A “Quick and Dirty” Meet-in-the-middle Approach for migrating to SOA

Filippo Ricca* and Alessandro Marchetto**

* Unità CINI at DISI, University of Genova, Italy
** Fondazione Bruno Kessler-IRST, Trento, Italy
The context

• Companies are in process of moving their applications toward SOA

• When executed, the migration (should) brings a lot of well-known advantages:
  – Reusability
  – Business agility
  – ...

• Migrating legacy applications toward service-oriented systems is a challenge for IT Professionals
  – It is a difficult and error prone task

• Several approaches proposed in literature but:
  – too time-consuming
  – too complex to apply
  – .... several issues are still open!

Existing approaches

• **Four steps** (SOA implemented with Web services):
  1. Discovering potential Web services from the existing application
     – potential Web services = fragments of code realizing the desired functionality
  2. Evaluating the potential Web services
     – e.g., maintainability, testability, reusability
  3. Extracting the Web services code
  4. Wrapping and deploying them to be reused as Web services.

• **Main difficulties:**
  – identifying fragments of code associated to the desired functionality
  – eliminating potential dependencies between the code actually implementing the target functionality and its contour (code isolation).
  • tangling and scattering

“pictures taken by Prof. Di Lucca”
“Quick and dirty” approach

- Work inspired by a real industrial case study
  - **Lesson learnt:**
    - approaches based on code isolation = too complex and time-consuming
      * non realistic waiting years for having a running implementation!

- Reinterpretation/extension of [Inaganti2007] (preliminary work)
  - a lot of details are missing
  - several issues are open

- “Quick and dirty”
  - “not optimal” solution but working in short time

- “Quick and dirty” useful for several purposes and in different contexts:
  - to verify that the migration toward SOA is feasible
  - to implement a prototype for evaluating the architecture and executing some trials on it (e.g., performance analysis)
  - to participate to a public international competition which requires a SOA-based solution
  - as a first step toward a SOA migration project
The running example: ATM application

- **It is a distributed system:**
  - Several ATMs connected with the Bank (unique)
  - Exchange of message
- **Automated Teller Machine:**
  - It provides the following functionalities:
    - Authentication
    - Deposit, withdraw, balance
    - Move Funds
    - Phone cards acquisition with external services
    - [Logging]
      * each transaction is logged
- **ATM software:**
  - GUI (the screen)
  - business logic (portion)
  - communication
- **Bank:**
  - Stores the transactions
Migration task

- **Task:** making the ATM software more light moving the business logic in a set of reusable services

- In the end, the ATM software should only:
  1. implement the “orchestrator” that invokes the Web services
     - e.g., authentication, deposit, withdraw
  2. and provide the GUI
1. Design the core business process and express it in BPMN (BPD)

2. Refine the sub-processes of the main BPD that could contain services invocations (if any)
   - The result is a BPD for each expanded sub-process

3. Identify which tasks from the BPDs could represent service invocations (service tasks)
   - In our approach each service task correspond to a single operation of a service
     - e.g., authenticate(accountNum, pin):boolean
BPD of ATM

"+" = activity further decomposable
"simple box" = atomic activity
Identify sub-processes to refine and service invocations

Refining (step 2)

Service invocation (step 3)
Withdraw cash refining

- Show withdraw menu
- Receive the withdraw amount or cancel transaction
- Insufficient cash in the ATM or insufficient funds in the account
- Service invocation
- Execute transaction [draw]
- Dispense cash
- Customer
4. Identify the **points of functionality** in the legacy that may be useful to implement the needed services
   - i.e., finding among the several programs the one that contains the functionality of interest

5. Expose, by means of wrappers/adapters, the identified points of functionality as services.
   - **built services** = adapters that are waiting for a request
     - they use the points of functionality
   - Cloning an entire program is possible
     - program = executable file “part of” the target legacy code

6. **Map the service tasks** within the BPDs with the built services.
“Quick and dirty” Macro steps (3)

7. **Complete the mapping** between service tasks and services. At the end of this step each service task in the BPDs will be associated to exactly one service

   • Admitted operations:
     1. Realizing a new service “linking” two or more points of functionality
        * e.g., the **Move funds service** that uses the points of functionality Draw and Deposit
     2. Building from scratch a new service. This happens when no points of functionality are present in the legacy code to realize what we need
        * e.g., the service **Logger** has been built from scratch because this functionality was not present in the legacy code
     3. Adapting the BPDs splitting the target service task into two or more service tasks with the same meaning
        * one-to-n mapping is not allowed

8. Generate the executable BPEL that invokes the implemented Web services starting from the BPDs.
Program 1

Program 2

DB

Authenticate

Draw

Move funds

Deposit

Balance

Logger

DB

Legacy

Realizing a new service “linking” two or more points of functionality

Building from scratch a new service
Open issues, problems and considerations

• **Issues to be considered:**
  – handling concurrent requests
  – identification mechanism
    • to preserve the interactive session between “orchestrator” and services
  – stateful or stateless services?
  – where is maintained the state of the application?

• **Difficulties:**
  – identifying points of functionality
  – code isolation (avoided in our approach)
    • e.g., separating input/output from business logic

• **Considerations:**
  – our approach privileges applications with fine-grained and stateless services
“Program cloning”

• Three choices:
  – Wrapping “all”
  – Code isolation and wrapping
  – Program cloning and wrapping
    • Using an entire program only for a subset of functionalities

• Program cloning
  – Pro:
    • first step toward a migration producing independent services
    • no code isolation
  – Against:
    • overall dead code of the application augmented
Debate questions

• **Is it possible to use approaches based on code isolation in real SOA migration projects?**
  – *Tangling and scattering are always present!*


**Code isolation approaches:**

1. Discovering potential Web services from the existing application
2. Evaluating the potential Web services
3. Extracting the Web services code
4. Wrapping and deploying them to be reused as Web services.
Feedback questions

• Is the idea of cloning more times entire programs composing the legacy applicable? make sense?
1. useful as first step toward a real SOA migration project?

ATM code

- authentication
- draw
- deposit

ATM code (cloned)

balance
Thanks for the attention!

Questions?
Before - After

A portion of the business logic is moved in the services (e.g., check of the pin)
References


