General Adaptation Framework

(GAF)

"Framework for Semantic Adaptation of Maintenance Resources"

SmartResource **Tekes Project**
Deliverable D 1.2

Industrial Ontologies Group

Agora Center, University of Jyväskylä

July – October, 2004
Jyväskylä, Finland

IOG

GENERAL ADAPTATION FRAMEWORK (PART I)

Technical report

SmartResource: Proactive self-maintained resources in Semantic Web

2/24/2005

University of Jyväskylä

Agora Center

Author: Industrial Ontologies Group

Contact Information: e-mail: vagan@it.jyu.fi

Title: General Adaptation Framework

Work: Technical report

Status of document: working draft

Number of Pages: 45

Keywords: General adaptation framework, Semantic Web, Agent, Device, Human, Expert, Software Interface, Industrial Maintenance, Global Understanding Environment

Abstract: This document represents conceptual, operational and architectural characteristics of software components and their cooperation using family of UML diagrams. Report covers technical aspects of General Adaptation Framework and is dedicated to describe domain in semi formal way to allow unified understanding of all concepts related to adaptation. This report is a starting point of software development process and captures design features.

Abbreviations

OWL	Web Ontology Language
RDF	Resource Description Framework
RDFS	RDF Schema language
XML	eXtensible Markup Language

Contents

1 INTRODUCTION 5
1.1 TASKS AND GOALS 5
1.2 BACKGROUND 6
1.2.1 Framework for semantic adaptation of resources 6
1.2.2 Data integration 6
1.2.3 Software integration 7

2 DESCRIPTION OF CONCEPTS 8
2.1 SMARTRESOURCE 8
2.2 REAL WORLD RESOURCE 8
2.3 WEB SERVICE 8
2.4 HUMAN 9
2.5 ADAPTER 9
2.6 AGENT 9
2.7 GLOBAL UNDERSTANDING ENVIRONMENT 9
2.8 ADAPTATION 9
2.9 GENERAL ADAPTATION 9
2.10 PLACE OF SMARTRESOURCE 10
2.11 RDF 10
2.12 RSCDF 10
2.13 ONTOLOGY 11
2.14 GUN ADAPTER 11

3 ONTOLOGY DESIGN 13

4 SCENARIOS OF INTERACTION 16
4.1 SMARTRESOURCE INTERNAL SCENARIOS 17
4.1.1 Agent to RWR 18
4.1.2 RWR to Agent 21
4.1.3 SmartResource internal interoperation 26

5 ADAPTER SOFTWARE COMPONENT DESIGN FOR SEMANTIC ADAPTATION 27
5.1 ADAPTER – ABSTRACT REALIZATION 27
5.1.1 Adapter Class Diagram 27
5.1.2 Adapter creation 28
5.1.3 Protocol Class Diagram 29
5.1.4 Agent to RWR communication 30
5.1.5 Abstract view on semantic adaptation process 31

5.2 ADAPTER WITH CONCRETE REALIZATION 32
5.2.1 Concrete Adapter Class Diagram 32
5.2.2 Concrete Adapter creation 33
5.2.3 Concrete Protocol Class Diagram 33
5.2.4 Agent to Rwr communication with concrete realization 34
5.2.5 Semantic adaptation with concrete realization 34
5.2.6 Partitioned logic of semantic adaptation 35
5.2.7 Run-time concrete realization loading 38

6 ADAPTATION OF HUMAN, DEVICE AND WEB SERVICE USING GAF 40
6.1 HUMAN ADAPTATION 40
6.2 DEVICE ADAPTATION 40
6.3 WEB SERVICE ADAPTATION 40
6.3.1 W3C stack 40

REFERENCES 43

1 Introduction

Rapid development of new technologies and implementation of new innovations bring to industry new possibilities for conducting its business. Recently on-research stage technologies are available for implementation already. Wide data range wireless transition, increasing computation power and decrease the price of components are result of recent science achievements.

However today's state of affairs show us improvements of data processing and acquisition from one hand, from another hand it's still difficult to process data by intelligent software that allows integration of heterogeneous systems.

Data, represented in systems is in its own format, has no semantic description, often non-interoperable.

The main objective of general adaptation framework is to design generic approach for building resource adapters and development of appropriate ontologies for semantic adaptation.

Taking into account wide variety of possible resource types, data formats and ways of accessing and acquisition, adaptation of such resources in unified resource management environment is important development challenge.

1.1 Tasks and Goals

Tasks of stage:

- Development of general framework for semantic adaptation of resources.
- Development and implementation of semantic adapter for real world resource.

Goals of stage:

- To study approaches for semantic adaptation of resources; design generic software components for adaptation of different real world resources.

- Design, development and implementation of a prototype of ontology-based device adapter

1.2 Background

1.2.1 Framework for semantic adaptation of resources

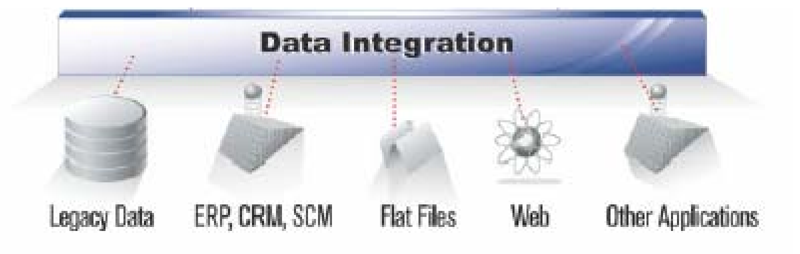
There is a variety of resources intended for integration into maintenance environment. Originally, as it thought, all resources were divided into three base classes: devices, services and humans. These resources represent real world objects, which should interact in some way. The adaptation of such resources in common sense lies in providing an environment for heterogeneous resources which would allow them to communicate in a unified way via standard protocol.

Originally the task of adaptation is extremely difficult and leads to the big challenge. There is a variety of organizations and projects working in field of application and data integration.

Basically we can consider adaptation from two sides: adaptation of heterogeneous applications and adaptation of heterogeneous data which is in different formats. Both types of integrations are intended to be implemented in prototype of general adaptation framework.

1.2.2 Data integration

Let’s suppose the tracing of data such as data lineage, mappings, and transformations. The picture bellow depicts different types of data which need to be integrated such as flat files, XML-based data, data from specific applications in specific format.



The integration process may include the following key functions:

- Extracting, transformation and loading – for building data warehouse or operation data stores and giving end-user/sources/applications possibility to proceed integrated data
- Data replication, to allow multiple heterogeneous servers and databases to share data in real time
- Data Synchronization – to allow the sharing of data between servers and remote devices when connectivity is temporary

  It’s intended in the term of this project to investigate and develop adaptation framework for extracting data and transformation it into specific designated format.

1.2.3 Software integration

Application integration – another part of general adaptation task. The data is generated by different resources with specific applications. Considering this part of integration we can distinguish following application specific features:

- application functions
- application APIs
- application interfaces

All variations of these features have effect on process of adaptation and architecture of adaptation framework.

2 Description of concepts

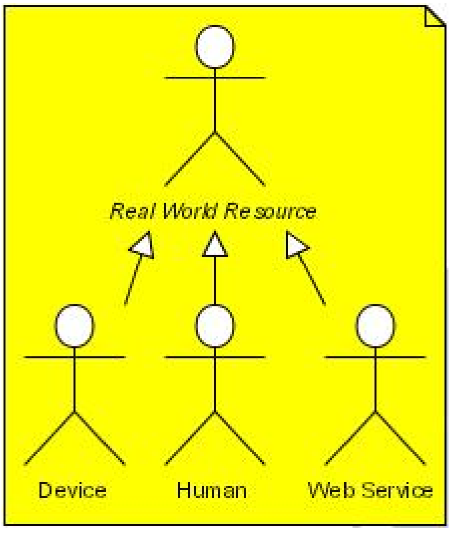
Definition of concepts is given in context of Adaptation task thus some definitions can expand ones given in previous project papers.

2.1 SmartResource

Under SmartResource we will understand conjunction of Real World Resource (RWR), Adapter and Agent. Concept SmartResource represents exactly one RWR. It has one Adapter and one Agent

2.2 Real World Resource

Real World Resource is a complex concept compounded from first part software component which provides access interface to second part real world entity. In wide sense notion of RWR is to represent some entity in real world for which adaptation framework can be applied. In context of industrial self maintenance goals of this project real world entity is Device, Human or Web Service.



2.3 Web Service

Web Service – WEB based software application that performs specific functions for anyone ordered this service [SWGuide]. Service class is subset of RWR.

2.4 Human

Human – any person (expert, operator, dispatcher) who interacts with other smart resources. Human class is subset of RWR.

2.5 Adapter

Adapter is a software component which provides bidirectional bridge between software component specific interface of RWR and General Interface to Agent.

2.6 Agent

Agent is a software component which represents interests of RWR in Semantic Web environment in wide sense, and in Global Understanding eNvironment (GUN) in sense of this project.

2.7 Global Understanding eNvironment

Global Understanding Environment – virtual environmental with appropriate architecture for unified interaction between Smart Resources

2.8 Adaptation

Adaptation in wide sense is a process of enabling RWR to participate in GUN. This definition indicates requirements to functionality of both components of SmartResource - Adapter and Agent; requirements to protocol of interaction between Adapter and Agent, Agent to Agent; requirements to overall infrastructure of GUN.

Adaptation in narrow sense is a process of enabling interaction between RWR and Agent through Adapter.

2.9 General Adaptation

All consideration of previous subchapter remains valid. Term “General” points on combined wide sense of Adaptation process and wide sense of RWR notion.

Two main layers are planned to use for general adaptation framework design:

The fist is well thought-out object-oriented design, based on design patterns. This layer is aimed to elaborate the architecture and components interaction for reuse of already developed components in adapter construction process. In other words, the adapter should be split into elementary “bricks”. The components of adapter can be divided into reusable components, which are common for certain types of resources and into specific ones, which might cover resource-specific parts.

Second layer concerns the design of semantic annotation for process of assembling the adapter. This aims at minimization of the human involving into process of adapter assembling. Now it’s impossible to omit human’s involving in assembling process at all, but with well designed ontology and assembling framework there is possibility to ease this process. We distinguish two kinds of ontologies: first ontology which describes semantics of problem domain – so-called upper ontology, and concrete ontologies which describe semantics of characteristics of particular resources. Every resource should be semantically annotated to provide knowledge for inference.

2.10 Place of SmartResource

We can call our research SmartResource centric. In higher abstract level we will deal with ad-hoc network of SmartResources.

2.11 RDF

The Resource Description Framework (RDF) is a framework for representing information in the Web [RDF]. It is intended to integrate a variety of application using XML for syntax and URIs for naming [SemanticWeb].

2.12 RscDF

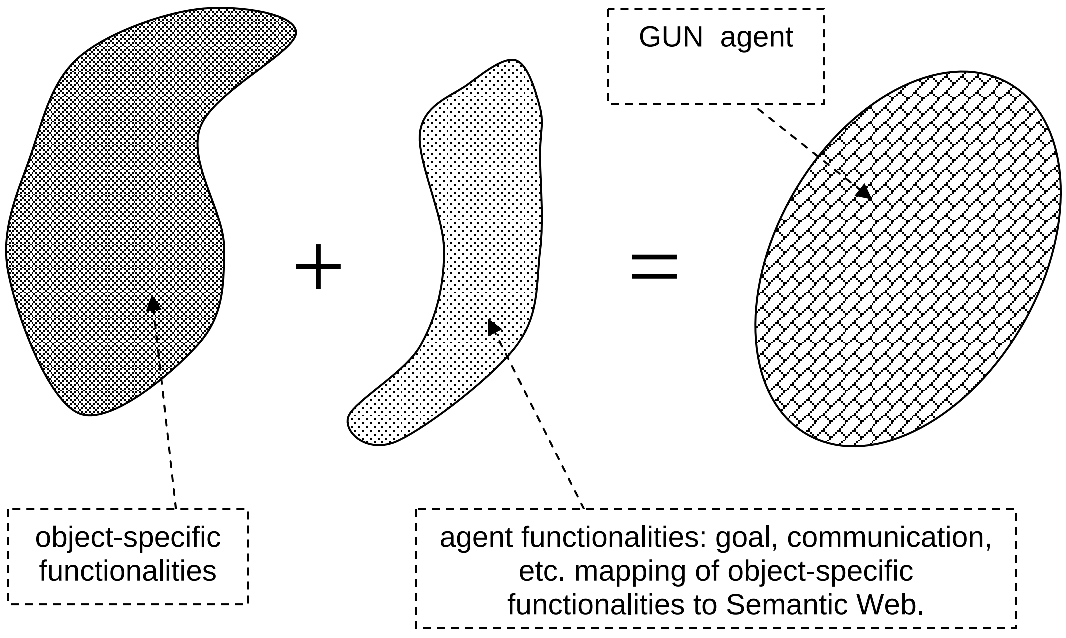
Resource State Condition Description Framework – enhancement for standard RDF, which reflects specifics of industrial domain

2.13 Ontology

Ontology are about vocabularies and their meaning, with explicit, expressive, and well-defined semantics. [SWGuide]

2.14 GUN Adapter

The concept of GUN (Global Understanding eNvironment) Adapter assumes an adaptation of every object from physical world to Semantic Web environment. GUN Adapter is represented by integrated software/hardware components, which on the one hand implement object-specific functionalities and on the other hand – common for whole Semantic Web environment functionalities. The Adapter translates interaction activities from device-specific format to Semantic Web one and vice versa. Adapter also supplements real-world object with agent functionality, implicit purpose of the object becomes explicit goal of an agent.


GUN agent
+
=
object-specific functionalities
agent functionalities: goal, communication, etc. mapping of object-specific functionalities to Semantic Web.

The ideal GUN Adapter must adapt to a specific object automatically. The set of GUN agents can be joined into cluster (OntoShell) [OntoShell] and the cluster will be represented for external world as a single entity. Example: industrial plant GUN agents (adaptated field devices) are joined into a cluster and other plants sense it as a single entity [GUN].

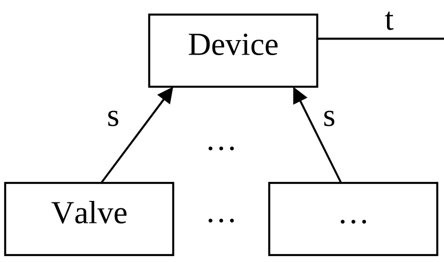
As for implicit purpose of the object we can remember pills: they were manufactured for certain diseases, has strict application instructions. There can be a supplier of this product – some store, method, price and scope of delivery, business description. If to supplement the pills to the GUN agent and place it in some environment that supports such agents owners of the pills can forget about this object – agent will take care about it.

Present Web resources don’t have their purpose explicit: who can find it, what should be noticed. OntoShell is an active resource; Adapter supplements the passive resource with active functionalities. As a result Semantic Web is populated by active, goal-oriented agents.

3 Ontology design

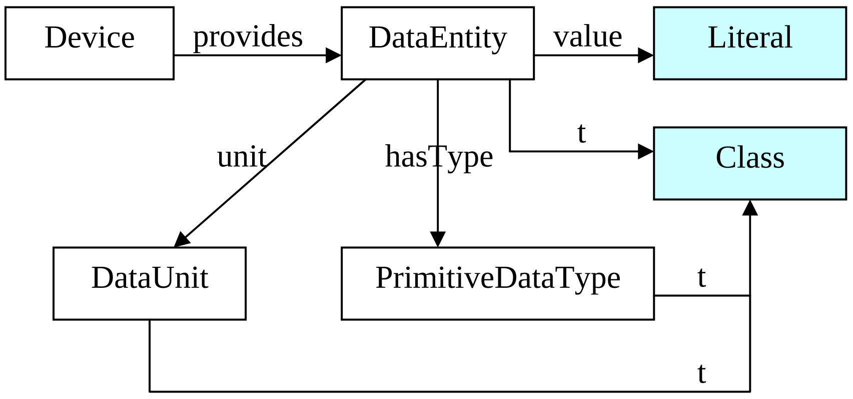
The following ontologies must be developed:

1. 1. Industrial Devices ontology, which will include metadata about industrial devices.



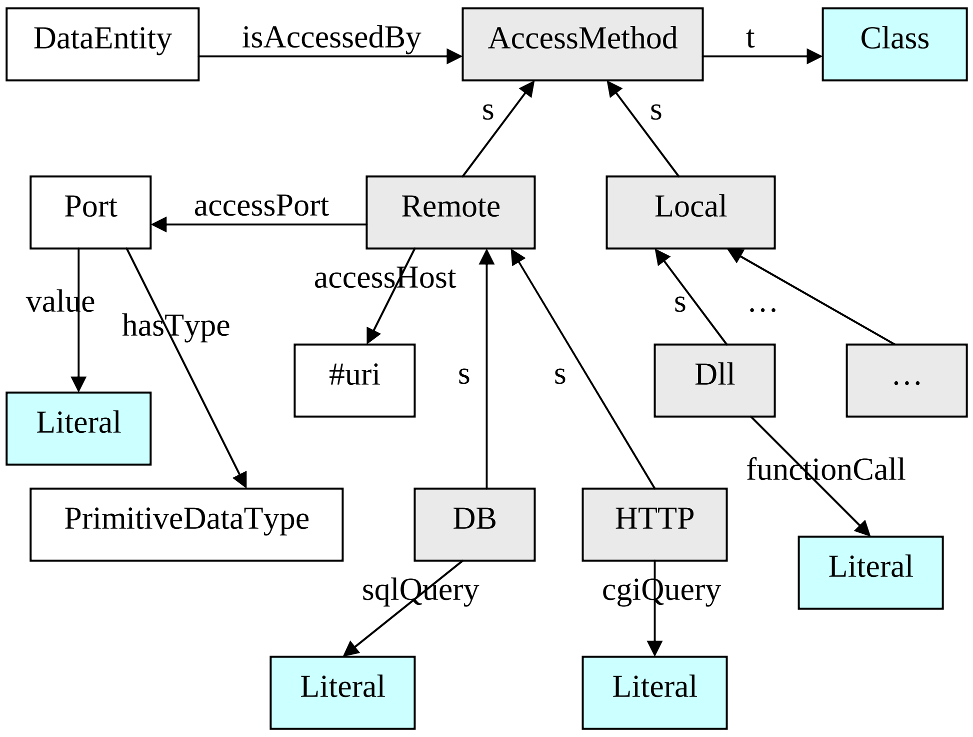
Industrial Devices upper ontology

2. Industrial Device Data ontology. It will reflect the requirements for data entities provided by Field Device. The ontology might look like:



Industrial Device Data upper ontology

3. Industrial Device Data Access Methods ontology. It will classify all existent methods of programming access to Field Device data. Example of such ontology:

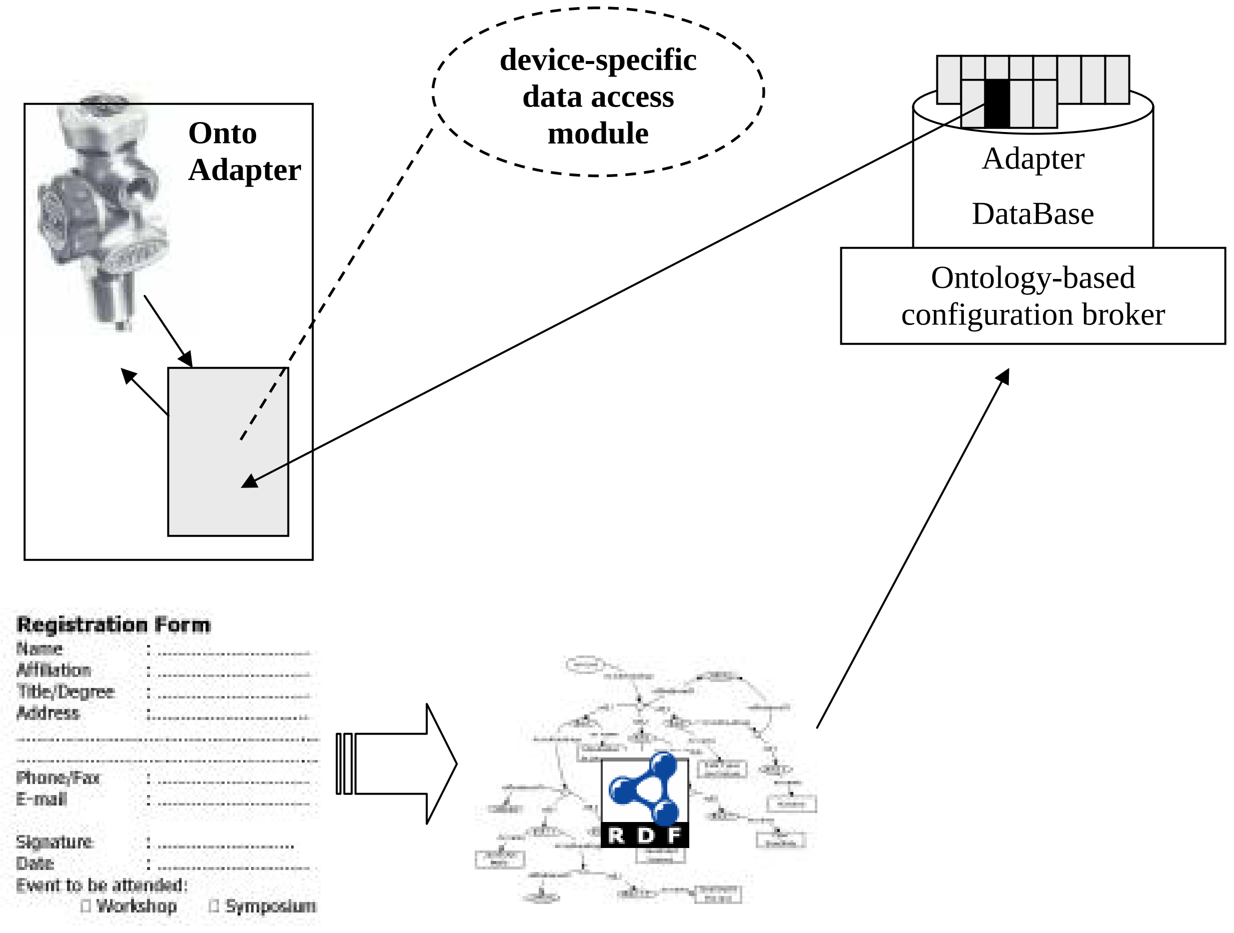


Industrial Device Data Access Methods ontology

As we can see, all the ontologies depicted above have relations one to other. The last ontology – Industrial Device Data Access Methods – is necessary just for configuration of Adapter. When somebody wants the Adapter to access data coming from some industrial device he must select appropriate AccessMethod from this ontology. After the method has been defined the additional attributes correspondent to the method must be defined too. If, for instance, user selected DB class as access method to the value of temperature in the valve, he must define an sqlQuery, which retrieves necessary from database. When all necessary data has been provided in the form correspondent RDF-instance is created. This RDF-instance is sent to software module, which tailors Adapter for specific industrial device. Semantic annotation of such data allows to compose Adapter automatically.

