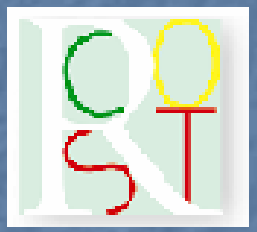
Migrating Interactive Legacy Systems To Web Services



Porfirio Tramontana

Anna Rita Fasolino

Giovanni Frattolillo

Dipartimento di Informatica e Sistemistica

University of Naples Federico II, Italy

Gerardo Canfora

RCOST – Research Centre on Software Technology

University of Sannio, Benevento, Italy

Motivation

- Growing interest is currently being devoted to Service Oriented Architectures
  - IDC estimates that worldwide spending on Web services-based software projects will reach $11 billion by 2008, compared to $1.1 billion in 2003 (*Neal Leavitt, IEEE Computer, November 2004*)
- Possible approaches for obtaining Web Services
  - Developing them from scratch
  - Reusing existing software
- Legacy Systems pervade fundamental productive activities:
  - Public administration, bank, tourism, customer relationships, ...
- A relevant issue: migrating legacy system functionalities toward Web Services

Three basic questions ...

1. What to expose as a Web Service?
2. When the migration is convenient?
   - G. Lewis, E. Morris and D. Smith have approached this question in the yesterday tutorial and in the previous talk ...
   - S. Tilley, J. Gerdes, T. Hamilton, S. Huang, H. Muller, K. Wong also outline the challenges inherent in migrating to Web services
3. Which approaches for the migration?
   - Sneed and Sneed present a tool supported process to make accessible selected sections of legacy code as Web Services;
   - E. Stroulia, M. El-Ramly, P. Sorenson propose methods based on the analysis of screen features and on the tracing of user interactions to reverse engineering interfaces of an interactive legacy system in order to support the migration

A specific problem:
the migration of interactive legacy system functionalities toward Web Services

Comparing Interaction paradigms...

Form based Interactive Systems

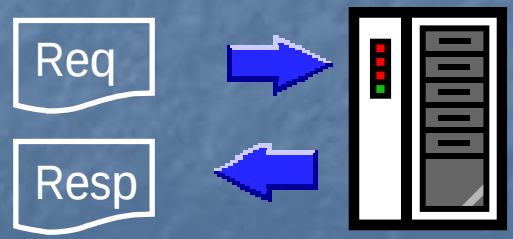
Users query the system by inputting data and sending commands, by interacting with the user interface.

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.

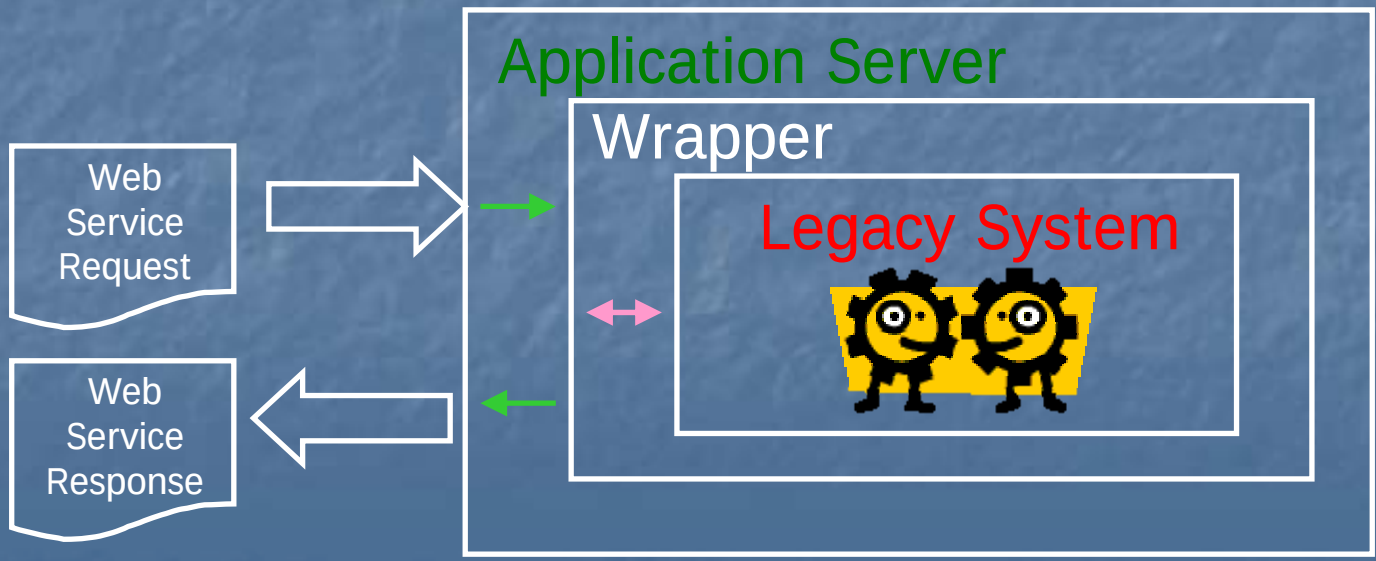


Which approaches for the migration?

