Use Cases in Scribble

The Scribble Team

April 24, 2012

How to read this document

This document presents our experience in expressing several protocols in Scribble. The examples are gathered from different papers and projects. Each use case is explained with a citation from its source. Then the Scribble code matching the use case is illustrated. Some source descriptions are general and do not provide any implementation details. However, they are sufficient for our aims. We have tried to capture the high level workflow of each example rather than the concrete implementation details.

Contents

1. Purchasing protocol from Red Hat Savara project [9] 2
2. Network protocol (EmployabilityNetwork) [1] 3
7. Amazon e-commerce service for Item Search [4] 8
8. SQL service [8] 9
1 Purchasing protocol from Red Hat Savara project [9]

**Description:** "The example provides a simple choreography where a buyer interacts with a store to buy an item. The store checks the buyer’s credit before instructing a logistics organization to deliver the item. In the process of carrying out this business transaction, the credit agency may return a fault if the customer is not known, or the store reject the purchase request if the credit rating is too low." [9]

**Description:** The use case is taken from the webpage of Savara project. The Savara team has provided Java implementation for each role in the protocol.

```java
/*Import of types:
type="{http://www.jboss.org/examples/store}BuyRequest"
type="{http://www.jboss.org/examples/creditAgency}CreditCheckRequest"
type="{http://www.jboss.org/examples/creditAgency}CreditRating"
type="{http://www.jboss.org/examples/logistics}DeliveryRequest"
type="{http://www.jboss.org/examples/store}BuyConfirmed" */

protocol Purchasing (role buyer, role store, role creditAgency, role logistics) {
    buy(BuyRequest) from buyer to store;
    creditCheck(CreditCheckRequest) from store to creditAgency;
    choice at creditAgency{
        CreditRatingType(rating: int) from creditAgency to store;
        choice at store{
            Deliver(DeliveryRequest) from store to logistics @rating > 5;
            DeliveryConfirmed from logistics to store;
            BuyConfirmed from store to buyer;
        } or { BuyFailed from store to buyer @rating <= 5;}
    } or {CustomerUnknown from creditAgency to store;}
}
```

Figure 1: Purchasing example from SAVARA project
2 Network protocol (EmployabilityNetwork) [1]

Description: "The EmployabilityNetwork provides on-line access to an internships management and work placement system for the students in a University. Specifically, this scenario foresees that each department of a University has a Placement Service Coordinator (PSC) that forwards applications from students to one or more Placement Provider (PP) looking for a match." [1]

```plaintext
/*PSC = Placement Service Coordinator PP = Placement Provider */
protocol EmployabilityNetwork (role Student, role PSC, role PP) {
    Application from Student to PSC;
    Application from PSC to PP @has_match(Student, PSC, PP);
}
```

Figure 2: EmployabilityNetwork example
3 Vehicle subsystem protocol [5]

Description: "The Central Locking System (CLS) is a well-studied and documented example of one common automotive vehicle subsystem. For reasons of brevity, we present a simplified and abstract adaptation of the CLS. In this paper, we focus on the CLS functions for locking and unlocking vehicle doors triggered by a wireless entry device; this involves user identification, signaling by flashing external lights, and user preset loading for on-board multimedia devices. The main interacting entities we call them roles of the CLS are remote entry key fob (KF), controller (CONTROL), lock manager (LM), security module (SM), lighting system (LS), and database (DB) with driver presets. These logical entities communicate locally or over the network to provide the above-mentioned functions. Fig. 3 shows roles and their communication links." [5] as well as the Scribble code for both protocols.

```
protocol LockAndSignaling (role KF, role Control, role LM, role LS) {
    unlock from KF to Control;
    unlock from Control to LM;
    ok from LM to Control;
    door_unId_sig from Control to LS;
    ok from Control to KF;
}

protocol TransferDriverID (role KF, role Control, role SM, role DB) {
    unlock from KF to Control;
    handle_id from Control to SM;
    get_id from SM to KF;
    id from KF to SM;
    id from SM to DB;
    id_handle from SM to Control;
    ok from Control to KF;
}
```

