

JRegistry: An extensible UDDI Registry

Hafedh Mili

Radhouane Ben tamrout

Abdel Obaid



The Problem

- UDDI:
 - A standard for the description of enterprise web services and for querying a registry of such services
 - A number of implementations of the standard exist: Apache, IBM, SAP, etc.
- The problem:
 - Standard => lowest common denominator
 - Limited querying capabilities

Strategies for extending UDDI Registries

- Strategy 1:
 - Same API
 - extending UDDI entry and query formats
 - modifying existing implementations
- Pros:
 - Transparency for existing clients
- Cons:
 - Limited by API bandwidth
 - New implementations of UDDI
 - Migrating existing registries

Strategies for extending UDDI Registries

- Strategy 2:
 - Extend the UDDI API with the new functionalities
 - extending UDDI entry and query formats
 - modifying existing implementations
- Pros:
 - Wider range of extensions
- Cons:
 - New implementations of UDDI
 - Migrating existing registries

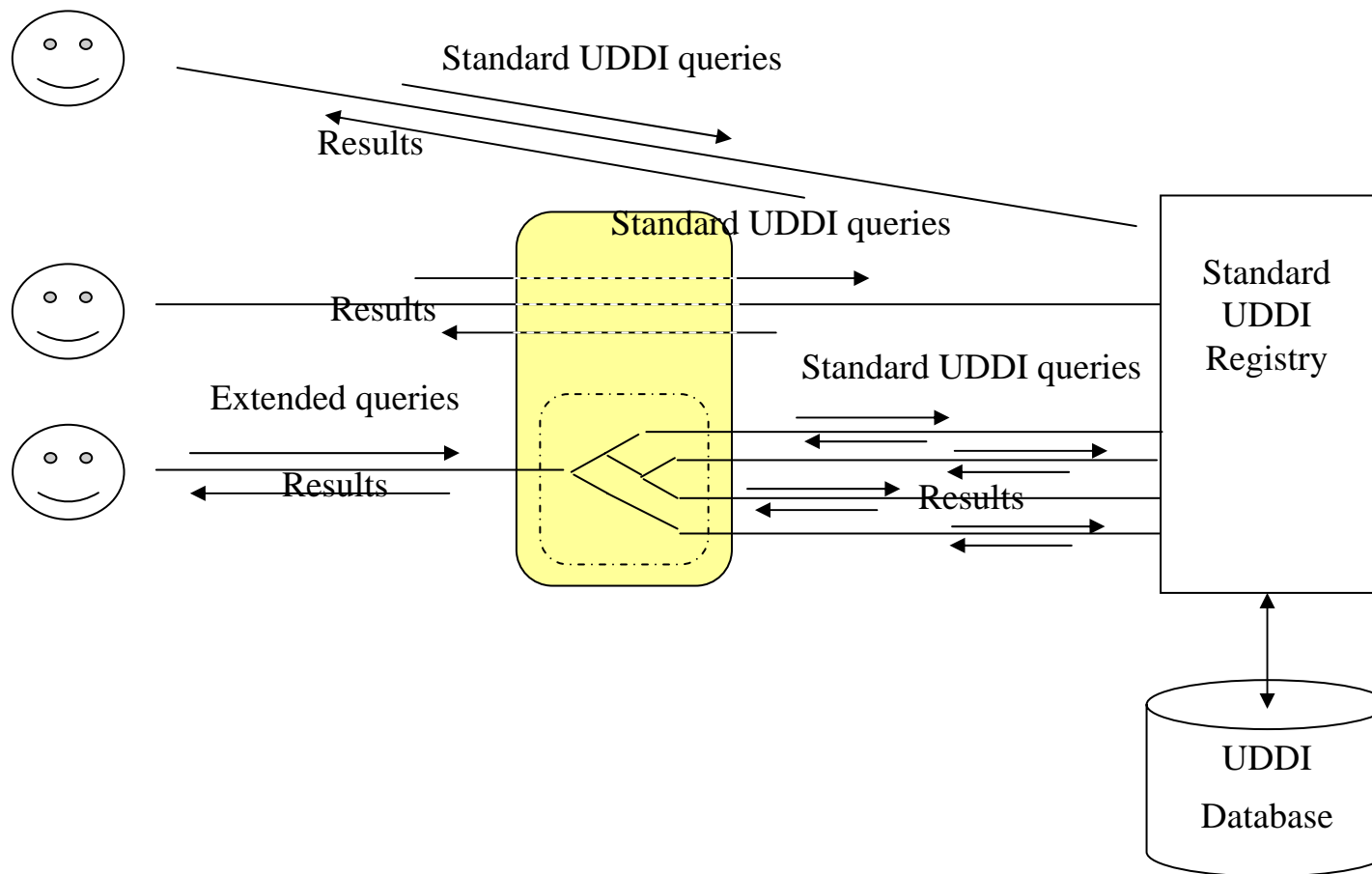
Strategies for extending UDDI Registries