Wrapping

The Wrapper

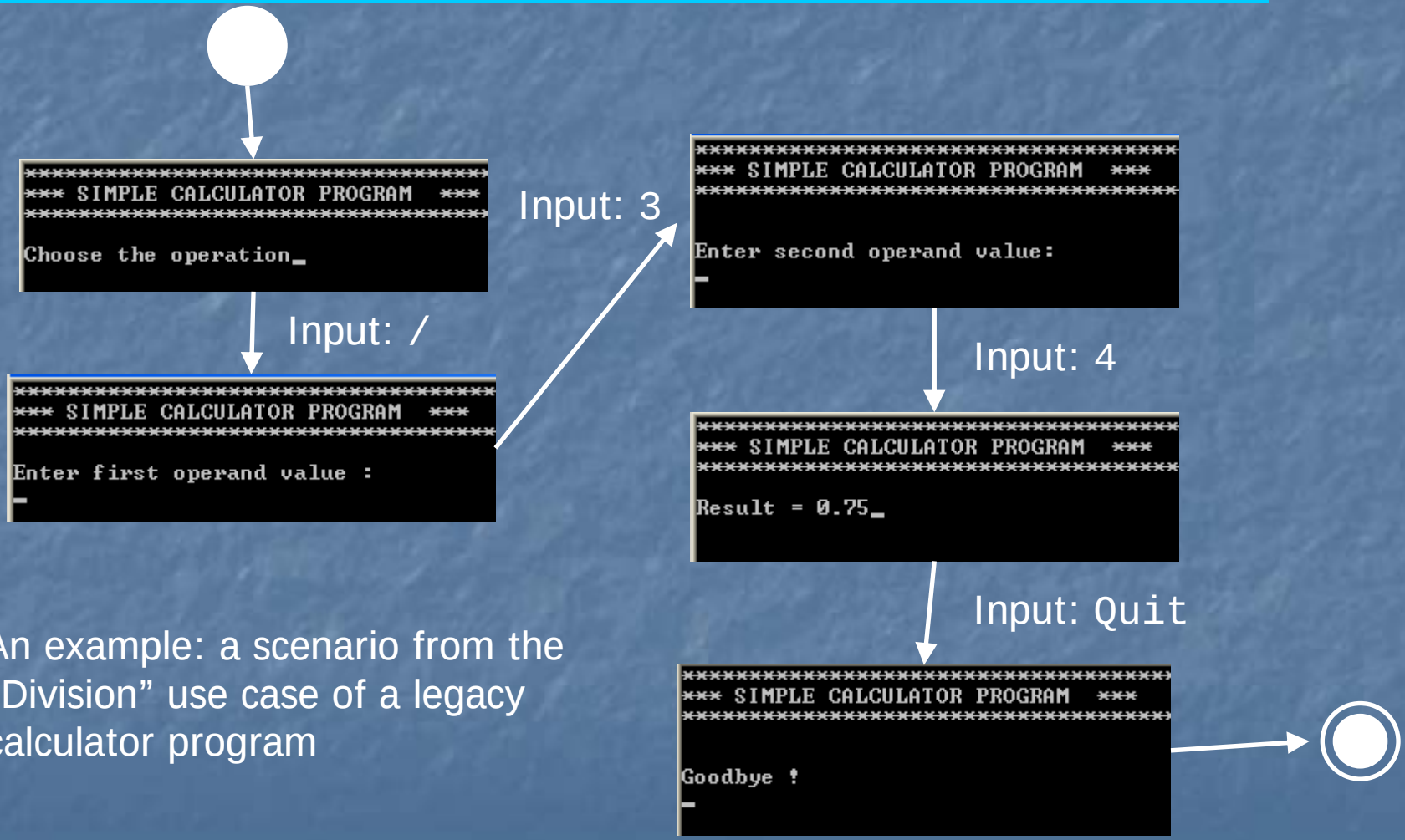
- The goal of the wrapper is to drive the legacy system during the execution of each possible interaction scenario associated with the use case to migrate, by providing it with the needed flow of data and commands.
- The wrapped legacy system use case is accessible as a Web Service



A key requirement of the Wrapper

- The wrapper must be reusable for migrating different use cases, so...
- The wrapper behavior requested for each use case will not be embedded in the wrapper...
- But it will be separately specified for each use case
- A key question: obtaining for each use case a complete model of the interaction between the legacy system and the user
- A Reverse engineering problem!

