Санкт-Петербургский государственный университет
Математико-механический факультет

Кафедра системного программирования

Система мониторинга веб-сервисов

Курсовая работа студента 445 группы
Фефелова Алексея Андреевича

Научный руководитель — Строкан Павел Владимирович
(Руководитель отдела, T-Systems CIS)

Санкт – Петербург
2012

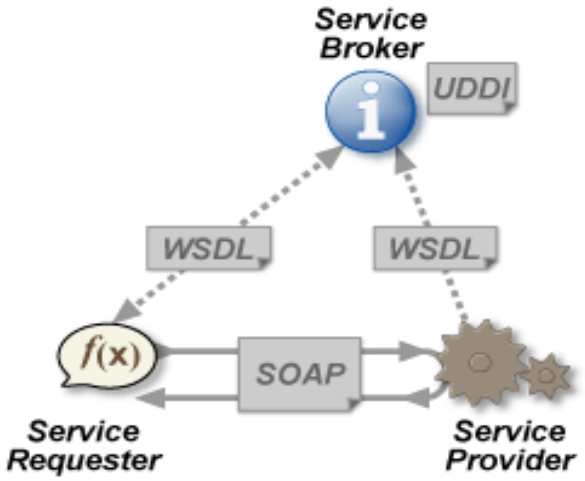
Оглавление

Введение

- Веб-служба, веб-сервис — идентифицируемая веб-адресом(URI) программная система со стандартизированными интерфейсами.
- SOAP - протокол обмена сообщениями на базе XML.
- WSDL - язык описания внешних интерфейсов веб-службы на базе XML.

Для того, чтобы отправлять и получать сообщения веб-сервису, клиенту, реализованному на платформе Java EE, необходимо иметь:

1) wsdl-описание этого веб-сервиса;
2) java-классы, соответствующие веб-сервису, написанные самостоятельно или сгенерированные автоматически;
3) soap – сообщения, сгенерированные системой разработки веб-сервисов с помощью java-классов из 2 пункта, которые являются запросами веб-сервису, и получив которые, веб-сервис должен послать ответ.



(Рисунок 1 – Веб-сервис)

С одной стороны, клиент может и вовсе не дождаться ответа в силу разных обстоятельств (разрыв интернет соединения, веб-сервис выключен или вообще не существует), а с другой – ответ может быть некорректным (не

соответствует wsdl-описанию, синтаксически неверный и т.д.) Когда происходят такие случаи, неизвестно, как поведет себя клиент. Он может никак не прореагировать на некорректный ответ, а может и вовсе упасть сервер, на котором работает клиент. Поэтому, в ходе разработки клиента, чтобы как можно быстрее исключить такие ситуации из его поведения, и нужна система мониторинга веб-сервисов, о реализации которой написано ниже.

1.Постановка задачи

- Изучить технологии, с помощью которых веб-сервисы взаимодействуют между собой
- Реализовать систему, которая выполняет наблюдение за работой веб-сервисов и дает возможности для анализа данные, которыми веб-сервисы обмениваются.
- Встроить систему в реальный проект.

2.Область применения

Система мониторинга служит для ускорения процесса реинжиниринга. Зачастую веб-сервисы и клиент, работающий с ними, реализуют разные люди, поэтому процесс реинжиниринга становится довольно трудным. Но благодаря системе мониторинга можно в более короткие сроки осуществлять следующие задачи:

- Поиск багов и уязвимостей;
- Оптимизация;
- Добавление новой функциональности;
- Тестирование.

3.Обзор существующих решений

Изучив несколько систем мониторинга (сетей, серверов, системных процессов) я практически не нашел среди них систем, умеющих осуществлять мониторинг веб-сервисов. Классифицировав полученные результаты, оказалось, что большинство существующих системы умеют мониторить:

- процессы (наличие, количество потребляемых ресурсов)
- сетевые хосты (пинг и коннект на определенный порт по определенному протоколу)
- количество входящего и выходящего трафика.

К таким системам можно отнести: PRTG, CACTI, Zabbix, Nagios и другие.