- Strategy 3:
 - Use a middle-tier that implements the extensions
 - Standard UDDI: talk to the source
 - Extension: talk to the middle tier
- Pros:
 - Least intrusive
- Cons:
 - Different middle-tiers for different extensions

Our approach

- Use the third strategy
- Support ad-hoc extensions
- Support dynamic extensions

Our approach



Low-cost implementation

- Modify a public domain implementation
 - We chose Apache's JUDDI
- Do not re-implement standard functionality
 - We modified a JUDDI client proxy implementation, which forwards to the JUDDI server, anyway

Supporting run-time extensions

- Define new query formats *declaratively*
- Use code generation, AOSD, and dynamic linking to integrate those functionalities into the running system
 - JUDDI's *framework* for handling queries made this *relatively* simple

JUDDI's framework for handling queries (1)

- Each query is characterized by a <request, response> pair
- The registry is itself implemented as a web service:
 - Requests and responses are exchanged as SOAP messages
- For each of “request” and “response”, we need:
 - A handler to take care of the marshalling / unmarshalling of requests
 - We need a function that implements the actual query

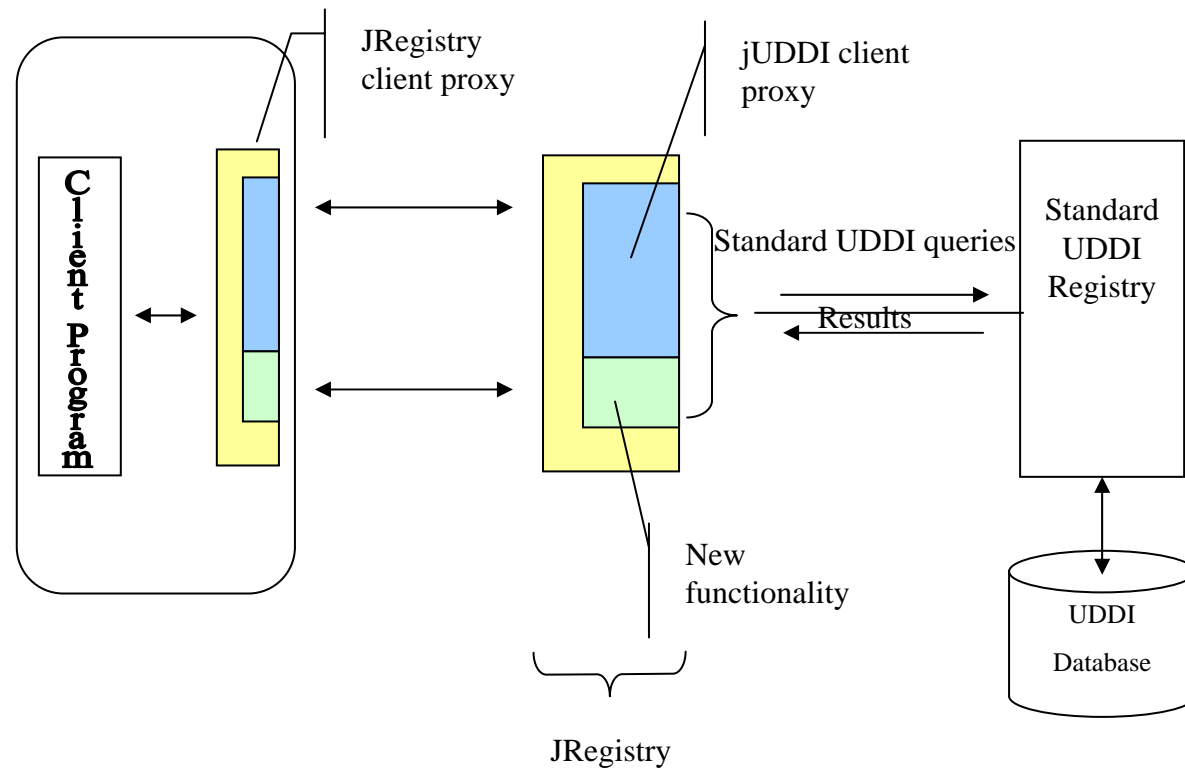
JUDDI's framework for handling queries (2)

- Both the handlers and the function are *registered* within a dispatch function/object that, upon recognizing the *type* of query, will call upon the appropriate handlers and execute the appropriate function
- The dispatch table is initialized statically
 - In the singleton constructor ...

