

A Flexible Wrapper For The Migration Of Interactive Legacy Systems to Web Services

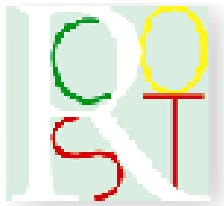


P. Tramontana

A.R. Fasolino

G. Frattolillo

Dipartimento di Informatica e Sistemistica
University of Naples Federico II, Italy




G. Canfora

RCOST – Research Centre on Software Technology
University of Sannio, Benevento, Italy

Introduction

Motivation

- Service-Oriented Architecture (SOA)
 - expresses a perspective of software architecture that defines the use of services to support the requirements of software users.[...] Most definitions of SOA identify the use of **Web services** in its implementation (Wikipedia)
- Possible approaches for obtaining Web Services
 - Developing from scratch
 - *Reusing existing software* ← 
- Form-based Legacy Systems pervade fundamental productive activities:
 - Public administration, bank, tourism, CRM, ...
- There is a great request for migration of interactive legacy system functionalities toward Web Services and SOA

Comparing Interaction paradigms

Form based Systems

User types input data and issues commands by interacting with the user interface.

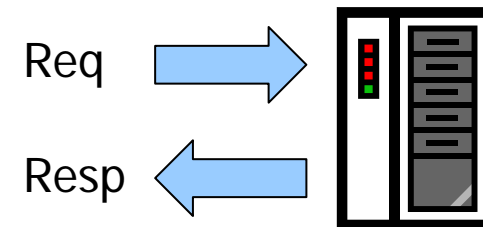
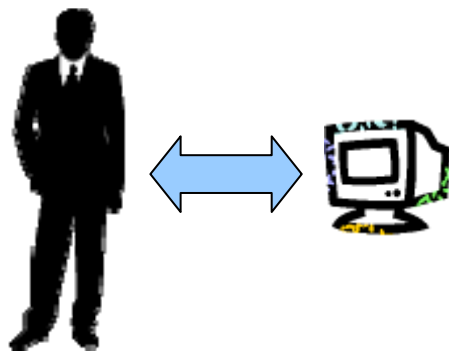
System answers by producing a response screen, containing output values and new input fields and command buttons.



Web Services

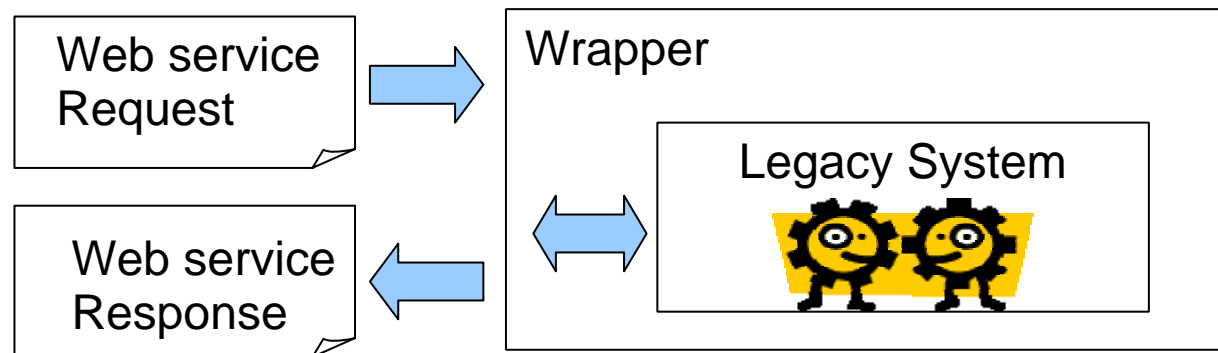
A Client party invokes a service implemented by a provider party, using a request message.

The provider processes the request and sends a response message with the obtained results.



The Wrapper Tool

- Uses a black-box approach
- The goal of the wrapper is to make legacy systems accessible as Web Service
- Legacy system use-case constitutes the base of a Web Service
- The Wrapper drives the legacy system during the execution of each possible interaction scenario associated with the use case to migrate, by providing it with the needed flow of data and commands.



