Overview

As part of the Connect synthesis enabler, we implemented the mapping-based approach to automatically generate the mediator model. This implementation is made up of three modules:

- (i) The ontology-encoding module,
- (ii) The interface-mapping module, and
- (iii) The mediator-synthesis module.

The ontology-encoding module associates numerical codes to ontology concepts in order to optimize the subsumption testing required during interface mapping. It further takes into account union of concepts. It uses Pellet to reason about the ontology.

The interface-mapping module identifies the semantic correspondence between the actions of the components' interfaces. We formalize interface mapping as a Constraint Satisfaction Problem (CSP) for which we use Choco (an open-source java library for CSP solving) to find all the possible solutions.

The mediator-synthesis module relies on these mappings to generate the mediator. In a first step, we generate the parallel composition of the mapping processes and verify that the overall system successfully terminates using the LTSA (Labelled Transition System Analyser) model checker. In a second step, we are improving the algorithm so as to deal with ambiguous mappings, i.e., when an action from one component may semantically be mapped to different actions from the other component.

The MICS tool has been implemented using Java.

Contributors

- Amel Bennaceur
- Valérie Issarny

Supporting Grant

- Connect -- IST FP7 FET IP - Emergent Connectors for Eternal Software Intensive Networked Systems

Related Research Projects

- Dynamic Synthesis of Connectors

Download

- MICS tool

Case Studies

File Management

GMES

Description

We consider the scenario presented in the Connectproject involving GMES-based systems that are a representative example of Systems of Systems.
GMES
(Global Monitoring for Environment and Security) is the European Programme for the establishment of a European capacity for Earth Observation.

In particular, the emergency management thematic directs efforts towards a wide range of emergency situations involving different European organisations due to, e.g., the cross-border location or criticality. The target GMES system involves highly heterogeneous networked system, which are connected on the fly as they appear in the environment.

We more specifically concentrate on the connection with a video capturing networked system, which may have various concrete instances, ranging from fixed cameras to robots with video sensing capabilities (UGV: Unmanned Ground Vehicle) or flying drones (UAV: Unmanned Aerial Vehicle). In addition, the videos may be accessed from other heterogeneous systems, including applications run on the mobile handheld devices of the various actors on site, and the ones executed by Command and Control (C2) centres.

Ressources

gmes_preliminary.zip (ongoing)

Purchase Order Mediation

Description

To illustrate our approach and provide insight into the benefits of using it to support interoperability between heterogeneous systems, we used the Purchase Order Mediation scenario from the Semantic Web Service (SWS) Challenge . It represents typical real-world problems that are as close to industrial reality as practical. They are intended as common ground to discuss semantic (and other) Web Service solutions and make different solutions becoming comparable with respect to the set of features supported for a particular scenario. The Purchase Order Mediation scenario describes two commercial systems that have implemented using heterogeneous industrial standards and protocols.

The first system, called *Blue*, is a customer ordering products. It initiates a purchase process by starting an order and adding items to it. Then, it places the order giving its client identifier. Finally, it expects a confirmation for each individual item belonging to the order. The exchanged information is formatted using the RosettaNet . RosettaNet is an XML-based standards for the global supply chain and interaction across companies.The interface signature for Blue (abstracted from WSDL 2.0) is given below, where we provide only the ontology concepts associated with the syntactic terms embedded in the interface:

- createOrder(Ø,{orderID})
- addToOrder({orderID,item},Ø)
- placeOrder({clientID,orderID},Ø)
- confirmItem({orderID,item},Ø)

The second system, called *Moon* uses two backend systems to manage its order processing, namely a Customer Relationship Management system (CRM) and an Order Management

System (OMS). First, a client contacts the Customer Relationship Management (CRM) System to obtain relevant customer details. This details are used by the OMS to assess if the client is eligible, i.e. if the customer is known and authorized to creating order. Then, individual items can be added to the order created. First an item is selected, the needed quantity is specified and the the addition to the order is confirmed. Once all the items have been submitted, the order can be closed. The interface signature for Moon is given below, where we provide only the ontology concepts associated with the syntactic terms embedded in the interface:

- login({clientID},{client})
- startOrder({client},{orderID})
- selectItem({orderId,itemID},Ø)
- setItemQuantity({orderId,quantity},Ø)
- confirmItem({orderId,itemID,quantity}, {acknowledgment})
- closeOrder({orderId},{itemID})

A client developed for the Blue Service cannot communicate with the Moon Service due to the following mismatches:

- *Data mismatches*: While Blue specifies its interface using the RosettaNet standard, Moon uses a propriety legacy system in which data model and message exchange patterns differ from those of RosettaNet.
- *Behavioral mismatches*: In the Blue implementation, the client provides its identifier while placing the order whereas in the Moon implementation it has to login before performing any operation. In addition, in the Blue implementation, an item is directly added and only once the order is placed then confirmations are sent while in Moon, first the item is selected, the quantity is specified and then confirmed. Hence, there is no need to send confirmations once the order has been closed.

The SWS-Challenge provides relevant information about the systems involved in two forms: using current Web Service description (WSDL) and natural language text annotations. We interpreted the information, defines the ontology and annotated the description. Indeed, the SWS-Challenge participants are asked to extend the syntactic descriptions in a way that their algorithm/systems can perform the necessary translation tasks in a fully automatic manner. The Moon and the customer Web Services (Blue) are provided by the SWS-Challenge organizers and can not be altered (although their description may be semantically enriched).

Ressources

gmes_preliminary.zip

Travel Agency

Description

This scenario illustrates the role of ontologies in handling heterogeneity both at application and middleware layers. For this purpose, we consider two travel agency systems that have heterogeneous application interfaces and are implemented using heterogeneous middleware protocols (one is implemented using SOAP and the other with HTTP REST). We use application-specific and middleware ontologies to reason about the matching of both application and middleware behaviour.

The first networked system, called EUTravelAgency, is developed as an RPC-SOAP web service. Thus, data is transmitted using SOAP request and response envelopes transported using HTTP Post messages. The service allows users to perform the following operations concurrently:

- Selecting a flight. The client must specify a destination, a departure and a return date. The service returns a list of eligible flights.
- Selecting a hotel. The client indicates the check-in and check-out dates. The service returns a list of rooms.
- Selecting a car to rent. The user indicates the period of rental and their preferred model of car. The service then proposes a list of cars.
- Making a reservation. Once the user has chosen a flight and/or a hotel room and/or a car, they confirm their reservation. The service returns an acknowledgment.

The interface signature for EUTravelAgency (abstracted from WSDL 2.0) is given below, where we provide only the ontology concepts associated with the syntactic terms embedded in the interface:

- SelectFlight({destination,departureDate, returnDate},flightList)
- SelectHotel({checkIndate,checkOutdate,pref}, roomList)
- SelectCar({dateFrom,dateTo,model},carList)
- MakeReservation({flightID,roomID,carID}, Ack)

The second system is called USTravelAgency and allows users to perform the following two operations:

- Finding a trip. The client specifies a destination, departure and return date. The service finds a list of “packages” including a flight and hotel room and car.
- Making a reservation. The user selects a trip package and confirms it. The service acknowledges the reception of the selection.

The interface signature, although giving only embedded ontology concepts, is abstracted as follows:

- FindTrip({destination,departureDate,returnDate,needCar},flightList)
- ConfirmTrip(tripID,Ack)

The USTravelAgency service is implemented as a REST web service over the HTTP protocol.

