Service Composition Implementation Details

Validation Checks

The following validation checks are performed on the service composition request submitted by the user. If any of these checks fail, the service composition process is aborted immediately.

- A composition request constraint must have exactly 3 elements (separated by the pipe symbol) in the sequence: type, operator and literal value. E.g. “float : Price | < | 100.0”.
- Operators that can be used in a constraint currently include <, >, =, <= and >=. These are validated against the Operator enumeration defined in the service repository implementation.
- A composition request must provide at least 1 input.
- A composition request must request at least 1 output.
- Quality of Service features that can be used in a composition request currently include COST, RESPONSE_TIME, RELIABILITY and AVAILABILITY. These are validated against the QualityOfService enumeration defined in this service composition implementation.
- A composition request constraint must be applied only on (i.e., should have as type) the parameters provided as input, output or QoS in the request.

Below are some additional validation checks performed once a valid composition request is created. If any of these checks fail, the service composition process also fails.

- The service repository must contain at least 1 service.
- The composition problem should be solvable based on the given request, repository and algorithms, i.e. at least 1 solution should be obtained at each step of the composition process.
- The composition problem should not be solvable by a single service present in the given service repository, thereby rendering service composition unnecessary.

Differences between Algorithm and Implementation

- In the algorithm, composition request and set of available services are provided as input. However, in the implementation, user is prompted to provide the composition request inputs, outputs, QoS features and constraints and the service repository file path. These details are then used to create a composition request and parse the repository file to obtain the list of available services.
  Reason: In the implementation, this class is the starting point and the driver for the entire service composition process. Therefore, the inputs cannot be provided as parameters to the method; they need to be fetched from the user.

- The algorithm did not include any specific validation checks. However, in the implementation, several checks are performed on the composition request, service repository and results of the various stages of the service composition process.

Reason: These validation checks ensure that the composition process continues after completing each step only if all the execution parameters are available and valid and if it is worth triggering the next step so as to minimize the effort involved in case of a failure.

- The loops around certain algorithms that are triggered by the service composition algorithm are not implemented. Instead they are included in the respective triggered algorithms themselves.
  Reason: This has been done for better modularity and lower coupling. Since the complete implementation of the subsequent algorithms is now contained within their own classes, any future modifications in those processes (if required) would not affect the service composition implementation.

Notes

- As per the Constraint class defined in the service repository implementation, each constraint object must have the service name data member populated. However, the constraints provided in a composition request are not associated with any individual service. Therefore, in the implementation, a dummy service name "CompositeService" is associated with the requested constraints.
- Comparison of input, output and QoS during validation consider the case of the parameter, i.e., the comparison is case-sensitive.
- The input and output names for any service (atomic or composite) have 2 parts separated by a colon preceded and succeeded by a space character: data type and name. E.g., "float : Price", "string : ProductName", etc.
- This format is followed even when an input or output appears in a constraint. E.g., "string : Delivery Address | = | Montreal".
- The data types currently handled are: int, char, float, string and boolean.
- Since there is no processing being done on the QoS features as of now, no data types have been associated with them.

Future Work/Limitations

- The implementation currently accepts only XML file representations of a service repository. It could be made more versatile by allowing the user to choose from several available options and then using the appropriate repository parser for the chosen medium.

- Currently, the operators that can be used in constraints include only <, >, =, <= and >=. More operators can be added to the Operator enumeration and means to validate and evaluate them can be implemented.

- Currently, the QoS features that can be used in composition requests include only COST, RESPONSE_TIME, RELIABILITY and AVAILABILITY. More features can be added to the QualityOfService enumeration and means to validate and process them can be implemented.

- Currently, int, char, float, string and boolean data types are being handled for input and output parameters. The functionality can be extended in future to handle more data types. Also, if QoS features are processed in future, data types would need to be associated with each of them.