Replacement Policies for Service-Based Systems

Khaled Mahbub and Andrea Zisman

Software Engineering Research Group
School of Informatics
City University London
{k.mahbub, a.zisman}@soi.city.ac.uk

Presenter: Ricardo Contreras

Outline

- Motivation
- Overview of Framework
- Replacement Policies
- Implementation Aspects
- Evaluation
- Conclusion and Future Works

Motivation

- There are several situations that may trigger the need to change service-based systems during execution time, including
  - changes in the context of the service-based system environment or their participating services
  - changes in functional and quality aspects of services participating in service-based systems
  - failures in or unavailability of services participating in service-based systems
  - emergence of new services
  - changes in or emergence of new requirements

Motivation

- Many issues to be considered for dynamic adaptation of service based systems
  - What needs to be changed (*what*)
    - replacement of a service by another service, or a composition of services
    - changes in the execution process (e.g., conditions, loop statements, variables, exception handlers)
  - The ways that the changes need to be executed (*how*)
    - stop the system, make the necessary changes, and resume the system
    - use proxy services as place holders for the services in a composition, instead of having concrete services referenced in the system
    - use an adaptation layer based on aspect oriented programming with information about alternative services
  - The moment that the changes should be performed in the systems (*when*), to avoid
    - Unnecessary changes
    - Side effects introduced by the change