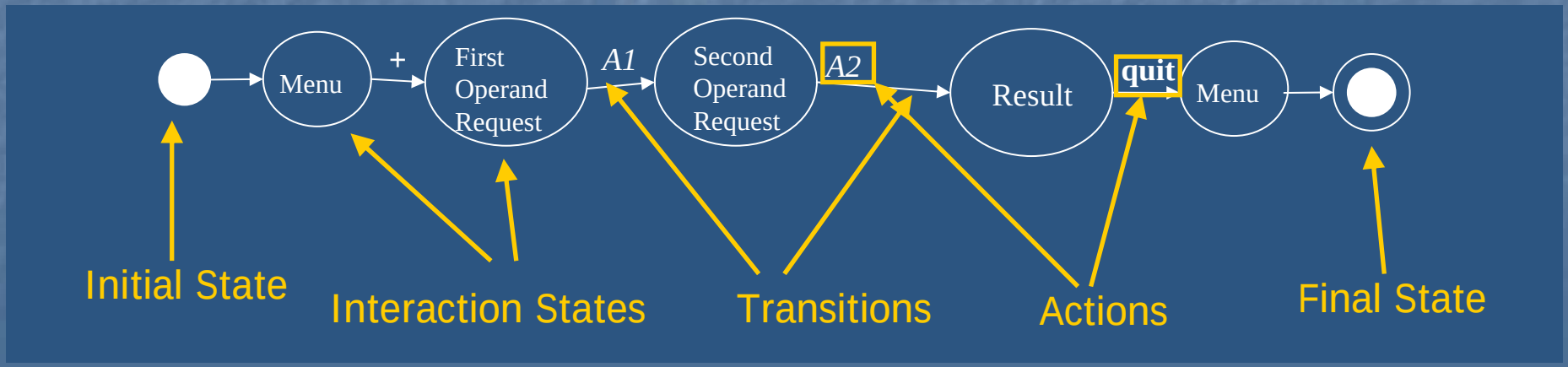
Modelling Interactions between User and Legacy System



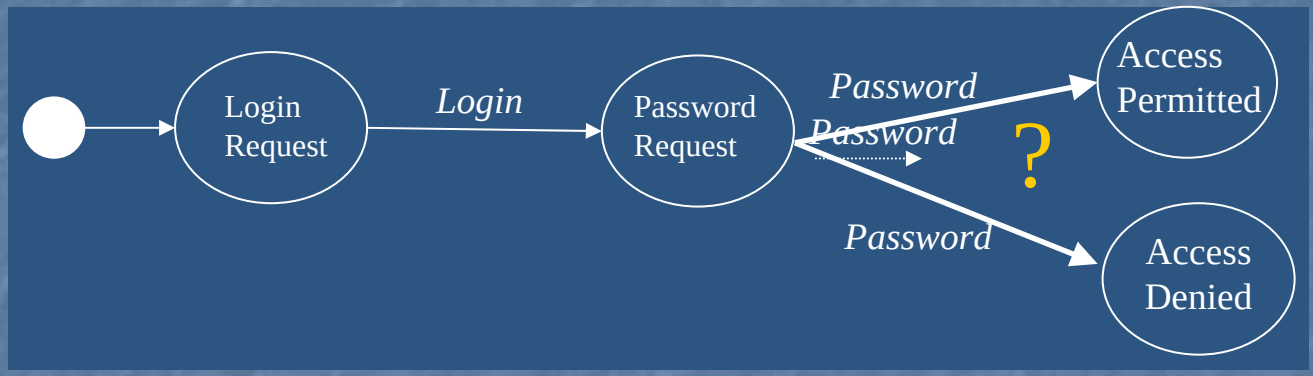
An example: a scenario from the "Division" use case of a legacy calculator program

The Model of the Interaction

- A Finite State Automaton FSA= (S, T, A, Sin, Sfin) where:
  - S is the set of Interaction States,
  - A is the set of Actions performed by the user when an Interaction State occurs,
  - T is the set of Transitions between states,
  - Sin and Sfin are the Initial and Final states of the interaction.