Thus, for every specific method of access to device data must be implemented a piece of code. If suddenly user while configuration of Adapter doesn't find appropriate access method in the ontology it means that correspondent piece of code haven't been implemented yet. In this case Adapter has to be configured for this type of device.

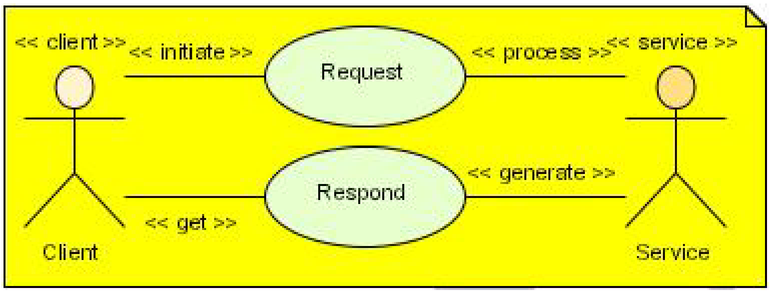
The process of Adapter configuration can be depicted as the following:



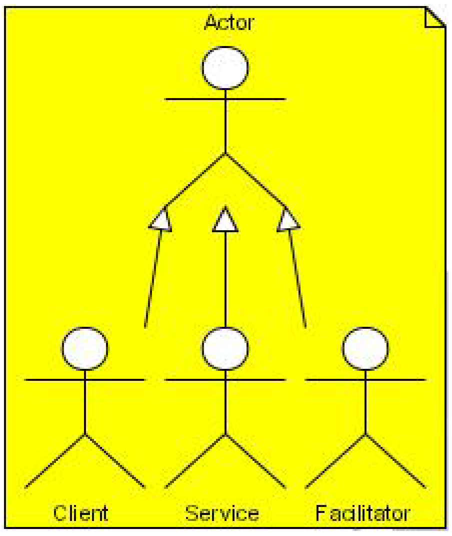
Process of Adapter configuration

4 Scenarios of Interaction

Main purpose of scenario is to capture features of interoperation in sense of Actor's roles and methods of communicational infrastructure. All scenarios are considered using client-service paradigm of communication. Essential part of this paradigm is a request-response pattern of communication Operational logic of semantic adaptation is encapsulated in request and response objects. Scenarios are presented using UML Use Case diagrams in which our project software components are represented as Actors. And methods of underlying connection infrastructure are represented as usecases. To distinguish usecases which are used to depict Adapter functionality in this chapter they are referred as scenarios. Figure shows reference Use Case diagram of client-service scenario.



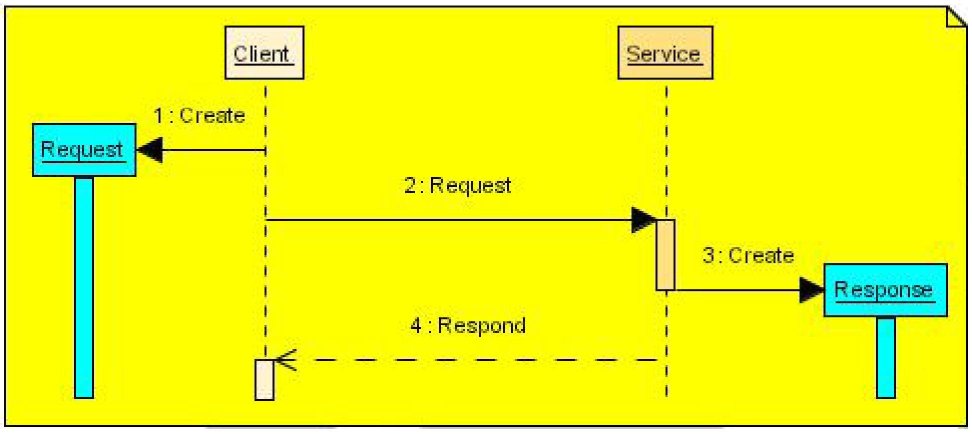
Agent, Adapter and Resource became Actors and have semantic of some of three roles depending on context of interaction. Taxonomy of Actor concept is depicted in figure



These roles are

- Client is an Actor which usually initiate a communication scenario making request to other actors
- Service processes and generate responses on clients requests
- Facilitator performs an action either client or server role on behalf of another actor.

Sequence diagram of typical client-service interaction is illustrated on figure .



4.1 SmartResource internal scenarios

Before modeling of Adapter functionality firstly this subchapter defines internal limited by SmartResource concept scenarios of interaction between components such as Agent, RWR and Adapter without considering cases when interaction of these components is initiated by other resources.

The main idea is that

- Adapter represents an Agent for a RWR in scenarios where RWR initiates communication.
- Adapter represents a RWR for an Agent in scenarios where Agent initiates communication.

Thus Adapter on the level of communication between Agent and RWR performs role of facilitator.

One more comment that Agent to RWR and RWR to Agent scenarios are a reverse from each other. But in report all description is repeated to explicitly define interoperation among SmartResource components because of some nuances.

4.1.1 Agent to RWR

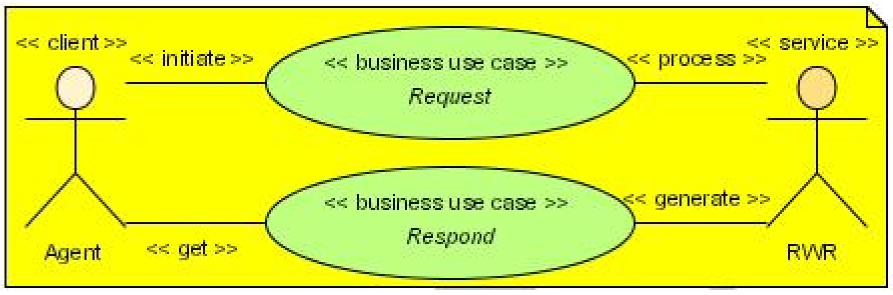
Scenario name: Agent2RWR business view

Scenario: Agent initiates communication by sending a request to RWR. RWR processes request to generate response. RWR responds to request of Agent by sending a response.

Initiator: Agent

Service: RWR

Scenario description: *Request* and *Respond* are usecases of delivering request and response messages over communication channel. *Request* and *Respond* are abstract usecases because of existing of Adapter which mediates communication.



Agent to Adapter

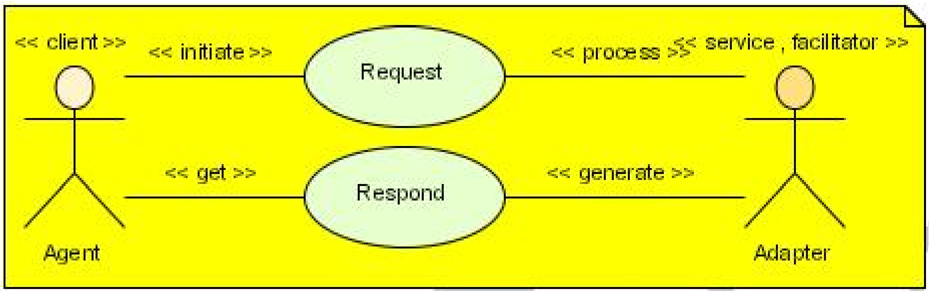
Scenario name: Agent2Adapter

Scenario: Agent initiates communication by sending a request to Adapter. Adapter processes request to generate response. Adapter responds to request of Agent by sending a response. Adapter represents an RWR in case when Agent requests for interaction with RWR. Thus Adapter is in role of facilitator on behalf of RWR. However Adapter can process Agents requests on behalf of itself.

Initiator: Agent

Service: Adapter

Scenario description: Request and Respond are usecases of delivering request and response messages over communication channel.



Adapter to RWR

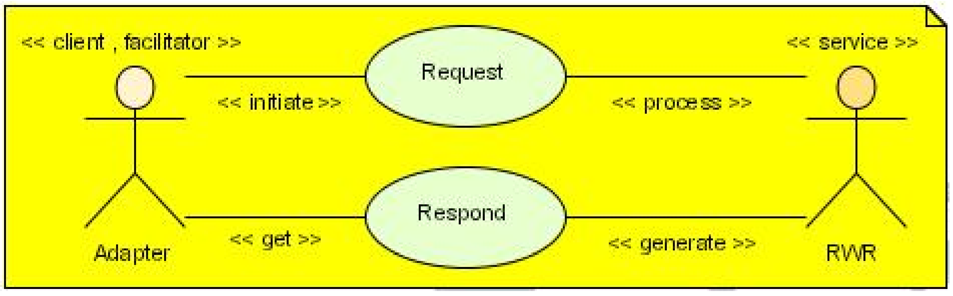
Scenario name: Adapter2RWR

Scenario: Adapter initiates communication by sending a request to RWR. RWR processes request to generate response. RWR responds to request of Adapter by sending a response. Adapter is always in role of facilitator in sense of operating on behalf of Agent and in role of client in sense of initiating communication with RWR.

Initiator: Adapter

Service: RWR

Scenario description: Request and Respond are usecases of delivering request and response messages over communication channel.



Agent to RWR communication

Scenario name: Agent2RWR

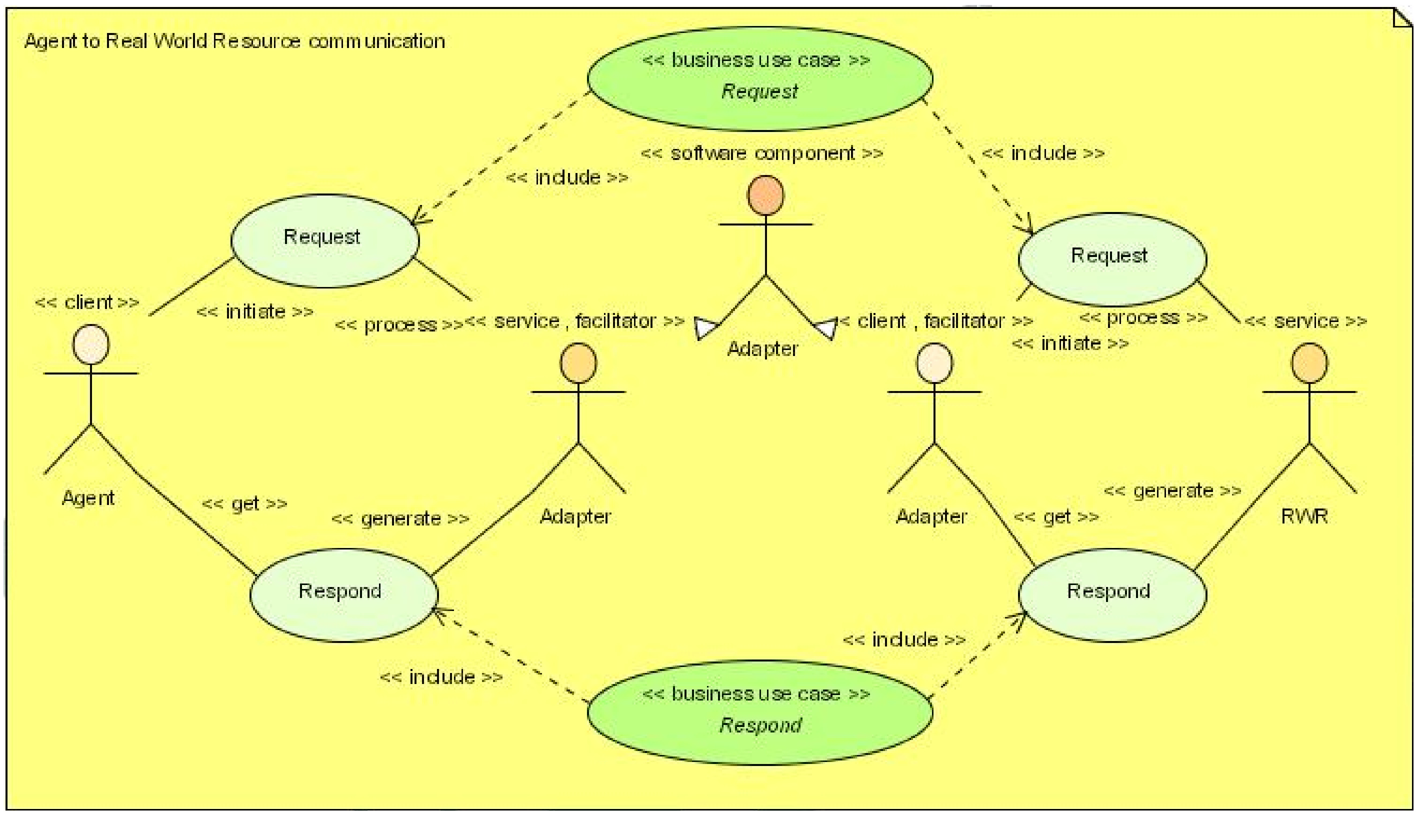
Scenario: Agent initiates communication by sending a request to RWR. To do this accordingly toAgent2Adapter scenario Agent requests Adapter which operates as service on behalf of RWR. Then Adapter forwards as client request to RWR on behalf of Agent accordingly to Adapter2RWR scenario. RWR processes request to generate response. RWR responds to request of Agent by sending a response. To do this accordingly to RWR2Adapter scenario RWR responds to Adapter which represents Agent and operates as client. Then Adapter forwards response to Agent on behalf of RWR operating as service.

Initiator: Agent

Facilitator: Adapter

Service: RWR

Scenario description: *Request* and *Respond* are usecases of delivering request and response messages over communication channel. *Request* and *Respond* are abstract usecases because of existing of Adapter which mediates communication. *Request* includes Agent2Adapter scenario Request and Adapter2RWR scenario Request. *Respond* includes RWR2Adapter scenario Respond and Adapter2Agent scenario Respond. Adapter as a software component appears in roles of client and service performing also representative role of facilitator.



4.1.2 RWR to Agent

Scenario name: RWR2Agent business view

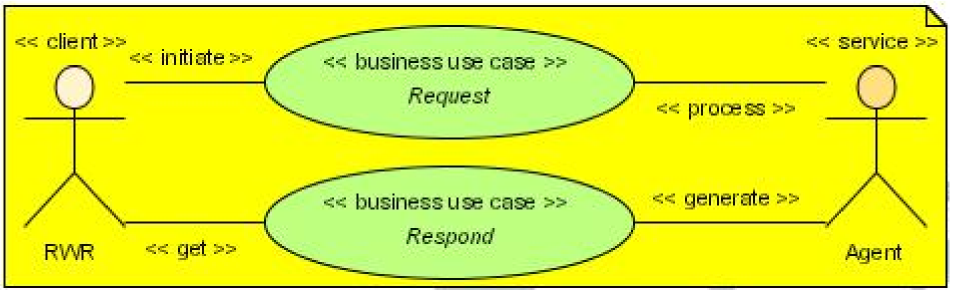
Scenario: RWR initiates communication by sending a request to Agent. Agent processes request to generate response. Agent responds to request of RWR by sending a response.

Alternative: RWR initiates communication by sending a request to Agent. Agent processes request. In such case interoperation goes without feedback link of response.

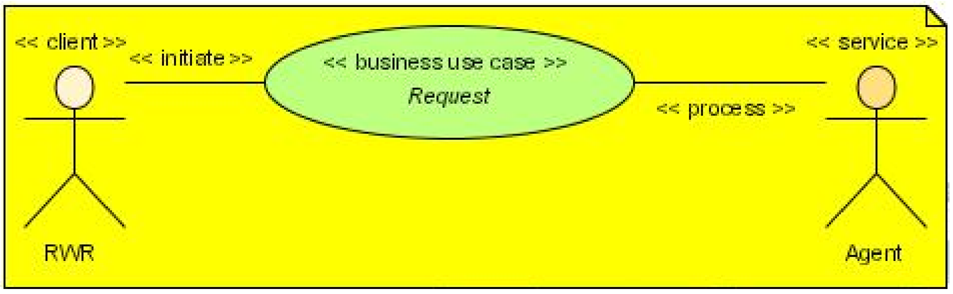
Initiator: RWR

Service: Agent

Scenario description: *Request* and *Respond* are usecases of delivering request and response messages over communication channel. *Request* and *Respond* are abstract usecases because of existing of Adapter which mediates communication.



Alternative scenario: RWR initiates communication by sending a request to Agent. Agent processes request. In such case interoperation goes without feedback link of response.



RWR to Adapter

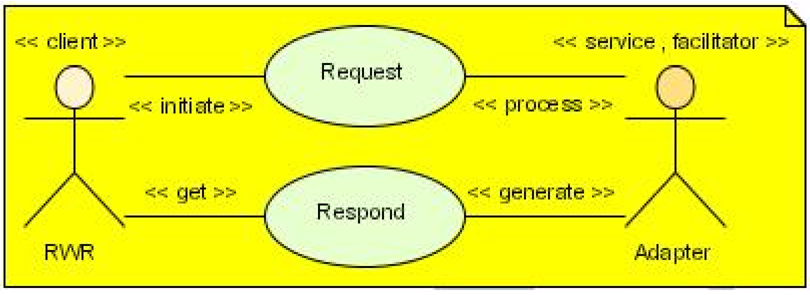
Scenario name: RWR2Adapter

Scenario: RWR initiates communication by sending a request to Adapter. Adapter processes request to generate response. Adapter responds to request of RWR by sending a response. Adapter is always in role of facilitator in sense of operating on behalf of Agent and in role of service in sense of processing the requests of RWR.

