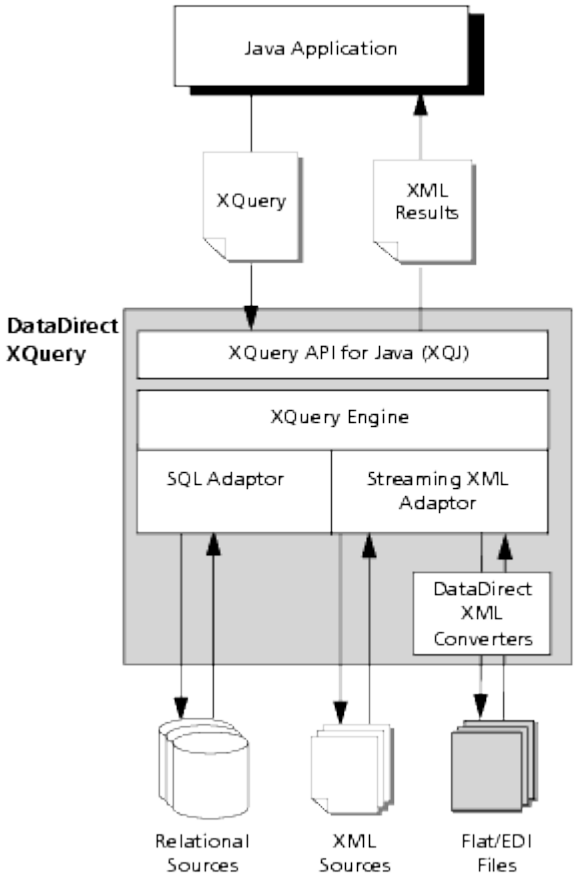
3.2 DataDirect XQuery

DataDirect е компания, занимаваща се с решения, свързани с обработката на данни при големите предприятия. Един от техните продукти представлява XQuery процесор, даващ възможност за интеграция на данни от различни източници [20]. Този процесор е базиран на Saxon, но е значително оптимизиран и разширен по отношение на работата с големи обеми от данни в най-различен формат.

На фигура 19 е показана архитектурата на DataDirect XQuery.

Потребител на процесора представлява Java приложение (Java Appliaction), което доставя XQuery заявката, заедно с конфигурационна информация за това от къде ще се четат данните и по какъв начин. Тъй като DataDirect XQuery поддържа програмния интерфейс XQJ, той представлява контракт между потребителското приложение и останалата част от модулите на процесора.

В сърцето на процесора стой XQuery двигател (XQuery Engine), който е базиран на Saxon и се грижи за семантичната и синтактична обработка на заявката.



Фигура 19: Архитектура на DataDirect XQuery

Достъпа до данните се осъществява чрез т.нар. адаптери. Съществуват два основни адаптера:

- за SQL (SQL Adaptor) – основната му функция е да превежда части от XQuery заявката в SQL. С негова помощ се избягва зареждането на всички данни от базата в паметта преди да се изпълни XQuery заявката. Благодарение на него се зарежда само информацията, която е наистина необходима и по този начин се пести много памет;
- Адаптер за XML под формата на поток от данни (Streaming XML adaptor) – той доставя XML форматирани данни на процесора, за да изпълни

заявката върху тях. Характерно за него е че XML информацията постъпва под формата на поток, т.е. в RAM паметта никога не се зарежда целия входен XML а само частта, която е необходима за изпълнение на съответния елемент от заявката. По този начин се пести памет и са дава възможност за обработка на обеми от данни далеч надхвърлящи размера на оперативната памет.

Освен адаптери в DataDirect XQuery се въвежда понятието конвертори (Convertor). Целта на конверторите е да преобразуват информация в произволен формат (например формата за обмяна на електронни документи EDI) в XML и по този начин да я направят достъпна за изпълняващата се заявка [21].

Заедно с процесора, който изпълнява XQuery заявки е разработен и богат потребителски интерфейс (Stylus Studio) [22], даващ възможност за графично представяне на трансформации от структури от различни източници на данни към дадена изходна структура. Разработените с редактора трансформации се записват директно в XQuery формат. Друга важна характеристика на редактора е, че позволява проследяване за грешки (debug) на XQuery трансформации и заявки.

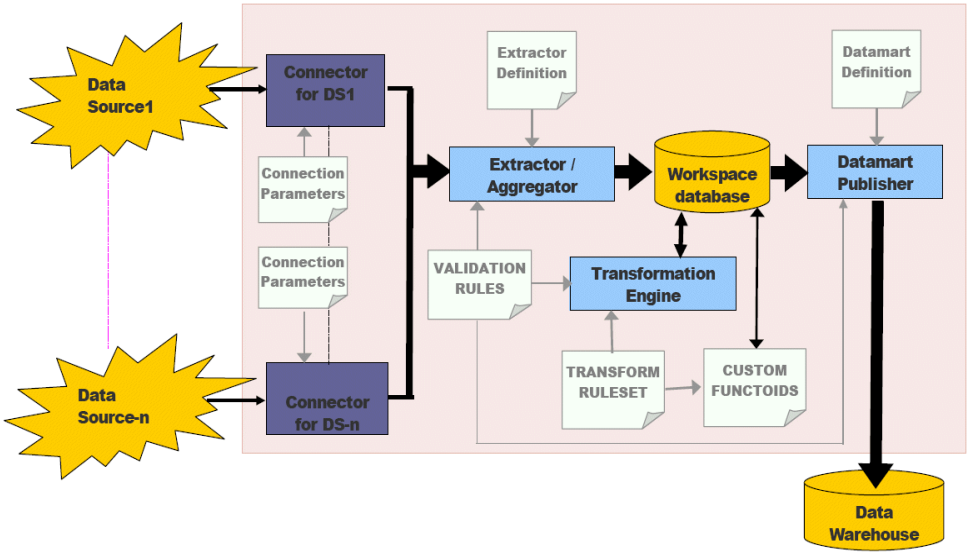
3.3 SUN Open ESB

SUN Open ESB е имплементация с отворен код на JBI предоставена от SUN. Тя беше разгледана в главата, описваща JBI спецификацията. Тук ще бъдат разгледани тези части от нея, които са свързани с интеграция на данни

и адаптиране на интерфейси.

3.3.1 ETL SE

Първият компонент, който е предназначен за интегриране на данни се нарича ETL SE (Extract, Transform and Load service engine) [23]. На фигура 20 е показана принципната схема на неговата работа: т.нар агрегатор (Extractor/Aggregator) извлича данните от различни източници (Data Sources) посредством свързващи модули (Connectors) и ги записва във временна база от данни (Workspace Database). Върху данните от тази база се прилагат дефинираните трансформации и се извършва агрегиране. Полученият резултат се записва посредством специален софтуер (Datamart Publisher) в склад за данни (Data Warehose), които в последствие се използват от JBI-базираното приложение.



Фигура 20: Принципна схема на ETL SE

За създаване на необходимите трансформации, които трябва да се изпълнят се използва SQL като има редактор,

позволяващ графичното им дефиниране.

3.3.2 XSLT SE

Друг двигател на услуги, който е част от Open ESB платформата се нарича XSLT SE (XSLT service engine) [24] и се използва за адаптиране на WSDL интерфейси между извикващи се услуги. С негова помощ се извършват XSLT трансформации на съобщения в следните шаблони на извикване (message exchange patterns – MEP):

- Заявка – отговор (“requestReplyService”), като заявката се изпраща към трансформационната услуга и съдържа съобщението, което трябва да се трансформира, а отговора съдържа трансформираното съобщение;
- еднопосочно извикване (“filterOneWay”). При него заявката съдържа съобщението, което трябва да се изпрати към съответния адресат, а NMR-а отговаря това съобщение да се филтрира от XSLT двигателя по неговия път;
- двупосочно извикване (“filterRequestReply”). При него, както и при еднопосочното извикване се извършва трансформация на съобщението, представляващо заявка към дадена услуга, но NMR-а е отговорен за трансформация и на отговора от извикването на тази услуга.

3.3.3 EDMS SE

Enterprice Data Mashup Services двигател за услуги (EDMS SE) дава възможност за изпращане на SQL заявки към различни източници на данни като по този начин се

създава единен образ на тези източници [25]. За реализиране на тази задача се използва виртуална база от данни, с която клиента работи посредством JDBC. Тази база от своя страна превежда SQL заявките в зависимост от типа на източника, към който са насочени и ги препредава. Резултата се оформя като JDBC ResultSet.

EDMS позволява достъп до различни типове източници на данни: релационни бази от данни, обикновени файлове, XML файлове (за достъп до които се извършва трансформация от SQL заявка към XQuery заявка) и др. Въпреки това тази услуга не предоставя възможност за създаване на нов модел на данните, т.е. модела трябва да се създаде с използването на друга технология (като EJB например).

3.3.4 SQL SE

SQL двигателя за услуги (SQL SE) [26] дава възможност да се предоставят услуги, които са имплементирани посредством SQL като този SQL се изпълнява върху базата от данни и резултата се връща като отговор от извиканата операция. SQL заявките се описват в .sql файл, като на един такъв файл съответства услуга с една операция. С помощта на SQL SE може да се създаде изглед на данните в базата под формата на уеб услуги като DB типовете на данните се транслират към XSD типове.

3.3.5 Адаптиране на типове при BPEL бизнес процеси

За адаптиране на типовете, връщани от дадена услуга към типове, които се изискват в даден бизнес процес се използва XPath като е предоставен графичен редактор за

дефинирането на XPath изразите. Това решение е значително по ограничено от използването на XQuery, тъй като липсват синтактични елементи като условие, цикъл и др. Освен това е невъзможно да се обединят резултатите върнати от повече от една услуги в един резултат.

Като цяло може да се заключи, че в SUN Open ESB за интеграция на данни се използва езика SQL, а за адаптиране на интерфейси – XSLT и XPath.

3.4 Интеграция на данни и процеси в Software AG

Software AG предлагат добре дефиниран стек от продукти за разработка на SOA приложения. В това число се включват:

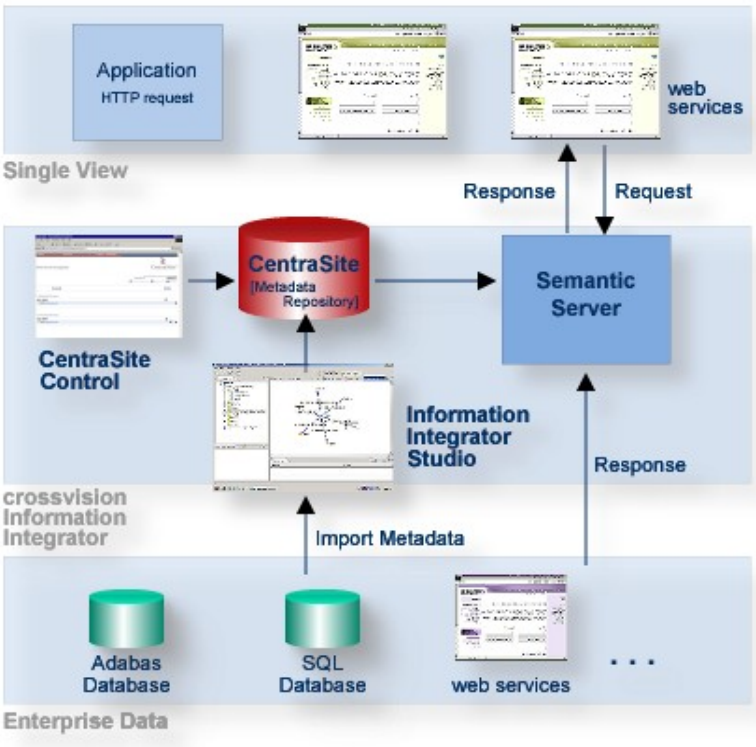
- Application Composer – предназначен за разработка на уеб-базиран потребителски интерфейс;
- Business Process Manager – използва се за създаване и изпълнение на бизнес процеси;
- Service Orchestrator – представлява ESB решение;
- Information Integrator – технология за интеграция на данни от различни източници в един общ домейн модел;
- Legacy Integrator – позволява използването на съществуващи системи посредством набор от адаптери;
- CentraSite – представлява централно хранилище (repository) за метаданни, свързани с разработваното SOA приложение;
- Tamino – XML сървър, даващ възможност за

XQuery-базиран достъп до XML форматирани данни.

От разгледаните технологии най-голям интерес за целите на дипломната работа представляват Infromation Integrator и Service Orchestrator.

3.4.1 Information integrator

Архитектурата на Information Integrator [27] е представена на фигура 21. С помощта на Infromation Integration Studio се осъществява зареждане на метаданни от различни източници (бази от данни, уеб услуги и др.) и съхранението им в CentraSite. Също с помощта на студиото се изгражда новия модел на данните (домейн модела), който е базиран на онтологии и се съхранява в CentraSite. Достъпът до новия модел става посредством семантичния сървър (semantic server), чрез изпълнението на различни клиентски заявки (queries).



Фигура 21: Архитектура на Information Integrator

Следните понятия се използват за дефиниране на домейн модели:

- Проект (project) – представлява колекция от онтологии, правила и заявки;
- Онтология (Ontology) – дефинира пространство от имена за класове, свойства, правила и заявки;
- Клас (Class) – представлява информационната единица на онтологията – представя понятията от домейна;
- Обектни свойства (Object properties) – това са свойства на даден клас, които представят връзката му с друг клас;
- Свойства за данни (Data properties) – свойствата на даден клас;
- Правила (Rules) – описват връзки между класовете на различните онтологии;
- Заявки (Queries) – използват се за извличане на информация от онтологиите.

За дефиниране на правила и заявки се използва декларативния език F-logic. Повече информация за този език може да се намери в [28]. Важно е да се спомене, също така, че е създаден език, представляващ комбинация между F-logic и XQuery, наречен XCerpt, чиято идея е да направи XQuery по-декларативен [29].

Information Integrator Studio предлага потребителски интерфейс, позволяващ зареждането на схеми от бази данни, WSDL, XSD и др. Също така се дава възможност за

дефиниране на онтологии, класове и връзки между класовете. Има редактори позволяващи графичното създаване на правила и заявки дори от потребители, които не разбират F-logic.