Figure 3: Vehicle Subsystem. MSC and its Scribble correspondence.
4 Travel web service protocol [3]

Description: "The process starts with John looking for suitable locations for his get-away weekend, locations must satisfy certain requirements (they must be close to where he lives, by a lake, near the mountains, etc, for simplicity we check only the distance). Using his office computer, John interacts with a fairly simple orchestrated process that guides him in finding the location, booking the rooms in a hotel, etc. Given a request that species the departure location (i.e., a location name and GPS coordinates), a maximum traveling distance the client is willing to go, and an articulate description (the format of which is omitted for simplicity)of key interests, such as proximity to a lake, mountains, etc., the web service responds with an array of possible holiday locations." [3]

Comment  The protocol is a RPC with assertions on a travel distance. For Assertion Plug-in we use a Python predicate evaluator.

Challenges: passing complex data types and assertions on complex types. The protocol is not that interesting in terms of a structure, but in terms of assertions. Note that we import the data types from Python modules. A better approach would be to specify the types in yml and to use common serialization module when treating the types. Scribble still does not support assertions of complex data types between different languages.

Source  page 246. 247

```plaintext
import dataTypes.RequestSpec
import dataTypes.LocationResults
protocol LocationFinder(role client, role service) {
    findLocation(spec:RequestSpec) from client to service;
    (locations:LocationResults) from service to client
    @for l in locations:
        correct = (((spec.departureLocation.Coordinate_Easting - 1.Coordinate_Easiting) *
        (spec.departureLocation.Coordinate_Easting - 1.Coordinate_Easiting) -
        (spec.departureLocation.Coordinate_Northing - 1.Coordinate_Northing) *
        (spec.departureLocation.Coordinate_Northing - 1.Coordinate_Northing))
        ≤ spec.maxLocationDistance*spec.maxLocationDistance)
        if correct == False: break;
        return correct;
    }
}
```

Figure 4: Travel Service example case
5 Map web-service protocol [3]

Description  "The example scenario states that John and his wife decide to travel by car. The car is equipped with a haptic device to communicate with remote services for entertainment reasons (e.g., purchase a multimedia stream), or for gathering useful information from the environment. John decides to use his device to obtain a map illustrating how to reach the vacation resort. The device can show only certain image formats with a given resolution. Therefore, it is important that the map returned by the external service satisfies both requirements. Suppose that Johns workflow has a pre-installed binding to a free-of-charge external service that does not always guarantee fulfillment of the requirement. It may, in fact, sometimes deliver maps whose format or resolution are invalid for the haptic device. The monitor is therefore turned on to allow for delivery of unacceptable maps to be trapped. A suitable reaction to a detected anomaly might consist of switching to another service provider who provides maps under payment." [3]

Comment  The challenge in the following example is that the monitor applies a recovery policy, which is still not yet supported by our monitor. However, we can implement the recoverability by using another service. Since we do not support an optional clause we insert an extra OK message to simulate the choice.

Source  page 247. 248

```
1 protocol MapService(role Device, role FreeMapService,
2          role PayedMapService) {
3     GetMap(location) from Device to FreeMapService;
4     Map(format, resolution) from FreeMapService to Device;
5     choice at Device
6     { OK from Device to FreeMapService;}
7     or {GetMap(location) from Device to PayedMapService;
8         Map(format, location) from Device to PayedMapService;
9         OK from Device to PayedMapService}
10  );
11 }
```