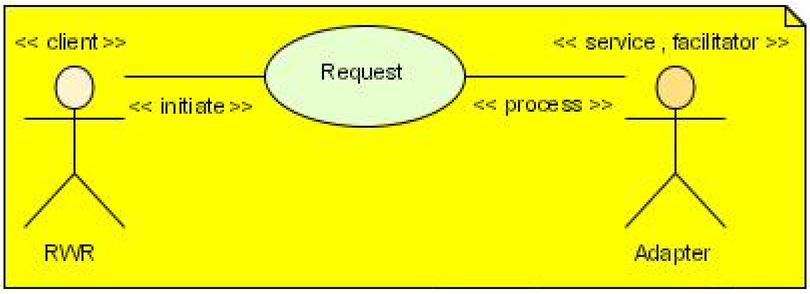
Initiator: RWR

Service: Adapter

Scenario description: Request and Respond are usecases of delivering request and response messages over communication channel.



Alternative scenario: RWR initiates communication by sending a request to Adapter. Adapter processes request.



Adapter to Agent

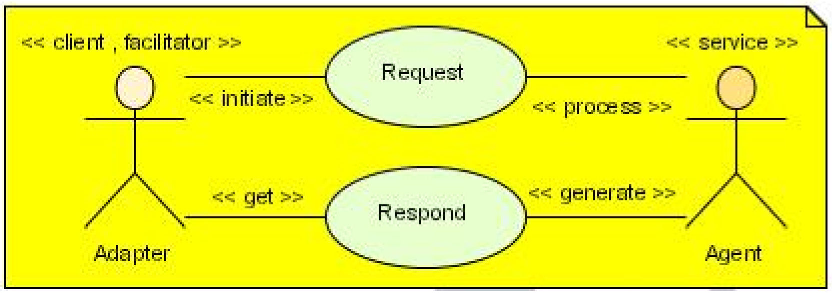
Scenario name: Adapter2Agent

Scenario: Adapter initiates communication by sending a request to Agent. Agent processes request to generate response. Agent responds to request of Adapter by sending a response. Adapter operates on behalf of an RWR and thus in facilitator role. However Adapter can generate request on behalf of itself.

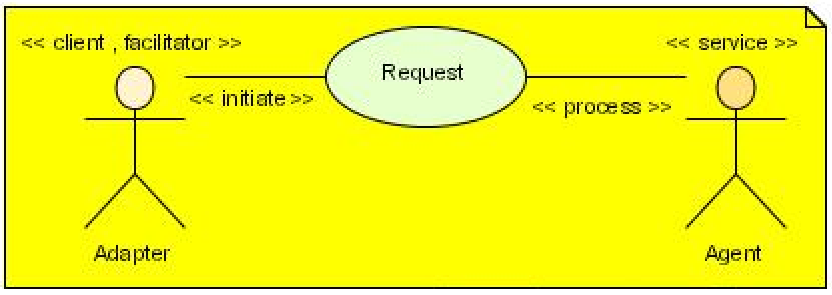
Initiator: Adapter

Service: Agent

Scenario description: Request and Respond are usecases of delivering request and response messages over communication channel.



Alternative scenario: Adapter initiates communication by sending a request to Agent. Agent processes request.



RWR to Agent communication

Scenario name: RWR2Agent

Scenario: RWR initiates communication by sending a request to Agent. To do this accordingly toRWR2Adapter scenario RWR requests Adapter which operates as service on behalf of Agent. Then Adapter forwards as client request to Agent on behalf of RWR accordingly to Adapter2Agent scenario. Agent processes request to generate response. Agent responds to request of RWR by sending a response. To do this accordingly to Agent2Adapter scenario Agent responds to Adapter which represents RWR and operates as client. Then Adapter forwards response to RWR on behalf of Agent operating as service.

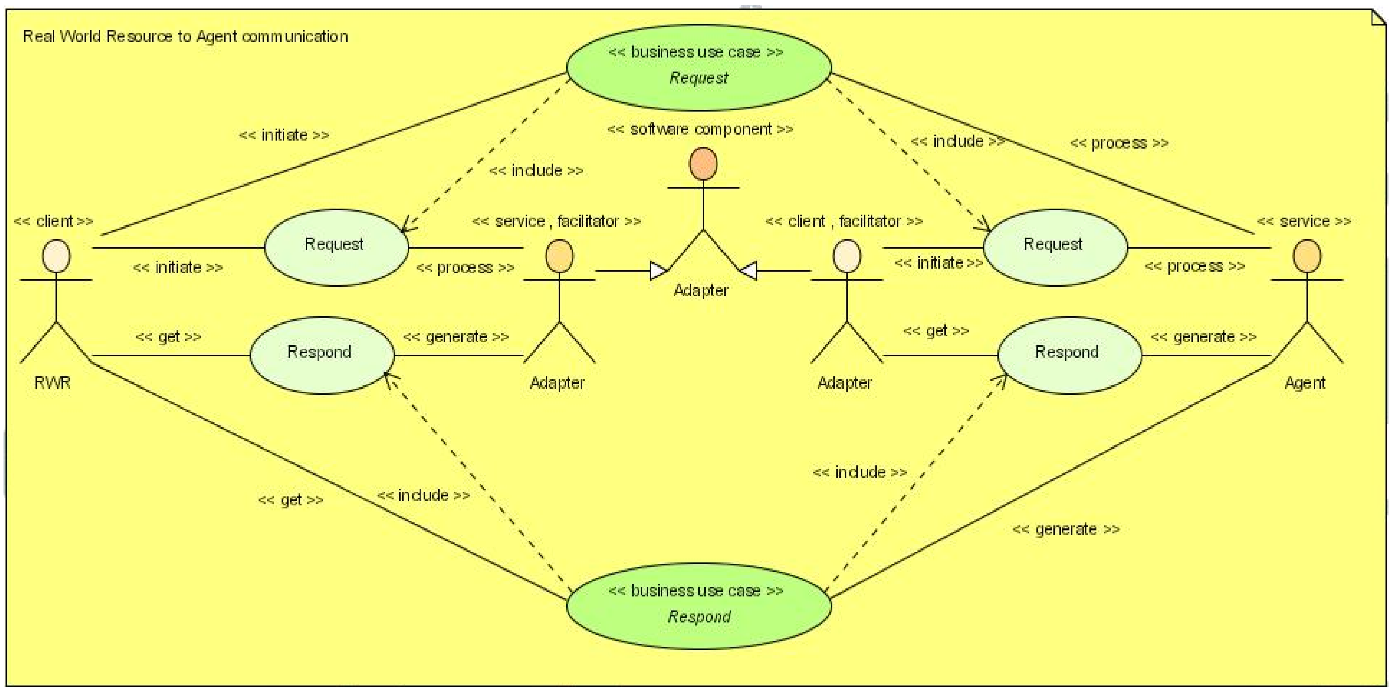
Initiator: RWR

Facilitator: Adapter

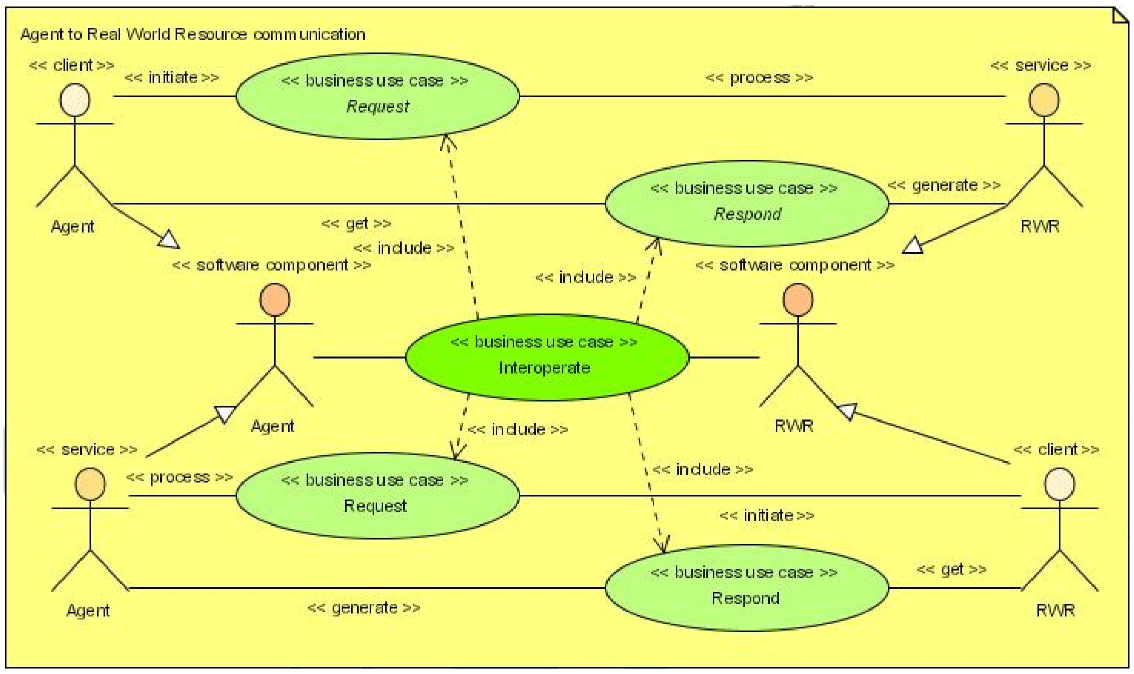
Service: Agent

Scenario description: *Request* and *Respond* are usecases of delivering request and response messages over communication channel. *Request* and *Respond* are abstract usecases because of existing of Adapter which mediates communication. *Request* includes RWR2Adapter scenario Request and Adapter2Agent scenario Request. *Respond* includes Agent2Adapter scenario Respond and Adapter2RWR scenario Respond. Adapter as a software component appears in roles of client and service performing also representative role of facilitator.

Alternative scenario: RWR initiates communication by sending a request to Agent. To do this accordingly toRWR2Adapter scenario RWR requests Adapter which operates as service on behalf of Agent. Then Adapter forwards as client request to Agent on behalf of RWR accordingly to Adapter2Agent scenario.



4.1.3 SmartResource internal interoperation



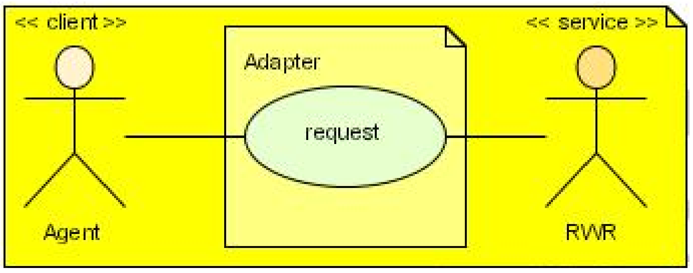
5 Adapter software component design for semantic adaptation

We consider at the beginning of the chapter only case when an Agent initiates communication to a Resource. So everywhere interoperation is considered from an Agent point of view. In other cases comments are given.

5.1 Adapter – abstract realization

Form Adapter as a software component only one functionality is required, that is to provide unified interface to resource. Agent performs requests to Resource to get data about its state. So Adapter is a software component which serves one Agent to access one Resource. Adapter as software component is referred further as Adapter application.

This subchapter captures process of requesting Resource. Figure represents this using Use Case diagram.



On this figure Agent is an external actor

5.1.1 Adapter Class Diagram

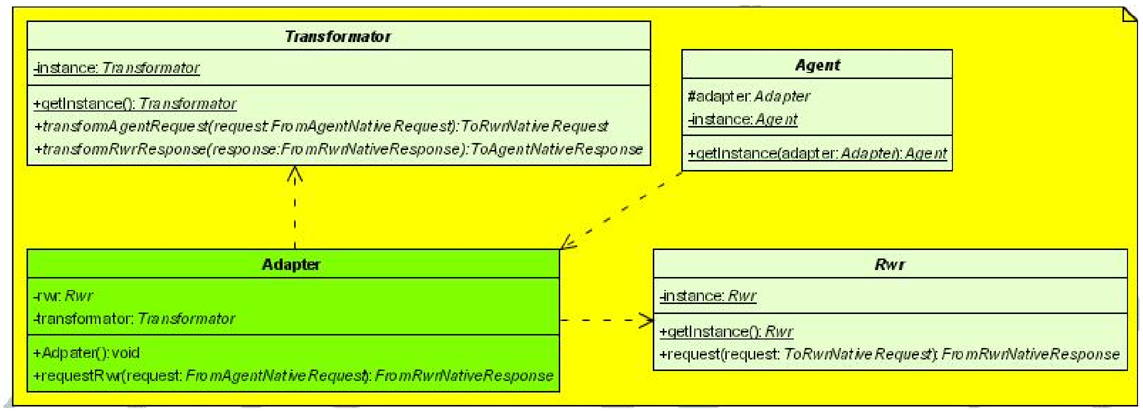
“Adapter” is a class which implements adaptation functionality and in this case represents whole Adapter application which creates instance “adapter”. The Adapter class is an implementation of “Facade” pattern of software design [1]. This class contains reference to an instance “rwr” of “Rwr” class and an instance “transformator” of “Transformator” class.

“Agent” is a class to represent external actor Agent in a role of client for the Adapter application. This is an abstract class and is a base class for creating concrete class to implement functionality of interaction with external actor Agent to perform requests to external actor RWR. Agent class contains a reference to Adapter class to be able to invoke method requestRwr.

“Rwr” is a class to represent external actor RWR in a role of service for the Adapter application. This is an abstract class and is a base call for creating concrete class to implement functionality of interaction with external actor RWR to process requests of external actor Agent and to generate response to the actor Agent.

Agent and Rwr are the “border” classes in sense of capturing all important features of external to Adapter application actors.

“Transformator” is an abstract class to define also in abstract way transformation methods of from Agent request to Rwr request and from Rwr response to Agent response transformation. This class is dedicated to define a starting point for subsystem which encapsulates semantic adaptation.


Transformator
-instance: Transformator
+getInstance(): Transformator
+transfomAgentRequest(request: FromAgentNativeRequest):ToRwrNativeRequest
+transfomRwrResponse(response:FromRwrNativeResponse):ToAgentNativeResponse
Agent
#adapter: Adapter
-instance: Agent
+getInstance(adapter: Adapter): Agent
Adapter
-rwr: Rwr
-transfomator: Transformator
+Adpater():void
+requestRwr(request: FromAgentNativeRequest): FromRwrNativeResponse
Rwr
-instance: Rwr
+getInstance(): Rwr
+request(request: ToRwrNativeRequest): FromRwrNativeResponse

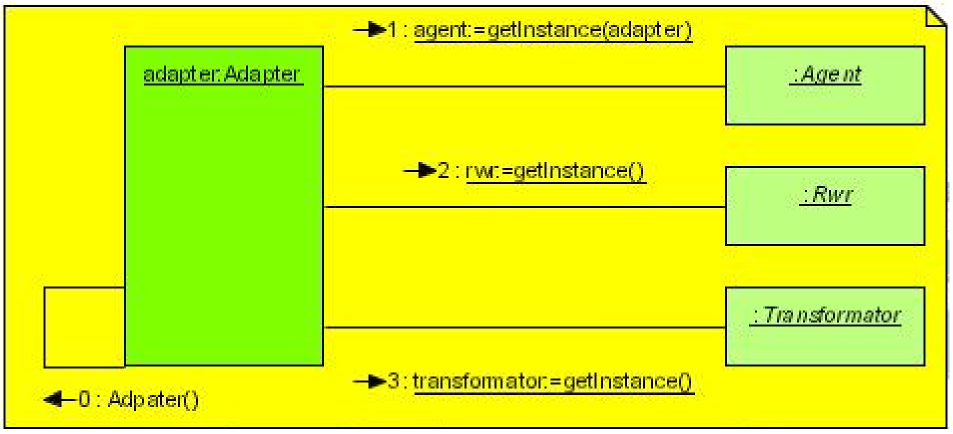
5.1.2 Adapter creation

As it was said before Adapter is a “Facade” class for whole application, this means that instance of Adapter has references to instances of other “border” classes Agent and Rwr and “utility” class Transformator.

So at the point of Adapter instance creation all other instances should be created too. Agent, Rwr and Transformator classes are designed accordingly to “Singleton” pattern of software design. This pattern ensures that only one instance of a class exists in the system. Thus we meet logic of the domain that adapter serves one agent and one resource.

getInstance is a static method in all classes. As Agent, Rwr and Transformator are abstract classes, they cannot have instances. So this method returns instances of the classes which extend these abstract ones and implement concrete logic. Process of creation instances of subclasses and code of this method is described by subchapter “Run-time Adapter configuration”

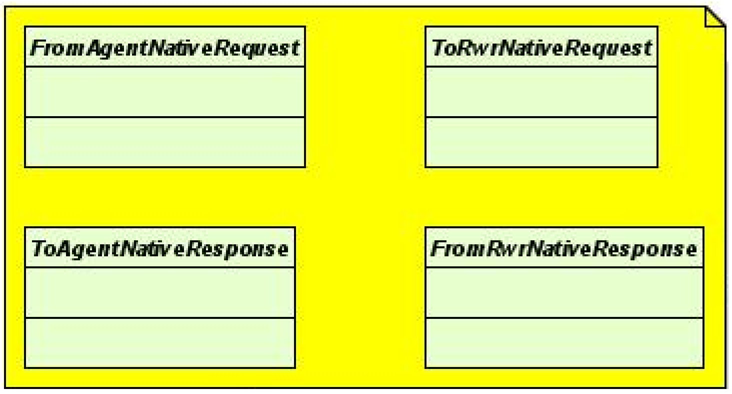
Figure shows collaboration diagram of Adapter creation. If Adapter as class is a part of Agent application then creation is performed by calling constructor of Adapter class from some point of Agent application. If Adapter class is standalone application then it has static method main as entry point and creates instance of itself in this method by calling a constructor too. Then instance of Adapter class invokes getInstance methods of Agent, Rwr and Transformator classes to serve interaction between them. Instance of Agent class gets reference to instance of Adapter class to be able to invoke Adapter class method requestRwr.



5.1.3 Protocol Class Diagram

Figure introduces abstract classes of requests and responses to define in abstract way process of semantic adaptation of external actor Agent to external actor RWR. This is achieved by including four classes without fields and methods. Notion of these classes is to support signature of request and transformation methods in Adapter, Agent, Rwr and Transformator classes.

Concrete realization of request and response logic depends from nature of external actors Agent and RWR and this logic is encapsulated in subclasses which are defined in subchapter “Adapter with concrete realization”. So informally these classes serve as dummies on this stage of software design.



5.1.4 Agent to RWR communication

Sequence of methods invocation to perform request from Agent class instance to Rwr class instance is shown in figure. The agent is an instance of the class Agent, the adapter is an instance of the class Adapter and the rwr is an instance of the class Rwr.