3.4.2 Service Orchestrator

Service Orchestrator представлява ESB решение на Software AG. При него обработката на съобщенията, минаващи през ESB се определя в зависимост от конфигурационен файл, наречен документ на последователността (sequence document). В този документ могат да се дефинират различни стъпки на обработка, през които може да мине съобщението.

Във връзка с трансформацията на съобщението две стъпки могат да се дефинират, като и при двете се използва XSLT като език за трансформация. Тези стъпки са:

- Междинна трансформация – при нея се изпълнява XSLT шаблон върху съобщението, като този шаблон може да приема входни параметри. Няколко трансформации могат да се свържат верижно;
- Крайна трансформация – при нея се изпълнява пак XSLT трансформация, но резултатното съобщение е предназначено за предаване извън рамките на Service Orchestrator.

3.5 Microsoft BizTalk сървър

Microsoft BizTalk сървър представлява платформа за интеграция на процеси (process integration platform), която дава възможност за дефиниране и изпълнение на бизнес процеси. В рамките на тези бизнес процеси се налага

адаптиране на интерфейсите на услугите, които биват извиквани, така че те да могат да се използват в рамките на процеса. За целта се използва BizTalk mapper, представляващ средство, с което могат да се описват графично трансформации (mappings) между различни типове от данни и след това тези трансформации да се изпълняват [30].

За описването на трансформации се използва графичен редактор, който позволява създаването на сложни преобразувания, в които могат да участват различни функции, част от които вградени, а друга част – дефинирани от потребителя.

След като е описана трансформацията от графичния редактор тя се компилира, като при това се извършва преобразуване на формата, в който се съхранява трансформацията по време на дизайн към XSLT, който се използва по време на изпълнение. При тази компилация се извършва и верификация на трансформацията, като при наличие на проблеми се показват съобщения за грешки.

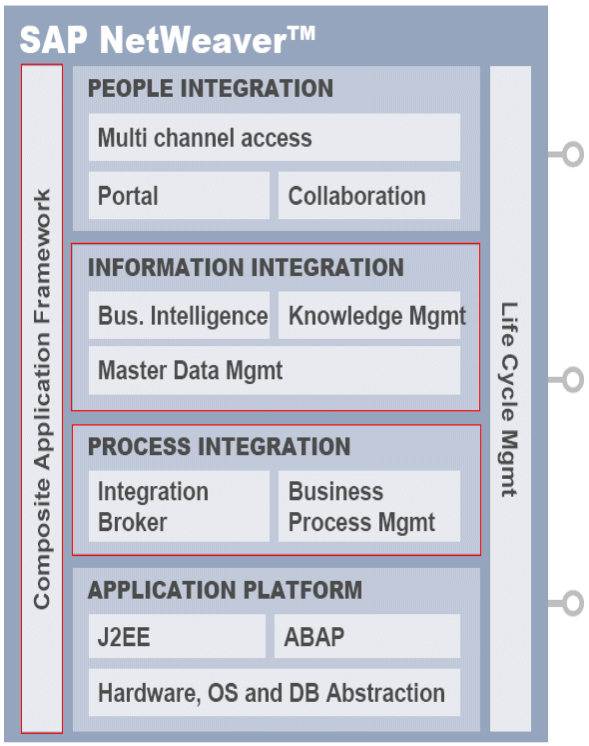
Недостатък на този редактор е, че не поддържа обратното преобразуване: от готова XSLT трансформация да изгради нейното графично представяне.

3.6 Интеграция на данни и процеси в SAP

На фигура 22 са показани елементите изграждащи SAP NetWeaver платформата, която е предназначена за създаване на SOA приложения. В основата седи платформа за изграждане на приложения (Application Platform), която е базирана както на JEE, така и на ABAP. За интегрирането на

различни услуги и създаването на бизнес процеси, отговарящи на изискванията на потребителите се използва платформата за интегриране на процеси (process integration). За обединение на информацията от различни източници на данни в един общ модел се използва платформата за интеграция на информация (Information Integration). На най-високо ниво в стека стоят платформите за обслужване на потребителя посредством създаване на графичен интерфейс (people integration).

Заедно с описаните елементи на NetWeaver стека на SAP, се включва и средството за композиране на услуги – Composite Application Framework, както и система за мониторинг и мениджмънт на приложенията – Life Cycle Management.



Фигура 22: SAP NetWeaver платформа

За интеграция на информация се използва приложение

наречено Business Intelligence (BI) [31]. Неговите цели са:

- Предоставяне на унифициран модел на данните;
- Доставка на данни от различни източници;
- Трансформация на данните;
- Изпълнение на заявки спрямо данните.

Може да се отбележи, че за извършване на гореописаните операции се използват вътрешно фирмени стандарти. Това прави технологията по-трудна за разбиране и използване от потребителите.

Има създадени графични редактори за дефиниране на трансформации и заявки. Резултатите се записват в специален формат, който се използва в последствие при изпълнение.

За интеграция на услуги и дефиниране на процеси, SAP са създали платформа, наподобяваща много JBI като концепция. Тази платформа се нарича Exchange Infrastructure (SAP XI) [32] и позволява маршрутизиране и трансформация на съобщения, както и интеграция с различни по тип източници на данни като бази от данни, уеб услуги, RFC (това е специфичен за SAP формат за извикване на ABAP модули) и др.

За реализацията на трансформации се използва специален редактор, чрез който се задава как трябва да се преобразуват, съответно входящото (input) съобщение, изходящото (output) съобщение и съобщението за грешка (fault). След като се опишат графично трансформациите, от тях се генерира Java код, който се използва директно в

средата за изпълнение. Предимство на решението с генериран код е бързината, с която се изпълнява трансформацията. Платформата, която изпълнява трансформации е организирана така, че входните данни влизат като поток и изходящите данни излизат като поток, което позволява обработката на големи количества информация с малко памет. Правени са успешни експерименти за обработка на данни с размер до 5 GB.

Освен Java генериран код, за трансформиране може да се използва и XSLT, но за него няма вграден графичен редактор, като изпълнението на трансформацията е доста по-бавно.

3.7 Интеграция на данни и процеси в Oracle

Oracle предлагат множество продукти свързани с изработката на SOA базирано приложение. В основата на техния стек стои Oracle базата от данни, която предлага пълен XQuery интерфейс, което я прави особено популярна за интеграционни сценарии, базирани на XQuery. Множество фирми, между които BEA и DataDirect предлагат адаптери за интеграция с Oracle.

Освен базата данни, Oracle предлагат и няколко продукта, които са от интерес за проучването в настоящата глава. Това са:

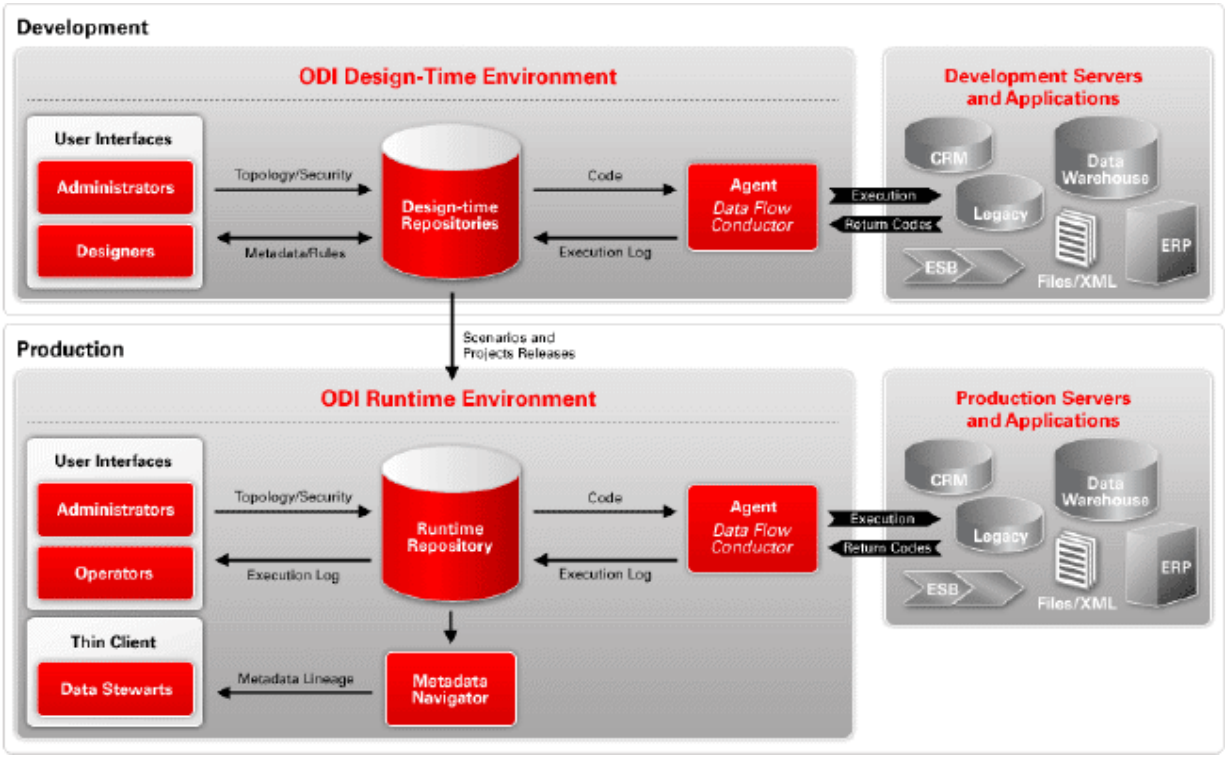
- Oracle Data Integrator
- Oracle ESB
- Oracle XML Query Service

Тези продукти ще бъдат разгледани в следващите

точки.

3.7.1 Oracle Data Integrator (ODI)

Oracle Data Integrator се използва за извличане на данни от различни входни източници с различни структури, трансформиране на данните в друг формат и записването им в изходен източник [33]. На фигура 23 е показана архитектурата на интегратора.



Фигура 23: Архитектура на ODI

Архитектурата е разделена на две части:

- среда за ипълнние на ODI (ODI Design-Time Environment). В основата и седи хранилище (Designtime repositories), което съхранява схемите на входящите и изходящите източници на данни и трансформациите. Работата с хранилището се извършва посредством приложенията Topology Manager (Adminstrator) и Designer, които ще бъдат разгледани в последствие.

Самата интеграция се изпълнява от т.нар. агенти (Agents);

- среда за изпълнение на ODI (ODI Runtime Environment). Разликата при средата за изпълнение се състои в това, че достъпа до хранилището е само за четене и се осъществява от т.нар. Data Stewarts, които се свързват с уеб-базирания Metadata Navigator. Освен това има клиент за разглеждане и управление на задачи за интеграция (runtime integration jobs).

Приложението Topology Manager служи за дефиниране на информация за свързване с различни източници на данни. За целта е необходимо да се дефинират т.нар. сървъри за данни (data server) – по един за всеки източник на данни, който участва. Тук източника на данни може да е входящ, изходящ или служебен, който се използва от интегратора. Поддържат се различни типове от източници като бази от данни, файлове, уеб услуги и др. Връзката със съответния източник трябва да е JDBC-базирана тъй като SQL е езика който се използва за описване на интеграцията.

След като са дефинирани източници и физическата връзка с тях се използва дизайнера (Designer). Неговата работа се базира на съвкупност от модули за знания (knowledge modules) основните измежду които са:

- Reverse-engineering knowledge modules – служат за извличане на модела на данните от различни източници;
- Journalizing knowledge modules – водят статистика за това какви данни са се променили във входящите

източници. Използват се за да се ускори интеграцията като се извършват само тези промени в изходните източници, които отговарят на променените данни;

- Loading knowledge modules – използват се за ефикасно извличане на данни от входящите източници;
- Check knowledge modules – използват се за проверка за валидност на входящите данни спрямо някакви зададени правила;
- Integration knowledge modules – извършват трансформация на данните и записа им в целевите източници;
- Service knowledge modules – дават възможност за представяне на данните под формата на уеб услуги.

Примерен процес за използване на дизайнера може да се опише по следния начин:

- Извличане на метаданни от входящи и изходящи таблици посредством reverse-engineering knowledge module;
- Ръчно описване на метаданни от входящ CSV файл, например. Ръчното описване става, чрез вграден в дизайнера редактор;
- Задаване на слушалка за промяна на входните данни посредством Journalizing knowledge module и регистриране на процедура, която да се изпълни при тази промяна;
- Описване на процедурата, като се задават трансформации между входните и изходните източници

с използване на езика SQL. Тези трансформации в термините на ODI се наричат интерфейси;

- Описване на процедура за първоначална трансформация.

3.7.2 Oracle ESB

В Oracle ESB [34], както и във всички други ESB имплементации се налага да се дефинират трансформации на съобщенията, които се предават между услугите. Освен това се дава възможност за интеграция с източници на данни, например – прочитане от/записване в XML или обикновен файл.

За реализиране на тази цел се използват XSLT трансформации като има графичен редактор за тяхното дефиниране. Чрез този редактор може да се визуализира произволен XSLT документ, независимо дали е създаден ръчно или не. Редакторът предоставя и възможност за автоматично свързване на елементите на входните и изходните данни (mapping) на базата на имената им или на техния тип.