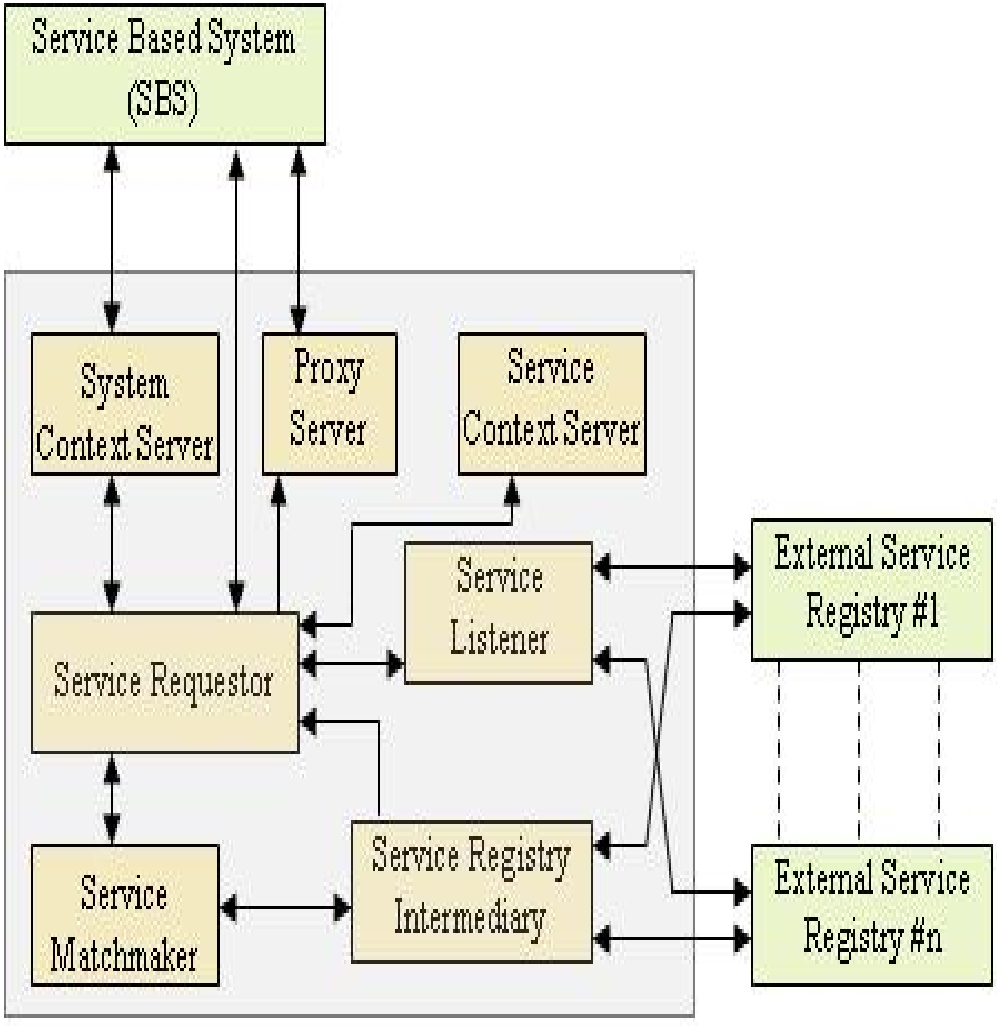
Motivation

- Most existing approaches replace services in service-based systems with alternative services, or composition of services. However these approaches do not consider
  - the problem that triggers the need for changes
  - when to execute the changes
  - how to execute the changes

Overview of Framework

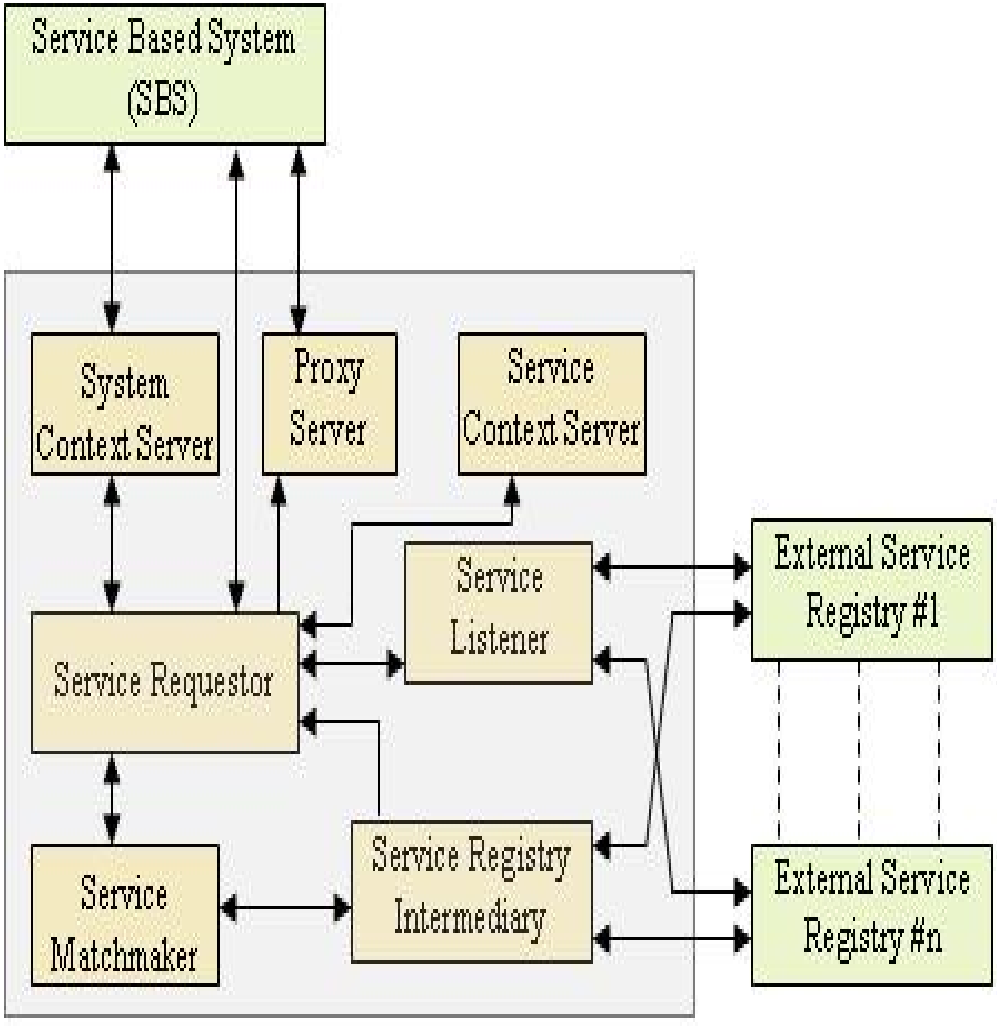
- We use a proactive service discovery framework to identify replacement services in parallel to the execution of the system due to the triggering cases described before.
- We use proxy server to support changes in the system during execution time, avoiding changes in the original service-based system
- Changes in a service-based system consist of replacing a service participating in the system by another service