Лишь одна система, из тех, с которыми я ознакомился, умеет мониторить сервисы – это AggreGate Network Manager. Эта система имеет следующие достоинства:

1)Имеет панель статуса сервисов (См. рис. 2)

Панель представляет собой матрицу, где с помощью цветового кодирования отражается состояние всех сервисов на всех отслеживаемых сетевых устройствах.

2) Оповещение о сбоях в работе приложений.

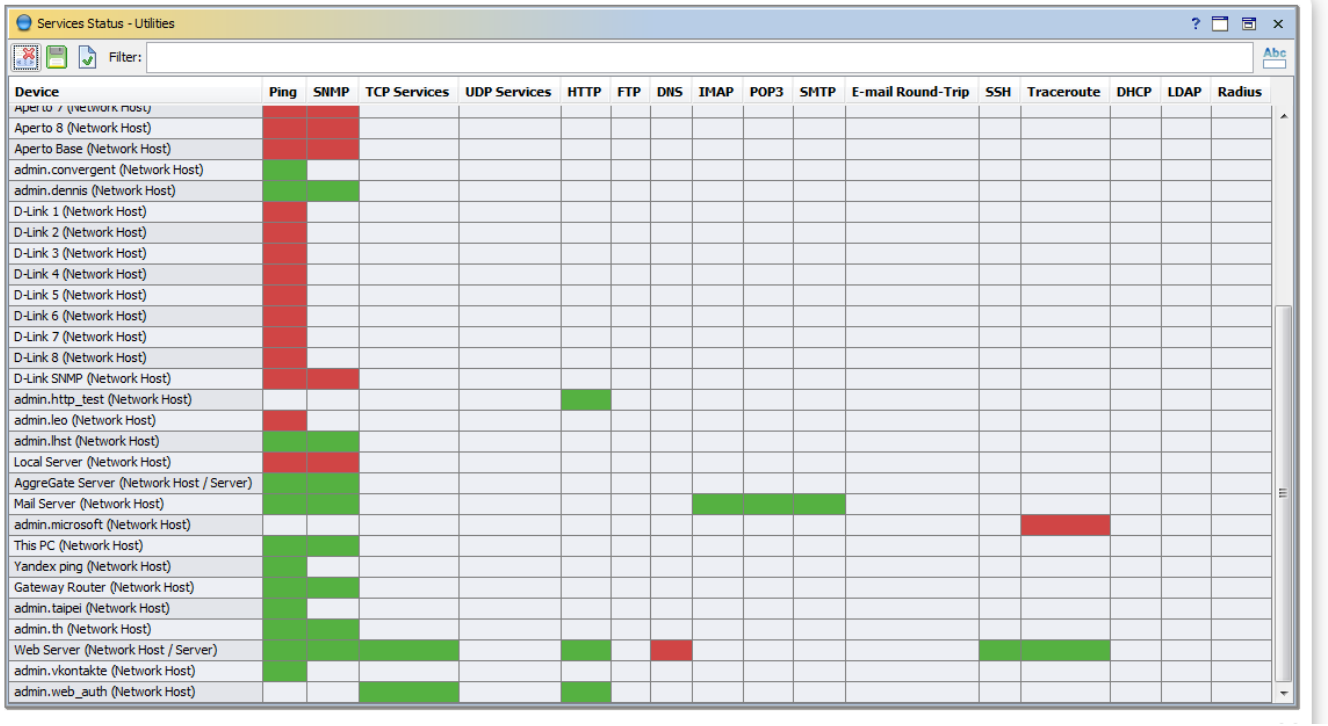
AggreGate Network Manager включает в себя готовые тревоги для типичных проблем, таких как слишком большое количество запросов к веб-серверу Apache или нехватка места на FTP-сервере.

3) Обнаружение приложений и сервисов.

В результате сетевого обнаружения система находит как известные ей, так и неопознанные ("обобщенные") сервисы, последние путем сканирования TCP/UDP портов. Мониторинг для найденных активных сервисов активируется по умолчанию при создании учетной записи устройства. При этом для каждого активированного сервиса автоматически выбираются оптимальные настройки мониторинга.

4) Сбор сообщений и уведомлений об ошибках.

Система удаленно собирает сообщения приложений и уведомления об ошибках, используя для этого Syslog и журнал событий Windows. Полученные сообщения сохраняются в центральной базе данных на сервере, обеспечивая возможность проведения долгосрочного аудита.



