Assignment: Modelling Web Services in mCRL2 and UPPAAL

Software Architecture and Calculi - 2018-19

To do: Write a report using LaTeX including the answers to the exercises below. The complete mCRL2 and UPPAAL code for the project must be supplied in annex so that it can be fully tested.

To submit: The report in PDF by email to lsb@di.uminho.pt.

Deadline: 6 June 2019 @ 23h59 (Friday)

Objectives

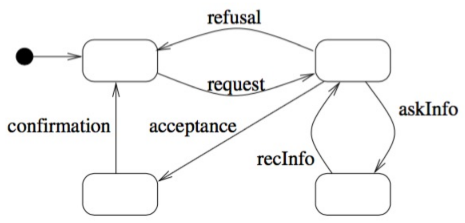
Developing web services (WSs) raises many software engineering issues, some of which are new and some of which have been recurrent problems already encountered in previous programming paradigms. Implementing WSs is error-prone, because of the complex interactions and message exchanges that have to be specified.

A more unusual specificity that distinguishes them from more traditional software components is their being accessed through the internet. WSs are distributed, independent processes which communicate with each other through the exchange of messages. The central question in WS engineering is therefore to make a number of processes work together to perform a given task.

Such systems need to match the requirements expressed by their users, and it is therefore needed that these requirements be stated accurately. In this project, you are required to model an instance of a WS-based system, according to the informal requirements below, in mCRL2 and UPPAAL. You are completely free to design such a system in a non trivial way. In any case, you should

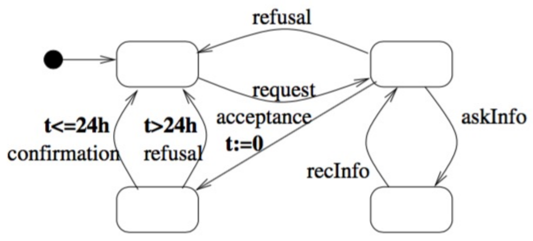
- introduce WS resources as independent, concurrent processes;
- make use of at least a user-defined data type in mCRL2;
- test your system against a number of (non trivial) properties formulated in the extended Hennessy-Milner logic implemented in mCRL2 and verified with its model checker;
- develop, simulate and verify a timed version of your system in UPPAAL.

WSs are essentially processes and any meaningful representation of services should take into account behavioural information. An essential reason behind this need is that the interaction between WSs is typically more complex than, for instance, simple (Remote) Procedure Calls. Knowing the signature of the parameters expected by a RPC is essentially sufficient to use it. On the contrary, even very simple services like the basic hotel booking service shown on the diagram below, involve more complex interactions, since the service can dialogue with the caller, for instance to ask for complementary information.



Clearly, it is not sufficient to know that the service receives three types of messages (booking requests, confirmation and extra information) to understand the way it interacts with other services, because its behaviour may follow different scenarios depending on different context conditions. Typically, for example, in a hotel reservation system, one needs to know that the booking request comes first, and that it can be followed by a number of requests for more information, by a refusal or by a confirmation.

The real-time dimension, on the other hand, cannot be swept under the carpet. Even a very simple scenario may require the specification of waiting times for specific events, and the verification that the concerned web service will never wait for a response for longer than the specified duration. This could be useful, for instance, for a hotel which does not wish to wait for more than one day for a confirmation — cf, the diagram below.



The system's minimal requirements

The example to be handled in this assignment is related to the field of public welfare and extracted from a larger domain analysis concerning the local government of a municipality. When implemented, this software system aims at supporting elderly citizens in receiving sanitary assistance from the public administration.

This problem involves several actors. In particular, it is composed of services involved when elderly people apply for a sanitary assistance: the sanitary agency satisfying requests, the transportation service, the meal delivery, the bank managing funds.

- A **citizen** posts a request, exchanges information with the agency, waits for a response, and if accepted receives a service and pays fees.
- A **sanitary agency** has to manage requests submitted by elderly citizens. First, it asks some information to the citizen who has posted the request. Depending on that, it sends either a refusal and waits for a new request (as reflected by a recursive definition), or the request is accepted. In the latter case, a synchronization is performed with the cooperative controller (it controls in some way both cooperatives) to order the delivery of concrete (transportation or meal) services. Then, the agency pays some public fees to the bank and waits from the bank component for a signal indicating that the transaction4 is completed.
- The **transportation** (respectively, **meal**) **cooperative** provides its service, warns the bank to start the payment, and waits for the reception of its fees. A controller receives a notification of the agency and launches one of the possible services.
- The **bank** receives a firing message from the cooperative involved in the current request. The different payments are then performed: payment of public fees by the agency and of private fees by the citizen, payment of the cooperative by the bank. Note that the specified strategy for this service is to collect the money first and only then to pay for the service.

The specification of time constraints on the system is left to you; in any case such requirements must be clearly explained and formalized.