Overview of Framework



- The discovery framework supports the identification of services based on structural, behavioral, quality, and contextual criteria represented as complex queries.
- The queries are specified in an XML-based query language named SerDiQueL [7].
- *External service registries* contains service specifications as multi-faceted descriptions including service interfaces represented as WSDL, behavior specifications represented as BPEL, and quality and contextual specifications represented as different XML schema
- The *service requestor* orchestrates the functionality offered by the other components in the framework.

Overview of Framework



- The *service matchmaker* parses the different criteria of a query and evaluate these criteria against service specifications in the service registries.
- The *service context server* and the *system context server* disseminate context information of the services and system environment
- The *service listener* notifies the service requester about a new service that becomes available or about changes in the characteristics of a service.
- The *proxy server* supports replacement of services in service-based systems.
- The *service registry intermediary* provides an interface to access services from various registries.

Replacement Policies

- Replacement policies to replace participating services in a service based system due to
  - unavailability or malfunctioning of services
  - changes in the structural, functional, quality, or context of services
  - availability of a new service
  - changes in the context of the system's environment, changes in requirements, or emergence of new requirements
- The replacement policies take into consideration the position of a service S that may need to be replaced with respect to the current execution point of the system. There are three different positions that are considered:
  - *not_in_path*: when service S in not in the current execution path of the system, i.e., S appears in a different branch of the system's execution path or before the current point in the execution path;
  - *current*: when service S is in the current execution point of the system;
  - *next_in_path*: when service S is in the current execution path of the system, and will be invoked some time in the future.
- In our framework we consider cases when
  - changes are required to be performed so that the system can continue its operations
  - changes that can wait to be performed after the current execution of the system; and
  - no changes are required.

Replacement Policies

- Notations
  - P the process of the service-based system being executed
  - S_P a subscribed service being used in P that may need to be replaced
  - Q a subscribed query associated with S_P that is composed of structural, behavioural, quality or contextual constraints
  - d(Q,S) the distance between a service S and a query Q
  - $d_{max}(Q)$ the threshold of acceptable distance values between services and query Q
  - Set_S the set of subscribed candidate services for Q, including S_P, ranked in ascending order of the distances between the services and query Q
  - $d_{delta}(Q)$ the threshold used to decide about the replacement of S_P in P in different situations
  - *pos(S)* a function that returns the position of service S in process P. The returned values of *pos(S)* can be *not_in_path*, *current*, or *next_in_path*

Replacement Policies

- **A subscribed service S becomes malfunctioning or unavailable**

```
E(unavailable/malfunctioning, Q, S)
If S is being used in P then //S == Sp
 If S_Set is empty then
       // There are no available candidate services to replace SP
    If pos(Sp) == not_in_path then
      mark Sp for replacement when  Sp is accessed in a future execution of P;
    If pos(Sp) == current then
      If exist_exception(P) then  execute exception handler in P;
      else throw exception(unavailable/malfunctioning,S) ;
    If pos(Sp) == next_in_path then
      mark Sp for replacement when  Sp is accessed in the current execution of P;
   If S_Set is not empty then
    If pos(Sp) == not_in_path then
       mark Sp for replacement when  Sp is accessed in a future execution of P;
    If pos(Sp) == current then  replace Sp by S0;
    If pos(Sp) == next_in_path then
       mark Sp for replacement when Sp is accessed in the current execution of P;
```