(Рисунок 2 – скриншот работы AggreGate Network, панель статуса сервисов)

Но несмотря на неоспоримые преимущества, эта система не может показывать контент сообщений, которыми обмениваются наблюдаемые сервисы.

Возможно, это потому, что для доступа к сообщениям нужен доступ к веб-сервису, а у системы его нет.

4.Реализация

Вся разработка велась согласно MVC- концепции, согласно которой модель данных приложения, пользовательский интерфейс и взаимодействие с пользователем разделены на три отдельных компонента так, что модификация одного из компонентов оказывает минимальное воздействие на остальные.

4.1Model

Была создана база данных Oracle , которая имеет одну таблицу MONITORING (см. таблицу 1).

Name	Data Type	Size	Scale	Nullable
ID	NUMBER			☐
TYPE	NUMBER			☑
MESSAGE	VARCHAR2	4000		☑
MESSAGEDATE	VARCHAR2	50		☑
MESSAGEER...	VARCHAR2	1024		☑
ADRESS	VARCHAR2	100		☑

Таблица имеет следующие поля:

1) Id – идентификационный номер таблицы.
2) Type – тип сообщения. Бывают следующие типы сообщений:

- Correct message – в случае, когда пришедшее клиенту сообщение синтаксически и семантически (соответствует wsdl-представлению) верное;
- Connect error – если клиент не может отправить сообщение на веб-сервис;
- Data error – в каком-то из элементов сообщения находятся неправильные данные, которые не могут быть преобразованы в тип, описанный в wsdl-представлении;
- Unexpected character – в ответе встретился лишний символ;
- Unexpected element – в ответе встретился тег, которого нет согласно wsdl-представлению веб-сервиса.
- Incorrect structure – в случае, если сообщение неправильной структуры, в которой не хватает каких-либо элементов(тегов), несмотря на то, что

синтаксически верное.

3) Message – контент сообщения;

4) MessageDate – дата, когда было получено сообщение;

5) Adress – URI – адрес веб-сервиса.

Прослойку между БД и Java обеспечивает технология Hibernate, которая очень широко применяется в Java EE проектах. Ее особенностью является то, что в отличие от JDBC , можно работать не только с нативным SQL-кодом, но и с объектами, представляющими таблицы в базе данных, так называемыми Entity. Таблице MONITORING соответствует Entity - класс Monitoring.

Для работы с базой данных создан интерфейс IMonitoringDao и класс его реализующий, MonitoringDaoImpl, представляющие из себя DAO (Data Acsess Object), то есть нужные исключительно для доступа к данным.

4.2View

Пользовательский интерфейс в системе реализован с помощью фреймворка JSF (Java Server Faces) . Главной отличительной особенностью jsf от других MVC - фреймворков является использование компонентов. Состояние компонентов пользовательского интерфейса сохраняется, когда пользователь запрашивает новую страницу и затем восстанавливается, если запрос повторяется. В моем проекте используется лишь одна jsf-страница, благодаря которой полностью реализован пользовательский интерфейс.