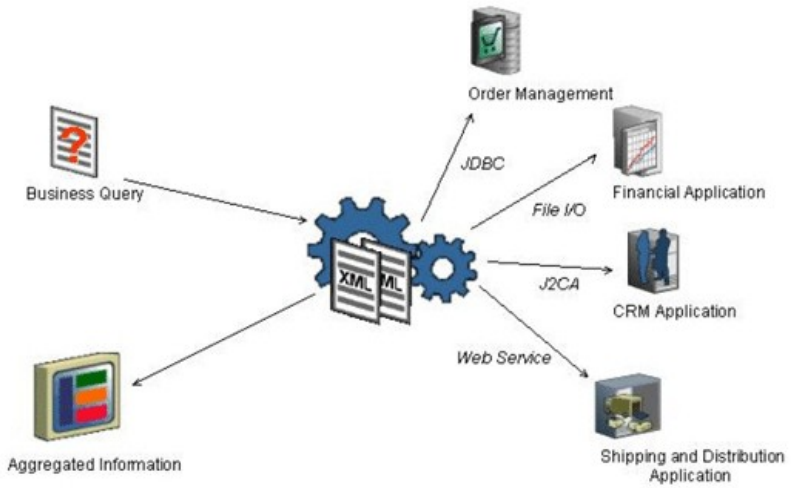
3.7.3 Oracle XML Query Service (Oracle XQS)

Oracle XML Query Service (преди наричана Oracle XML Data Synthesis) [35] е част от Oracle Application Server (JEE платформата на Oracle) и се използва за XQuery-базирана интеграция на данни от различни източници. На фигура 24 е показано схематично множеството от източници, които могат да участват в интеграцията.

Няма графични редактори за описване на интеграцията, а всичко се извършва ръчно чрез описания в

текстови файлове. Основните файлове свързани с интеграцията са:

- xds-config.xml файла, който се поставя в директория META-INF/xds/config/ в архива на приложението. Този файл описва използваните източници на данни и връзката с тях;
- XQuery файл, описващ самата трансформация.



Фигура 24: Интеграция чрез Oracle XQS

Изпълнението XQuery заявката може да се извика по три начина:

- чрез клиентския програмен интерфейс (Java client API);
- от JSP като библиотека от тагове (tag library);
- от сесийни бийнове (Session EJB).

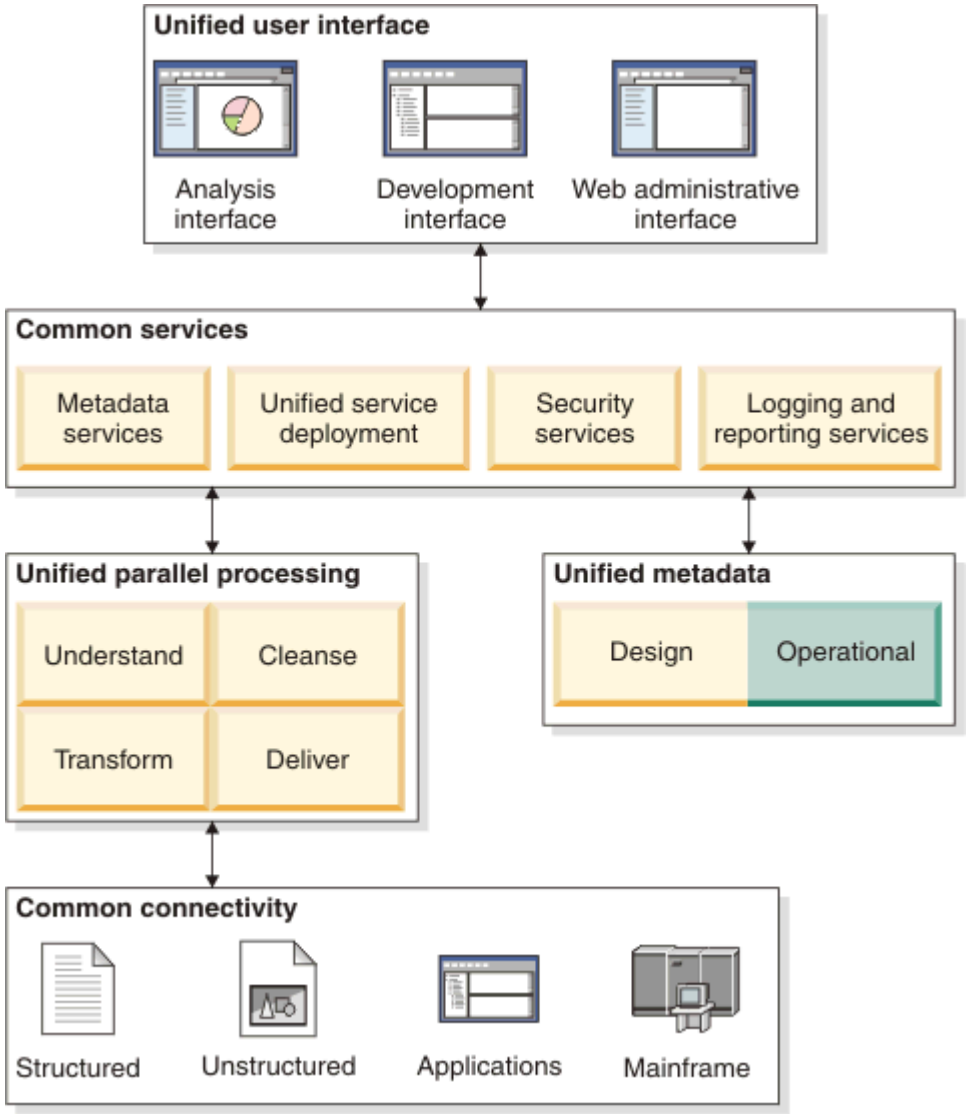
3.8 Интеграция на данни и процеси в IBM

IBM предлагат голямо разнообразие от продукти, свързани с интеграцията. Част от тях се застъпват като

функционалност, друга част представляват усъвършенствана версия на по-предишни продукти. В следващите точки те ще бъдат разгледани в подробности.

3.8.1 IBM Information Server

IBM Information Server [36] е решението за интеграция на данни на IBM. То е базирано на принципа на федерация, при който се създава виртуален образ на данните от различни източници. На фигура 25 са показани компонентите, които изграждат продукта.



Фигура 25: Компоненти на IBM Information Server

Първата група от компоненти, които извършват основната работа в сървъра са обединени от т.нар. Unified parallel processing двигател. Този двигател дава възможност за паралелно изпълнение на потоци от данни (data flow). Един поток от данни може да има различни стъпки на обработка на тези данни, като всяка от стъпките може да е свързана с: анализ на данни (Understand), семантично преобразуване на данни в друг модел (Cleanse), извършване на трансформация на данните (Transform) и изпълнение на федерирана заявка към данните (Deliver). За всяка една от тези стъпки съществува продукт, който изпълнява съответната задача:

- WebSphere Information Analyser – позволява извършване на анализ върху входните за интеграционния процес данни и дава възможност за установяване на метрики определящи тяхното качество;
- WebSphere QualityStage – дава възможност за премоделиране и реинженеринг на данните, като по този начин се повишава качеството на информацията. QualityStage продукта е тясно свързан с DataStage;
- WebSphere DataStage – използва се за извършване на комплексни трансформации на данните;
- WebSphere Federation Server – изпълнява задачите по виртуализация на данните постъпващи от различни източници, като достъпа до тях се извършва посредством федерирани заявки, базирани на езика SQL. Заедно с Federation Server-а, за федерация на

данни се използват: Federation Server for z/OS, предназначен за интеграция с IBM мейнфрейми и WebSphere Information Integrator Content Edition специализиран за интеграция на информация от обикновени файлови източници.

Информацията необходима за работата на Information Server-а, както за всичките му компоненти се съхранява в общо хранилище (Unified Metadata), което е достъпно за всички под формата на услуга (Metadata Service). В това хранилище се съхраняват два типа метаданни:

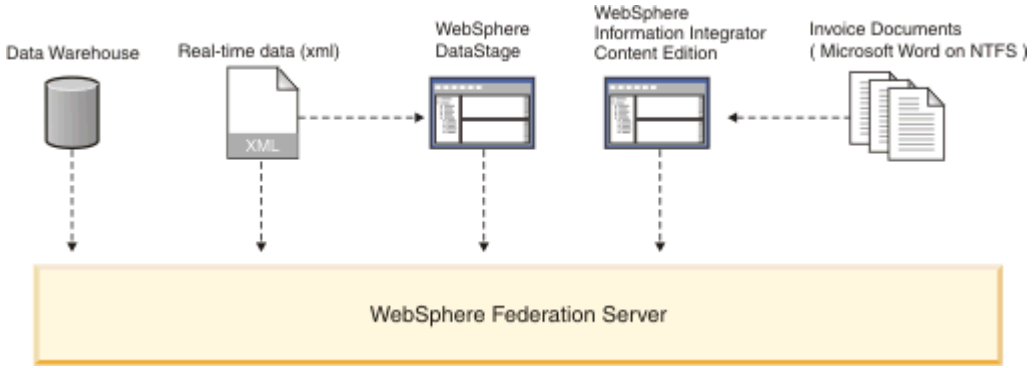
- Такива свързани с дизайна на интеграционното приложение (Design) – това включва схеми на входни източници на данни и на изходни структури, трансформации и др;
- Такива свързани с работата на приложението по време на изпълнение (Operational) – включват мониторинг на производителността, профилиране, писане на служебни съобщения и др.

За достъп до различните източници на данни е създадена обща основа за връзка (common connectivity), в която могат да се добавят произволен брой компоненти осъществяващи достъпа (connectors). Тези компоненти отговарят за това да доставят метадани и данни от съответния източник, както и да осигурят превеждане на федерираните SQL заявки към такива, които да са разбираеми за съответния източник на данни.

В IBM Information Server служебната функционалност съществува под формата на услуги (Common Services). В

тези услуги се включват такива, осигуряващи достъп до метаданните (metadata services), даващи обща инфраструктура за logging и reporting, услуги свързани със сигурността, като аутентикация и оторизация и услуга, позволяваща добавянето на нови услуги, които са свързани с разработеното приложение (Unified service deployment). Тези специфични за приложението услуги се създават с помощта на WebSphere Information Services Director, с чиято помощ задачи като анализа на данни, трансформацията на данни и изпълнението на федерирани заявки може да се представят като услуги.

За да се илюстрира работата на IBM Information Server и изграждащите го компоненти на фигура 26 е показан пример за архитектура на приложение, чиято цел е да верифицира верността на дадена фактура (invoice document) спрямо текущата информация за доставка пристигаща под формата на XML (xml real time data) и данни за историята на доставките поместени в склад за данни (data warehouse).

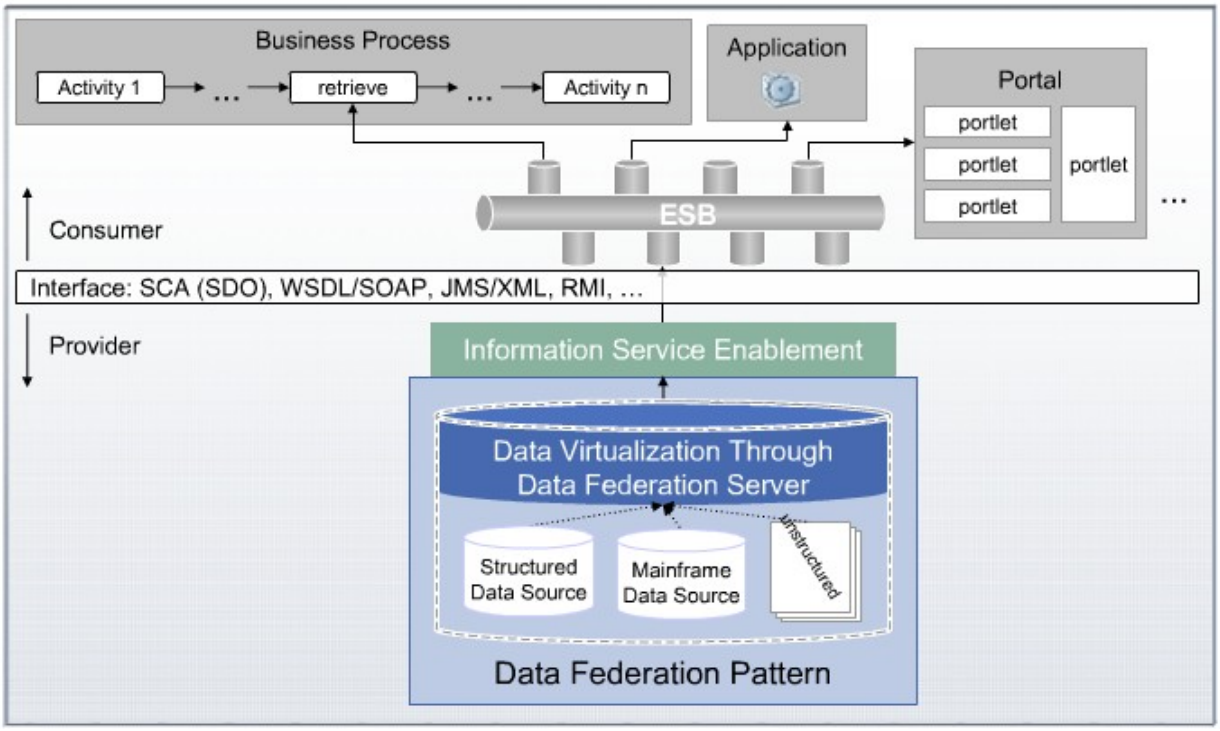


Фигура 26: Приложение работещо с IBM Information Server

Както се вижда от фигурата се използват следните компоненти на Information Server-а:

- WebSphere DataStage – за извикване на задача предназначена за трансформирането на данните в постъпващия XML файл с информация за доставката. Използва се същата задача, чрез която се попълва склада за данни;
- WebSphere Information Integrator Content Edition – за разчитане на документа на фактурата;
- WebSphere Federation Server – за създаване на общ SQL-базиран изглед на цялата информация.

Както беше споменато по-горе за да може цялата тази функционалност, предлагана от Information Server-а да стане достъпна за SOA приложения се използва WebSphere Information Services Director, който позволява създаването на услуги, извикващи компонентите на сървъра. Например една услуга би могла да изпълни DataStage задача. Друга услуга може да извика федерирана SQL заявка и т.н. Освен това, създадените с директора услуги могат да се използват по различен начин в зависимост от тяхната свързаност (SCA, WSDL, JMS, RMI). На фигура 27 е показано как може Information Server-а да се интегрира с ESB посредством услуги.