Replacement Policies

- **A new service S becomes available**

```
E(new,S):
If S is in Set_S then  //the new service is a candidate service
   If S0 != S then  do nothing;  // the new service S is not the best service
   If S0 == S //the new service S is the best service
       If d(Q, Sp) – d(Q, S0) <= d_delta(Q) then
          mark Sp for replacement when  Sp is accessed in a future execution of P;
       If d(Q, Sp) – d(Q, S0) > d_delta(Q) then
      If pos(Sp) == not_in_path then
           mark Sp for replacement when  Sp is accessed in a future execution of P;
      If pos(Sp) == current then  replace Sp by S0 ;
      If pos(Sp) == next_in_path then
        mark Sp for replacement when  Sp is accessed in the current execution of P;
If S is not in Set_S then do nothing; //the new service is not a candidate service
```

Implementation Aspects

- A prototype tool of the framework has been implemented in Java.
- The tool is available as a web service and can be deployed by any client that can produce service requests in the format required by the framework.
- The subscription of the services is supported by WS-Eventing and by an event receiver.
- The external service registry uses eXist database.
- Communication with the registry is through the use of Remote Method Invocation (RMI).
- The proxy server has been implemented as an HTTP server using Java socket programming.

Evaluation

- The work was initially evaluated by comparing times
  - to execute a service-based system without the need for changes
  - to execute the system when changes are required using our replacement policies
- In the evaluation we used a *Route-Planner* service-based system, specified as a BPEL process that allows to plan optimal route between two locations. This systems composed of web services including
  - S_GPS : Global Positioning Service that provides exact location of a car
  - S_e-AZ: a web service that provides the coloured electronic map of an area.
  - S_RT: a web service that provides traffic information in and area.

Evaluation

- A query was specified in SerDiQueL concerned with the identification of candidate services to replace the Global Positioning Service (S_GPS) in the system.
- The query was executed for the following situations
  - service S_GPS becomes unavailable
  - there is a change in service S_GPS
  - a new better service S_GPS' becomes available

No required changes	Service unavailable	Change in service	New service
0.48	0.56	0.52	0.54

Conclusions and Future Works

- Conclusions
  - We presented a set of replacement policies to support changes in service-based systems due to different situations such as failures in or unavailability of services participating in service-based systems or changes in functional and quality aspects of services participating in service-based systems.
  - The replacement policies consider the cases in which changes need to be performed so that the system can continue its operations; changes can wait to be performed after the current execution of the system; and no changes are required
  - Initial experiment of the work has shown that the use of the replacement policies does not cause an overhead in the performance

- Future Works
  - Extension of replacement policies to support the adaptation of the service based system by dynamic modification of the composition process.
  - Extension of replacement policies to support adaptation of the service based system by replacing a service by a composition of services.

References

1. Ardagna, D., Comuzzi, M., Mussi, E., Pernici, B., Plebani, P.: PAWS: A Framework for Executing Adaptive Web-Service Processes. IEEE Software, 24 (6), (2007).
2. Baresi, L., Ghezzi, C., Guinea, S.: Towards Self-Healing Compositions of Services. Studies in Computational Intelligence, v. 42, Springer (2007).
3. Baresi, L., Di Nitto, E., Ghezzi, C., Guinea, S.: A Framework for the Deployment of Adaptable Web Service Compositions. Service Oriented Computing and Applications Journal (to appear).
4. Colombo, M., Di Nitto, E., Mauri, M.: SCENE: A Service Composition Execution Environment Supporting Dynamic Changes Disciplined through Rules, 4th International Conference on Service Computing, ICSOC, USA, December (2006).
5. Hielscher, J., Kazhamiakin, R., Metzger, A., Pistore, M.: A Framework for Proactive Self-Adaptation of Service-based Applications Based on Online Testing, 1st European Conf. Towards a Service-Based Internet, LNCS vol. 5377, ServiceWave, Spain, (2008).
6. Li, L., Horrock, I.: A Software Framework for Matchmaking based on Semantic Web Technology, WWW Conference, Workshop on E-Services and the Semantic Web (2003).
7. Zisman, A., Spanoudakis, and Dooley, J.: A Query Language for Service Discovery, 4th Int. Conference on Software and Data Technologies, ICSOFT, Bulgaria, July (2009)