A problem

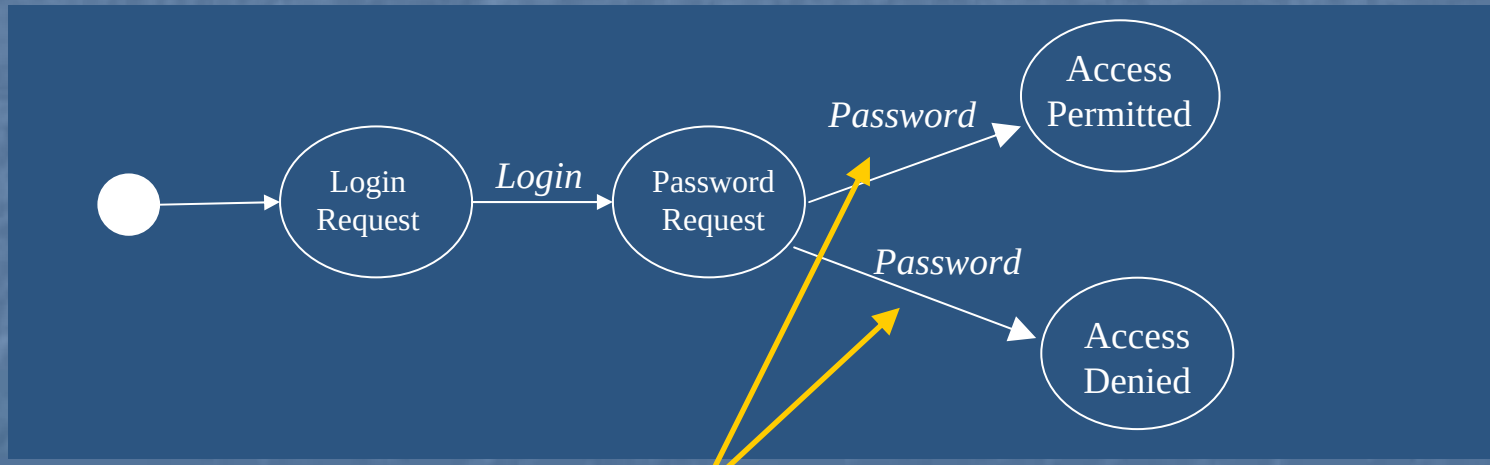


What the next State?

- The next state depends on the internal logic or on the internal state of the legacy system.
- A solution: Non Deterministic Finite State Automata
  - a Non Deterministic Finite State Automaton (NFA) is a finite state machine where for each pair of state and input symbol there may be several possible next states

Another Wrapper Requirement

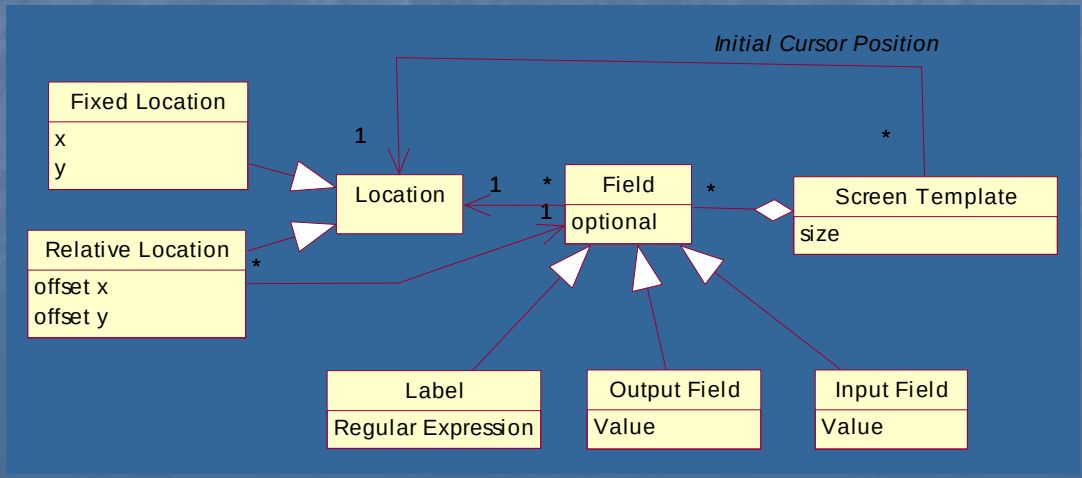
- The wrapper must know the list of the possible Next States of a given State
  - Possible successors of `Password Request` State are `Access Permitted` and `Access Denied` states
- The wrapper must be able to identify the current state on the basis of the returned screen
  - Wrapper must discriminate among `Access Permitted` screen and `Access Denied` screen



Same Action, but different Transitions!