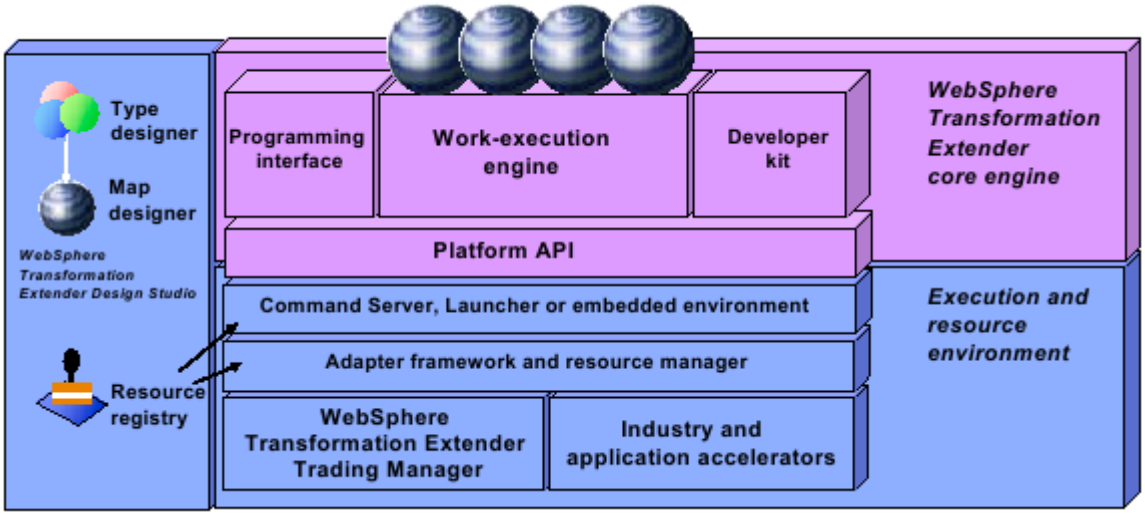


Фигура 27: SOA интеграция на IBM Information Server

3.8.2 IBM WebSphere Transformation Extender

IBM WebSphere Transformation Extender [37] е средство за интеграция на данни от различни източници. Той е разработен на базата на WebSphere DataStage и представлява самостоятелен продукт, включващ, както среда за разработка, така и за изпълнение на интеграционни задачи. На фигура 28 са показани изграждащите го компоненти.

В основата на продукта седи двигателя за изпълнение на задачи (Work-execution engine). Той изпълнява зададените му поредици от трансформации като достъпа до него се осъществява посредством програмният интерфейс (Programming Interface) и платформения интерфейс (Platform API).



Фигура 28: Архитектура на Transformation Extender

В допълнение към двигателя се включват:

- т.нар. акселератори (Industry and Application accelerators), които спомагат свързването към специфичен за дадена индустрия формат на данните или пък към специфични приложения като SAP, Siebel и др.;

- с помощта на рамка от адаптери (adapter framework) се осъществява връзка между адаптерите към различни приложения и двигателя за трансформации. С помощта на ресурс мениджъра (resource manager) се осъществява съответствие между логически имена на ресурси и техните физически местоположения;

- команден сървър (command server) – дава възможност за стартиране на трансформации от командния ред;

- стартер (launcher) – стартира трансформации при настъпването на определени събития (промяна на

данните или на файловете и др.);

- Търговски мениджър (Trading manager) – позволява B2B интеграция между предприятията.

По време проектиране се използва WebSphere Transformation Extender Design Studio. Основните компоненти, чрез които се реализират трансформационните приложения са:

- Дизайнер на типове (Type Designer) – чрез него се създават типовете на данните, които участват в трансформациите, като тези типове се наричат бизнес обекти (business objects). Бизнес обектите съответстват на XSD дефиниции;

- Дизайнер на интерфейси за бази от данни (Database interface designer) – дава възможност за зареждане на метаданни от база от данни;

- Дизайнер на връзки (map designer) [38] – дава възможност за определяне на връзките между входни и изходни интерфейси и дефиниране на трансформации между типовете. Дизайнера на връзки използва собствен формат за записване на таблиците на връзките във файл с разширение .mms. В последствие тези таблици се компилират във формат .mmc, след което могат да се изпълняват и да се проследяват за грешки (debug). Към всяка връзка, която е дефинирана чрез дизайнера може да се прикрепи трансформация. Всяка трансформация са записва под формата на генериран Java код;

- дизайнер на интеграционния поток (Integration flow designer) – чрез него се описват последователности от таблици на връзки, чрез които се осъществява пълната трансформация от входа към изхода.

3.8.3 WebSphere Business Integration Server

WebSphere Business Integration Server [39] представлява решение за интеграция на приложения (EAI), базиращо се на следните основни компоненти:

- InterChange Server and Toolset – дава възможност за реализиране на бизнес процеси и тяхното изпълнение. Осигурява комуникацията на тези процеси с различни услуги и приложения;
- WebSphere Business Integration Adapters – осигуряват достъп на приложенията, изпълняващи се от InterChange сървъра, до различни типове ресурси и приложения;
- WebSphere Business Integration Message Broker – осигурява маршрутизиране и трансформация на съобщения между различни приложения. Трансформацията при него се осъществява посредством разширена версия на езика SQL – ESQL;
- WebSphere Studio – предоставя среда за разработка на EAI приложения.

В InterChange сървъра бизнес процесите се дефинират с понятието колаборация (collaboration). Информацията между колаборацията и услугите и приложенията, подобно на WebSphere Transformation Extender се обменя под формата на бизнес обекти. Там където се налага

трансформация и създаване на таблици на връзките (maps) се използва същият Map Designer, както в Extender-а.

3.8.4 WebSphere Process Server

WebSphere Process Server представлява съвременния вариант на WebSphere Business Integration Server. В него са заложени последните стандарти в областта на SOA като ESB, SCA, BPEL и др.

Като основна технология за реализация на SOA се използва SCA. Все още съществуват понятия като бизнес обекти и интерфейси на услуги, но те вече са поставени в контекста на SCA. Следните основни продукти изграждат WebSphere Process Server-а:

- WebSphere Process Server – осигурява изпълнението на BPEL- базирани бизнес процеси;
- WebSphere Enterprise Service Bus – осигурява маршрутизиране и трансформация на съобщения;
- WebSphere Adapters – подобно на адаптерите от WebSphere Business Integration Server осигуряват връзка с външни приложения и източници на данни;
- WebSphere Integration Developer – осигурява среда за разработка на SCA – базирани SOA приложения.

В WebSphere Process Server дефиниции на таблици с връзки се създават посредством редактор, който е част от продукта Rational Application Developer, който пък от своя страна е част от WebSphere Integration Developer. Този редактор позволява дефиниране на следните видове трансформации:

- Между релационна база данни и XML. При това се поддържат следните видове връзки
  - Между дефиниция на таблица от базата данни и схемата на XML формата. Тогава информацията от тази таблица ще се представи в необходимия XML формат;
  - Между SQL заявка и XML. В този случай резултата от дадената SQL заявка ще бъде форматиран като XML с необходимата схема.

При този тип трансформации като резултат се генерира файл с формат DAD, който може да се изпълнява на DB2 XML Extender и осигурява преобразуването на XML съдържание в такова за базата данни и обратното.

- Между два XML формата, зададени със съответните им схеми. На базата на трансформацията се генерира XSLT трансформация, която може да се публикува на сървъра.

Съществуват няколко вида таблици на връзки, като всяка една от тях може да се инсталира на съответния контейнер като SCA компонент, който е част от SCA композита на приложението:

- връзка между два интерфейса – инсталира се върху WebSphere Process Server-a;
- връзка и трансформация между два бизнес обекта - инсталира се върху WebSphere Process Server-a;
- медиаторна трансформация – XSLT трансформация на съобщение в рамките на WebSphere Enterprise

Service Bus. При дефинирането и могат да се използват външни функции написани на Java и по този начин да се настройва медиаторната логика.

3.9 Интеграция на данни и процеси в IONA

IONA разработва два продукта свързани със SOA. Единият се казва Artix и е представлява комерсиален продукт на компанията. Той включва слените компоненти:

- Artix ESB – Enterprise Service Bus решение;
- Artix Registry/Repository – хранилище за метаданните на различни услуги;
- Artix Orchestration – средство за създаване и изпълнение на BPEL бизнес процеси;
- Artix Data Service – решение за интеграция на данни;
- Artix Mainframe – средство за интеграция на legacy системи;
- SOA Management – дава възможност за управление и мониторинг на SOA приложения;

Другият продукт се казва FUSE и представлява колекция от продукти с отворен код на организацията Apache, които се предлагат като интегрирано решение от IONA. В тези продукти се включват:

- Fuse ESB – базиран на Apache ServiceMix – представлява ESB решение, отговарящо на JBI спецификацията;
- Fuse message Broker – базиран на Apache ActiveMQ

– предлага среда за разпространение на съобщение на базата на различни протоколи;

- Fuse Service Framework – базиран на Apache CXF – предлага среда за разработка и изпълнение на уеб услуги;

- Fuse Mediation Router – базиран на Apache Camel – дава възможност за маршрутизиране и трансформация на съобщения между услугите.

В следващите точки ще бъдат разгледани част от тези продукти и тяхната връзка с интеграцията на данни и интерфейси.

3.9.1 Artix Data Service

Artix Data Services [40] е решение за интеграция на данни от различни източници и изграждане на нов модел на тези данни, който е специфичен за домейна на разработваното SOA приложение. Следните модули се включват в Artix Data Services:

- Artix Data Services Designer – предоставя възможност за графично моделиране на услуги за данни и дефиниране на трансформации, които след фаза на компилация се превеждат към високо ефективен Java код. Поддържа работа в конкурентна среда, както и средство за сливане на промени от различни потребители по даден модел (merge tool);

- Metadata Management – дава възможност за зареждане на метаданни то различни формати като XSD, XML Instance, обикновени файлове (напр. в CSV формат), интроспектиране на Java класове, схема на

таблици и заявки към бази от данни;

- Semantic Constraints and Validation Rules – позволява дефинирането на зависимости и условия между данните чрез средствата на XPath. Също така дава възможност за писане на валидации на Java;

- Artix Data Services API – представлява йерархия от Java класове, генерирана на базата на дефинираните в Artix Data Services Designer модели. Именно тази йерархия реализира услугите за данни (data services). За достъп до релационни бази данни тези услуги използват Hibernate. Клиента може да работи с йерархията по някой от следните начини: използвайки DOM или SAX, за да чете данните изразяващи се от услугите за данни; да използва XPath и XQuery за изпълнението на заявки върху данните изразяващи се от тези услуги; да използва XSLT за дефиниране на трансформации между различни модели;

- Artix Data Services Standards Libraries – представлява колекция от предефинирани модели на услуги за данни, които са характерни за дадени индустрии като SWIFT, SEPA, FpML и др.;

- Artix Data Services Reference Implementations – представлява набор от примери за използване на Artix Data Services.

3.9.2 Artix Enterprice Service Bus

Artix Enterprice Service Bus [41] се използва за интеграция на приложения, работещи с различни протоколи на пренос на информация и тип на данните.

За осъществяване на медиация в Artix ESB се използва специална услуга, изпълняваща XSLT трансформации. Чрез тези трансформации може да се извършва както медиация на предавани параметри между операции, така и трансформация на съобщения, обменящи се между услугите. Трансформацията се дефинира на базата на входен и изходен интерфейс, дефинирани с WSDL.

3.9.3 Apache Camel

Apache Camel [42] е проект с отворен код, целящ реализацията на средство за маршрутизиране и трансформация на съобщения. Основава се на език Java DSL, с чиято помощ се могат да се описват пътища на преминаване на дадено съобщение и фази на неговата трансформация.

За описване на пътищата се използват крайни точки (end points), от които тръгва и пристига съобщението. Всяка крайна точка представлява дефиниция на компонент и на URI. Apache Camel поддържа различни компоненти в зависимост от типовете крайни точки. Примери за такива компоненти са RMI, JPA (за достъп до бази от данни), Mail, Queue, File и др.

За описване на трансформации и въобще на всякакви предикати и изрази може да се използва множество от езици, между които SQL, XPath, XQuery, JavaScript и др. Трансформации могат да се извършват и с помощта на Java класове. Следващия код показва пример за това какво представлява едно Java DSL описание:

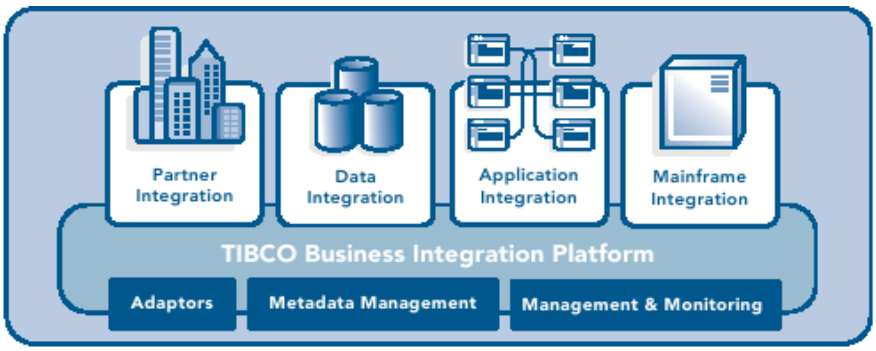
```
public class EtlRoutes extends SpringRouteBuilder {
    public void configure() throws Exception {
        from("file:src/data?noop=true").convertBodyTo(PersonDocument.class)
        //  .intercept(transactionInterceptor())
            .to("jpa:org.apache.camel.example.etl.CustomerEntity");

        // the following will dump the database to files
        from("jpa:org.apache.camel.example.etl.CustomerEntity?consumeDelete=
false?consumer.delay=3000&consumeLockEntity=false")
            .setHeader(FileComponent.HEADER_FILE_NAME,
el("${in.body.userName}.xml"))
            .to("file:target/customers?append=false");
    }
}
```

3.10 Обединена платформа за интеграция на TIBCO

Tibco Business Integration Platform [43] представлява единно решение на Tibco, за интеграция както на процеси, така и на данни. На фигура 29 са показани основните компоненти, които изграждат платформата.