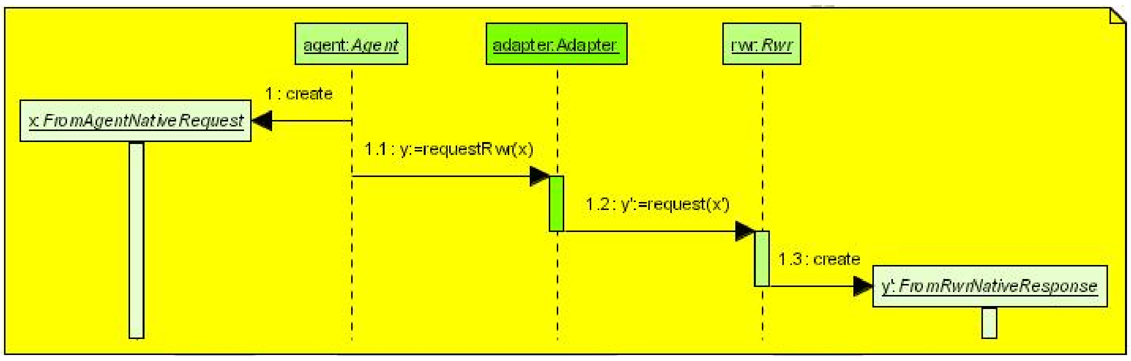
1 The agent creates instance x of the class FromAgentNativeRequest

1.1 The agent invokes method requestRwr using reference of to the adapter (see subchapter 4.1.2) passing x as a parameter and getting an instance y of the class ToAgentNativeResponse

1.2 The adapter after a stage of request transformation (see next subchapter) has an instance x’ of the class ToRwrNativeRequest and invokes method request using reference to the rwr passing x’ as a parameter and getting an instance y’ of the class FromRwrNativeResponse

1.3 The rwr on method request invocation generates an instance y’ of the class FromRwrNativeResponse

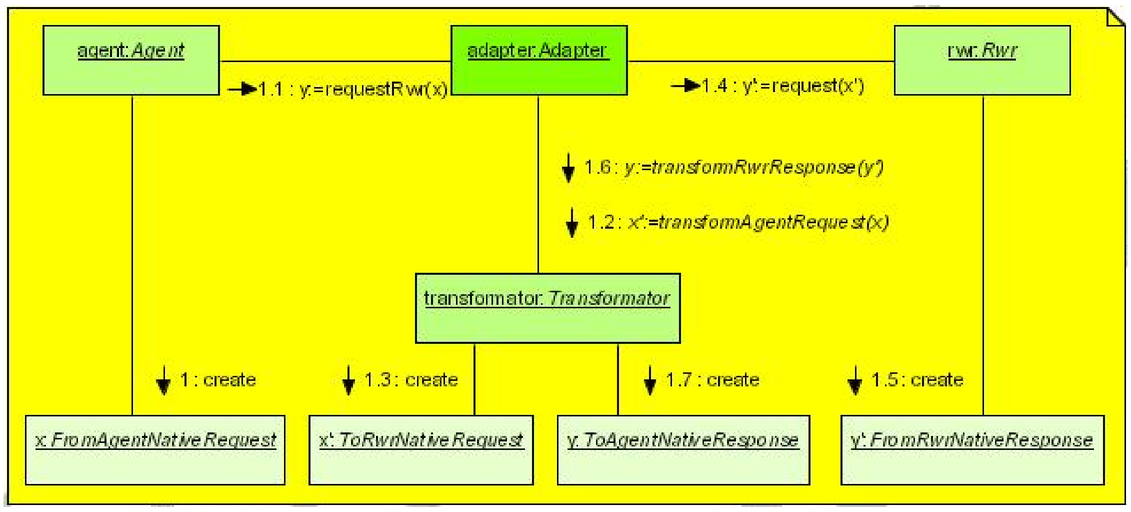
More precisely this abstract process of the semantic adaptation is described in next chapter



5.1.5 Abstract view on semantic adaptation process

Figure gives abstract view using Collaboration diagram on process of semantic adaptation. The agent, adapter, rwr, x and y' are the same instances as in Figure in previous chapter. The transformator is an instance of the class Transformator and the x' is an instance of the class ToRwrNativeRequest and the y is an instance of the class ToAgentNativeResponse.

1 The agent creates x
1.1 The agent invokes method requestRwr of the adapter passing x as a parameter
1.2 The adapter invokes method transformAgentRequest of the transformator forwarding x as a parameter
1.3 The transformator creates and returns x' performing semantic adaptation of agent native request model which is encapsulated in x to rwr native request model which is encapsulated in x'
1.4 The adapter invokes method request of the rwr passing x' as a parameter
1.5 The rwr creates y' and returns it to the adapter as response
1.6 The adapter invokes method transformRwrResponse of the transformator forwarding y' as a parameter
1.7 The transformator creates and returns y performing semantic adaptation of rwr native response model which is encapsulated in y' to agent native response model which is encapsulated in y

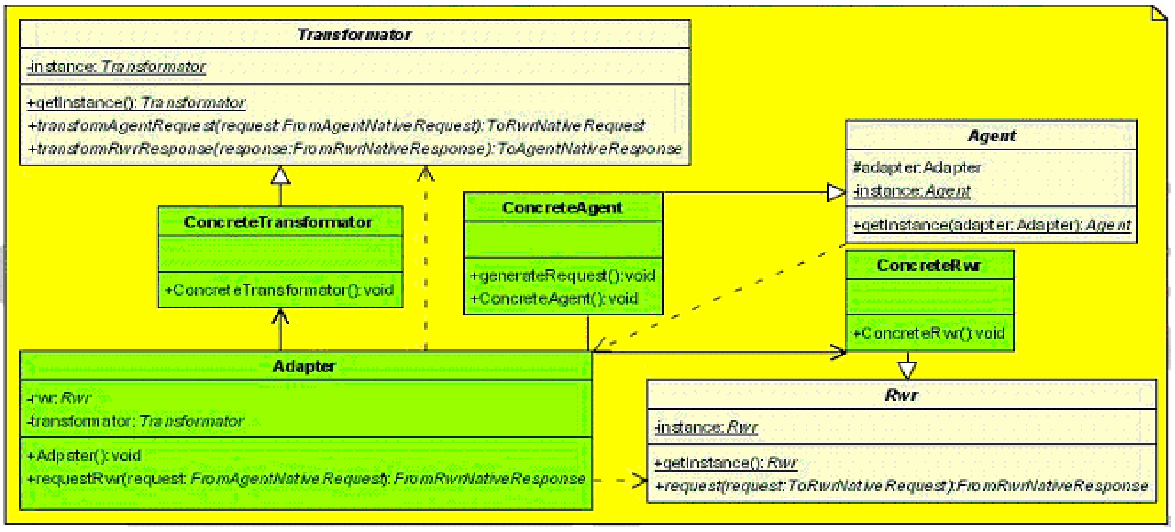


So as you can see instances of protocol classes are real bearers of the Agent to RWR request and response semantics.

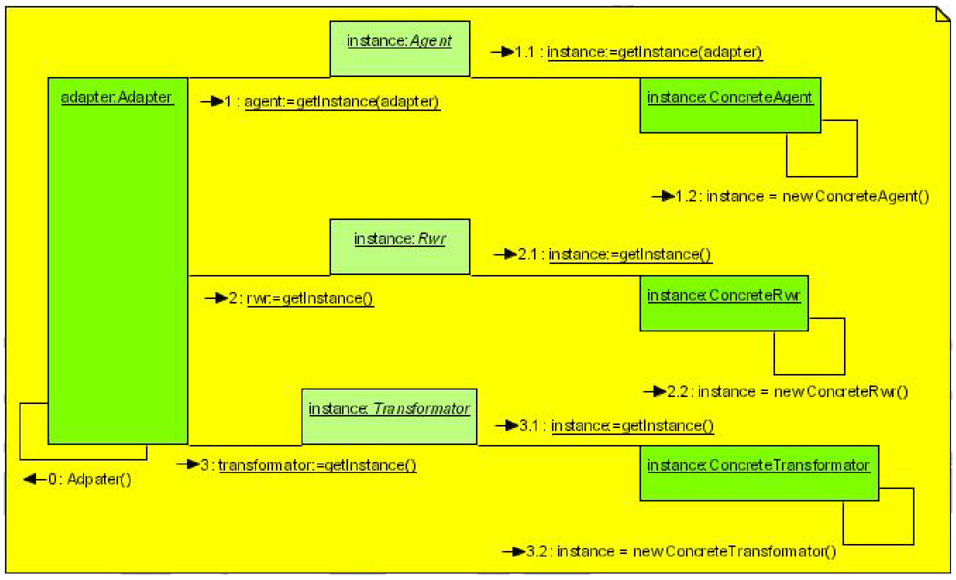
Creation of the instances of the concrete protocol classes which capture semantic of request and response interfaces of external actors Agent and RWR is a subject of the next subchapter.

5.2 Adapter with concrete realization

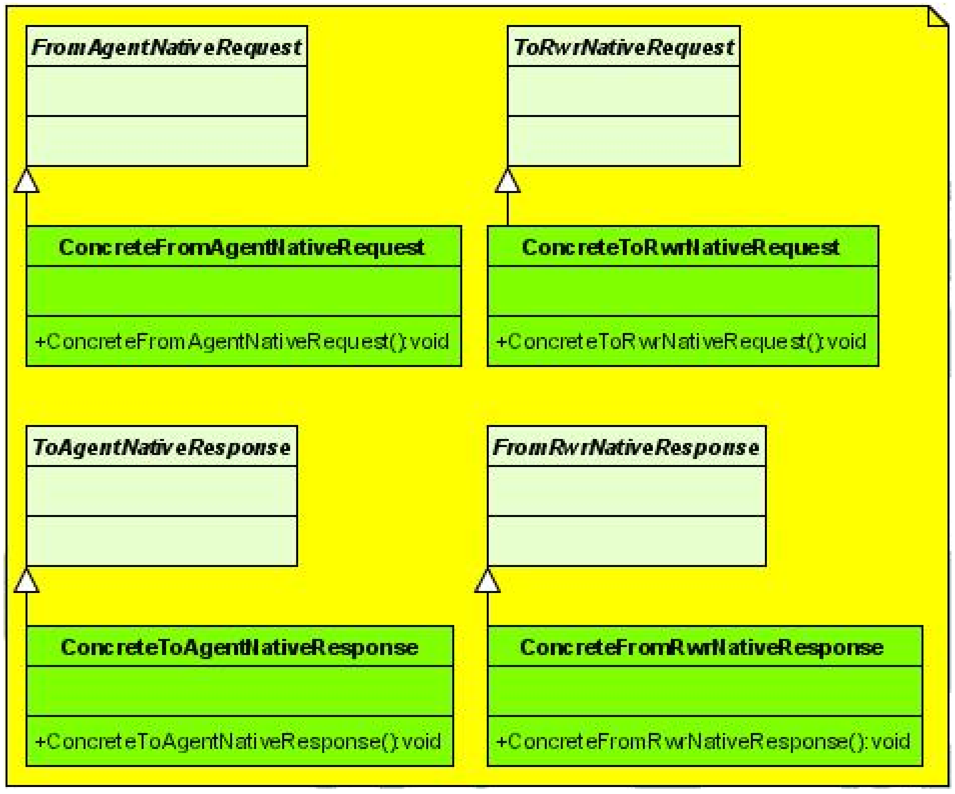
5.2.1 Concrete Adapter Class Diagram


Transformator
-instance: Transformator
+getInstance(): Transformator
+transfomAgentRequest(request: FromAgentNativeRequest): ToRwrNativeRequest
+transfomRwrResponse(response: FromRwrNativeResponse): ToAgentNativeResponse
Agent
#adapter: Adapter
-instance: Agent
+getInstance(adapter: Adapter): Agent
ConcreteTransformator
+ConcreteTransformator(): void
ConcreteAgent
+generateRequest(): void
+ConcreteAgent(): void
ConcreteRwr
+ConcreteRwr(): void
Adapter
-rwr: Rwr
-transformator: Transformator
+Adpater(): void
+requestRwr(request: FromAgentNativeRequest): FromRwrNativeResponse
Rwr
-instance: Rwr
+getInstance(): Rwr
+request(request: ToRwrNativeRequest): FromRwrNativeResponse

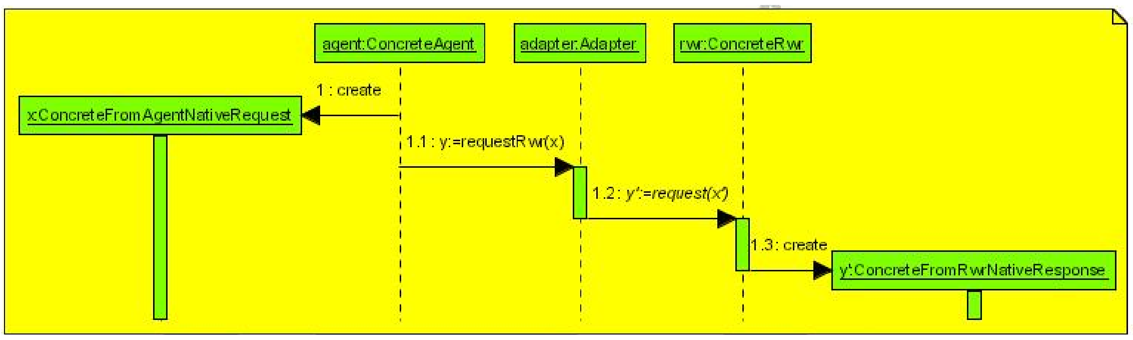
5.2.2 Concrete Adapter creation


instance: Agent
1.1 : instance:=getInstance(adapter)
adapter: Adapter
1 : agent:=getInstance(adapter)
instance: ConcreteAgent
1.2 : instance = new ConcreteAgent()
instance: Rwr
2.1 : instance:=getInstance()
2 : rwr:=getInstance()
instance: ConcreteRwr
2.2 : instance = new ConcreteRwr()
instance: Transformator
3.1 : instance:=getInstance()
3 : transfomator:=getInstance()
instance: ConcreteTransformator
0 : Adpater()
3.2 : instance = new ConcreteTransfomator()

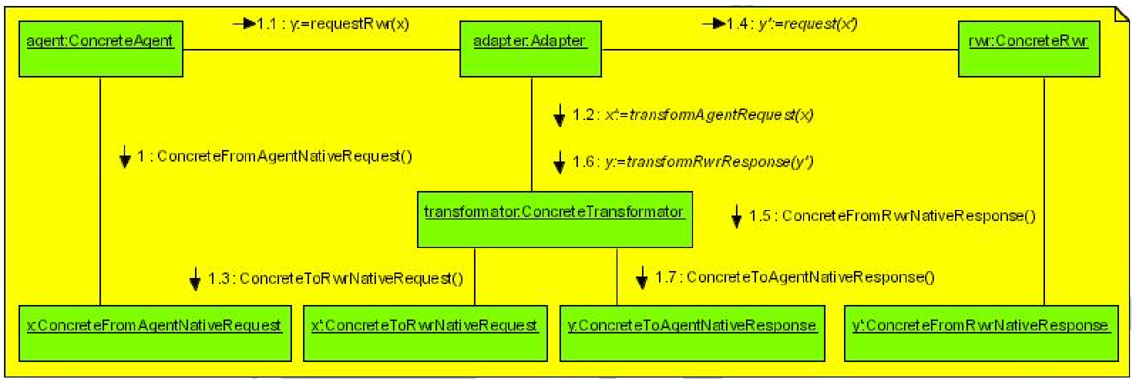
5.2.3 Concrete Protocol Class Diagram


FromAgentNativeRequest
ToRwrNativeRequest
ConcreteFromAgentNativeRequest
+ConcreteFromAgentNativeRequest():void
ConcreteToRwrNativeRequest
+ConcreteToRwrNativeRequest():void
ToAgentNativeResponse
FromRwrNativeResponse
ConcreteToAgentNativeResponse
+ConcreteToAgentNativeResponse():void
ConcreteFromRwrNativeResponse
+ConcreteFromRwrNativeResponse():void

5.2.4 Agent to Rwr communication with concrete realization



5.2.5 Semantic adaptation with concrete realization



agent:ConcreteAgent
1.1 : y:=requestRwr(x)
adapter:Adapter
1.4 : y':=request(x')
rwr:ConcreteRwr
1.2 : x':=transformAgentRequest(x)
1 : ConcreteFromAgentNativeRequest()
1.6 : y:=transformRwrResponse(y')
transformator:ConcreteTransformator
1.5 : ConcreteFromRwrNativeResponse()
1.3 : ConcreteToRwrNativeRequest()
1.7 : ConcreteToAgentNativeResponse()
x:ConcreteFromAgentNativeRequest
x':ConcreteToRwrNativeRequest
y:ConcreteToAgentNativeResponse
y':ConcreteFromRwrNativeResponse

Sequence of method invocation and request/response creation for semantic adaptation process with pieces of source code.

1:

ConcreteFromAgentNativeRequest x = new ConcreteFromAgentNativeRequest();

return x;

1.1:

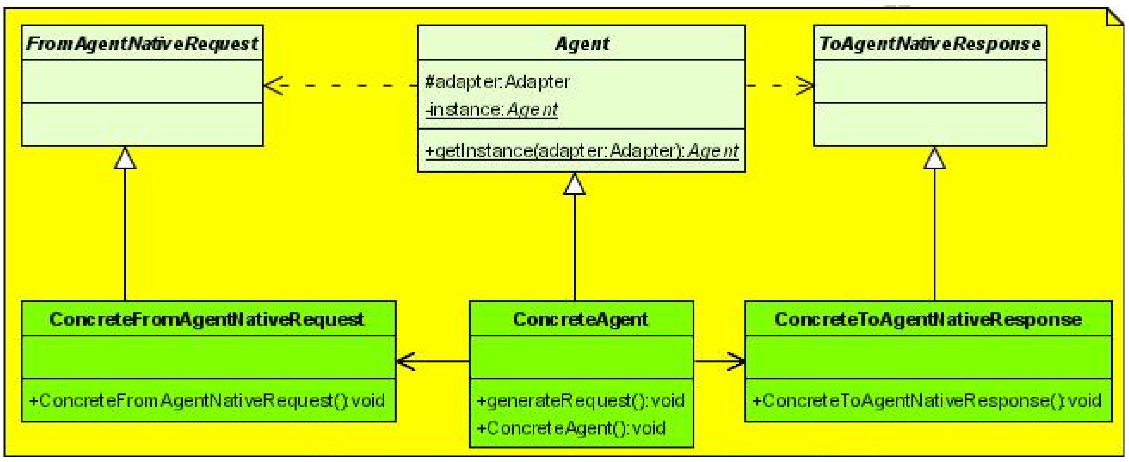
ConcreteToAgentNativeResponse y = (ConcreteToAgentNativeResponse)super.adapter.requestRwr(x);

1.2:

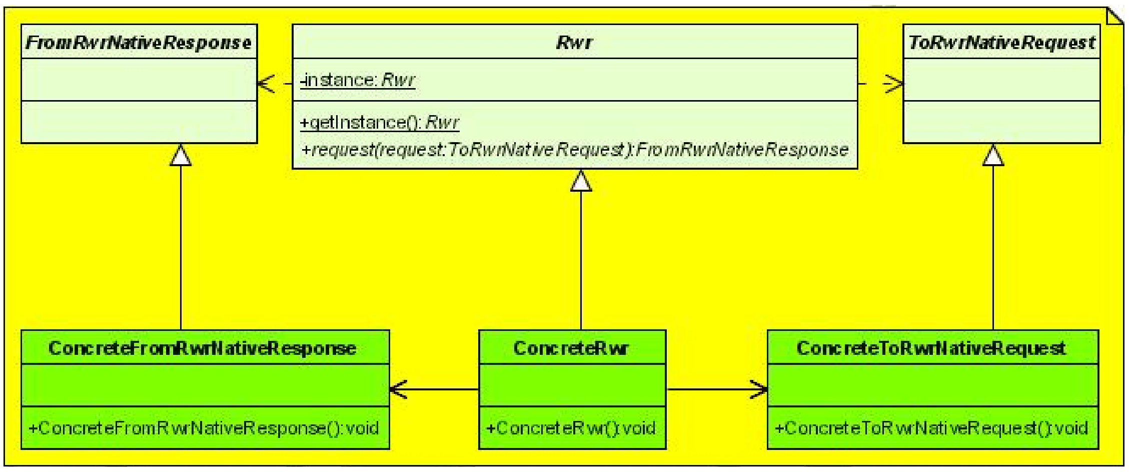
```
ToRwrNativeRequest x' = transformator.transformAgentRequest(x);

1.3:

ConcreteToRwrNativeRequest x' = new ConcreteToRwrNativeRequest();

return x';

1.4:

FromRwrNativeResponse y' = rwr.request(x');

1.5:

ConcreteFromRwrNativeResponse y' = new ConcreteFromRwrNativeResponse();

1.6:

ToAgentNativeResponse y = transformator.transformRwrResponse(y');

1.7:

ConcreteToAgentNativeResponse y = new ConcreteToAgentNativeResponse();
```

5.2.6 Partitioned logic of semantic adaptation

ConcreteAgent class as an encapsulation of logic of an external actor Agent depends from realization of ConcreteFromAgentNativeRequest and ConcreteToAgentNativeResponse classes. Thus development of this domain can be performed independently from particular nature to an external actor RWR. So ConcreteAgent class captures model of request/response interface, model of underlying data which encapsulated in Agent side protocol concrete classes and logic of interoperation within external actor Agent as it is shown in figure.