Screen Templates

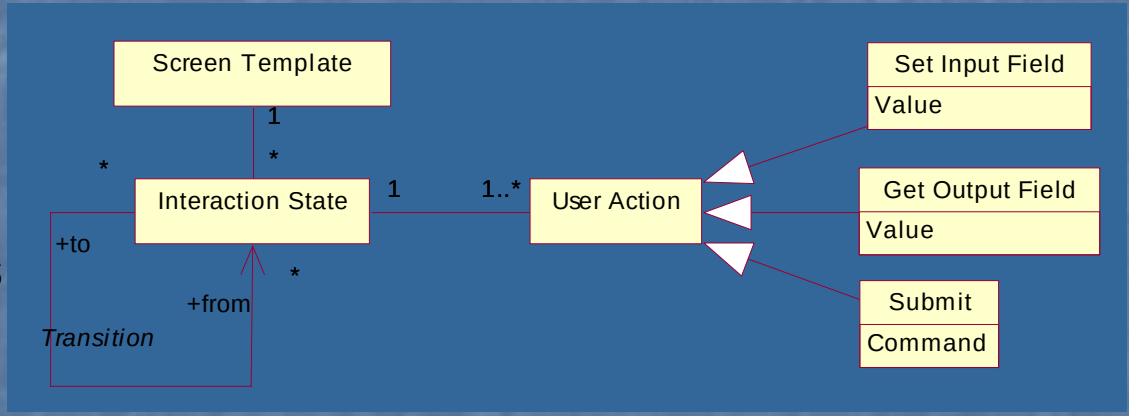
- A description of Legacy Screen is needed for the identification: Screen Templates
  - A Screen Template is a collection of Fields:
    - Labels;
    - Input Fields;
    - Output Fields;
  - Each field has a Location on the Screen. Location may be defined as a:
    - Fixed Location, i.e. coordinates of the field;
    - Relative Location, i.e. distance from another field.



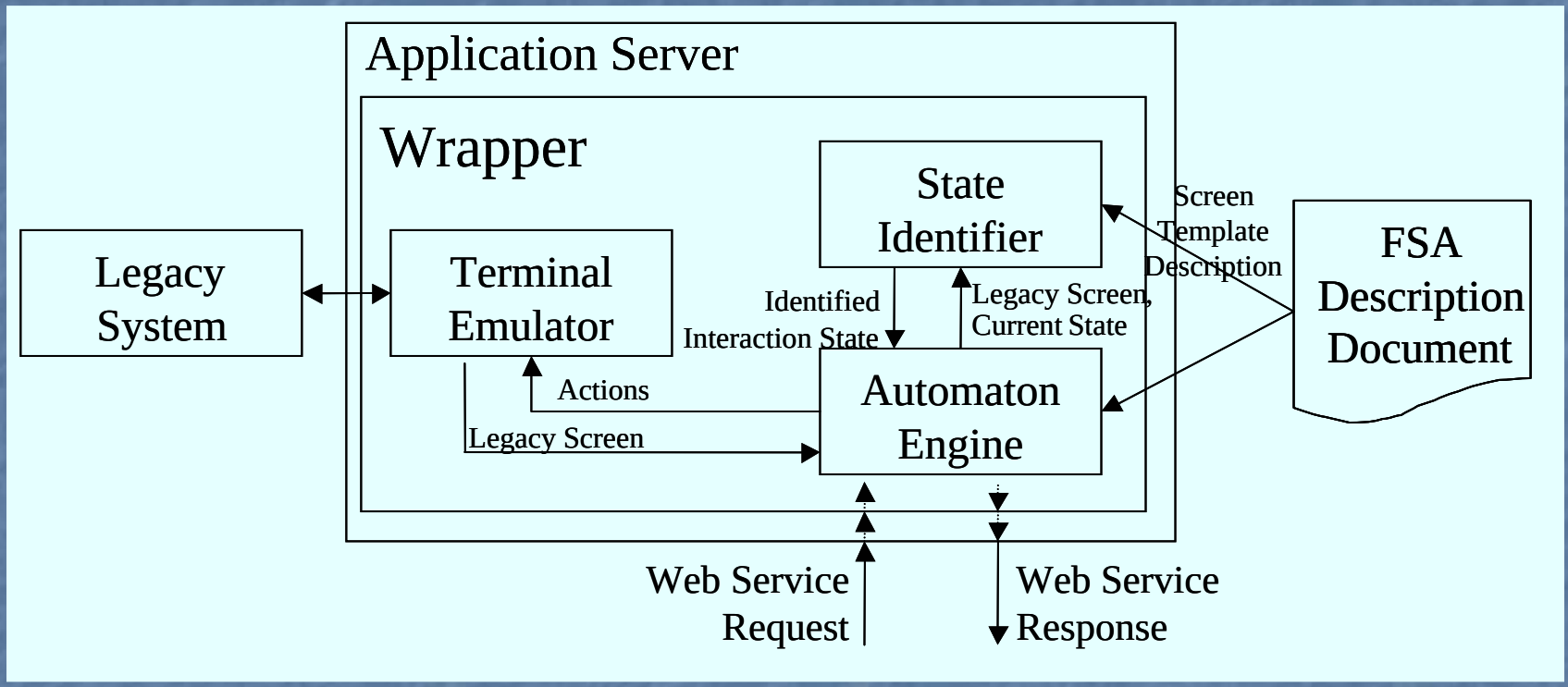
Characterising Interaction States

- An Interaction State is characterised by a Screen Template and a set of actions to perform on its fields, causing transitions to other Interaction States