The findTrip operation is performed as an HTTP GET and the confirmTrip operation is performed using an HTTP POST as shown below (the outputs of both service operations are formatted using JSON):

- GET http://ustravelagency.com/rest/tripervice/findTrip/{destination}/{departureDate}/{returnDate}/{needCar}
- POST http://ustravelagency.com/rest/tripervice/confirmTrip/{tripID}

A client of the EUTravelAgency cannot interact with the USTravelAgency, and similarly a client developed for the USTravelAgency cannot communicate with the EUTravelAgency due to the aforementioned heterogeneity dimensions:

- *Application data.* The EUTravelAgency refers to the Flight, Hotel and Car concepts, whereas the USTravelAgency makes use only of the Trip concept. Additionally, the EUTravelAgency specifies the departure and the return dates using Greenwich Mean Time (GMT), while the USTravelAgency uses Pacific Standard Time (PST) to describe them.

- *Application behaviour.* In the EUTravelAgency implementation, users can independently select a flight, a room and a car, whereas in the USTravelAgency implementation all of them are selected through a package.
- *Middleware data format.* The data exchanged in the EUTravelAgency implementation are

encapsulated in a SOAP message, while the input data of the USTravelAgency are passed through a URL and the output data are formatted using JSON.

- *Middleware behaviour.* REST and RPC-SOAP are different architectural styles and induce heterogeneous control and communication models.

Ressources

travelAgency.zip

Photo Sharing

Description

We study the scenario of photo sharing within a public space such as a stadium. Considering the ever-growing base of content-sharing applications for handhelds, numerous versions of the photo sharing application may be available on the spectators' handhelds, thus calling for appropriate interoperability solutions that mediate interaction protocols from the application down to the middleware layer.The target environment allows for both infrastructure-based and ad hoc peer-to-peer photo sharing.

In the former implementation, a photo sharing service is provided by the stadium, where only authenticated photographers are able to produce pictures while any spectator may download and even annotate pictures. Three types of networked are identified, which are respectively associated with the producer, consumer and server systems. The definition of the systems' actions specify the associated SOAP functions, i.e., the client-side application actions are invoked though SOAP middleware using the SOAP-RPCInvoke function, while they are processed on the server side using the two functions SOAP-RPCReceive and SOAP-RPCReply. The specific applications actions are rather straightforward. For instance, the producer invokes the server operations Authenticate and UploadPhoto for authentication and photo upload, respectively. The consumer may possibly search for, download or comment photos, or download comments. Finally, the actions of the photo sharing server are complementary to the. The producer interface is abstracted as follows:

- Authenticate(login,authenticationToken)
- UploadPhoto(photo,acknowledgment)

The consumer interface is abstracted as follows:

- SearchPhotos(photoMetadata,photoMetadataList)

- DownloadPhoto(photoID,photoFile)
- DownloadComment(photoID,photoComment)
- CommentPhoto(photoComment,acknowledgment)

The peer-to-peer implementation allows for photo download, upload and annotation by any spectator, who are then able to directly share pictures using their handhelds. The peer-to-peer-based implementation defines a single interface signature, as all the peers feature the same capability. It further illustrates the naming of actions after domain data types of the application data instead of operations since the actions are data-centric and are performed through functions of the Lime tuple-space middleware:

- <Out,PhotoMetadata,Ø,photoMetadata>
- <Out,PhotoFile,Ø,photoFile>
- <Rdg,PhotoMetadata,photoMetadata,photoMetadataList>
- <Rd,PhotoFile,photoID,photoFile>
- <Rd,PhotoComment,photoID,photoComment>
- <Out,PhotoComment,Ø,photoComment>
- <In,PhotoComment,photoID,photoComment>
- <Rd,PhotoComment,photoID,photoComment>

In both cases, the spectator’s handheld would need to embed the appropriate software application, which may not be available due to the handheld’s specific platform. Further, the spectator may not be willing to download yet another photo sharing application, i.e., the proprietary implementation offered by the stadium, while one is already available on the handheld.While the photo sharing functionality is present in both implementations, it is unlikely that they feature the very same interface and behavior.In particular, the RPC interaction paradigm suits quite well the infrastructure-based service, while a distributed shared data space is more appropriate for the peer-to-peer version.

Ressources

photoSharing.zip

Instant Messaging

See Instant Messaging Interoperability

-