В основата на интеграцията на процеси и приложения е софтуера Tibco BusinessWorks, който представлява ESB. Чрез него се извършва маршрутизиране и трансформация на съобщения между различни приложение, като се поддържат различни среди за разпространението на тези съобщения (Tibco Rendezvous, JMS и др.). За трансформация се използва XSLT, като има създаден графичен редактор на връзки между интерфейсите и структурите. Този редактор е част от средата за разработка BusinessWorks ISE (Integrated Services Environment).



Фигура 29: Платформа за интеграция на Tibco

В допълнение към BusinessWorks се предлага и BusinessWorks SmartMapper, предназначен за изграждане на услуги, които извършват трансформация на данни по определени правила между различните приложения. За дефинирането на правила и трансформации е разработен графичен интерфейс. Дефиницията на структури от данни се базира на XSD, а трансформациите на XSLT и XPath.

За интеграция на данни от различни източници, Tibco предлага ETL (extract-transform-load) решение, наречено Tibco DataExchange. При него се използва подход, при който се моделира домейна, който ще се използва от SOA приложението (Model Driven Approach – MDA). След това се дефинират интеграционни процеси и трансформации на данните. Интеграционите процеси могат да се стартират както при наличие на някакво събитие, така и като регулярна задача (batch job). За осигуряване на консистентност на данните се използва централно хранилище, което съдържа цялата моделна информация. За дефиниране на трансформации на данните, обединения на таблици (joins), сливания (merges) и др. се използват езиците SQL, Java и JavaScript.

3.11 Заключителен анализ

В тази, последна точка от главата посветена на интеграция на данни и медиация между услуги ще бъде направен сравнителен анализ на използваните технологии между различните производители.

В таблица 1 в приложение 1 е представено резюме на средствата за интеграция на данните при различните фирми. От него се вижда, че основно се използват два метода за интеграция: федерация и ETL, като повечето фирми предлагат федерация или и двете. Само Tibco и SAP нямат интеграция базирана на федерацията на данни. По принцип федерацията има съществоното предимство пред ETL, че не предизвиква репликация на данните.

Като цяло може да се отбележи, че основно се използват езиците SQL и XQuery, за реализиране на интеграцията, като част от фирмите използват други езици – Software AG използват F-logic, SAP – използват вътрешно фирмен език, а IONA – Java и XSLT. Това е нормално тъй като за интеграция на данни най-подходящ е език за заявки (queries). Използването на SQL преобладава пред използването на XQuery, като има два фактора обуславящи този факт:

- В ETL интеграцията е нормално да се използва SQL, тъй като се осъществява достъп само до релационни бази от данни;
- SQL е по-утвърден от XQuery (версия 1.0 на езика беше завършена 23 януари 2007).

От направеното разглеждане става ясно, че XQuery

представлява много удобно средство за федерация на данни, като предимството му пред SQL се изразява в по-свободния му и богат синтаксис. Освен това XQuery по своята същност дава възможност за директна работа с XML базирани източници на данни като уеб услуги, XML файлове и XML-бази от данни. Като се има предвид, че повечето големи доставчици на бази данни поддържат XML интерфейс и директна, високопроизводителна обработка на XQuery заявки, постепенно XQuery ще заеме централно място в средствата за федерация на данни.

В таблица 2 в приложение 1 е представено резюме на средствата за медиация между услуги. От анализа се вижда, че медиацията се използва основно в ESB решенията, където се извършва маршрутизиране и трансформация на съобщения. Почти монополист в реализацията на медиацията е езика XSLT. Той осигурява лесен и удобен начин за трансформация на XML документи каквито се обменят между услугите. Разбира се за целта би могъл да се използва и езика XQuery (както правят BEA), но той е по-неудобен и по-трудно приложим когато трябва да се трансформира XML документ от един формат в друг.

Въпреки предимствата си, обаче, XSLT има проблеми с производителността, затова някои фирми (SAP, IBM) предоставят генерация на Java код, който извършва трансформацията, което предполага по-добро бързодействие.

Като цяло може да се заключи, че за медиацията на данни най-подходящо е използването на генериран Java код

ако е необходимо да се обработват големи обеми от информация. Ако пък целта да се предостави решение, което предлага по-голямо удобство за разработчика и което е базирано на стандарти – може да се използва XQuery или XSLT. Във връзка с това може да се отбележи, че в бъдеще идеята за XQuery имплементационен тип за SCA може да се доразвие и в добавянето на XSLT тип, който да се използва само за нуждите на медиация на съобщения (но не и на федерация на данни, поради проблемите с производителността).

4 Глава 4. Спецификация на XQuery имплементационен тип за SCA

4.1 Въведение

Тази спецификация представлява разширение на SCA асембли модела (SCA Assembly Model [44]) с дефиниция за това как XQuery скрипт може да предостави имплементация на SCA компонент и как може да се използва в SCA като имплементационен тип.

4.2 XQuery имплементационен тип

В тази секция се описва как един XQuery скрипт може да предостави имплементация на SCA компонент, като определя неговите услуги, референции и входни точки за конфигуриране.

4.3 Услуги

Услугите, които имплементира даден XQuery скрипт се определят от дефиниции на пространства от имена в началото на файла. Чрез тези дефиниции може да се окаже, че скрипта имплементира определени Java или WSDL

интерфейси. По долу е показан пример за това как се дефинира услуга имплементираща Java интерфейс:

```
declare namespace quoteJoin="scaservice:java/xquery.quote.QuoteJoin";
```

Декларацията се състои от следните компоненти:

- “quoteJoin” - представлява префикс, който ще бъде използван при декларация на функциите, имплементиращи съответния интерфейс. По-долу е показан пример за такава функция;

```
declare function quoteJoin:joinPriceAndAvailQuotes($priceQuoteDoc,
$availQuoteDoc, $taxRate) {
```

- “scaservice” - префикс в URI-то, определящ декларация на SCA услуга;

- “java” - дефинира типа на интерфейса, който ще се имплементира. Друга възможна стойност тук е “wsdl”, например;

- “xquery.quote.QuoteJoin” - пълно име на интерфейс на услуга, който се имплементира. Ако типа на интерфейса е wsdl тук ще стои пълния URI на wsdl интерфейса.

Описаната в примера декларация има следната семантика в термините на компонентен тип:

```
<?xml version=”1.0” encoding=”ASCII”?>
<componentType xmlns=”http://www.osoa.org/xmlns/sca/0.9”>
      <service name=”QuoteJoin”>
            <interface.java interface=”xquery.quote.QuoteJoin”>
      </service>
</componentType>
```

Синтаксиса и правилата за описание на java интерфейсите на услугите са описани в [45].

4.4 Референции

За дефиниране на референции се използват, както при

услугите, декларации на пространства от имена в началото на скрипта. Формата на тези декларации е подобен на този на услугите с тази разлика, че в URI-то се използва префикс “scareference”. Пример за декларация на референция е показан по-долу:

```
declare namespace
quoteCalculator="scareference:java/xquery.quote.QuoteCalculator";
```

Освен декларацията на пространства от имена е необходимо да се декларира и променлива, на която ще бъде присвоен обект, чрез който да се извикват методите на референцията. По конвенция е възприето името на тази променлива да съвпада с името на префикса от декларацията на на пространствата от имена. В продължение на по-горния пример, тази декларация би изглеждала по следния начин:

```
declare variable $quoteCalculator external;
```

4.5 Входни точки за конфигуриране

Декларирането на входни точки за конфигуриране е аналогично на декларирането на референции. Единствената разлика е в начина на образуване на URI-то на декларацията на пространства от имена:

```
declare namespace
availQuoteDoc="scaproperty:xml/http://www.example.org/avail:availQuote";
declare namespace taxRate="scaproperty:java/java.lang.Float";
```

Примера съдържа две декларации: при първата, входната точка представлява XML-форматирано съдържание. В този случай елементите на URI-то имат следното значение:

- “scaproperty” - оказва декларация на входна точка;
- “xml” - указва, че входната точка приема данни

форматирани като XML;

- “http://www.example.org/avail” - указва пространството от имена на XML съдържанието;
- “availQuote” - указва името на основния елемент на XML съдържанието.

Втората декларация представлява пример за входна точка, приемаща java тип, като тя е аналогична на декларацията на референция с единствена разлика – името на префикса към URI-то: “scaproperty”.

За да могат да се ползват входните точки от скрипта е необходимо да се декларират и променливи, в които да се зареди тяхното съдържание. Конвенцията за тези променливи е аналогична на конвенцията за референциите и за настоящия пример е показана по-долу:

```
declare variable $availQuoteDoc external;
declare variable $taxRate external;
```

4.6 Използване на имплементационния тип

XQuery имплементационния тип се използва в рамките на SCA компонент по следния начин:

```
<component name="QuoteJoinExternalReferencesComponent">
      <implementation.xquery location="META-
INF/sca/quote_join_external_references.xq" />
      <reference name="quoteCalculator" target="QuoteCalculatorComponent" />
      <reference name="priceQuoteProvider"
target="PriceQuoteProviderComponent" />
</component>
```

За дефиниране на XQuery имплементационен тип се използва елемента “implementation.query” и се задава път до XQuery скрипта в неговия “location” атрибут.

4.7 Посоки на развитие на спецификацията

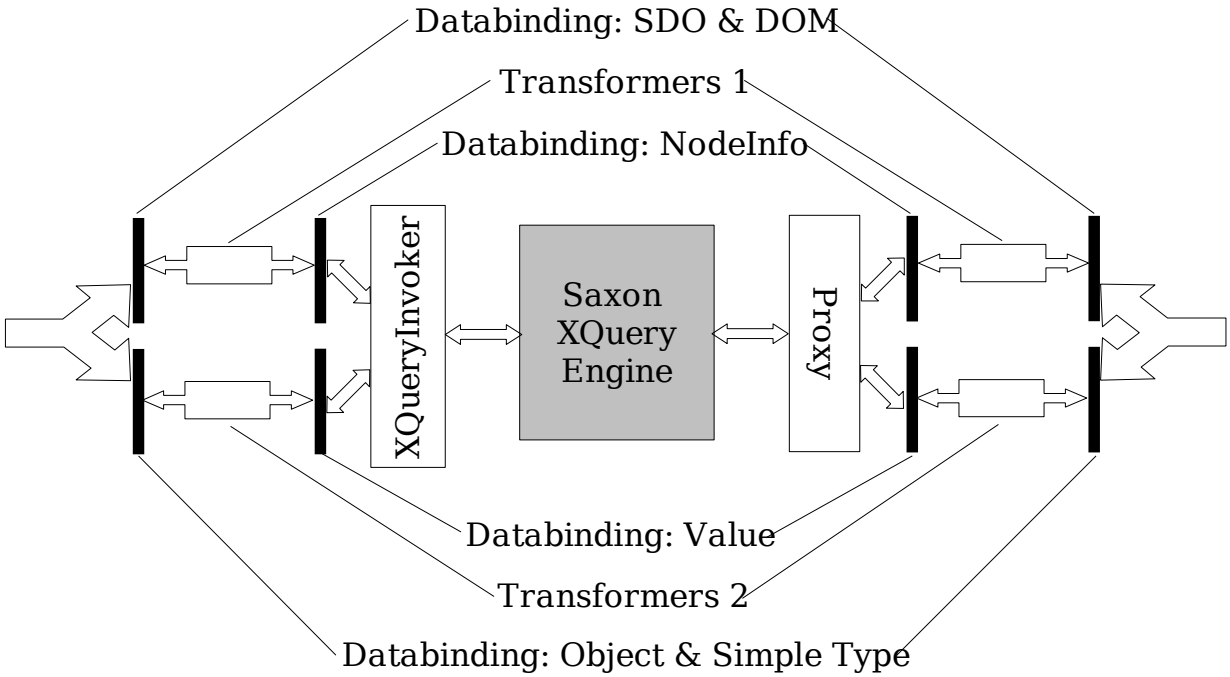
Настоящата спецификация трябва да бъде разширена в

следните посоки:

- Възможност за избор на желан (или предпочитан) XQuery процесор. Това може да се постигне с използването на policy рамката на SCA;
- Възможност за задаване на връзка към база от данни при използване на XQuery имплементационния тип, така че скрипта да може да се изпълнява директно с данни от базата. За реализация на тази задача може да се въведе нов имплементационен тип - “implementation.das”, чрез който да се осигурява SDO-базиран достъп до данните, като може да се дефинира XQuery подтип на DAS. Тази идея е предложена и развита в [46].

5 Глава 5. Приемерна имплементация на спецификацията чрез Apache Tuscany

За създаване на примерна имплементация на XQuery имплементационния тип за SCA е използван продукта на Apache Tuscany. Приложението се състои от два модула като първия (tuscany-implementation-xquery) представлява плъгин към Tuscany за нов имплементационен тип, а втория (tuscany-databinding-saxon) осигурява набор от свързаности за данни (databindings), конвертиращи типовете на данните постъпващи и излизащи от имплементационния тип в подходящ формат. Фигура 30 илюстрира съвместната работа на тези два компонента.



Фигура 30: Работа на компонент имплементиран с XQuery имплементационен тип

Всяка заявка към XQuery компонент преминава през трансформатори, които преобразуват входния тип на данните към съответен тип необходим за работата на Saxon XQuery процесора. След това тази заявка се предава към компонента XQueryInvoker, който има отговорността да изпълни съответния XQuery скрипт. След като се получи резултат от изпълнението, XQueryInvoker връща отговор, който от своя страна също минава през трансформатори, които да преобразуват Saxon типа към желания изходен тип. През време на изпълнението на скрипта може да се наложи извикване на операциите на някой от реферираните компоненти. Тогава се минава през подобен процес на трансформация, който се инициира от проксито (proxy) представляващо реферирания компонент.