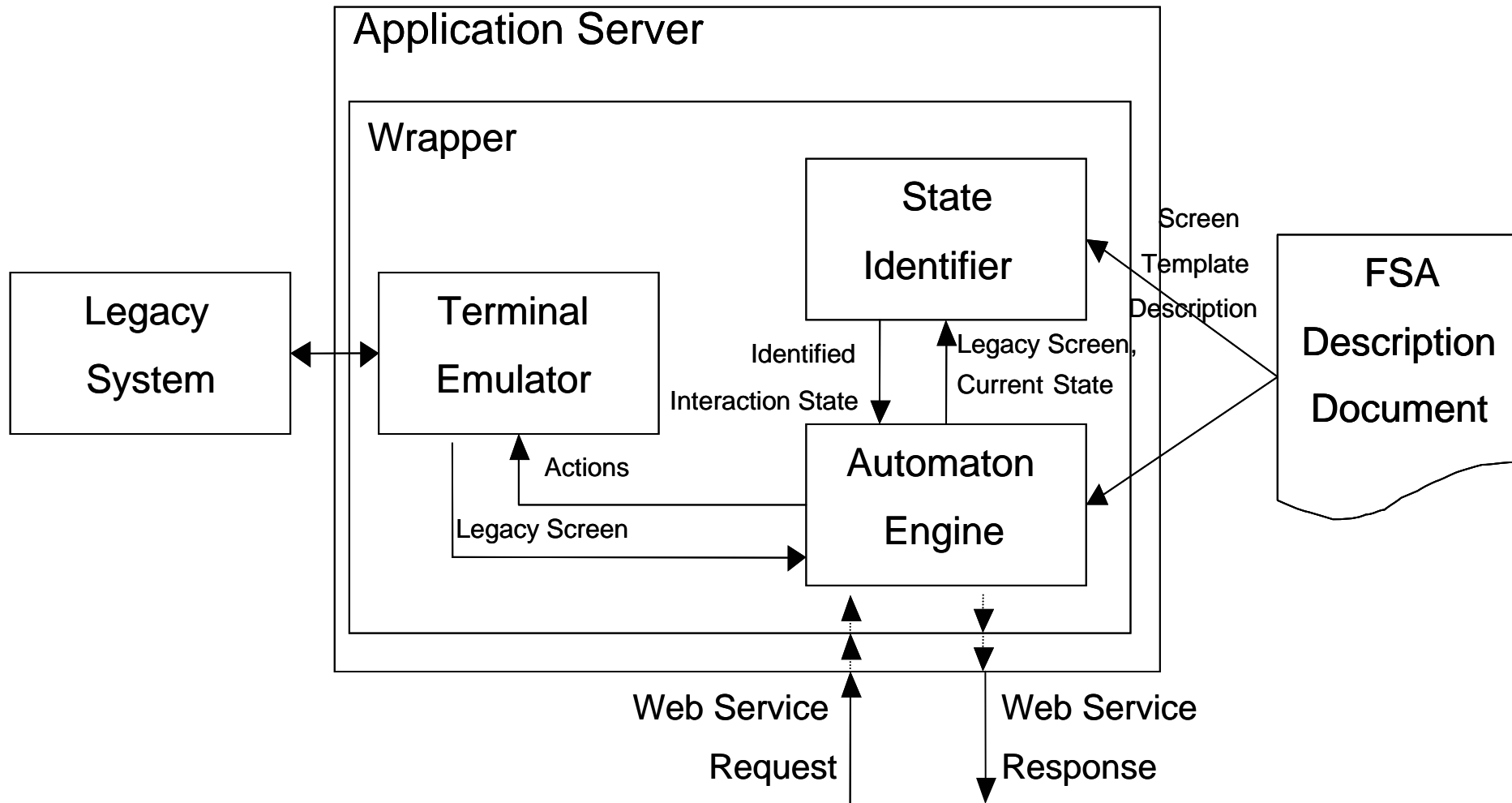


The Architecture

Features of the tool

- Platform independence
 - Developed using J2EE platform
- Extensible architecture
 - High modularity
- Supports WSDL and SOAP specification
 - For good interoperability
- Uses a black-box approach
 - No need for the source code of legacy system
- Supports a use-case driven approach
 - Iterative and incremental