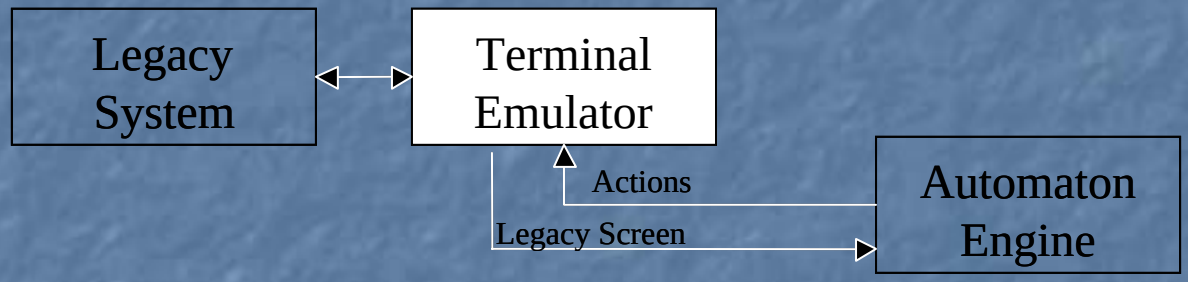
- User Actions may be:
  - Set Input Field Actions
  - Get Output Field Actions
  - Submit Command Actions



Wrapper Architecture



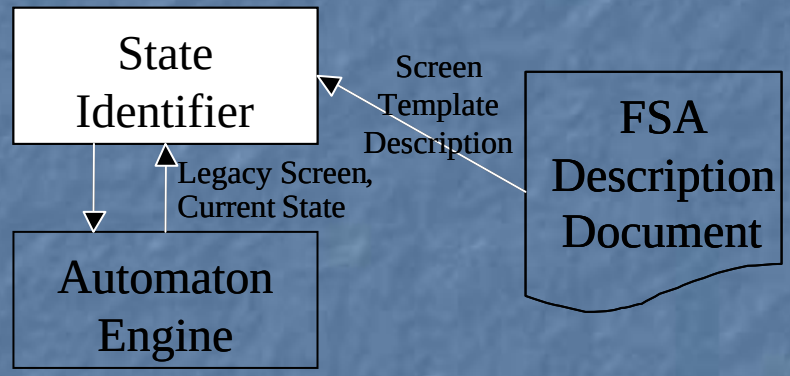
Terminal Emulator



- The Terminal Emulator component is responsible for the dialogue between the Wrapper and the Legacy System terminal
  - Different implementations of the Terminal Emulator are needed for different Legacy System Terminals
    - Stream Oriented terminals;
    - Block oriented terminals;
    - Web Applications.

State Identifier

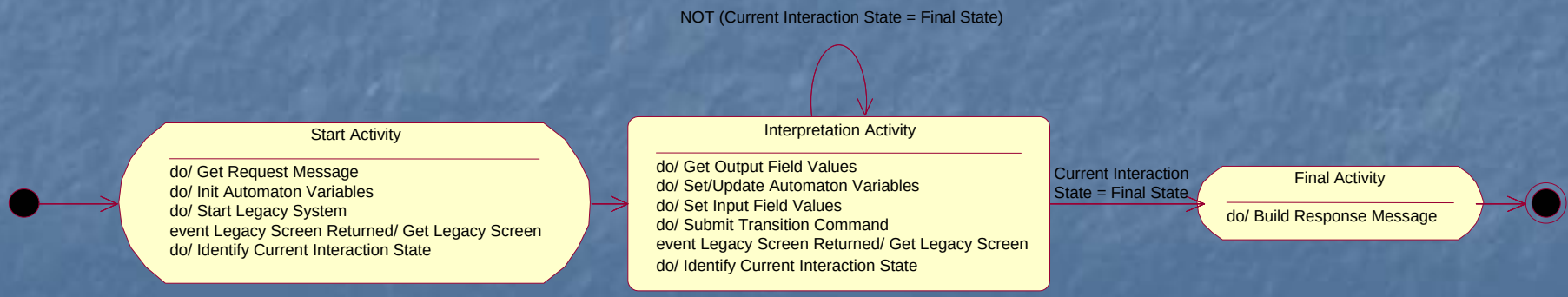
- The State Identifier component is responsible for the identification of the Interaction State reached by the Legacy System