Необходимите за прилагане трансформатори се определят от свързаностите за данни на типовете, стоящи

от двете страни. Ако тези свързаности са еднакви, тогава няма нужда от трансформация, но ако са различни, то се прилагат трансформатори, които да конвертират даден входящ тип към желания изходящ тип. Функцията по дефиниране на свързаностите от страната на XQuery компонента, както и на трансформатори за някои от основните типове се изпълнява от модула tuscany-databinding-saxon. В него са дефинирани следните два типа свързаности, които се изискват от XQueryInvoker-а:

- *net.sf.saxon.om.NodeInfo* – представя XML-базиран тип на данните;
- *net.sf.saxon.value.Value* – представя данни от прост тип или java обект, който не може да се идентифицира като XML тип.

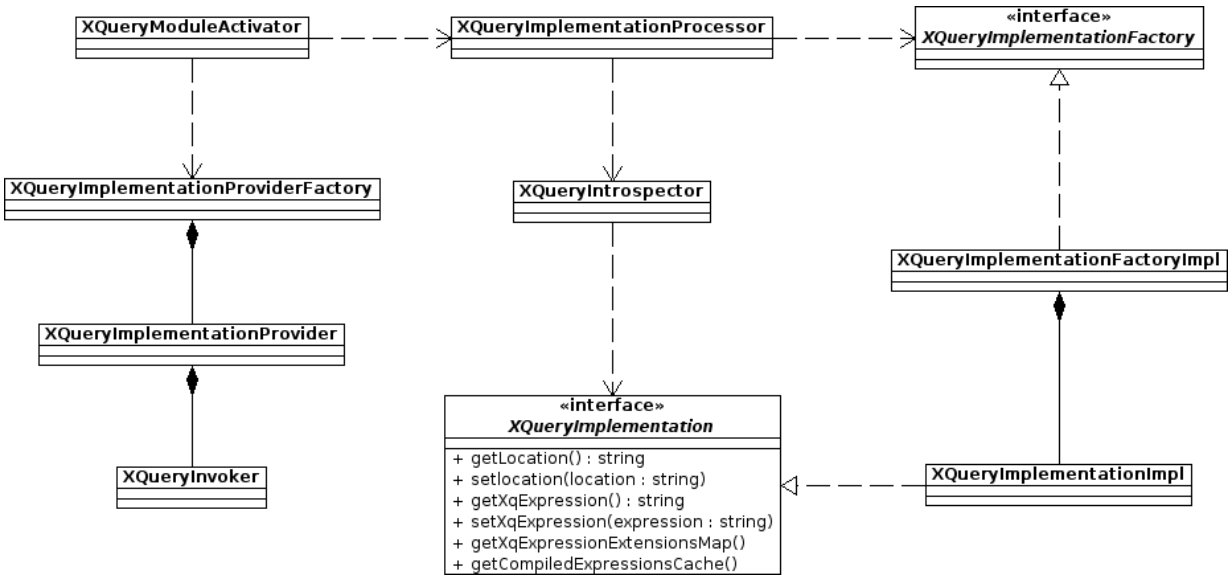
За да се осигури трансформация между съществуващите в Tuscany свързаности и по-горе дефинираните се използват следните трансформатори:

- *DataObject2NodeInfoTransformer* – осигурява конвертиране на SDO *DataObject* обекти към *NodeInfo*;
- *NodeInfo2DataObjectTransformer* - осигурява обратната на *DataObject2NodeInfoTransformer* конверсия – от *NodeInfo* към SDO *DataObject*;
- *Node2NodeInfoTransformer* – осигурява конвертиране на DOM *Node* обекти към *NodeInfo*;
- *NodeInfo2NodeTransformer* - осигурява обратната на *Node2NodeInfoTransformer* конверсия – от *NodeInfo* към DOM *Node*;

- *Object2ValueTransformer* - осигурява конвертиране на java обекти към *Value*;
- *Value2ObjectTransformer* - осигурява обратната на *Object2ValueTransformer* конверсия – от *Value* към java обекти;
- *SimpleType2ValueTransformer* - осигурява конвертиране на java прости типове към *Value*;
- *Value2SimpleTypeTransformer* - осигурява обратната на *SimpleType2ValueTransformer* конверсия – от *Value* към java прости типове.

Останалите трансформации (например от други XML типове като SAX поток и др.) се осигуряват от механизма за навързване на трансформатори осигурен от Tuscany. Например ако XQuery компонент се извиква като уеб услуга посредством Axis, се извършват следните трансформации на данните: от *XMLStreamReader* към SDO *DataObject* и след това от SDO *DataObject* към *NodeInfo*.

Другият модул на приложението – tuscany-implementation-xquery – осигурява създаването на XQuery компоненти в модела на Tuscany, зареждането на тези компоненти и извикването им посредством XQueryInvoker. На фигура 31 са показани класовете, изграждащи модула и техните взаимовръзки.



Фигура 31: Клас-диаграма на модула за XQuery имплементационен тип

Централно място заема класът *XQueryModuleActivator*, който представлява точка на разширение за Tuscany. Той се подчинява на следните условия:

- Имплементира специален интерфейс: *ModuleActivator*, който дефинира контракт между средата за изпълнение на Tuscany и разширението. Този интерфейс дефинира методите start() и stop(), където могат съответно да се регистрират/де регистрират разширения;

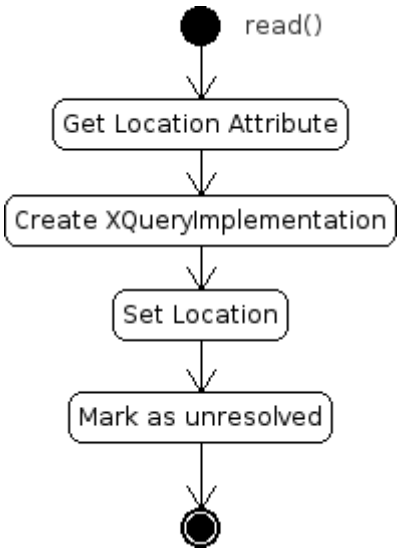
- За да се окаже на средата за изпълнение, че трябва да извика даден активатор е необходимо да се постави пълното java име на активатора в специален файл наречен *org.apache.tuscany.sca.core.ModuleActivator*, намиращ се в директория *META-INF/services* на архива с разширението.

Класът *XQueryModuleActivator* регистрира следните две

разширения, необходими за добавянето на нов имплементационен тип:

- *XQueryImplementationProcessor* – грижи се за разчитане на XML дефиницията на XQuery имплементацията в рамките на даден SCA компонент (дефиницията на *implementation.xquery* тага) и за създаване на обект представящ тази имплементация в модела на Tuscany. Този обект е необходимо да имплементира интерфейс, наследяващ *Implementation* интерфейса и в случая това е интерфейса *XQueryImplementation.* Инстанции на този интерфейс се създават посредством фабриката *XQueryImplementationFactory*;
- *XQueryImplementationProviderFactory* – грижи се за създаване на инстанции на *XQueryImplementationProvider* като всяка една от тези инстанции представлява конфигурираната по време на изпълнение имплементация за даден SCA компонент (т.е. за всяка една инстанция на SCA компонент има по една инстанция на *XQueryImplementationProvider*).

Класът *XQueryImplementationProcessor* изпълнява своята функция на два етапа. При първия етап се извършва прочитане на съответния откъс от XML-а на композита, като се създава инстанция на *XQueryImplementation* и се инициализира със стойностите на прочетените атрибути. В случая атрибута е само един – *location* – и то указва местоположението на XQuery скрипта (виж фигура 32).

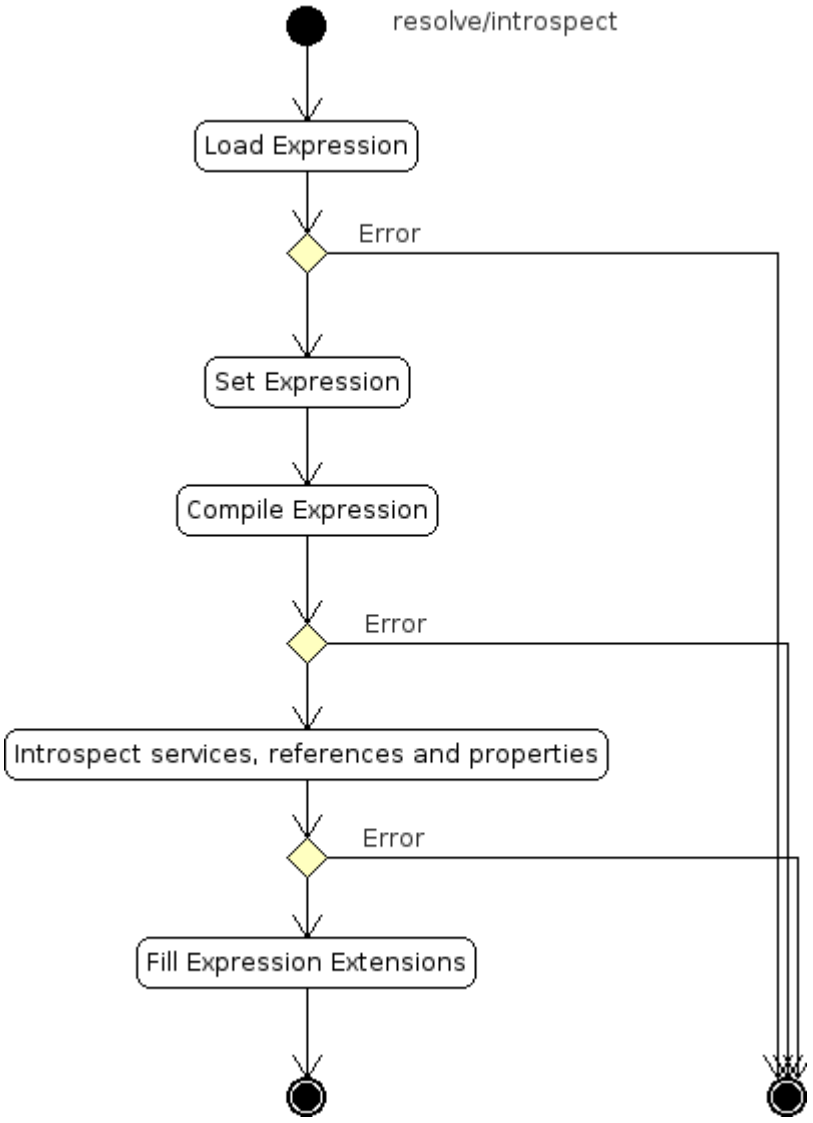


Фигура 32: Работа на XQueryImplementationProcessor

При втория етап се извършват довършителни работи по инициализацията на имплементацията, като например определяне на услугите, референциите и входните точки за конфигуриране. Задачите по този втори етап се делегират на класа *XQueryIntrospector*, който зарежда и прочита скрипта и от него определя услугите, референциите и входните точки за конфигуриране, така както е дефинирано в спецификацията за XQuery имплементационен тип. На фигура 33 са показани стъпките, които се изпълняват от *XQueryIntrospector*.

Първоначално се зарежда скрипта от адреса, зададен от *location* атрибута. След това заредения скрипт, под формата на *String*, се установява като атрибут на инстанцията на *XQueryImplementation*. Скриптът се компилира и на базата на резултата се определят дефинираните в него услуги, референции и входни точки за конфигуриране. Последната стъпка от процеса е

създаването на специално разширение на скрипта за всяка една от функциите, които са дефинирани в него. Всяко от разширенията е предназначено за извикване на една конкретна функция. За да се изпълни дадена функция е необходимо да се конкатенира заредения от файл скрипт и съответното разширение и след това получения скрипт да се компилира и изпълни.

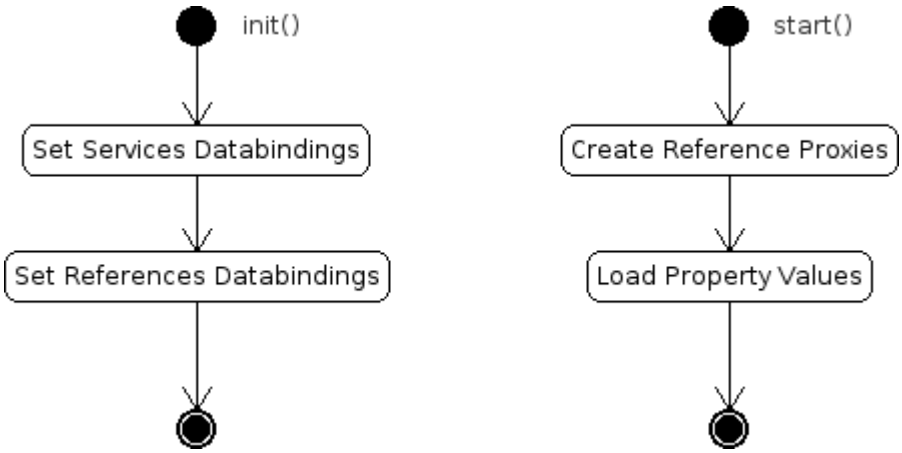


Фигура 33: Работа на XQueryIntrospector

По време на фазата на стартиране на средата за изпълнение на Tuscany се създават инстанции на

XQueryImplementationProcessor за всеки един компонент, който използва XQuery имплементационен тип. *XQueryImplementationProcessor* има следните отговорности:

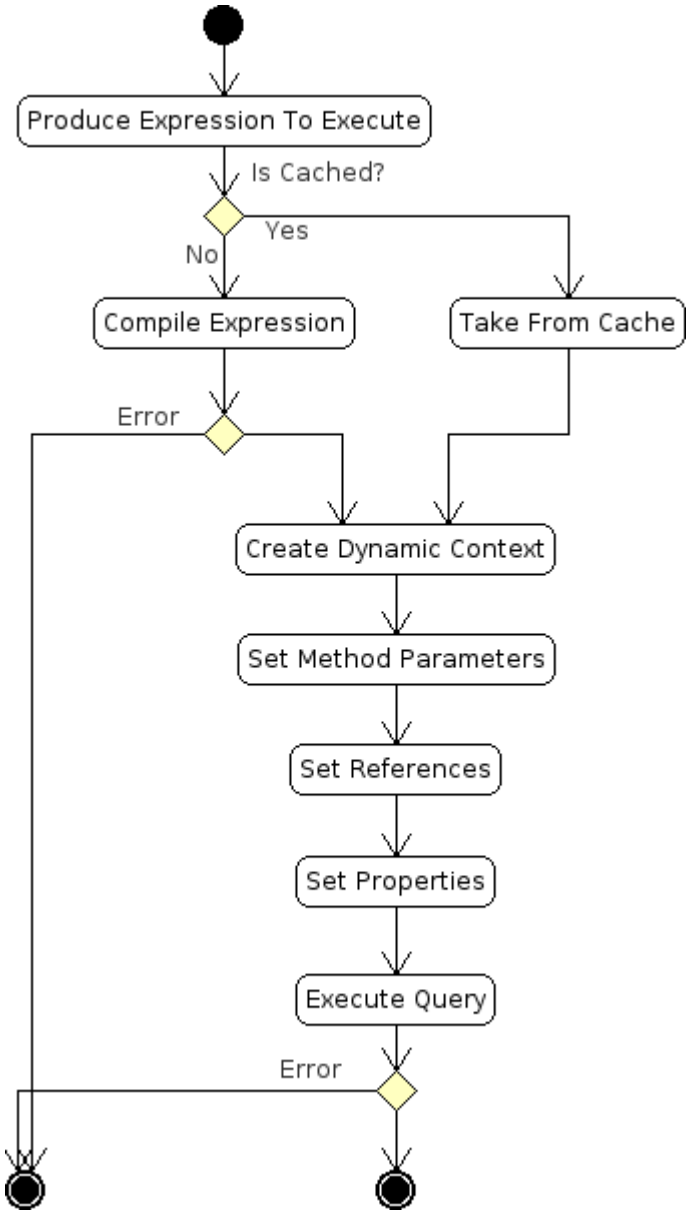
- да зададе свързаности на данните на услугите и референциите от страната на XQuery компонента (фигура 34);
- да създаде proxy обекти за всички компоненти, които се реферират от този компонент, като тези proxy обекти ще се използват на по-късен етап за инжектиране (injection) на стойности на променливите за референции в XQueryInvoker (фигура 34);
- Да зареди стойностите на входните точки за конфигуриране на компонента, като тези стойности ще се използват на по-късен етап за инжектиране (injection) на стойности на променливите за входни точки в XQueryInvoker (фигура 34);



Фигура 34: Инициализация на XQueryImplementationProcessor

Също така отговорност на *XQueryImplementationProcessor* е да създава инстанции на *XQueryInvoker*, който осъществява реалния достъп до

XQuery имплементацията по време на изпълнение на SCA-базираното приложение. Основен за *XQueryInvoker,* е метода *doInvoke,* който се извиква тогава, когато някой прати заявка към XQuery компонента. Този метод получава входните параметри на заявката и връща като изходен параметър резултата от изпълнението на скрипта. На фигура 35 е представен подробна схема на неговата работа.



Фигура 35: Работа на XQueryInvoker

Изпълняват се следните стъпки:

- Извършва се конкатенация на основния скрипт със разширение, съответстващо на извикваната функция;
- Ако преди това тази функция е извиквана, то компилираната форма на XQuery заявката се съдържа в кеша и се взема от там, ако не – то заявката се компилира и се поставя в кеша;
- Създава се динамичен контекст за изпълнение на заявката;
- Този контекст се използва, за да се зададат стойности на:
  - Аргументите подавани към функцията;
  - Променливите, изразяващи референции;
  - Променливите, изразяващи входни точки за конфигуриране;
- Заявката се изпълнява, като резултата се връща от *doInvoke* метода.

6 Глава 6. Демонстрационно приложение

За демонстрация на използването на XQuery имплементационния тип е разработен интеграционен сценарий, в който е необходимо сливането на две съобщения, идващи евентуално от различни източници и оформянето на едно изходно съобщение.

Сценарият представлява следното: Необходимо е да се образува извадка, съдържаща следната информация за даден клиент (Quote):

- име на клиента;

- адрес на клиента (като един низ);
- За всеки поръчан от клиента продукт е необходима информация за:
  - идентификационен номер на продукта;
  - единична цена на продукта;
  - поръчано количество;
  - дали заявката може да се изпълни;
  - дата на, която трябва да се изпълни заявката или „BackOrder“, ако тя не може да се изпълни;
  - данъчна основа;
  - обща цена.

Тази извадка трябва да се образува на базата на две извадки: едната за продуктите, които е поръчал клиента и техните единични цени (PriceQuote), а другата за наличностите на съответните продукти (AvailQuote).

Първата извадка (PriceQuote) съдържа следната информация:

- Име на клиента;
- Структура, представяща адреса за доставки на клиента. тази структура има следните полета:
  - улица;
  - град;
  - държава;
  - пощенски код;
- За всеки поръчан от клиента продукт е необходима

информация за:

- идентификационен номер на продукта;
- единична цена на продукта.

Втората извадка (AvailQuote) съдържа следната информация:

- За всеки продукт, който се предлага:
  - идентификационен номер на продукта;
  - поръчано количество;
  - дали има наличност за поръчаното количество;
  - дата, на която трябва да се изпълни поръчката или „BackOrder“, ако тя не може да се изпълни.

Всяка една от извадките е описана посредством следните XSD типове: Quote.xsd, PriceQuote.xsd, AvailQuote.xsd (виж приложения програмен код).

Необходим е XQuery-базиран компонент, който да преобразува входящите извадки - PriceQuote и AvailQuote в изходяща заявка Quote, като той трябва да използва външна функция, написана на java за изчисляване на общата стойност на дадена поръчка за продукт.

За да се демонстрират интеграционните свойства на XQuery компонента, освен това, се поставят следните изисквания:

- да се реализира сценарий, при който на функцията за образуване на Quote се подават като параметри PriceQuote и AvailQuote обекти, чиито java типове са резултат от SDO генерация, базирана на съответните

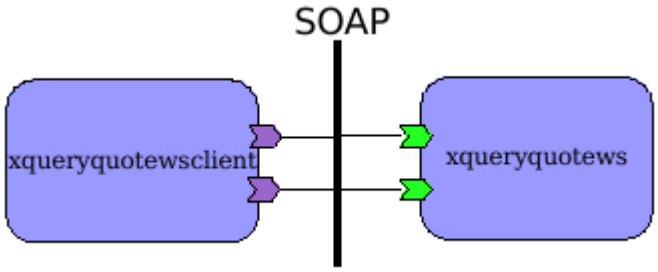
XSD типове, а също така се подава параметър от тип *float* изразяващ данъчната основа. Извикването на XQuery компонента да става по два начина:

- локално, т.е. в рамките на същата java виртуална машина;
- отдалечено - като уеб услуга;

- XQuery компонента трябва да има референция към java компонент, който предоставя функция за изчисляване на пълната цена на заявката;
- да се реализира сценарий, при който на функцията за образуване на Quote не се подават параметри, а информацията за PriceQuote, AvailQuote да е зададена като XML, а данъчната основа като параметър, чрез входните точки за конфигуриране на XQuery компонента;
- да се реализира сценарий, при който на функцията за образуване на Quote се подават параметър от тип *float* изразяващ данъчната основа. Освен това XQuery компонента има референция към java компонент, осигуряващ AvailQuote под формата на SDO обект, и достъпен като уеб услуга и референция към java компонент, осигуряващ PriceQuote под формата на SDO обект, и достъпен локално.

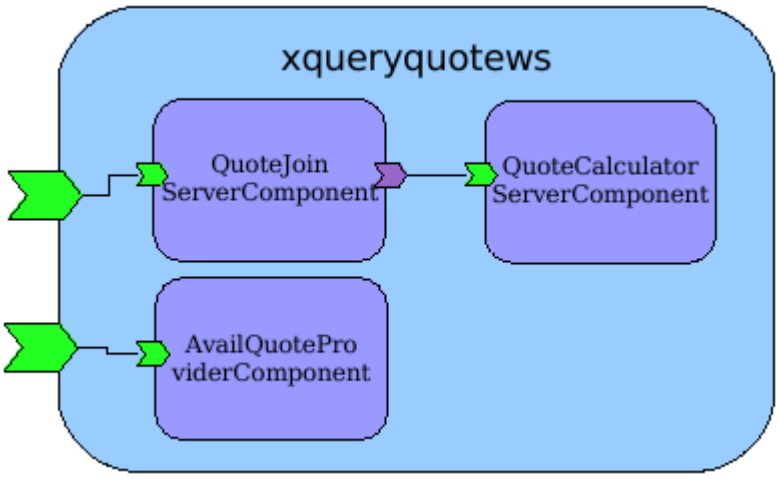
Демонстрационното приложение е реализирано от два композита, при които този наречен *xqueryquotewsclient* използва услуги под формата на уеб услуги от *xqueryquotews* (виж фигура 36).

xqueryquotews реализира две услуги. Първата се доставя от компонента *QuoteJoinServerComponent,* който представлява XQuery имплементация на компонент за сливане на *PriceQuote* и *AvailQuote* изадките. Вторият се нарича *AvailQuoteProviderComponent* и осигурява AvailQuote за сценария с приемане на информация от уеб услуга в *xqueryquotewsclient* композита.



Фигура 36: Демонстрационно приложение - общ изглед

Вътрешната структура на *xqueryquotews* е показана на фигура 37, като компонента *QuoteCalculatorServerComponent* осигурява изчислението на общата стойност на дадената заявка.

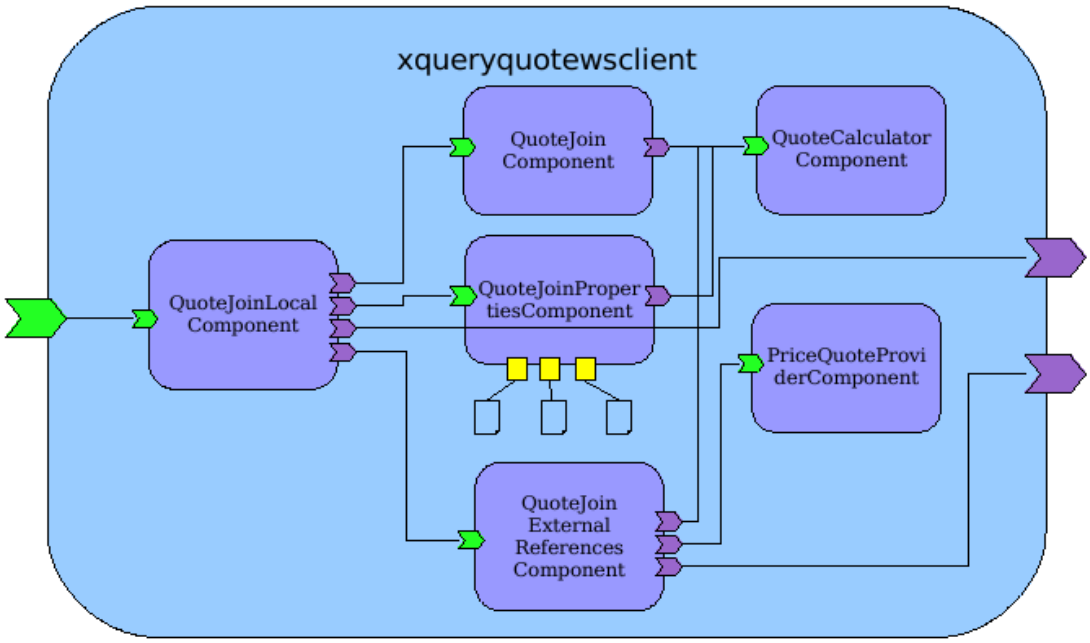


Фигура 37: Демонстрационно приложение - сървърен композит

На фигура 38 е представена реализацията на

xqueryquotewsclient композита. Като фасада в него се използва компонента *QuoteJoinLocalComponent*, който извиква останалите компоненти в зависимост от сценария който трябва да се изпълни. Това са:

- QuoteJoinComponent – изпълнява същото, което и *QuoteJoinServerComponent* но се извиква локално;
- *QuoteJoinPropertiesComponent* – използва се за реализация на сценария, при който необходимата информация се взема от входните точки за конфигуриране;
- *QuoteJoinExternalReferencesComponent* – използва се за реализация на сценария, при който XQuery компонент извиква външни компоненти, за да получи информация;
- Директна референция към уеб услугата *QuoteJoinServerComponent* от композита *xqueryquotews*.



Фигура 38: Демонстрационно приложение - клиентски композит

Компонента *QuoteCalculatorComponent*, подобно на *QuoteCalculatorServerComponent* осигурява изчислението на общата стойност на дадената заявка.

За изпълнението на тестовия сценарий се използва JUnit тест, който извиква последователно методите на интерфейса на *QuoteJoinLocalComponent*, като по този начин стартира изпълнение на всеки един от сценариите. Освен това, тестът осигурява примерни стойности на taxRate, PriceQuote и AvailQuote параметрите.

7 Глава 7. Идеи за интеграция на XQuery имплементационния тип със STP

В настоящата глава ще бъдат дадени някои насоки за това как да се създават и редактират XQuery имплементационни типове, като целта е да се предоставят набор от разширения към Eclipse STP проекта, с чиято помощ да се даде възможност на потребителя да създава SCA компоненти с XQuery имплементационен тип, както и да редактира тяхната имплементация.