Architectural overview

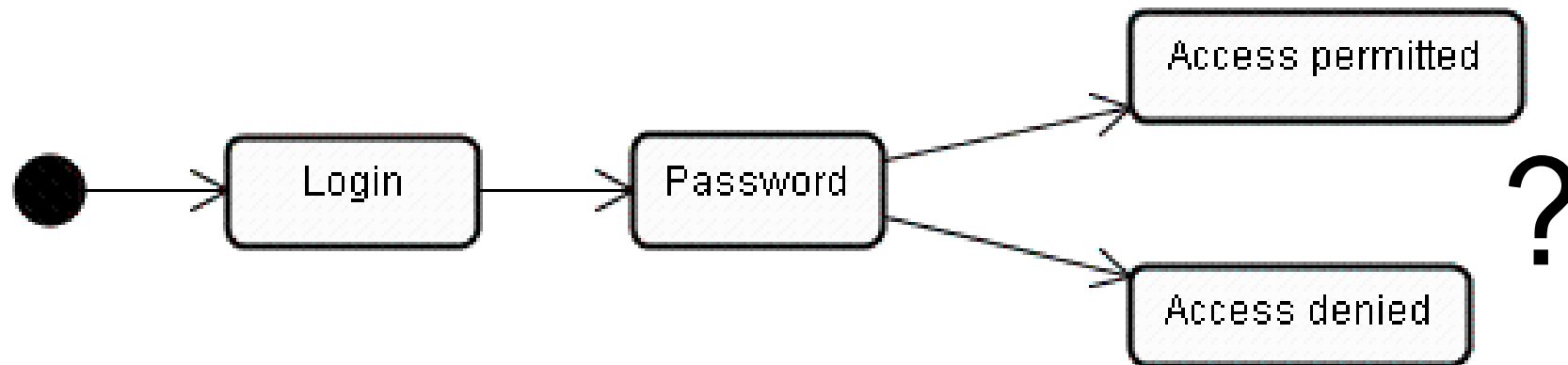


The “Terminal Emulator”

- Responsibilities
 - Manages the communication between the Wrapper and the Legacy
 - Is responsible for creating a virtual terminal and managing its virtual keyboard and display
 - Designed to support both stream-oriented and block-oriented terminals
- Implementation
 - Modular implementation. Developers can add other terminal protocol implementations
 - Vt100 module is based on Java Terminal Application
 - (<http://javassh.org/space/Plugin+Terminal>)

The “Automaton Engine”

- Responsibilities:
 - Coordinates the execution of NFA, driving the Terminal Emulator, the State Identifier and the FSA Repository
 - NFA is required because the next state may depend on the internal logic or on the internal state of the legacy system



- Implementation:
 - Custom

The “State Identifier”

- Responsibilities
 - Disambiguates the next state while Automaton Engine is executing the deterministic finite state automata (NFA)
 - The State Identifier must be able to identify the next state on the basis of the returned screen, trying to match the current screen of the legacy system with the Screen Templates associated with potentially reachable Interaction States
 - Localises Labels, Input Fields and Output Fields (getting their values) from the screen
- Implementation
 - Custom

The "FSA Repository"

- Responsibilities:
 - Stores the XML files containing the specification of the Finite State Automata associated with each wrapped use-case
 - Includes the descriptions of the Interaction States, Transitions, Screen Templates, and Actions to be provided to the Automaton engine
 - Helps the Wrapper to be independent of specific use-case
- Implementation
 - Xml and Xml-Schema languages
 - Castor Xml (binding Java-Xml)
 - File systems (persistence)

Demo

Software used

- Microsoft Windows Xp Professional Sp2
 - The OS installed on the laptop
- Cygwin
 - A Unix emulation layer over Win32 API
- Pine 4.64 and Inetd - telnet server
 - **The legacy system**
- Java 2 SDK 1.4.2
 - Java VM and Compiler
- Apache Tomcat Servlet Container
 - A J2EE Web Container
- Apache Jakarta Axis Web Application
 - The Web Service infrastructure

Identification Phase

- Use case of the legacy system is selected for the migration
 - Read the i -th message from a given folder
- The legacy system is exercised by user sessions covering the possible scenarios of that use case
 - Sample telnet session...
- Screens returned by the legacy system and actions performed by user are recorded
 - Sample screens and actions will be presented later...

Design Phase

- Required inputs and desired outputs are identified
 - Input : user, password, folder, message number
 - Output: email message (formatted in Xml)
- A Finite State Automaton is designed and stored in a XML document
 - FSA has to be able to replicate the behaviour of the user in the execution of any scenario of the selected use case
 - We'll see the resulting FSA later...

Deploy Phase

- The WSDL document describing the Web Service's interface is written and the Web Service is deployed
 - Starting up Tomcat...
 - Our WSDL... inside Axis...

Validation Phase

- A testing strategy is followed, in order to validate the correctness of the realised Web Service
 - Cover all possible scenarios associated with the migrated use-case
 - Folder not found
 - Folder with no messages
 - Message not found
 - ...
 - What could go wrong?
 - Forgetting a scenario
 - The “Folder is locked” scenario
 - The “Move messages in a dedicated folder” (asked only every month) scenario

Running the Demo

- A new message with Pine
 - Writing...
- Reading that message
 - Launching the Web service invoker
 - It runs for just a few seconds!
- A look inside generated log file
 - Animated automaton...

Q & A Session

Thank you