Figure 5: Map Service example case
protocol Bidding(role Client, role BidderService) {
  Login(userInfo) from Client to BidderService;
  Sell(sellNo: itemNo) from Client to BidderService;
  choice at Client {
    Bid(bidRef1: bidRefNo, bidNo: itemNo) from Client to BidderService @ {bidNo == sellNo};
    choice at Client {
      Bid(bidRef2: bidRefNo, bidNo: itemNo) from Client to BidderService @ {bidNo == sellNo};
      choice at Client {
        Bid(bidRef3: bidRefNo, bidNo: itemNo) from Client to BidderService @ {bidNo == sellNo};
        choice at Client {
          Retract(retractRef3: bidRefNo) from Client to BidderService @ {bidRefNo3 = retractRef3};
          Logout from Client to BidderService;
        } or {
          Retract(retractRef2: bidRefNo) from Client to BidderService @ {bidRefNo2 = retractRef2};
          Logout from Client to BidderService;
        } or {
          Retract(retractRef1: bidRefNo) from Client to BidderService @ {bidRefNo1 = retractRef1};
          Logout from Client to BidderService;
        } or {
          Logout from Client to BidderService;
        }
      } or {
        Logout from Client to BidderService;
      }
    } or {
      Retract(retractRef2: bidRefNo) from Client to BidderService @ {bidRefNo2 = retractRef2};
      Logout from Client to BidderService;
    } or {
      Retract(retractRef1: bidRefNo) from Client to BidderService @ {bidRefNo1 = retractRef1};
      Logout from Client to BidderService;
    } or {
      Logout from Client to BidderService;
    }
  } or {
    Logout from Client to BidderService;
  }
}

Figure 6: Bidder Example

6 Bidding protocol [6]

Description: "Let us consider an auctioneer service that provides auction services on the Web. The service is able to accept registrations from new bidders/sellers and hold auctions among registered bidders. It provides several operations to allow users to query the information of auction items, register and unregister themselves to the service, login and logout the service, bid or sell an item. The service also provides an operation allowing bidders to retract their previous bids.” [6] The properties that the service should satisfy (as given in the paper) are given below:

- \( \text{opBid exists at most 3 times after opLogin until opLogout} \)
- \( \text{opBid precedes opRetract after opLogin until opLogout} \)
- \( \text{opSell precedes opBid where opSell.itemNo = opBid.itemNo} \)

Comments: Expressing the above properties in Scribble is challenging because of the lack of support of an optional clauses and bounded recursions. Although the properties can be expressed in the current Scribble version, adding the above features will greatly simplify the protocol.
protocol ItemService(role Client, role ItemService) {
  choice at Client{
    ItemSearch(search_criteria:string) from Client to ItemService;
    (ProductList) from ItemService to Client;
  } or {
    CartCreate(uid, n:int) from Client to ItemService;
  } or {
    CartGet(cartID:int) from Client to ItemService;
  } or {
    CartModify(cartID:int) from Client to ItemService;
  } or {
    CartClear(cartID:int) from Client to ItemService;
  }
}

Figure 7: Amazon Item Service high level protocol

7 Amazon e-commerce service for Item Search

[4]

Description: For simplicity, in the paper the Amazon Item Service is expressed with several core operations: “ItemSearch searches Amazon’s database for items matching some keywords. It returns a list of products that match the search criteria; CartCreate takes an ASIN a unique identifier for that item in Amazon’s database and a positive integer n and creates a new shopping cart that represents a request for n copies of that item. The carts contents is stored and managed by Amazon, the SOAP operations refer to the cart by passing its unique ID; CartGet returns the content of a given cart; CartAdd adds a new row to a given cart requesting n copies of some ASIN; CartModify is used to change the quantity of some item in a given cart (0 deletes it); CartClear empties the cart with given ID.” [4]
8 SQL service [8]

Description: "SQL service can receive (req?) and answer (result!) requests, stops (halt!), or halts temporarily for maintenance purposes (maintenance? and activation?)" [8]

```
protocol SqlService(role Service, role Client1, role Client2, role Client 3) {
    parallel {
        halt from Client1 to Service;
    } and {
        request(data:string) from Client2 to Service;
        result(id:int, res:string) from Service to Client2;
    } and {
        maintainance from Client3 to Service;
        activation from Client3 to Service;
    }
}
```

Figure 8: An SQL Service example
9 Online store [8]

Description: The example study the compatibility of services. That is why Figure 9 gives two finite state machines (as given in the source paper): the service and one of the customers.

```
protocol OnlineStore(role Store, role Customer2, role Customer1) {
    parallel {
        search from Customer1 to Store;
        reply from Store to Customer1;
    } and {
        update from Customer2 to Store;
        confirm from Store to Customer2;
    }
}
```

Figure 9: Online store example
10 Online Shopping System [2]