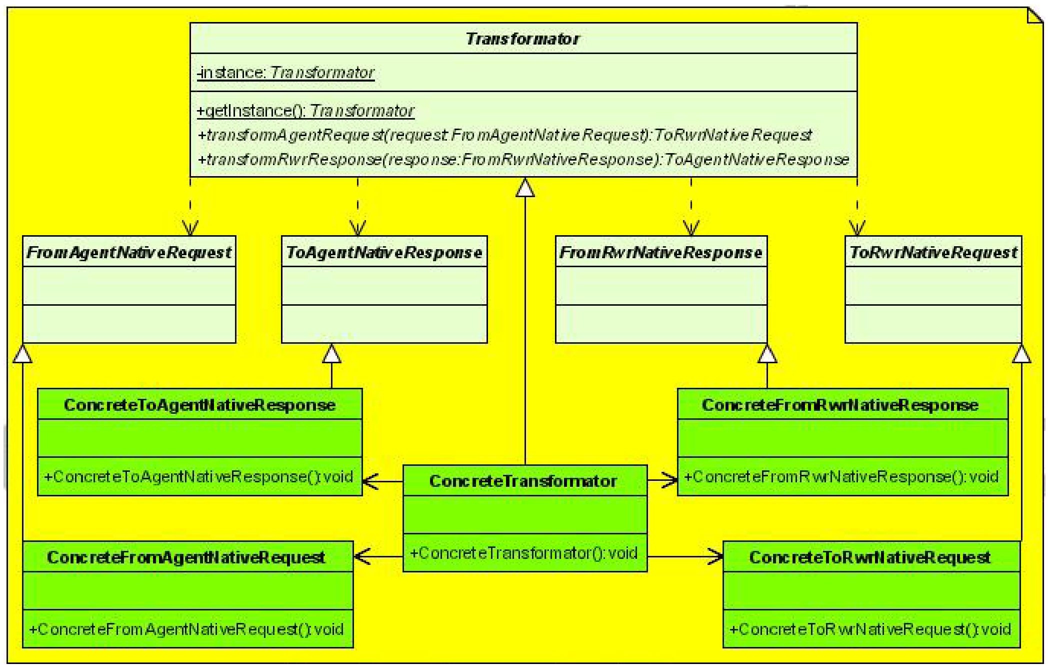

FromAgentNativeRequest
Agent
#adapter:Adapter
-instance:Agent
+getInstance(adapter:Adapter):Agent
ToAgentNativeResponse
ConcreteFromAgentNativeRequest
+ConcreteFromAgentNativeRequest():void
ConcreteAgent
+generateRequest():void
+ConcreteAgent():void
ConcreteToAgentNativeResponse
+ConcreteToAgentNativeResponse():void

ConcreteRwr class as an encapsulation of logic of an external actor Rwr depends from realization of ConcreteToRwrNativeRequest and ConcreteFromRwrNativeResponse classes. Thus development of this domain can be performed independently from particular nature to an external actor Agent. So ConcreteRwr class captures model of request/response interface, model of underlying data which encapsulated in Rwr side protocol concrete classes and logic of interoperation within external actor Rwr as it is shown in figure.


FromRwrNativeResponse
Rwr
-instance:Rwr
+getInstance():Rwr
+request(request:ToRwrNativeRequest):FromRwrNativeResponse
ToRwrNativeRequest
ConcreteFromRwrNativeResponse
+ConcreteFromRwrNativeResponse():void
ConcreteRwr
+ConcreteRwr():void
ConcreteToRwrNativeRequest
+ConcreteToRwrNativeRequest():void

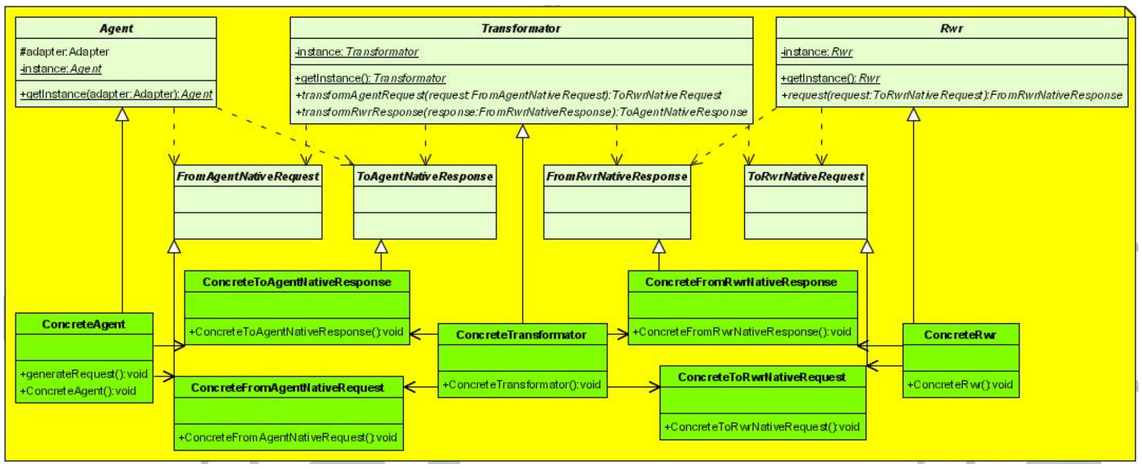
ConcreteTransformer performs the main job of semantic adaptation. This class depends only from the models of Agent class to Adapter class and Adapter class to Rwr class interfaces which are encapsulated in four concrete protocol classes.

This class implements method of Agent class request to Rwr class request adaptation and method for corresponding adaptation of responces.


Transformator
-instance: Transformator
+getInstance(): Transformator
+transformAgentRequest(request:FromAgentNativeRequest):ToRwrNativeRequest
+transformRwrResponse(response:FromRwrNativeResponse):ToAgentNativeResponse
FromAgentNativeRequest
ToAgentNativeResponse
FromRwrNativeResponse
ToRwrNativeRequest
ConcreteToAgentNativeResponse
+ConcreteToAgentNativeResponse():void
ConcreteFromRwrNativeResponse
+ConcreteFromRwrNativeResponse():void
ConcreteTransformator
+ConcreteTransformator():void
ConcreteFromAgentNativeRequest
+ConcreteFromAgentNativeRequest():void
ConcreteToRwrNativeRequest
+ConcreteToRwrNativeRequest():void

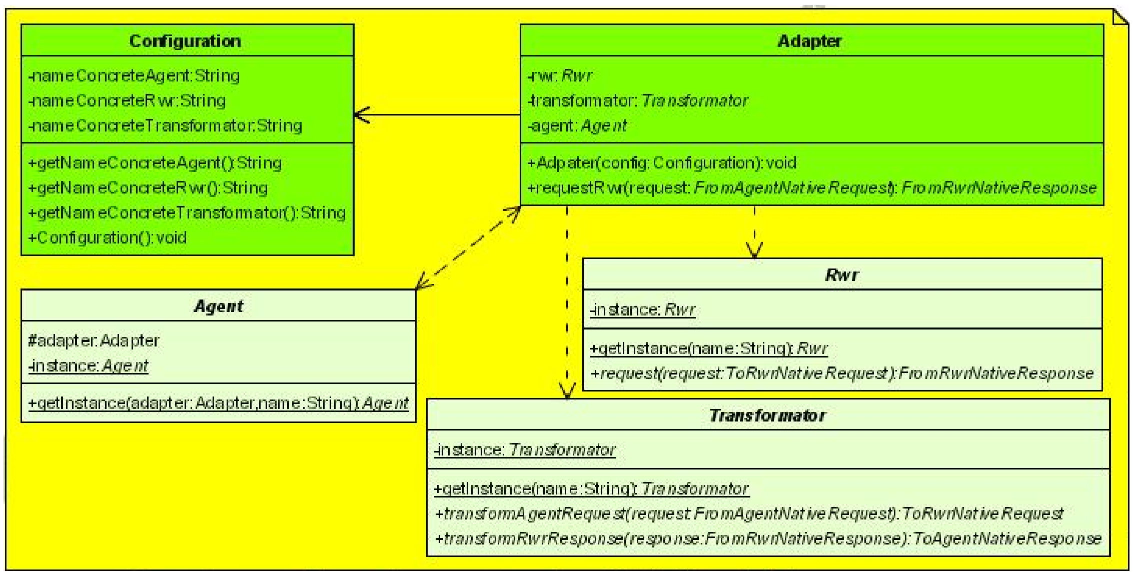
Thus by such software design we achieved:

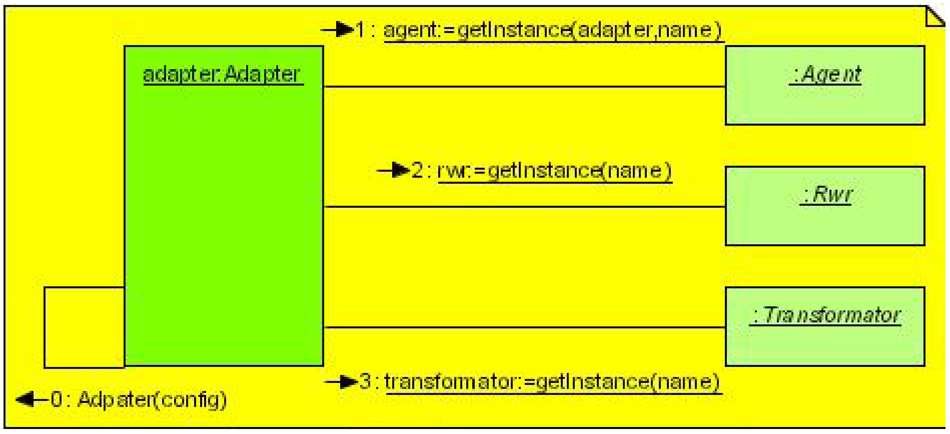
- Abstract level design to capture interoperation and task of semantic adaptation is proposed
- Concrete realization can be done by extending proposed abstract design
- Implementation of external actor Agent dependent part, external actor RWR dependent part and logic of semantic adaptation is separated from each other and can be performed for different options independently.


Agent
#adapter:Adapter
-instance:Agent
+getInstance(adapter:Adapter):Agent
Transformator
-instance: Transformator
+getInstance(): Transformator
+transformAgentRequest(request:FromAgentNativeRequest):ToRwrNativeRequest
+transformRwrResponse(response:FromRwrNativeResponse):ToAgentNativeResponse
Rwr
-instance: Rwr
+getInstance(): Rwr
+request(request:ToRwrNativeRequest):FromRwrNativeResponse
FromAgentNativeRequest
ToAgentNativeResponse
FromRwrNativeResponse
ToRwrNativeRequest
ConcreteToAgentNativeResponse
+ConcreteToAgentNativeResponse():void
ConcreteFromRwrNativeResponse
+ConcreteFromRwrNativeResponse():void
ConcreteAgent
+generateRequest():void
+ConcreteAgent():void
ConcreteTransformator
+ConcreteTransformator():void
ConcreteRwr
+ConcreteRwr():void
ConcreteFromAgentNativeRequest
+ConcreteFromAgentNativeRequest():void
ConcreteToRwrNativeRequest
+ConcreteToRwrNativeRequest():void

5.2.7 Run-time concrete realization loading

Runt-time concrete realization loading should be performed accordingly to configuration defined by ontologies described in chapter 3.


Configuration
-nameConcreteAgent:String
-nameConcreteRwr:String
-nameConcreteTransformator:String
+getNameConcreteAgent():String
+getNameConcreteRwr():String
+getNameConcreteTransformator():String
+Configuration():void
Adapter
-rwr: Rwr
-transformator: Transformator
-agent: Agent
+Adpater(config: Configuration): void
+requestRwr(request: FromAgentNativeRequest): FromRwrNativeResponse
Agent
#adapter: Adapter
-instance: Agent
+getInstance(adapter: Adapter,name:String): Agent
Rwr
-instance: Rwr
+getInstance(name:String): Rwr
+request(request: ToRwrNativeRequest):FromRwrNativeResponse
Transformator
-instance: Transformator
+getInstance(name:String): Transformator
+transfomAgentRequest(request: FromAgentNativeRequest):ToRwrNativeRequest
+transformRwrResponse(response:FromRwrNativeResponse):ToAgentNativeResponse


1 : agent:=getInstance(adapter,name)
adapter:Adapter
: Agent
2 : rwr:=getInstance(name)
: Rwr
: Transformator
3 : transformator:=getInstance(name)
0 : Adpater(config)

```
Class clas = Class.forName(name);

instance = (Agent)clas.newInstance();


Class clas = Class.forName(name);

instance = (Rwr)clas.newInstance();
```

```
Class clas = Class.forName(name);
instance = (Transformator)clas.newInstance();
```

6 Adaptation of Human, Device and Web Service using GAF

6.1 Human adaptation

At first sight it seems that human is the most difficult part for adaptation, but when we take a closer look we can distinguish basic roles of human as a proactive resource. First of all human may act as a web-service, for example for image recognition. So human can be annotated as a web-service with its inputs and outputs formalization. Second human may be looking for some service or data, then human's agent should contain certain functional features for human's orders execution. It can be for example search features, accounting or shopping. Human's agent should be extensible. In other words it must be extensible via plugins and of course configurable.

6.2 Device adaptation

6.3 Web Service adaptation

Service as a resource has its own specific features, which distinguish it from Device and Human. First, let's take a look at existing technologies in web-service integration. Below is the web-services stack proposed by W3C consortium.

6.3.1 W3C stack

The W3C Web Services Workshop, led by IBM and Microsoft, has agreed that the architecture stack consists of three components: Wire, Description, and Discovery.

Wire stack

The following table shows what layers constitute the Wire Stack.

Other "extensions"	
Attachments	Routing
Security	Reliability
SOAP/XML	
XML	

Table 1.1 – W3C Wire Stack

Wire Stack has extensions to two layers: SOAP and XML. This means whenever the SOAP is used as the envelope for the XML messages, they must be attached, secure, reliable, and routed to the intended service requester or provider. In the stacks of other organizations, SOAP and XML are not treated as "extensions." IBM, for instance, refers to SOAP as a tool for its stack layer, "XML-Based Messaging."

Description stack

The Description Stack, the most important component, consists of five layers:

Business Process Orchestration		
Message Sequencing		
Service Capabilities Configuration		
Service Description (WSDL)	Service Interface	WSDL
	Service Description	
XML Schema		

Table 2.5 – W3C Description Stack

This stack starts with orchestration of business processes from which the messages are sequenced, depending on how service capabilities are configured.

W3C uses WSDL to describe service interface and service implementation, neither of which is explicitly highlighted in other stacks.

Discovery stack

As the name implies, the Discovery Stack involves the use of UDDI, allowing businesses and trading partners to find, discover, and inspect one another in a directory over the Internet, as follows:

Directory (UDDI)
Inspection

Table 2.6 - W3C Discovery Stack

The Inspection Layer refers to WSIL (Web Services Inspection Language) and WS-Inspection specifications.

Putting all three stack-components together, we have the Architecture Stack.

	Other "extensions"	
	Attachments	Routing
	Security	Reliability
	SOAP/XML	
	XML	
Business Process Orchestration		
Message Sequencing		
Service Capabilities Configuration		
Service Description (WSDL)	Service Interface	WSDL
	Service Description	
XML Schema		
	Directory (UDDI)	
	Inspection	

Table 2.7 – W3C architecture stack

Today, SOAP (Simple Object Access Protocol), WSDL (Web Services Description Language), and UDDI are emerging as the Internet de facto standards for Web services. SOAP has been accepted and is being standardized by the World Wide Web Consortium (W3C). WSDL has been submitted to the W3C for standardization, and is emerging as the de facto standard language for the description of Web services. UDDI is poised to be the de facto standard for the Web service repository.

SOAP, WSDL and UDDI provide a “grammar” for web-service definition. In general they define certain ontology for service representation. This ontology can be reused in General Adaptation Framework and furthermore, can be extended via semantic unambiguous descriptions of parameters, for automation of service integration, orchestration and discovery.

References

[RDF] Resource Description Framework specification site, http://www.w3c.org/RDF/

[PracticalRDF] Shelley Powers, ”Practical RDF,” O’Reilly, 2003, 350 pages, ISBN 0-596-00263-7

[SemanticWeb] Semantic Web activity site, http://www.w3c.org/2001/sw/

[OWL] Web Ontology Language specification site, http://www.w3c.org/2004/OWL/

[XML] Extensible Markup Language specification site, http://www.w3c.org/XML/

IOG, 2004] Official Web-Site of Industrial Ontologies Group, http://www.cs.jyu.fi/ai /OntoGroup .

[DAML+OIL] DAML+OIL language web page, http://www.daml.org/2001/03/daml+oil-index.html

[DAML-S] DAML-S 0.7 Draft Release, http://www.daml.org/services/daml-s/0.7/

[Ermolayev et al., 2004] Ermolayev V., Keberle N., Plaksin S., Kononenko O., Terziyan V., <http://www.cs.jyu.fi/ai/papers/IJWSR-2004.pdf>Towards a Framework for Agent-Enabled Semantic Web Service Composition, International Journal of Web Service Research, Idea Group, ISSN: 1545-7362, Vol. 1, No. 3, 2004, pp. 63-87.

[GUN] Global Understanding Environment concept, http://www.cs.jyu.fi/ai/papers /HCISWWA-2003.pdf

[IBM WSCA] IBM Web Services Conceptual Architecture document, http://www-306.ibm.com/software/solutions/webservices/pdf/WSCA.pdf

[Kaykova et. al., 2004] Kaikova H., Khriyenko O., Kononenko O., Terziyan V., Zharko A., Proactive Self-Maintained Resources in Semantic Web, Eastern-European Journal of Enterprise Technologies, Vol. 2, No. 1, 2004, ISSN: 1729-3774, Kharkov, Ukraine, pp. 37-49

[SmartResource, 2004] Proactive Self-Maintained Resources in Semantic Web, Presentation of SmartResource Tekes Project, http://www.cs.jyu.fi/ai/OntoGroup /SmartResource.ppt

[Terziyan, 2003] Terziyan V., <http://www.cs.jyu.fi/ai/papers/HCISWWA-2003.pdf> Semantic Web Services for Smart Devices in a "Global Understanding Environment", In: R. Meersman and Z. Tari (eds.), On the Move to Meaningful Internet Systems 2003: <http://www-staff.it.uts.edu.au/~wgardner/HCI-SWWA.html> OTM 2003 Workshops, Lecture Notes in Computer Science, Vol. 2889, Springer-Verlag, 2003, pp.279-291.

[WSArchitect] Judith M. Myerson, “Web Service Architectures”, http://www.webservices architect.com/content/articles/webservicesarchitectures.pdf

[WSDL] Web Services Description Language submission, http://www.w3.org/TR/wsdl

[WSFL] Web Services Flow Language specification by IBM, http://www-306.ibm.com/software/solutions/webservices/pdf/WSFL.pdf

[W3C] World Wide Web Consortium site, http://www.w3.org/

[OntoShell] COMMUNITY FORMATION SCENARIOS IN PEER-TO-PEER WEB SERVICE ENVIRONMENTS, Olena Kaykova , Oleksandr Kononenko , Vagan Terziyan , Andriy Zharko

[SWGuide] Michael C. Daconta, Leo J. Obrst, Kevin T. Smith. The Semantic Web: A Guide to the Future of XML, Web Services, and Knowledge Management. John Willey & Sons. 2003. 281 p.

IOG

GENERAL ADAPTATION FRAMEWORK (PART II)

Technical report

SmartResource: Proactive self-maintained resources in Semantic Web

2/24/2005

Author: Industrial Ontologies Group

Contact Information: e-mail: vagan@it.jyu.fi

Title: General Adaptation Framework (Part II)

Work: Technical report

Status of document: working draft

Number of Pages: 26

Keywords: General adaptation framework, Semantic Web, Agent, Device, Human, Expert, Software Interface, Industrial Maintenance, Global Understanding Environment

Abstract: This document continues the work described in the Part I of the corresponding technical report and goes deeper in the development of the General Adaptation Framework.