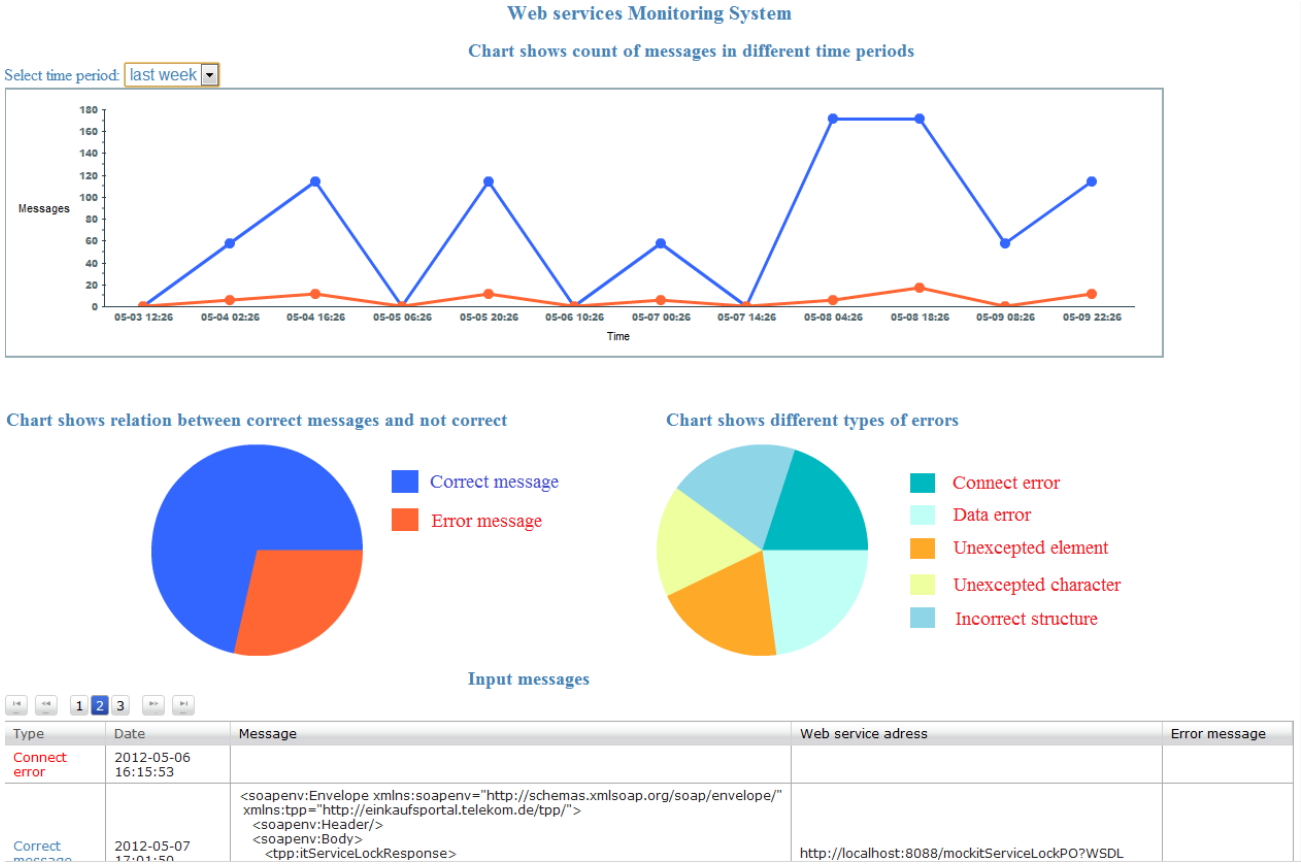
Для отрисовки графиков используется дополнение к jsf – Prime Faces.

С помощью этой технологии можно рисовать самые разные виды графиков, включая интерактивные и live - графики. Пользовательский интерфейс системы показан на рис.

4.3 Controller

Взаимодействие между пользовательским интерфейсом и моделью реализовано с помощью jsf – компонента ChartBean, java - класса, также реализующего

некоторую бизнес-логику. Этот бин с помощью методов DAO- класса получает данные из базы данных, обрабатывает их и посылает jsf - странице.



(Рисунок 3.Пользовательский интерфейс системы.)

4.4 Особенности реализации

Для того, чтобы получать или принимать сообщения, которыми обменивается клиент с веб-сервисом, я решил использовать стек web-сервисов Apache CXF, являющийся реализацией протокола JAX-WS – API для создания web-служб.

4.4.1 JAX-WS

Его особенностью является использование аннотаций, что устраняет необходимость создания дескрипторов веб-служб. Также большим преимуществом является то, что декларация конечных точек (endpoints) происходит непосредственно в классах Java.

Существуют следующие типы аннотаций:

- @WebService — указывает на то, что Java класс (или интерфейс) является веб-службой.
- @WebMethod — позволяет настроить то, как будет отображаться метод класса на операцию веб-службы.
- @WebParam — позволяет настроить то, как будет отображаться конкретный параметр операции на WSDL-часть (part) и XML элемент.
- @WebResult — позволяет настроить то, как будет отображаться возвращаемое значение операции на WSDL-часть (part) и XML элемент.