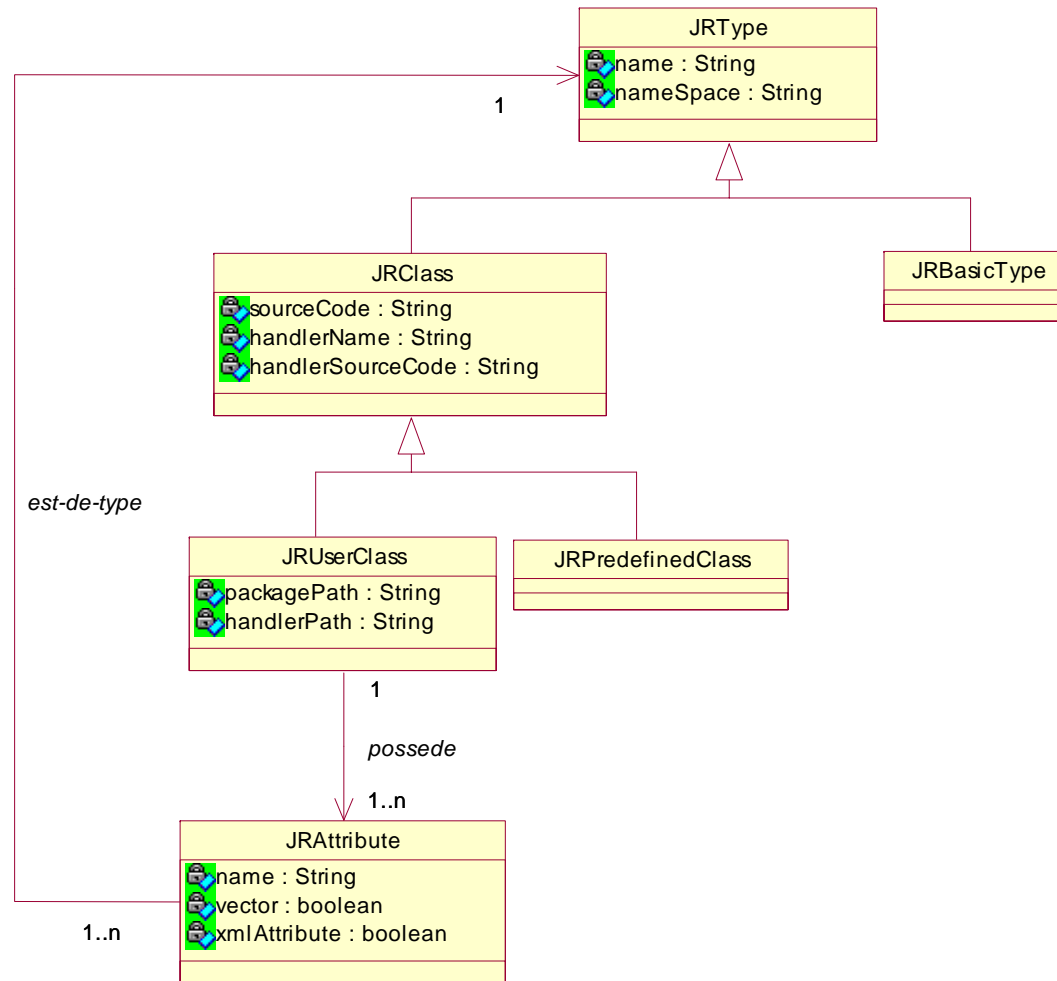
Our approach

- Use XSD to specify request/response structures.
Based on the XSD:
 - Generate the corresponding Java classes, by exploring, recursively, the type structure
 - Generate the corresponding handler (marshalling and unmarshalling)
 - Generate a stub for the function
- Use BPEL to describe the query processing that needs to take place at the middle-tier

Overall architecture



(Data) Types of requests and responses



Conclusion

- The rest is engineering