Abbreviations

OWL	– Web Ontology Language
RDF	– Resource Description Framework
RDFS	– RDF Schema language
XML	– eXtensible Markup Language

Contents

1 INTRODUCTION 1

2 APPROACH TO GENERAL ADAPTATION FRAMEWORK 2

2.1 DATA MODELS 3

2.2 PROCESSES IN GENERAL ADAPTATION FRAMEWORK 7

3 SEMANTIC ADAPTATION 9

3.1 SEMANTIC ADAPTATION EXAMPLE 10

3.2 ELABORATION OF SEMANTIC ADAPTATION APPROACH 13

3.3 ONTOLOGY-TO-ONTOLOGY MAPPING 16

3.4 MODEL-TO-MODEL MAPPING 17

3.5 ADAPTER AS SOFTWARE DESIGN 19

REFERENCES 22

7 Introduction

There is a diversity of heterogeneous systems, applications, data formats and ways of interaction. All those systems were tailored for particular tasks, purposes and goals. The world is heterogeneous and we face the challenge trying to integrate heterogeneous systems into a unified environment. The “Smart Resource” project has encountered exactly such kind of a problem.

“General adaptation” assumes a design of a sufficient framework for an integration of different (by structure and nature) resources into Global Understanding eNvironment (GUN). This environment will provide a mutual interaction between heterogeneous resources. Adaptation assumes elaboration of a common mechanism for new resource integration, and its provision with a unified way of interaction.

The main idea of adaptation is based on a concept of “adapter”, which plays role of a bridge between an internal representation of resource and a unified environment. Adapter is a software component, which provides a bidirectional link between a resource interface and an interface of the environment.

GUN assumes interoperability of SmartResources; by Smart Resource we mean a conjunction of Real World Resource (RWR), Adapter and Agent. By extending RWR within Adapter and Agent we make it GUN compatible. General Adaptation includes development of Adapter for RWR.

8 Approach to General Adaptation Framework

The primary intention behind the General Adaptation Framework (GAF) is a design of common framework for adaptation of heterogeneous resources. The design of the framework will be divided into two layers:

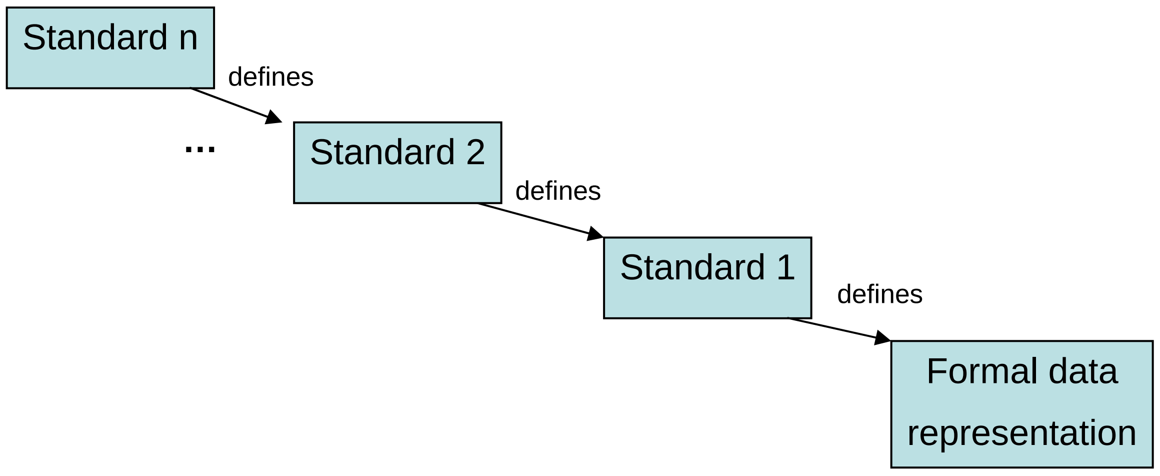
1. Structured software design for modules, classes, behavior and protocols;
2. Semantic adaptation of different formalizations of the industrial maintenance domain edges.

GAF includes the following components:

1. Model, which consists of the submodels:
   - Adapter Functionality;
   - Data representation standards;
   - Software interfaces;
   - Semantic Adaptation (data mapping model);
   - Adapter Configuration Properties.
2. Process, which consists of the subprocesses:
   - Adapter Development;
   - Adapter Composition;
   - Adapter Deployment;
   - Adapter Operation.
3. Tool set, which provide an UI for specification of problem domain features according to the GAF model; support of activity within GAF process and corresponding users
4. Scenarios that comprise roles of participants in Adaptation Processes and their interaction with Tool set and submodels.

8.1 Data models

Arbitrary number of standards exists, which define each other on different levels of abstraction and thus form a hierarchy:



One of the data models, which have recently gained wide adoption, is XML – Extensible Markup Language. The data representation using XML can be represented by the following figure (see Figure 1):

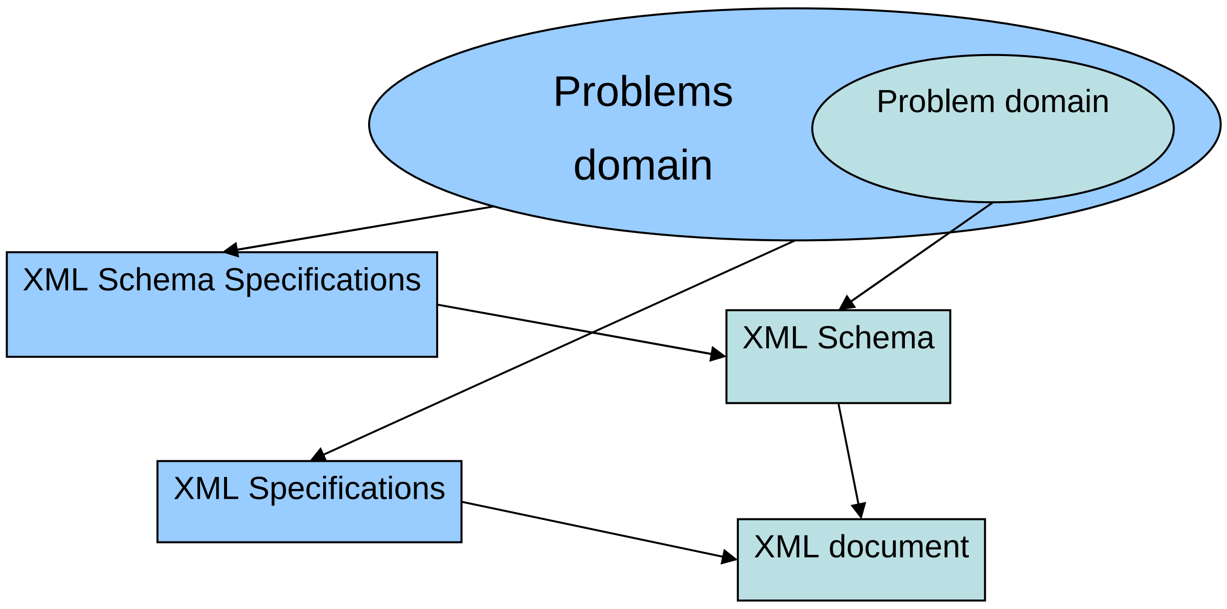


Figure 1 - XML data representation

The older and more tested data representation standard is Database Model (see Figure 2):

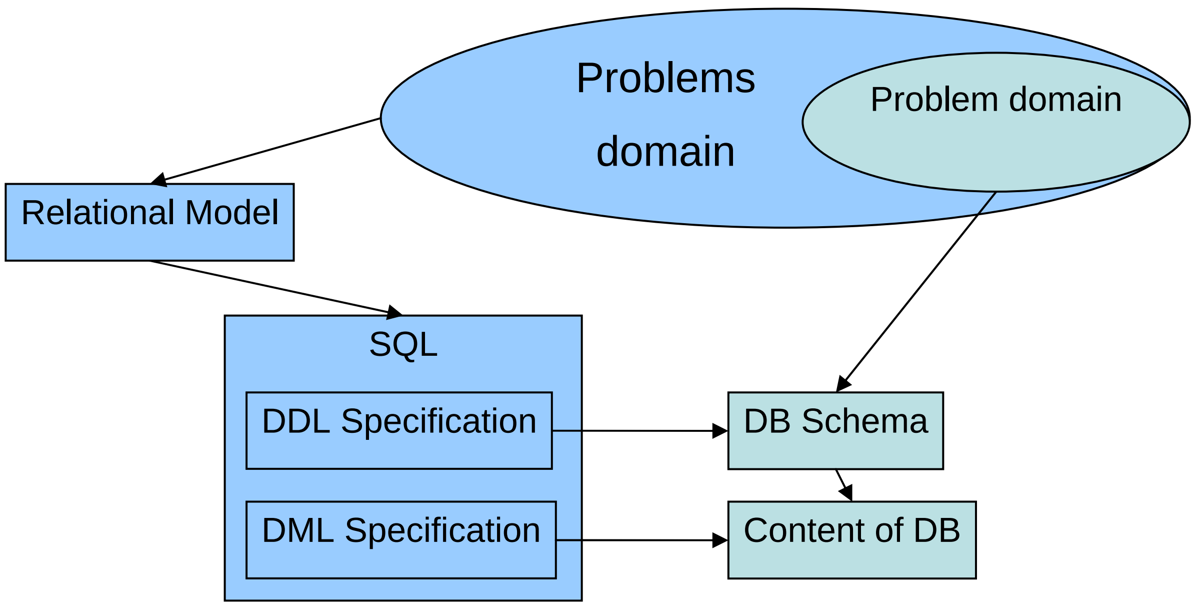


Figure 2 - Database data representation

The novel data representation standards, which focus primarily on semantics, are RDF and OWL (see Figure 3):

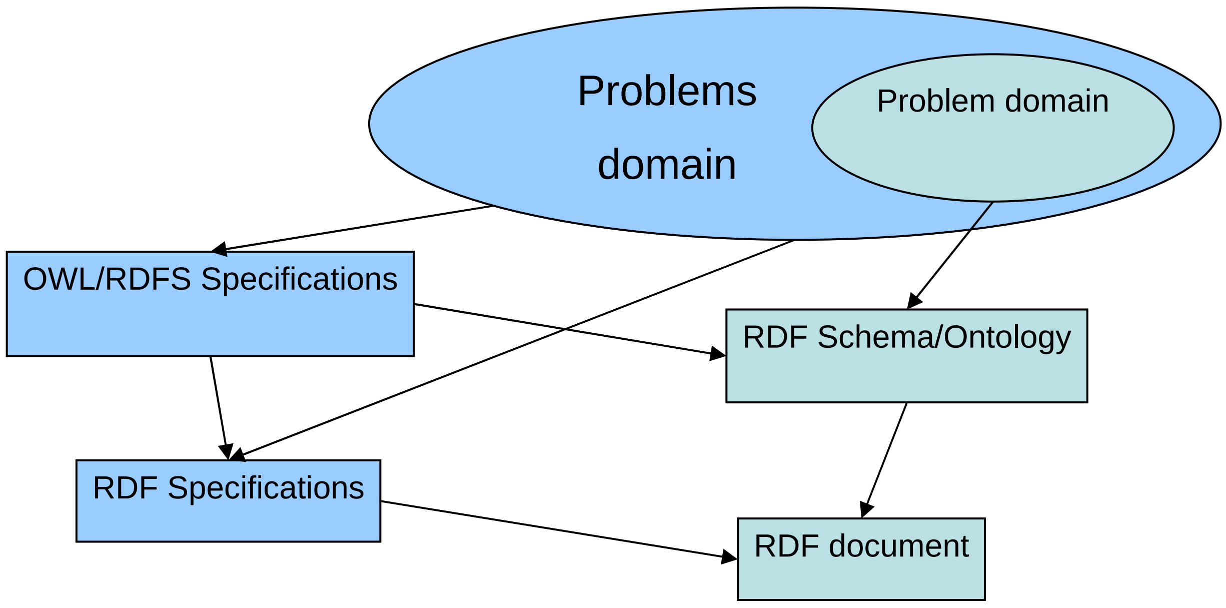


Figure 3 - Semantic data representation

In fact, arbitrary data representation schema looks like it is shown on Figure 4:

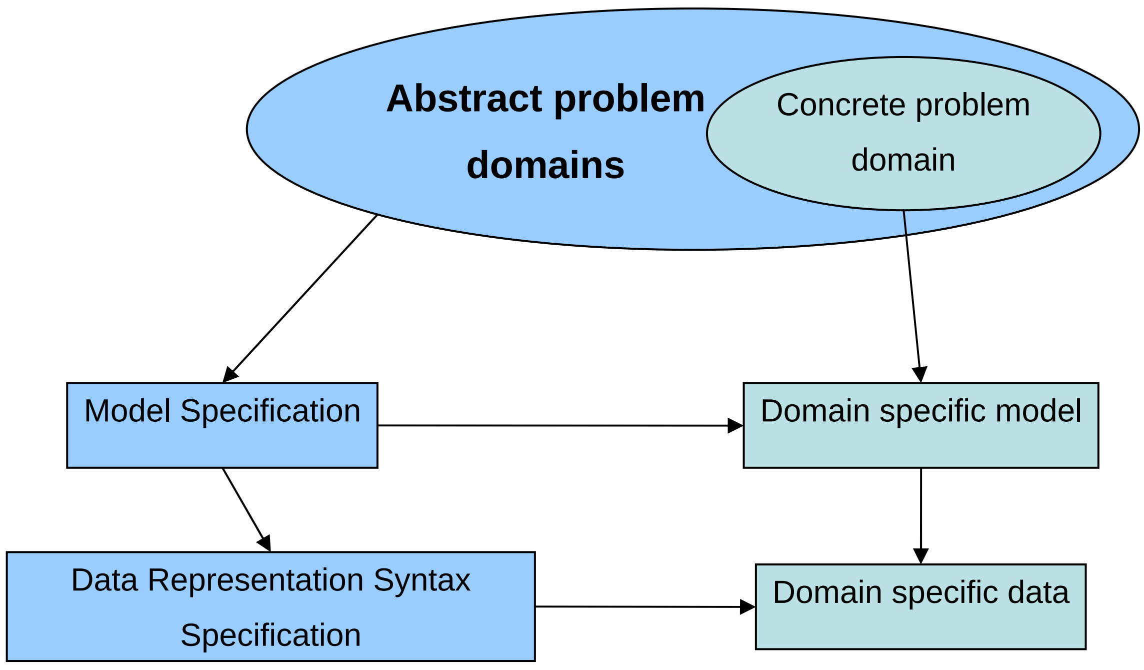


Figure 4 - Arbitrary data representation

More abstract models define more specific ones. In different cases arbitrary number of models can be found in chains and layers (see Figure 5).

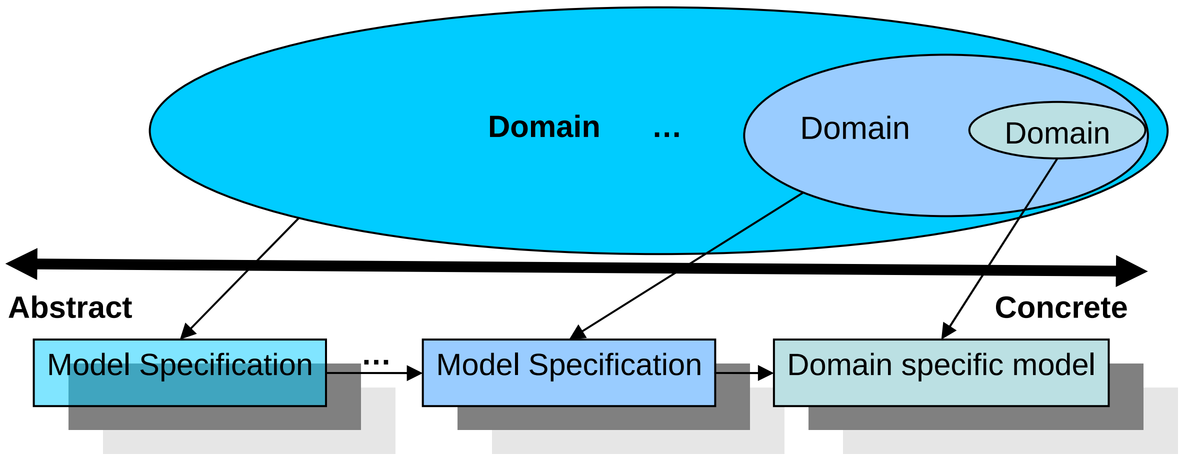


Figure 5 - Nested models

Semantic adaptation results in a mapping of data encoded according to one model to another model of data representation (see Figure 6):

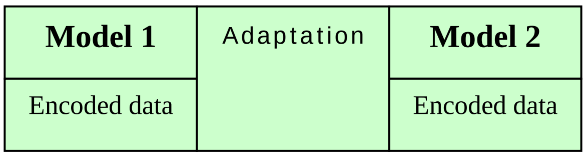


Figure 6 - Model-to-model adaptation

The most commonly used data representation standards are Relational model, XML-model and RDF-model. Thus, any problem domain can be formalized using these data models (see Figure 7):

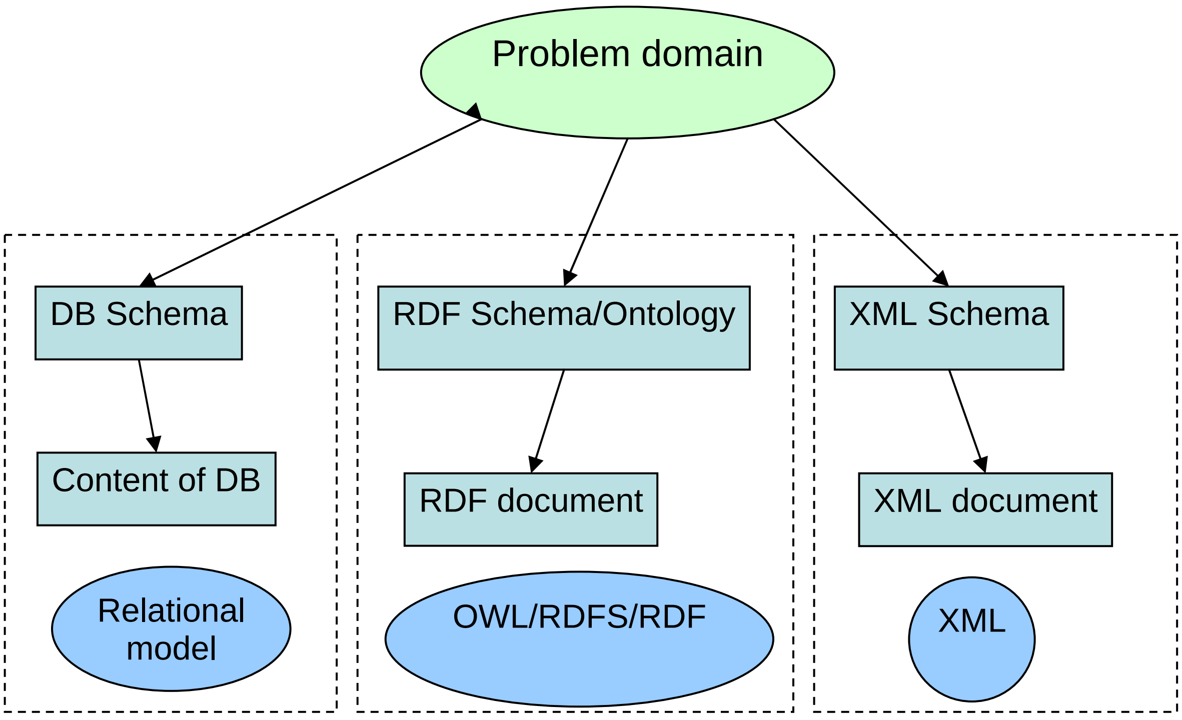


Figure 7 - Possible formalization of a domain

The problem domain of the SmartResource project utilizes RDF for its formalization and all its concepts are included into RscDF-schema (Resource State/Condition Description Framework). Finally, we get a Layered Cake of Specifications (see Figure 8):

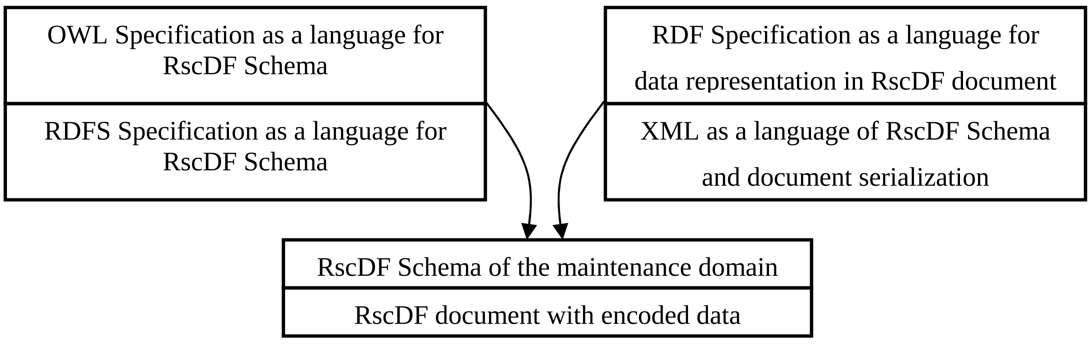

OWL Specification as a language for RscDF Schema
RDFS Specification as a language for RscDF Schema
RDF Specification as a language for data representation in RscDF document
XML as a language of RscDF Schema and document serialization
RscDF Schema of the maintenance domain
RscDF document with encoded data

Figure 8 - SmartResource Layered Cake of Specifications

8.2 Processes in General Adaptation Framework

The Processes that are included into GAF will be described according to the Template:

- Preconditions for process start;
- Process execution;
- Result.