Първата стъпка за реализация на проекта ще се базира изцяло на налични решения, които ще се интегрират. Следните разширения трябва да се направят:

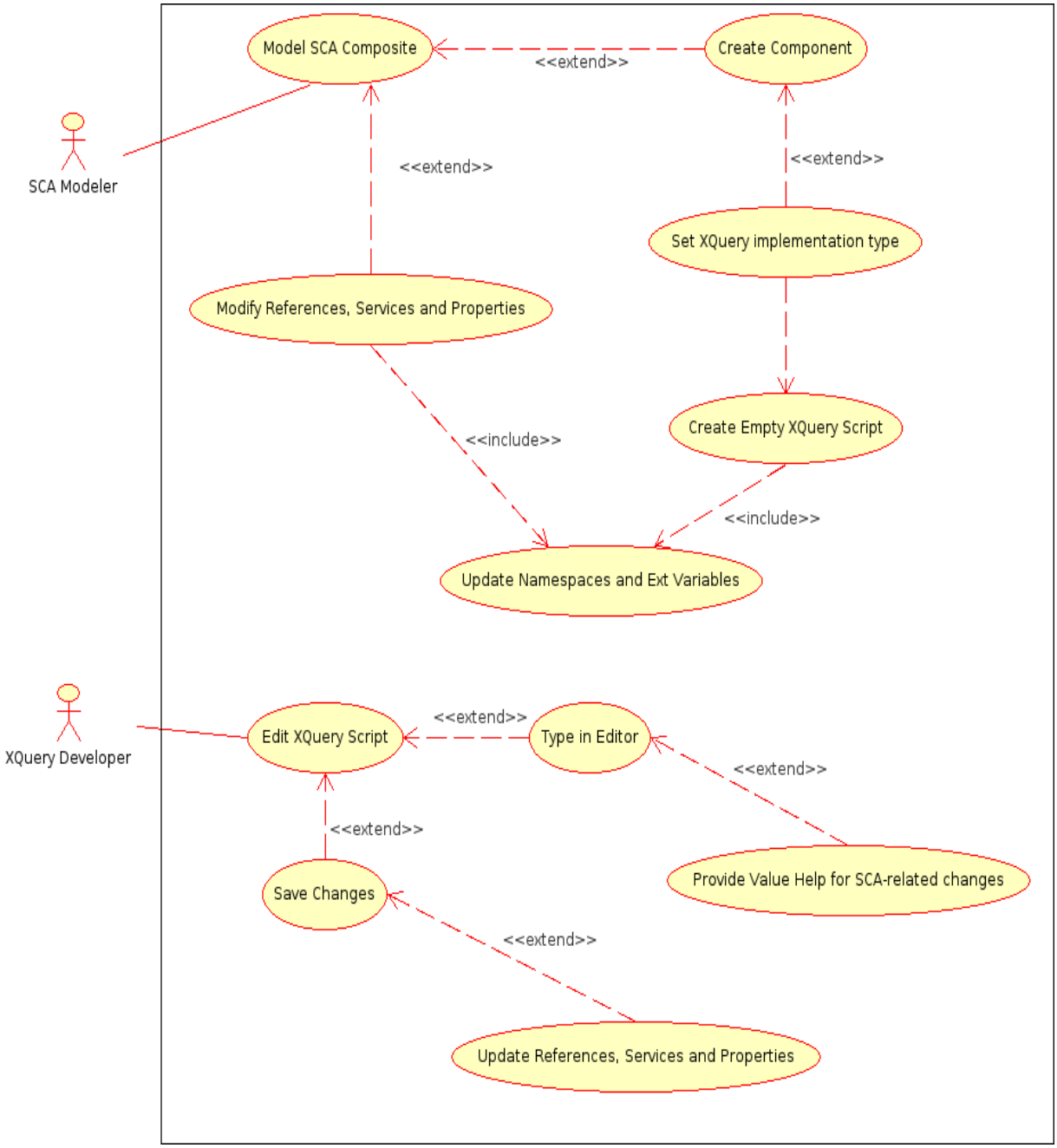
- Да се използва редактора за композити на Obeo [47], който. в момента е все още в процес на IP верификация и не е достъпен, като се допълни с възможност за създаване на имплементации от тип XQuery към даден компонент;
- да се използва редактора с отворен код xqIde [48], който позволява редактиране на XQuery скриптове. Той

трябва да се разшири с възможност за добавяне и верифициране на специфичните за XQuery имплементационния тип разширения, които са описани в спецификацията за този тип. Примери за такива разширения са автоматично добавяне/премахване на услуги и референции в зависимост от действията, които потребителя извършва в SCA редактора и обратното – автоматична промяна на услугите и референциите в зависимост от това как потребителя редактира XQuery скрипта.

Фигура 39 резюмира изискванията към първия вариант за интеграция на SCA с редактор за XQuery.

Разработчика на SCA модел може да създава компоненти (Model SCA Composite -> Create Component) като задава XQuery като техен имплементационен тип (Set XQuery implementation type). При това за него се генерира имплементация под формата на XQuery скрипт (Create Empty XQuery Script), като в този скрипт се поставят необходимите (за реализацията на услуги, референции и входни точки за конфигуриране) декларации на пространства от имена и външни променливи (Update Namespaces and Ext Variables). Освен това ако се промени нещо по референциите на дадения компонент с други компоненти, или пък се променят интерфейсите на услугите, които имплементира (Modify References, Services and Properties), тогава трябва да се актуализират и дефинициите на пространствата от имена и декларациите на външни променливи в XQuery скрипта (Update Namespaces

and Ext Variables).

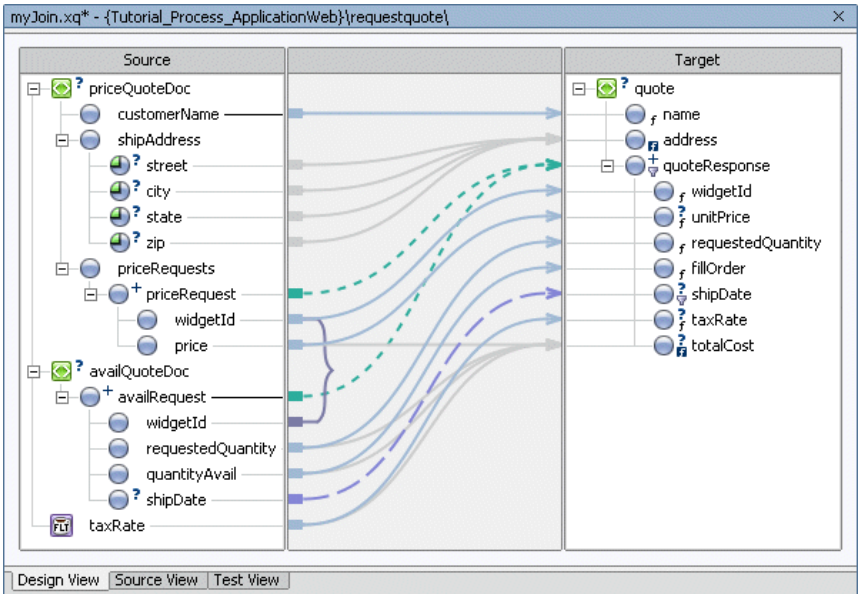


Фигура 39: Use Case диаграма на редактор за XQuery имплементационен тип

Разработчикът на XQuery скрипта, от друга страна, би могъл ръчно да дефинира, модифицира или изтрива услуги, референции или входни точки за конфигуриране (Edit XQuery Script), като при това редактиране, по време на писане (Type in Editor), би могъл да получава помощ за това

какви интерфейси съществуват в проекта (Provide Value Help for SCA-related changes). След като потребителя е модифицирал скрипта, той може да го запише (Save Changes) и ако са настъпили промени по услугите, референциите или входните точки за конфигуриране, това може да се отрази върху модела на SCA композита (Update References, Services and Properties).

Втората стъпка в реализацията на проект за интеграция на XQuery имплементационен тип към STP представлява разработката на собствен редактор, чрез който ще могат графично да се дефинират трансформации между входни и изходни интерфейси. По този начин, дори потребители, които не разбират от XQuery, ще могат да създават трансформации. Освен това този редактор може да се направи достатъчно разширяем, така че от него да се генерират различни трансформации: XQuery, XSLT, Java код или др. На фигура 40 е показан пример за това как може да изглежда такъв редактор.



Фигура 40: Потребителски интерфейс на редактор за XQuery имплементационен тип

Най-общо той може да е изграден от три части: най-отляво са показани всички входящи структури от данни, а най-отдясно – изходящите структури, като потребителя може да дефинира връзки между входните и изходните атрибути използвайки мишката. В средната част на редактора са изобразени вече направените връзки.

За реализацията на такъв редактор може да се дефинира метамодел, който е базиран на EBNF азбуката на XPath [49]. На базата на този метамодел ще се образуват модели, изразяващи дадена трансформация, които ще се описват и съхраняват с помощта на EMF. Освен това ще се дефинират трансформации от модела към XQuery, XSLT и др.

Заключение

В дипломната работа беше направен обзор на използваните технологии. След това бе извършен обстоен анализ на средствата за интеграция на данни и медиация на интерфейси, които се предлагат от основните производители на платформи за разработка на SOA-базирани приложения. На базата на този анализ бе аргументирана необходимостта от декларативно решение за интеграция в една SOA система. Във връзка с това бе отбелязана липсата на такова решение в една от най-консолидираните SOA спецификации в момента – SCA и една от нейните имплементации с отворен код – Tuscany. Като резултат от това бе специфициран специален имплементационен тип за SCA, базиран на езика XQuery. Този език бе избран във връзка с неговите безспорни предимства при интеграцията на данни от различни източници, както показва и направения в дипломната работа анализ. За да се докаже работоспособността на предложената спецификация, беше създадена примерна имплементация на XQuery типа за SCA контейнера на средата за изпълнение на Tuscany, както и сценарий за приложение, което да я използва. Дипломната работа завърши с описание на това как би могло да изглежда едно приложение, даващо възможност за дизайн на новопредложения тип, което е интегрирано с STP.

В момента разработения прототип е предоставен от дипломанта на организацията за отворен код Apache, където той продължава своето доразработване и служи като основа

за дискусии и развитие в областта на интеграцията на данни като част от SOA.

Библиография

1. SCA Main Page, http://www.osoa.org/display/Main/Service+Component+Architecture+Home
2. SOA principles, http://en.wikipedia.org/wiki/Service-oriented_architecture#SOA_principles
3. Thomas Erl, Service-Oriented Architecture Concepts, Technology, and Design, 2006
4. JBI Main Page, http://jcp.org/en/jsr/detail?id=208
5. OSOA Main Page, http://www.osoa.org/display/Main/Home
6. IBM WebSphere Process Server, http://www-306.ibm.com/software/integration/wps/
7. Apache Tuscany Main Page, http://incubator.apache.org/tuscany/
8. Eclipse STP Main Page, http://www.eclipse.org/stp/
9. Fabric3 Main Page, http://fabric3.codehaus.org/
10. Tuscany Architecture Guide, http://incubator.apache.org/tuscany/sca-java-architecture-guide.html
11. SOA Tools Platform Project, http://www.eclipse.org/stp/
12. STP Eclipsepedia, http://wiki.eclipse.org/STP
13. Erich Gamma, Design Patterns Elements of Reusable Object-Oriented Software, 2000
14. Open ESB Main Page, https://open-esb.dev.java.net/
15. XQuery Specification, http://www.w3.org/TR/xquery/
16. Comparing XSLT and XQuery, http://www.idealliance.org/proceedings/xtech05/papers/02-03-01/
17. Saxon Main Page, http://www.saxonica.com/
18. IBM Information Server Overview, ftp://ftp.software.ibm.com/software/data/is_server/4/index.htm
19. BEA AquaLogic Data Services, http://www.bea.com/framework.jsp?CNT=index.htm&FP=/content/products/aqualogic/data_services/
20. DataDirect XQuery, http://www.datadirect.com/products/xquery/index.ssp
21. DataDirect XML converters, http://www.datadirect.com/products/xmlconverters/index.ssp
22. Stylus Studio Main Page, http://www.stylusstudio.com/stylus_home_edition.html
23. ETL Service Engine, http://wiki.open-esb.java.net/Wiki.jsp?page=ETLSE
24. XSLT Service Engine, http://wiki.open-esb.java.net/Wiki.jsp?page=XSLTSE
25. EDMS Service Engine, http://wiki.open-esb.java.net/Wiki.jsp?page=EnterpriseDataMashup
26. SQL Service Engine, http://wiki.open-esb.java.net/Wiki.jsp?page=SQLSE
27. Information Integrator, http://documentation.softwareag.com/crossvision/xei/overview.htm
28. F-logic basics, http://documentation.softwareag.com/crossvision/xei/oflogic/oflogicover.htm
29. XCerpt introduction, http://www.xcerpt.org/about/intro/
30. BizTalk Mapper, http://msdn2.microsoft.com/en-us/library/aa559261.aspx

31. SAP BI, https://www.sdn.sap.com/irj/sdn/bi
32. SAP XI, https://www.sdn.sap.com/irj/sdn/pi
33. Oracle Data Integrator White Paper, http://www.oracle.com/technology/pub/articles/rittman-odi.html
34. Oracle ESB White Paper, http://www.oracle.com/technology/pub/articles/jellema-esb.html
35. Oracle XDS White Paper, http://www.oracle.com/technology/oramag/oracle/05-mar/o25xml.html
36. IBM Information Server, http://www-306.ibm.com/software/data/integration/info_server/
37. IBM Transformation Extender, ftp://ftp.software.ibm.com/software/websphere/integration/wdatastagetx/WebSphere_TX_wp_9-1.pdf
38. IBM Map Designer, ftp://ftp.software.ibm.com/software/websphere/integration/wdatastagetx/1005.pdf
39. WebSphere Business Integration Server, http://www-306.ibm.com/software/integration/wbiserver/library/?S_CMP=mav
40. Artix Data Services, http://www.iona.com/products/artix/data_services.htm
41. Artix ESB, http://www.iona.com/products/artix/esb.htm
42. Apache Camel, http://activemq.apache.org/camel/
43. TIBCO Integration White paper, http://www.tibco.com/resources/software/dataintegration/dataexchange_whitepaper.pdf
44. SCA Assembly Specification, http://www.osoa.org/download/attachments/35/SCA_AssemblyModel_V100.pdf?version=1
45. Java Component Implementation Specification, http://www.osoa.org/download/attachments/35/SCA_JavaComponentImplementation_V100.pdf?version=1
46. Declarative Data Access Services - Integrating DAS and SCA, http://lresende.blogspot.com/2007/01/declarative-data-access-services.html
47. Obeo Editor Demo, http://wiki.eclipse.org/SCA_Composite_Editor
48. xqIde Main Page, http://xqide.org/
49. XPath EBNF Notation, http://www.w3.org/TR/xpath20/#id-grammar

Приложения