**Description:** "The second system is an Online Shopping System (OSS) that implements a typical online shopping service. Two safety properties of OSS, correspond to "A premium customer always gets a discount on his/her purchase" and "An order cannot be billed before being marked complete by the customer", respectively. The liveness property of OSS, is "A completed order will be eventually billed"." [2]

**Comments:** Due to a lack of information for the concrete implementation details of the service we have choosen our own format of the service calls that can match the property description.

```
protocol OnlineShopping(role Customer, role Seller) {
    Order from Customer to Seller;
    choice at Customer{
        CompleteOrder from Customer to Seller;
        choice at Seller {
            BillWithDiscount from Seller to Customer;
            or {
                OrdinaryBill from Seller to Customer;
            }
            or {
                Cancell from Customer to Seller;
            }
        }
    }
}
```

Figure 10: Online Shopping Systems example
11 Travel Booking System [2]

Description: "A web-based travel booking system (TBS) acts as a broker offering its customers the ability to book all aspects of a trip. The workflow of TBS includes credit validation, flight/hotel/car reservation, and communication with the client. Customers can submit data about their desired travel plans and receive either a confirmation number or a failure message depending on whether the travel arrangements have been made successfully. The activity diagram in Figure 11 shows high level steps that are executed during the travel booking process. To fulfil its business goal, TBS needs to interact with several partners: CreditCardCheckingService, which validates the customers credit card data, FlightReservationService, which books a flight, HotelReservationService, which reserves a hotel room, and CarReservationService, which makes a car reservation. In a typical scenario, an Internet customer begins an interaction with TBS by entering data for his/her travel arrangements. The system then invokes CreditCardCheckingService, and if the credit card is valid, then up to three reservations, for the car, hotel and flight, are made. If all of the reservations are completed successfully, a confirmation number is generated and returned to the customer. Since the TBS system, like other web services, is a composition of several distributed business processes, its success depends on the correctness of its partners, and the interoperability between them is a major quality concern. For example, the system needs to guarantee that it processes travel reservations only for customers with valid credit, or that every request is acknowledged and none are lost or blocked indefinitely. Since each web service is a relatively simple process, analysis can concentrate on the message exchange between partners their conversations.” [2]
12 Banking example [7]

Description: This example had been used as a proof of concept for the integration of a ConversationMonitor in the OOI project infrastructures. The explanation of the example, Scribble code and links to the related code base can be found at OOI Conversation Governance Page [7]

References

/* We cannot verify that the booking is invoked at most 3 times and that each service can be invoked only once. Although we can express that property by writing the permutation of the three different reservations. For brevity (the protocol becomes too large), we have not express that requirement. */

protocol TravelBookingSystem (role Customer, role TBS, role CreditCardCheckingService, role FlightReservationService, HotelReservationService, CarReservationService) {
    TravelData from Customer to TBS;
    CheckCredit from TBS to CreditCardCheckingService;
    CreditResult from CreditCardCheckingService to TBS
    choice at TBS {
        rec ReservationLoop {
            choice at TBS {
                FlightReservation from TBS to FlightReservationService;
                ReservationLoop;
                or ( HotelReservation from TBS to HotelReservationService;
                    ReservationLoop;)
                or ( CarReservation from TBS to CarReservationService;
                    ReservationLoop;)
                or ( ConfirmedReservation from TBS to Customer;)
                or ( InValidCredit from TBS to Customer;)
            }
        }
    }
}

Figure 11: Travel Booking System example
protocol BuyBond(role client, role bank, role resource_registry) {
  buy_bonds from client to bank;
  read from bank to resource_registry;
  (Account) from resource_registry to bank;
  choice at bank{
    (Exception) from bank to resource_registry;
  } or {
    (Exception) from bank to resource_registry;
    (CustomerId) from resource_registry to bank;
    exercise(Order) from bank to trader;
    (Confirmation) from trader to bank;
    choice at bank {
      update from bank to resource_registry;
      (status:string) from resource_registry;
      (status:string) to client;
    } or {
      (status:string) from bank to client;
    }
  }
}

Figure 12: Bank (buy bonds) example.