Adapter operation process:

Preconditions: Deployed Adapter;

Process description: Automated on-line interoperability and data mapping between Agent and RWResource;

Result: GUN-compatible Resource.

Adapter deployment process:

Preconditions: Composed Adapter;

Process description: 1) Specification of Adapter runtime property values according to a submodel of Adapter Configuration Properties; 2) Adapter Installation.

Result: Deployed Adapter, ready to operate.

Adapter composition process:

Preconditions: 1) New combination of interoperating modules (e.g. another Network connection standard) or/and; 2) New data schema for already supported data model occurrence.

Process description: 1) Software Modules Composition: Modules selection; Modules assembling; Adapter Functionality Semantic Specification. 2) Semantic adaptation: New class definition (resource declaration in the ontology); Creation of new properties for a new class (if needed); Device's interface properties definition (connection type, data types, etc. Taken from ontology); Mapping of resource's data representation to RscDF data representation;

Result: Composed (Deployable) Adapter; Specified Adapter Functionality; Adapter Configuration Properties template (allowed values, etc.)

Adapter development process:

Preconditions: RWResource with a specified interface and a data format; Access to Semantic Adaptation, Data representation standards, Software interface models; Ontology mapping/editing tool (mapping to already existent standards of communication and data representation).

Process description: Software development process; Semantic annotation of the Developed Modules.

Result: Software modules for data access or transformation; Documented and registered in ontology.

9 Semantic Adaptation

During the data transformation process, Data Transformer involves format's metadata (schemas) and transformation rules. Schemas, rules and underlying ontologies constitute the semantic adaptation.

The tasks of Semantic Adaptation are the following:

1. Semantic Adaptation defines a functionality to work with semantics of:
   - Adapter Functionality (Services provided by the adapter);
   - Data representation standards and models of the adapted systems;
   - Software interfaces standards of the adapted systems;
   - Configuration properties of the adapter runtime environment.
2. Semantic Adaptation uses an Ontology-based approach to define the semantics mentioned above:
   - This involves associating commonly understood meaning to the definition of adapter properties, functionality, configuration, and associated meta-data standards.

Semantic Adaptation requires the following stages:

1. Analysis of problem domain and elaboration of a conceptual model;
2. Analysis of data representation formats;
3. Analysis of corresponding metadata (particular Database schema, for instance);
4. Analysis of a standard's specification (e.g. XML Schema Specification standard);
5. Elaboration of the model for transformations of particular standards of data representation;
6. Specification of data mapping rules;
7. Choosing and/or Development of the mechanism or tool of transformation (appropriate patterns, APIs etc).

9.1 Semantic Adaptation Example

To understand better the stages of Semantic Adaptation let us consider an example. The problem domain will be a paper machine and the process of paper manufacturing (see Figure 9). The first stage of the adaptation will be elaboration of a conceptual model for the selected domain. Firstly, the domain description in a natural language must exist. It can be made either separately, or existing specifications can be used. The main point is that this description must contain all important aspects of the problem domain. For our domain, the description can include such phrases as “a paper machine produces paper, uses cellulose”, etc.

After the mentioned domain description, domain decomposition follows based on the domain description. On this stage, entities, classes, properties, relations, behaviors of the problem domain are distinguished. After the necessary decomposition has been made, domain formalization is performed using any appropriate data models. It can be ER-diagrams (Entity Relationship), UML, Ontology, etc.

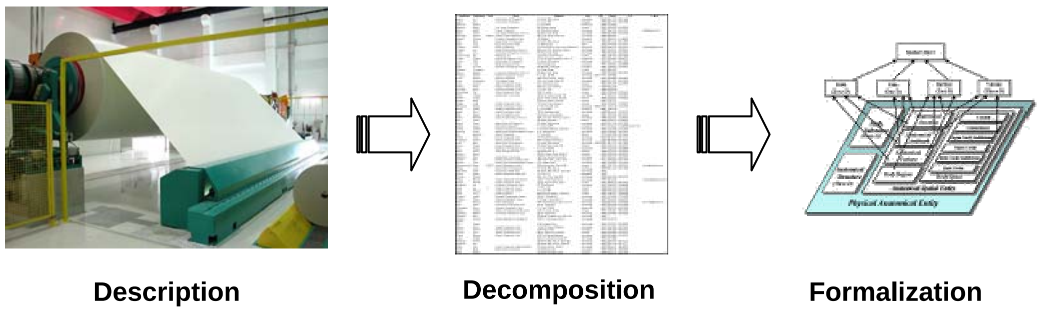
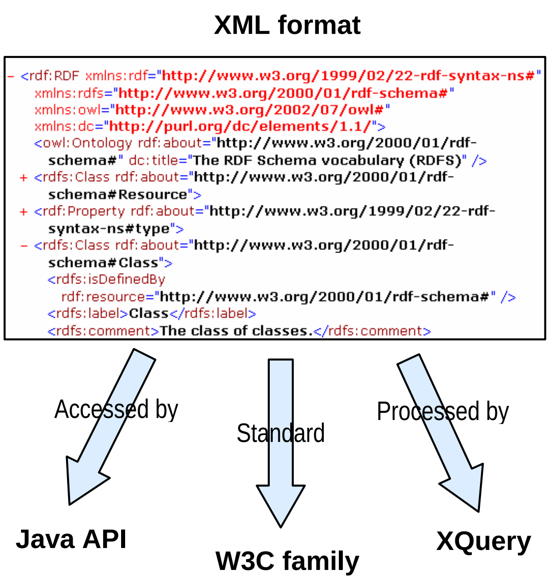


Figure 9 - Adaptation of paper machine domain

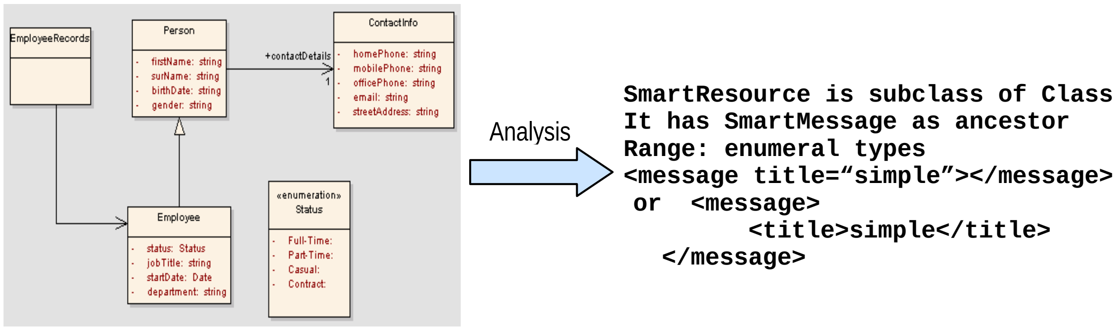
After the mentioned stages of the adaptation, analysis of data representation format follows (see Figure 10). It includes analysis of the data format type (XML, text file, Excel table, Oracle database, etc.), types of APIs that can be used in the domain (SQL-queries, Java DOM API, XQuery, etc.), access methods to data (JDBC, OLE, etc.), sorts of standards that are used to represent a format (ASCII, W3C-family standards).



```
- <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:owl="http://www.w3.org/2002/07/owl#"
   xmlns:dc="http://purl.org/dc/elements/1.1/">
   <owl:Ontology rdf:about="http://www.w3.org/2000/01/rdf-
     schema#" dc:title="The RDF Schema vocabulary (RDFS)" />
 + <rdfs:Class rdf:about="http://www.w3.org/2000/01/rdf-
     schema#Resource">
 + <rdf:Property rdf:about="http://www.w3.org/1999/02/22-rdf-
     syntax-ns#type">
 - <rdfs:Class rdf:about="http://www.w3.org/2000/01/rdf-
     schema#Class">
     <rdfs:isDefinedBy
       rdf:resource="http://www.w3.org/2000/01/rdf-schema#" />
     <rdfs:label>Class</rdfs:label>
     <rdfs:comment>The class of classes.</rdfs:comment>
```

Figure 10 - Analysis of data representation format

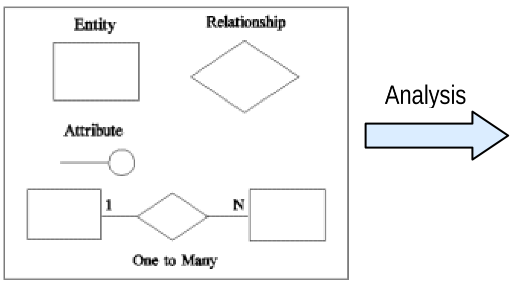
The next stage of Semantic Adaptation for our chosen domain is Metadata analysis (see Figure 11). This stage includes analysis of data schema used (elements, relationships, types, etc.), possible variations (XML tags or values, etc.), hierarchy of elements and restrictions (nesting of classes, range, etc.).



```
SmartResource is subclass of Class
It has SmartMessage as ancestor
Range: enumeral types
<message title="simple"></message>
 or  <message>
         <title>simple</title>
   </message>
```

Figure 11 - Metadata analysis

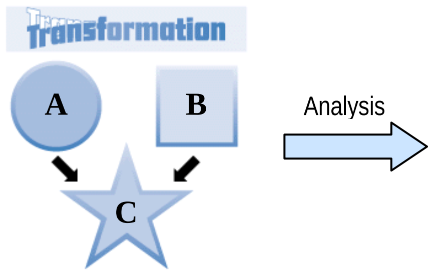
Further stage of the Adaptation is Analysis of standard (Figure 12). This stage includes analysis of standard specification (syntax, vendors, schema, etc.), analysis of existing formal theory (relational algebra, frame model, etc.), analysis of existing methods of transformation (XSLT, production rules, etc.), analysis of capabilities and restrictions (possibilities of formalization, querying, etc.).



Relational theory
SQL97 standard
Relational algebra (selection, projection, join)
transformable to object model, XML model

Figure 12 - Analysis of standard

Further, one must go on with a stage of Model transformation (see Figure 13). At this stage, the existing approaches to transformation have to be analyzed (XSLT, piping, etc.), the source/target resources have to be defined (XHTML to XML, Oracle DBMS to XML, XML to RDF, etc.), possibility for further extension must be taken into account (e.g. cost of extension during evolution), a role of metadata is also important (schema integration, compatibility, etc.).



- **A can be transformed to C with XSLT;**
- **A is the XML file, C is an RDF file;**
- **A's schema is dynamic;**
- **Schemas are interoperable.**

Figure 13 - Model of transformation

The next step of the adaptation process is concerned with data mapping rules (see Figure 14). This stage requires efforts for determining a protocol of transformation (elements and types matching), representation format for the rules (Ontology, XSLT, etc.), percentage of manual, semiautomatic and automatic matching actions.

The mechanism of transformation requires the following analyses to be done: analysis of possible approaches (tools, APIs, Services, etc.), estimation of cost for particular approach (time for development, price of the product, etc.), study of interoperability and extensibility of the chosen approach (supported platforms, extensible API, etc.). For transformation,

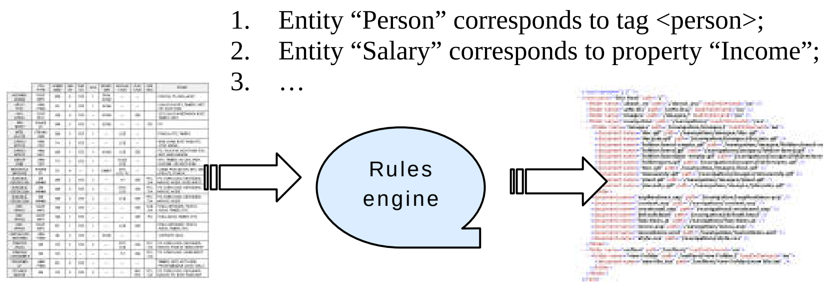

1. Entity “Person” corresponds to tag <person>;
2. Entity “Salary” corresponds to property “Income”;
3. …
Rules engine

Figure 15 - Data mapping rules

existing tools can be used or if reasonable these tools can be developed from scratch. The most popular APIs used in transformation of XML are XSLT, SAX and DOM. In case of RscDF the functionality for implementation must be defined: either it will be XML-to-RDF transformation, or more.

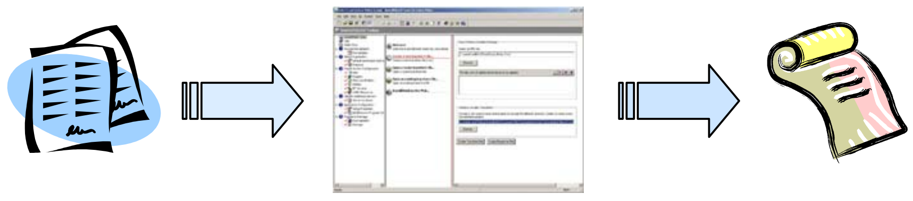


Figure 14 - Mechanism of transformation

9.2 Elaboration of Semantic Adaptation approach

One of the approaches to adaptation is a serialization of the RscDF format into intermediate well standardized and elaborated format. As the basis, it’s intended to use XML format for this approach. For this purpose, a unified mechanism of RscDF transformation into the XML format and vice versa has to be designed and developed (see Figure 16). This mechanism will allow mapping schemas and data from RscDF to XML (Figure 17).

There are some projects, which have elaborated pilot methods of transformation RDF to XML [1, 2]. Since RSCDF is enhanced subset of RDF it’s possible to adopt these methods.

The transformation is carried out by either by replacing XPATH expressions or by the set of XSLT style sheets (see Figure 17).

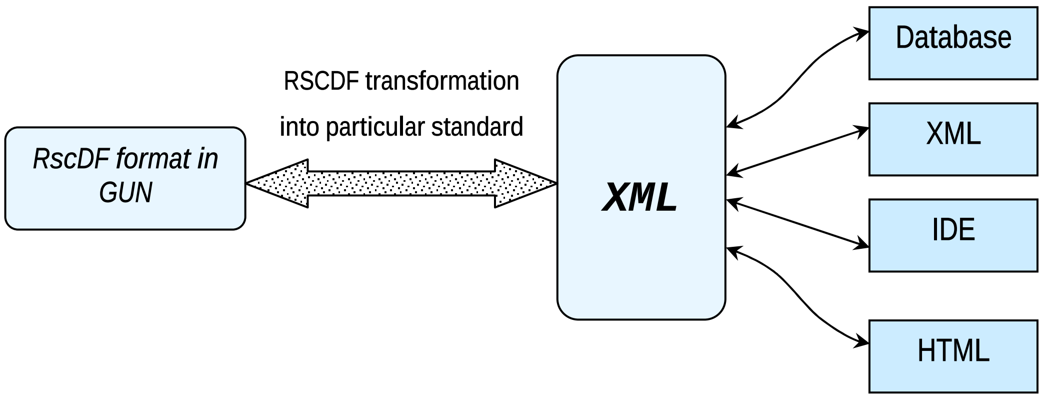


Figure 16 - Unified RscDF-to-XML transformation

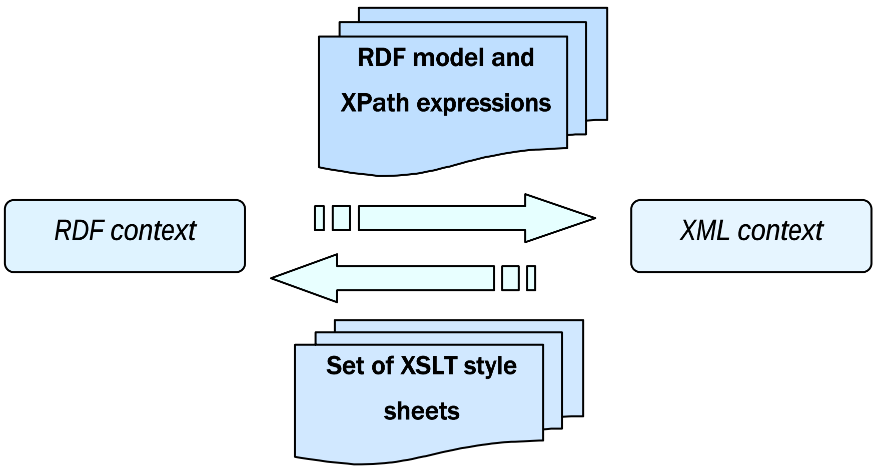


Figure 17 - XPath and XSLT in RDF-to-XML transformation

Once the mechanism of transformation from RSCDF to XML and XML to RSCDF has been designed it's possible to use standard approaches for future transformation (Figure 18). Choosing the XML format as the start point will allow unifying process of adaptation.

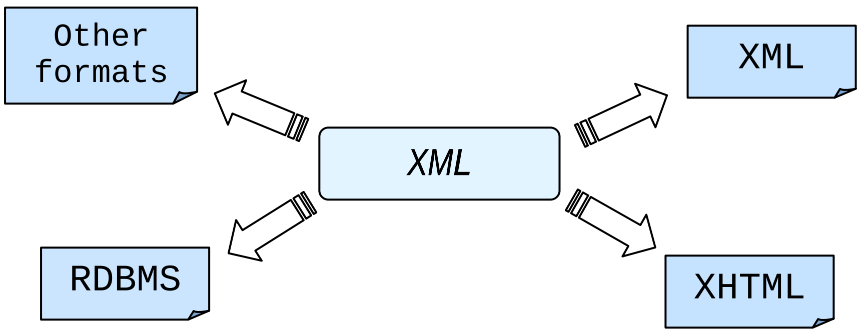


Figure 18 - Transformation of XML to other formats

From existing tools that provide transformation of XML to other formats, Altova MapForce can be mentioned [3]. This commercial tool allows XML to XML transformation based upon two XML schemas (Figure 19). It's also might be necessary to perform some processing functions to pipe data from source to target.