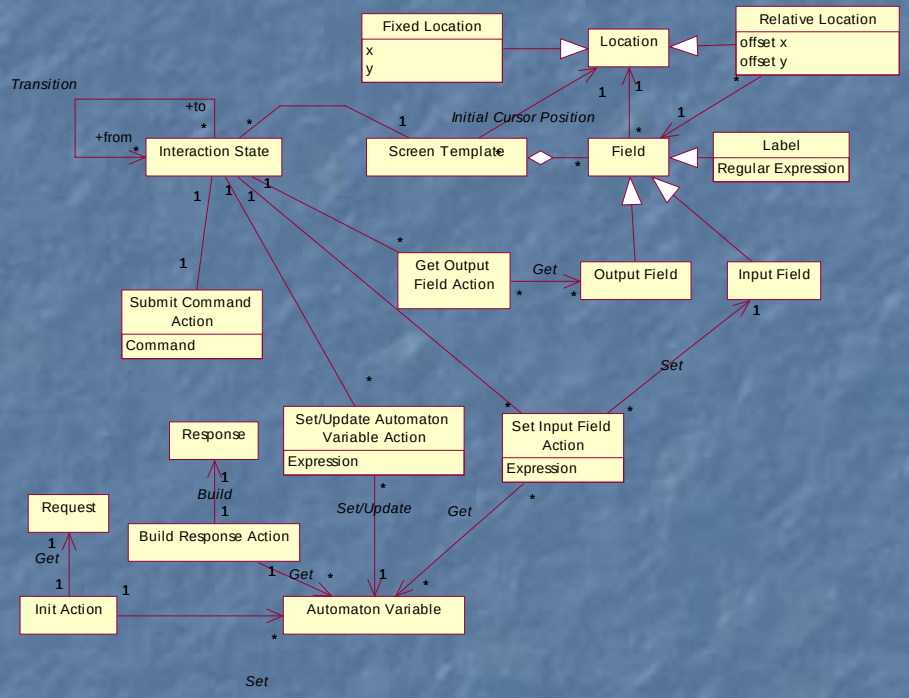
  - It has to match the current screen of the legacy system with the Screen Templates associated with potentially reachable Interaction States
  - The Screen Templates descriptions are part of the Automaton Description Document
- Moreover, the State Identifier
  - localises Labels
  - localises Input Fields
  - Localises Output Fields and read their values

Automaton Engine

- The Automaton Engine is responsible for interpreting the FSA associated with a given service offered by the legacy system. It:
  - Sends commands to and receives screens from the Terminal Emulator
  - Queries the State Identifier about the identification of the Current Interaction State
  - Interprets the request message received from the application server
  - Builds the response message and sends it to the application server
  - Manages Automaton Variables (i.e. temporary variables needed to save intermediate results of the execution of the Automaton)



Finite State Automaton Description Document



Automaton Description Document UML model

```
<automa>
 <automa-states>

....

 <state id="go_to_header"
type="automa"
screen="PineGoToHeaderScreen">
        <description>State
        </description>
        <layout>
          <location x="1" y="2"/>
          <size width="8" height="2"/>
        </layout>
        <actions>
          <set-fields-action>
            <field ref="prompt">
               <data ref="/root/header"/>
            </field>
          </set-fields-action>
        </actions>
        <next-states>
          <next-state ref="bad_header">
        </next-state>
          <next-state ref="header">
       </next-state>
       </next-states>
      </state>
```

```
<screen id="GoToHeader">
 <size width="80" height="25"/>
 <simple-field id="GoToHeaderId"
optional="false" input="false">
    <fixed-location>
       <point x="0" y="22"/>
    </fixed-location>
    <content pattern="Message number to
jump to" length="25"/>
 </simple-field>
 <simple-field id="prompt" optional="true"
input="true">
      <fixed-location>
         <point x="25" y="22"/>
      </fixed-location>
     <content pattern="" length="5"/>
    <focus order="1">
      <advance-key id="ENTER"/>
    </focus>
 </simple-field>
 <caret-location>
   <fixed-location>
      <point x="28" y="22"/>
   </fixed-location>
 </caret-location>
</screen>

...

</automa-states>
</automa>
```

An excerpt of an Automaton Description Document

A Case study

- A migration case study has been carried out according to a process including the following phases:
  - Identification, i.e. reverse engineering of the interaction model;
  - Design, i.e. defining the FSA describing the Wrapper behaviour;
  - Implementation, i.e. realisation of the XML FSA Description Document;
  - Web service deploy, i.e. deployment of the wrapper in the context of an application server;
  - Validation, i.e. testing of the scenarios of the migrated use case

A Case Study

- Legacy system: Pine (ver. 4.64)
  - client mail software, that allows a user to read, compose and manage e-mail messages from an existing message box.
- Pine is a form based legacy system based on *stream oriented* terminals.
  - Usually, Pine is accessible via the Telnet protocol.
- We submitted to the migration process the *Get Message* use case that allows the owner of a mailbox to get the text of a specific e-mail message contained in a specific mailbox folder.