4.4.2 Apache CXF

CXF - это стек Web-сервисов от Apache Software Foundation. .Этот стек позволяет создать Web-сервис на основе существующего кода Java или же сгенерировать код Java на основе WSDL-описания для использования или реализации сервиса. Я использовал второй вариант. Для того, чтобы сгенерировать класс web-сервиса на клиенте (web-сервис будет реализован с помощью jax-ws), сначала необходимо создать wsdl-файл, в котором будут определены сообщения, порты, типы данных и, конечно, связывания данных(binding) Затем в файле сборки(для системы сборки maven – pom.xml, для ant – build.xml) нужно прописать использование утилиты wsdl2java. Именно эта утилита от CXF и позволяет генерировать java – код. Затем нужно запустить конфигурационный файл с помощью своей системы сборки проекта(командой mvn или ant) и Java - код сервисов будет сгенерирован со всеми нужными аннотациями.

Как и в других стеках, в CXF обработка запросов и ответов осуществляется набором конфигурируемых компонентов. В CXF эти компоненты называются *перехватчиками* (intercepters), а не обработчиками (*handlers*), но, несмотря на разные названия, эти термины обозначают одни и те же компоненты. Для того, чтобы перехватывать входящие сообщения был написан перехватчик

ReceiveInterceptor, который имеет методы handleMessage и handleFault. Первый метод срабатывает всегда, когда приходит корректное сообщение, а второй – когда приходит сообщение с ошибкой. Оба метода полученные и обработанные данные заносят в базу данных с помощью DAO – классов.

4.4.3 Spring

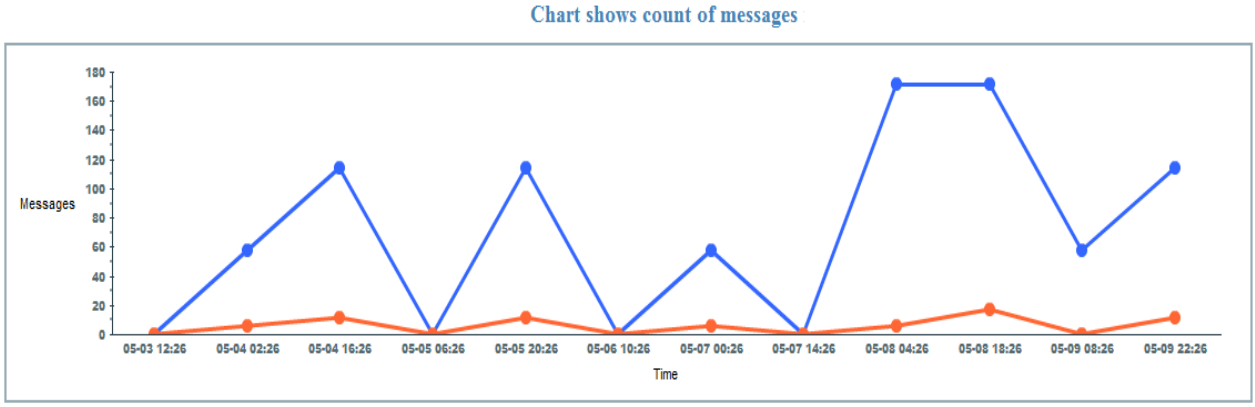
Чтобы перехватчик ReceiveInterceptor работал, его надо зарегестрировать в качестве бина в файле default-beans.xml. В этом файле обычно описаны все Spring - бины, имеющиеся в проекте. Spring – очень популярный фреймворк в java-сообществе, имеющий много разных достоинств, прежде всего – это фреймворк аспектно – ориентированного программирования, он работает с функциональностью, которая не может быть реализована возможнстями объектно-ориентированного программирования на Java без потерь.