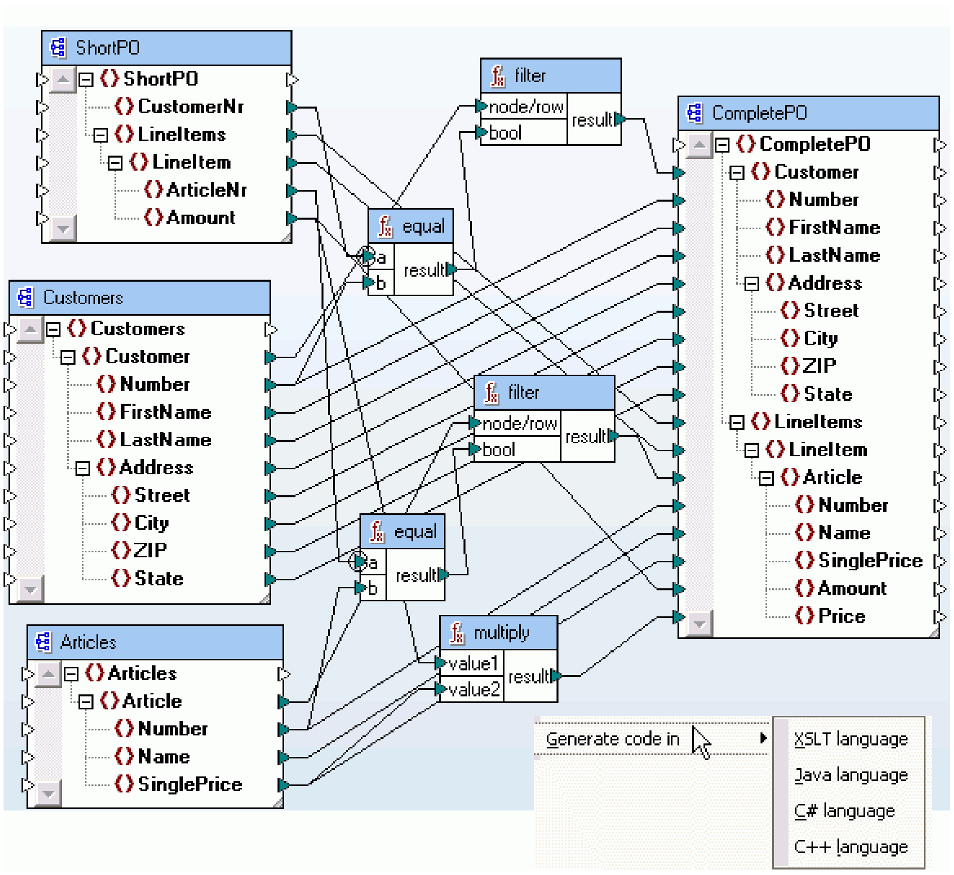


Figure 19 - XML-to-XML transformation in MapForce

MapForce allows mapping between XML and Relational database, too (Figure 20). The process of mapping starts from the loading of database scheme and XML schema. Then engineer manually fulfils matching between XML elements and database entities. While mapping it might be necessary to use processing functions.

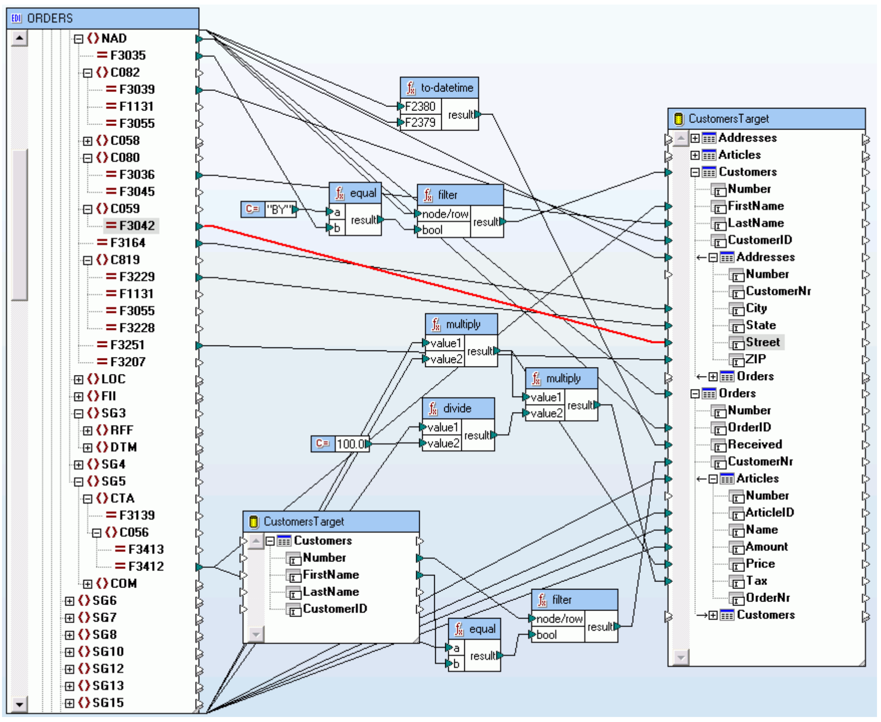


Figure 20 - XML-to-Database transformation in MapForce

9.3 Ontology-to-ontology mapping

When we deal with RDF-to-RDF transformation, we inevitably face with the challenge of ontology-to-ontology mapping and transformation. If all domain descriptions refer to a common vocabulary (World ontology – ideal case), mapping can be done explicitly (Figure 21). However, incompleteness of one ontology may cause inability to transform in both directions.

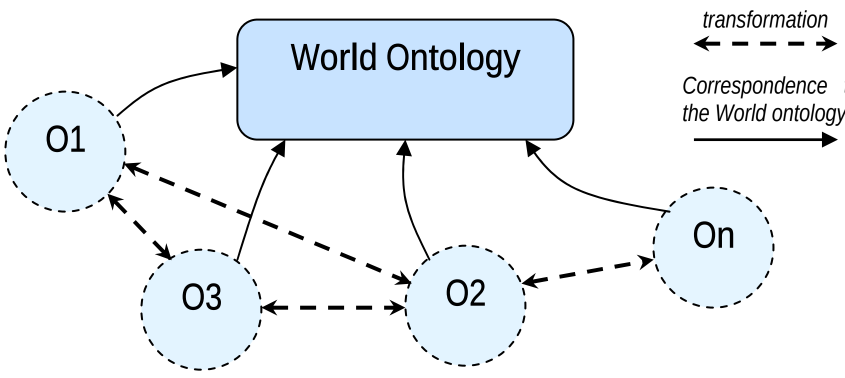


Figure 21 - Ontology-to-ontology transformation

In case of Peer-to-Peer ontology mapping, one ontology is mapped to other manually or in semi-automatic way (Figure 22).

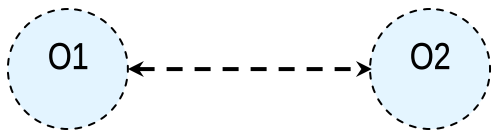


Figure 22 - Peer-to-Peer ontology mapping

Construction of mapping rules may meet the following problems:

- ❑ Different expert vision of problem domain;
- ❑ Models may be inconsistent conceptually;
- ❑ Paradigms the models are based on may cause hardly convertible schemas.

9.4 Model-to-model mapping

To develop unified adapter to a particular standard, the following formats and structures of data must be analyzed:

- Flat files (ASCII text files);
- Tables (Excel);
- Trees and taxonomies (xml, ontology-files);
- Marked up structures (HTML);
- Relational model (RDBMS);
- Object model (Classes and objects);
- Compound structure (any mixed specific structures).

The generic model mapping scheme is shown in Figure 23:

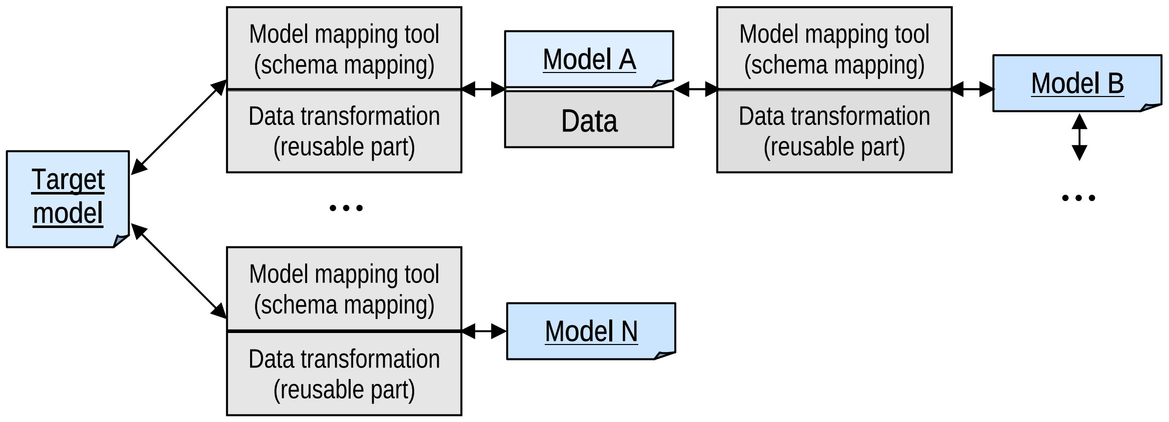

Model mapping tool (schema mapping)
Data transformation (reusable part)
Model A
Data
Model mapping tool (schema mapping)
Data transformation (reusable part)
Model B
Target model
...
...
Model mapping tool (schema mapping)
Data transformation (reusable part)
Model N

Figure 23 - Generic model mapping schema

For a concrete case the model mapping scheme will look like the following (Figure 24):

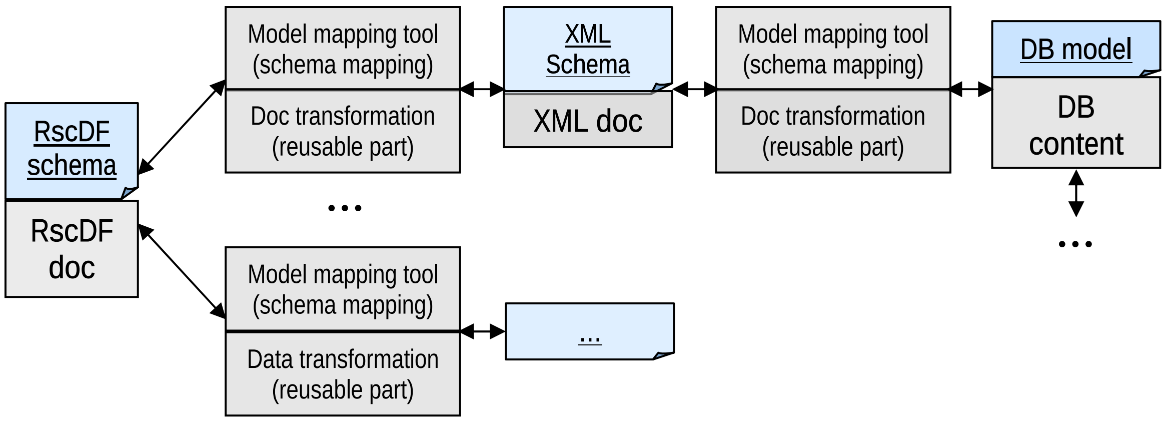

RscDF schema
RscDF doc
Model mapping tool (schema mapping)
Doc transformation (reusable part)
…
Model mapping tool (schema mapping)
Data transformation (reusable part)
XML Schema
XML doc
...
Model mapping tool (schema mapping)
Doc transformation (reusable part)
DB model
DB content
…

Figure 25 - Model mapping schema

Finally, the document transformation scheme is the following (Figure 25):

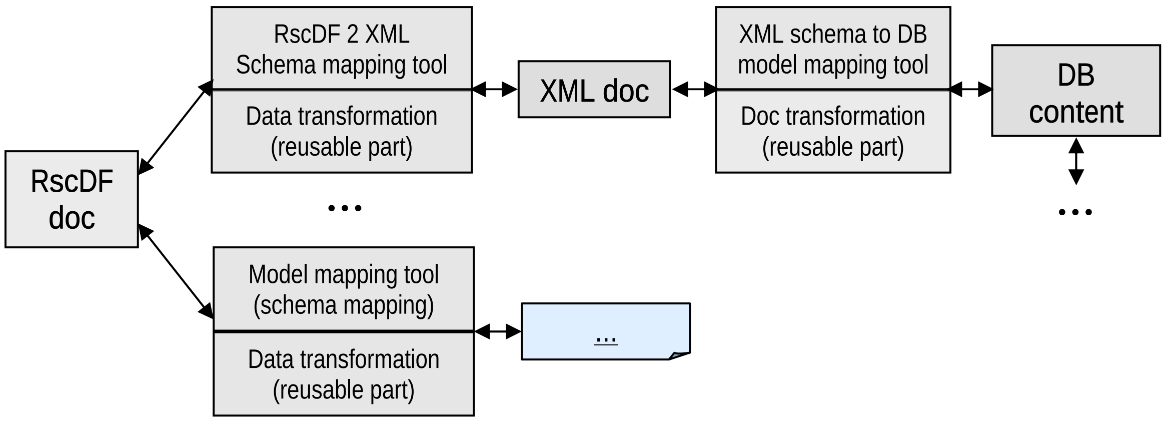

RscDF doc
RscDF 2 XML Schema mapping tool
Data transformation (reusable part)
…
Model mapping tool (schema mapping)
Data transformation (reusable part)
XML doc
...
XML schema to DB model mapping tool
Doc transformation (reusable part)
DB content
…

Figure 24 – Document transformation scheme

As for the mapping tools that can be used in the transformation process, the following existing ones are available:

- RDBMS 2 XML Schema;
- XML 2 XML (XSLT);
- RDF 2 XML;
- etc.

If to talk about automation of the adaptation, it is evident that fully automated semantic adaptation cannot be implemented. The question is what level of automation is possible and how to achieve it.

Given that unambiguous semantic description resources become machine processable, hence automated adapter composition is possible. However, unambiguous semantic description requires human to map the meaning of concepts and relations unless there is already existent common ontology. The tools will be needed to simplify the process of mapping for human. Tools will use faceted classification, adapted for each particular domain in order to make easily accessible the most relevant concepts.

The following cases are essential in a context of automated semantic adaptation:

- *case1*: Explicit mapping (human assisted);
- *case2*: Shared ontology (both resources use same ontology or at least are mapped to it);
- *case3*: Shared ontology lookup & composition (may be wrapped as a service or implemented as an embedded functionality).

9.5 Adapter as software design

The software design of the adapter will require Abstract design of Adapter Backbone using structural and behavioral patterns and Adapter concrete implementation using integration patterns (see Figure 26).

In order to simplify the complexity of Adapter, the following strategy has to be utilized:

- Use model based software development techniques:
    - Pre-defined software abstractions based on integration and design patterns provide a robust framework for developing adapters
- Partitioning logic of adaptation to multiple adapters even for one resource:
    - Integration function (Connection, Parsing, Transformation, etc.)
    - Support function (System Logging, Error handling, Audit Trail, etc.)
- Reuse all external transformers instead of developing transformer functionality in each adapter.

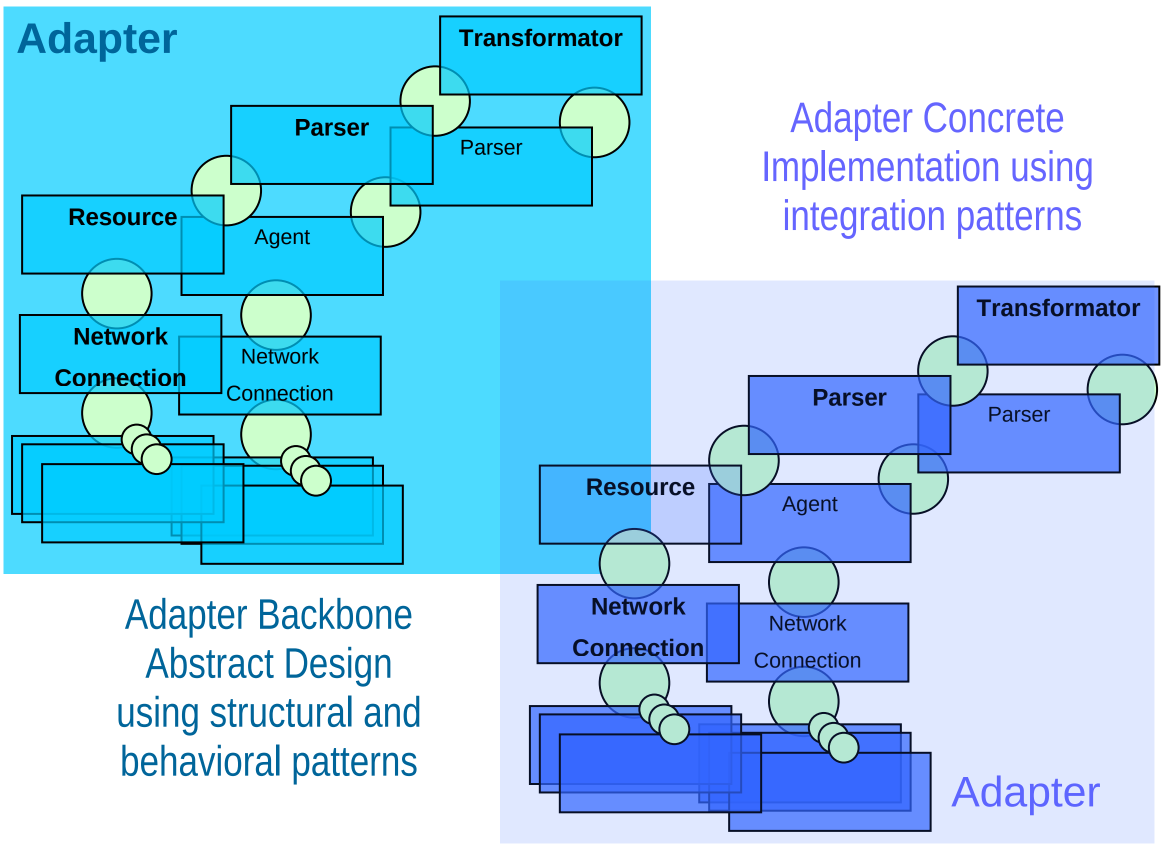


Figure 26 - Adapter design scheme

This part of design (software) includes techniques and methods for software development of components and modules. Different approaches exist for reusable and well structured software design - such as structural patterns, design patterns, etc. Pattern approach allows elaborating well-thought abstract adapter design with further reuse of it for concrete adapter implementation.

For clearer understanding, Data Piping Pattern can be considered as an example of the structural pattern. This pattern fits well to application-to-application adaptation. Each component of this pattern is responsible for a particular function (see Figure 27).

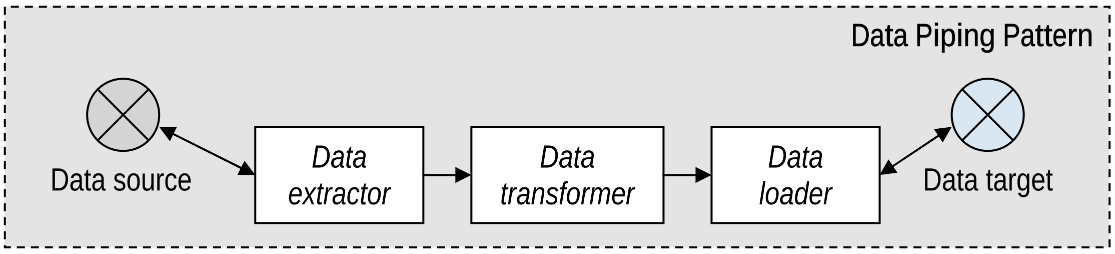


Figure 27 - Sample structural pattern

Data extractor is responsible for getting/extracting data from a source resource. Since we have heterogeneous resources with diversity of access methods (RS 232, Bluetooth, WLAN, LAN, etc), formats of data and APIs, each Data Extractor module must be developed for particular source of data.

After data extraction it is piped to the Data Transformer module. Data Transformer performs transformation involving metadata of formats (schemas) and data transformation rules. Format's metadata (schema) with data transformation rules (mapping) together constitute semantic adaptation.

After transformation process, data are ready to be stored in appropriate place. Data Loader performs this function.

References

[1] XR homepage: XML-to-RDF transformation format, http://w3future.com/xr/, last accessed 19th Oct 2004.

[2] E. Miller, C.M. Sperberg-McQueen: “On mapping from colloquial XML to RDF using XSLT”, Proc. of W3C Extreme Markup Languages 2004, August 3, 2004; Montreal, CA.

[3] MapForce homepage. http://www.altova.com/products_mapforce.html.