Use case:	Get Message
Preconditions	None
Input	Login, Password, Folder, Message Number
Output	Date, From, To, cc, Subject, Body, Exception
Postconditions	None

The Automaton: graphical view

7 different scenarios:

1) One Page Message Read
2) Two Pages Message Read
3) More than two Pages Message Read
4) Bad Message Number
5) Empty Folder
6) Folder Not Found
7) Incorrect Password

The Automaton: a tabular specification

Interaction State ID	Interaction State Description	Actions	Submit Command	Next State
START		**Init** (*Login, Password, Folder, Message Number*)		1
1	Authentication Menu (Ask Login)	**Set Input Field**: *Login*	<Enter>	2
2	Authentication Menu (Ask Password)	**Set Input Field**: *Password*	<Enter>	3,4
3	Authentication Menu (Incorrect Password)	**Set Automaton Variable**: *Exception* = "Incorrect Login and Password"	<Control>C	16
4	Main Menu		g	5
5	Go To Folder	**Set Input Field**: *Folder*	<Enter>	6,7,8
6	Folder Not Found	**Set Automaton Variable**: *Exception* = "Folder not found"	q	17
7	Empty Folder	**Set Automaton Variable**: *Exception* = "No messages in the folder"	q	18
8	Folder Open		j	9
9	Go To Message	**Set Input Field**: *Message Number*	<Enter>	10,12
10	Bad Message Number	**Set Automaton Variable**: *Exception* = "Incorrect Message Number"	<Enter>	11
11	Folder Open (Message not found)		q	19
12	Header		>	13
13	Message First Page	**Get Output Fields**: (*Date, From, To, Cc, subject,, Body*); **Set Automaton Variables**: (Output: (*Date, From, To, Cc, subject, Body*))	<Space>	14,15
14	Message Mid Page	**Get Output Field**: *Body* **Update Automaton Variable**: *Body* = *Body* + Output:*Body*	<Space>	14,15
15	Message End	**Get Output Field**: *Body* **Update Automaton Variable**: *Body* = *Body* + Output:*Body*	q	20
16	Main Menu (User not admitted)		q	21
17	Exit Confirm (Folder Not Found)		y	END
18	Exit Confirm (Empty Folder)		y	END
19	Exit Confirm (No Message)		y	END
20	Exit Confirm (Read Message)		y	END
21	Exit Confirm (User not admitted)		y	END
END		**Build Response**: (*Date, From, To, Cc, Subject, Body, Exception*)		

Testing Strategy

- A Test Suite comprehending 7 Test Cases had been selected in order to cover the 7 linear independent paths individuated on the FSA

TC#	TC Description	Interaction State Sequence
1	One Page Message Read	S-1-2-4-5-8-9-12-13-15-20-E
2	Two Pages Message Read	S-1-2-4-5-6-9-12-13-14-15-20-E
3	More Than Two Pages Message Read	S-1-2-4-5-8-9-12-13-14-14-15-16-21-E
4	Bad Message Number	S-1-2-4-5-8-9-10-11-19-E
5	Empty Folder	S-1-2-4-5-7-18-E
6	Folder Not Found	S-1-2-4-5-6-17-E
7	Incorrect Password	S-1-2-3-16-17-E

- We noticed that all the 7 scenarios of the migrated use case had been covered by the selected Test Suite

Conclusions

- A method and a tool supporting the wrapping of interactive legacy system have been presented
- A preliminary experiment showed the effectiveness of the approach
  - A more detailed description of the potentialities of the tool and of the experiment that have been carried out will be presented in the Tool Demo Session after the lunch!
- Many ideas for future works arises ...

Open problems and future works

- Definition of criteria for identifying migrable use cases
- Exploring the feasibility of the approach for the migration of different categories of legacy systems:
  - Legacy system using block oriented terminals
  - Legacy Web Applications
- Defining and validating reverse engineering approaches for obtaining the FSA specification (in particular Screen Template description)
  - Concept Analysis approaches
  - Feature Selection approaches
- Exploring the scalability of the approach for more complex functionalities to migrate
  - Identification of elementary functionalities to be migrated
  - Orchestration of migrated functionalities obtaining complex services


Time is over ... Are there any questions?