5. Результаты

В процессе работы были достигнуты следующие результаты:

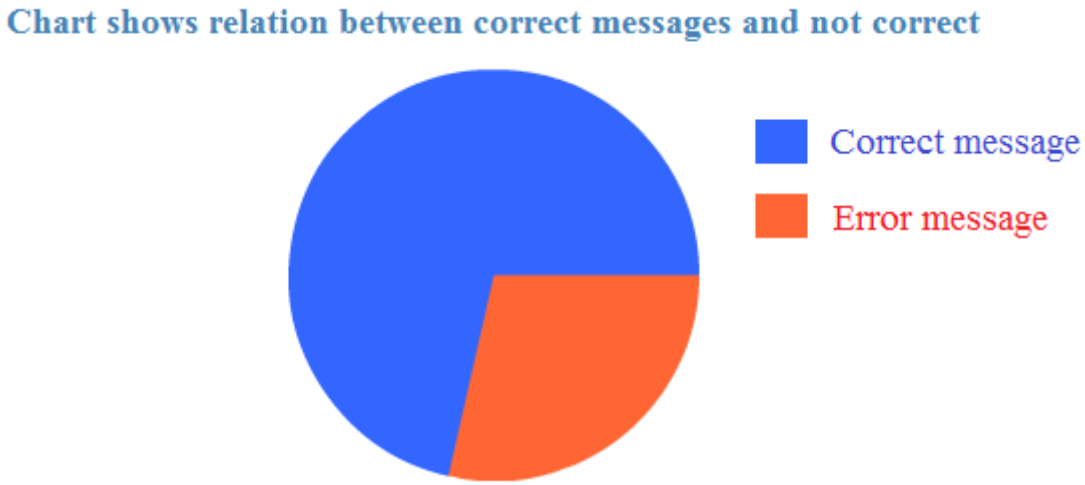
- Изучены технологии взаимодействия веб-сервисов
- Создано несколько веб-сервисов
- Реализована система мониторинга, умеющая :

1) Строить графики. В системе используются три графика, первый – типа line chart, для отображения количества всех сообщений и ошибочных сообщений в разные промежутки времени(см рис.)

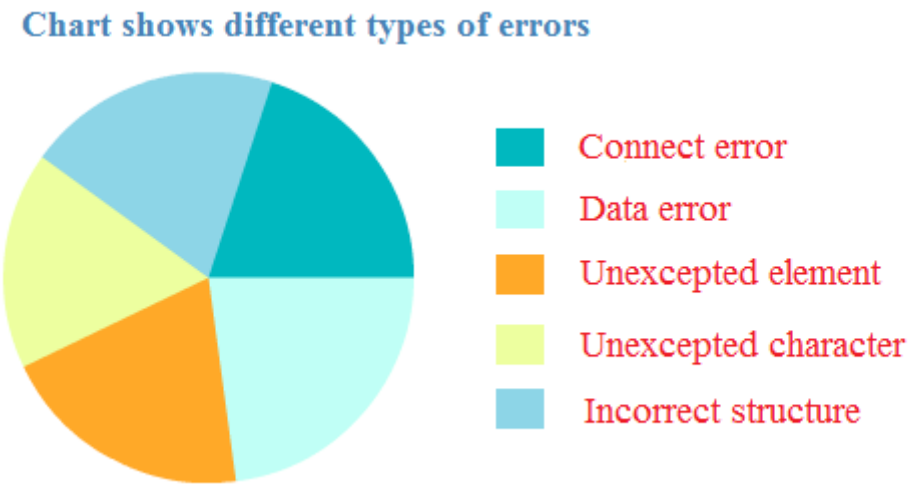


(Рисунок 4.График всех сообщений(синий) и ошибок(оранжевый))

Другие два графика типа pie chart: первый служит для классификации всех типов ошибок(см. рис.), второй – для визуализации соотношения количества ошибочных и верных сообщений (см. рис.)



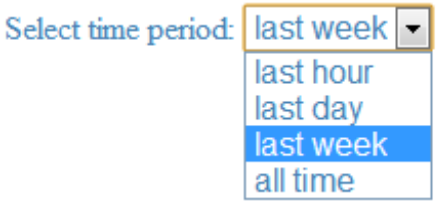
(Рисунок 5.)



(Рисунок 6.)

Все три графика являются live-графиками и запрашивают новые данные у бина каждые 10 секунд.

Можно делать выборку сообщений по следующим промежуткам времени: за последний час, последний день, последнюю неделю или за все время (с учетом удаления устаревших данных – сообщений старше 1 года).



(рисунок 7.)

2) Анализировать данные, пришедшие с веб-сервисов, за определенный промежуток времени. Все сообщения из периода выборки, попадают в таблицу (см. рис.) Если сообщение ошибочно, то кроме самого сообщения, система показывает сообщение об ошибке , а также место ошибки в сообщении (для типов ошибок Data error, Unexcepted element, Unexcepted character). Благодаря этому разработчик может существенно ускорить поиск багов и повысить производительность своего труда.

1 2 3

Type	Date	Message	Web service adress	Error message
Data error	2012-05-08 16:47:59	<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:tpp="http://einkaufsportal.telekom.de/tpp/"> <soapenv:Header/> <soapenv:Body> <tpp:itServiceLockResponse> <out>okdd</out> </tpp:itServiceLockResponse> </soapenv:Body> </soapenv:Envelope>	http://localhost:8088/mockitserviceOrderBasketSOAP?WSDL	Unmarshalling Error: Not a number: ok Location: line 5
Incorrect structure	2012-05-08 16:51:50	<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:tpp="http://einkaufsportal.telekom.de/tpp/"> <soapenv:Header/> <soapenv:Body> <tpp:itServiceLock> <purchaseOrder>?</purchaseOrder> <position>?</position> <leistungsZeile>?</leistungsZeile> <locked>?</locked> </tpp:itServiceLock> </soapenv:Body> </soapenv:Envelope>	http://localhost:8088/mockitServiceLockPO?WSDL	unexpected element (uri:"", local:"purchaseOrder"). Expected elements are <{}out>
Correct message	2012-05-08 19:39:24	<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:tpp="http://einkaufsportal.telekom.de/tpp/"> <soapenv:Header/> <soapenv:Body> <tpp:itServiceLockResponse> <out>111</out> </tpp:itServiceLockResponse> </soapenv:Body> </soapenv:Envelope>	http://localhost:8088/mockitServiceLockPO?WSDL	
Unexpected character	2012-05-09 16:15:53	<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:tpp="http://einkaufsportal.telekom.de/tpp/"> <soapenv:Header/> <soapenv:Body> <tpp:itServiceLockResponse> <out111</out> </tpp:itServiceLockResponse> </soapenv:Body> </soapenv:Envelope>	http://localhost:8088/mockitServiceLockPO?WSDL	Unmarshalling Error: Unexpected character '<' (code 60) excepted space, or '>' or "/>" at [row,col {unknown-source}]: [5,18]

1 2 3

(Рисунок 8.)

Заключение

В ходе работы были изучены различные технологии проектирования и разработки веб-сервисов: API для создания веб-сервисов JAX-WS и JAX-RS, их реализация Apache CXF, язык описания веб-сервисов WSDL, протокол обмена сообщениями SOAP.

Была реализована и встроена в реальный проект система мониторинга, использующая технологию JAX-WS для создания веб-сервисов и ее реализацию – библиотеку Apache CXF.

В дальнейшем систему возможно улучшить до полноценной системы мониторинга, умеющей определять входящий и выходящий трафик, проверку доступности веб-сервисов, возможность работы не только на клиенте, но и на сервере. Также хороший шаг к улучшению системы: хранение контента сообщений не в базе данных, а в файлах. В базе данных предполагается хранить лишь ссылки на эти файлы. Это уменьшит объем базы данных во много раз, а следовательно – увеличит скорость работы с ней.

Список литературы

[1] http://www.ibm.com/developerworks/ru/library/j-jws12/ (Статьи об Apache CXF)

[2] http://habrahabr.ru/post/140181/ (Основы JAX-RS)

[3] http://www.oracle.com/technetwork/articles/javase/index-140168.html (Java Architecture for XML Binding (JAXB))

[4] http://primefaces.org/ (PrimeFaces Ultimate JSF Component Suite)

[5] http://www.javaportal.ru/java/articles/java_Server_Faces.html (Введение в Java Server Faces)

[6] http://www.springsource.org/ (Spring Framework)

[7] http://www.wikipedia.org/

[8] http://www.hibernate.org/ (Hibernate - JBoss Community)

[9] http://www.w3schools.com/soap/ (SOAP Tutorial

[10] http://www.w3.org/TR/wsdl Web Services Description Language (WSDL) 1.1

[11] http://aggregate.tibbo.com/solutions/ (Aggregate – Система мониторинга и управления сетями)