Remote Web Services Composition and Orchestration

Seminars in Software Engineering
(Semantic Data and Service Integration)

Prof. Giuseppe De Giacomo

Prof. Massimo Mecella

Authors: Constantin Moldovanu
Emanuele Tatti

Project goals

- Remote composition and orchestration of existing web services
- Automatic deploy of the resources required for the composition
- Extension of existing tools (WSCE, Devilish Web Services Composition and Orchestration Engine)
- Integrate into an unique project and provide means for remote access

What is a Web Service

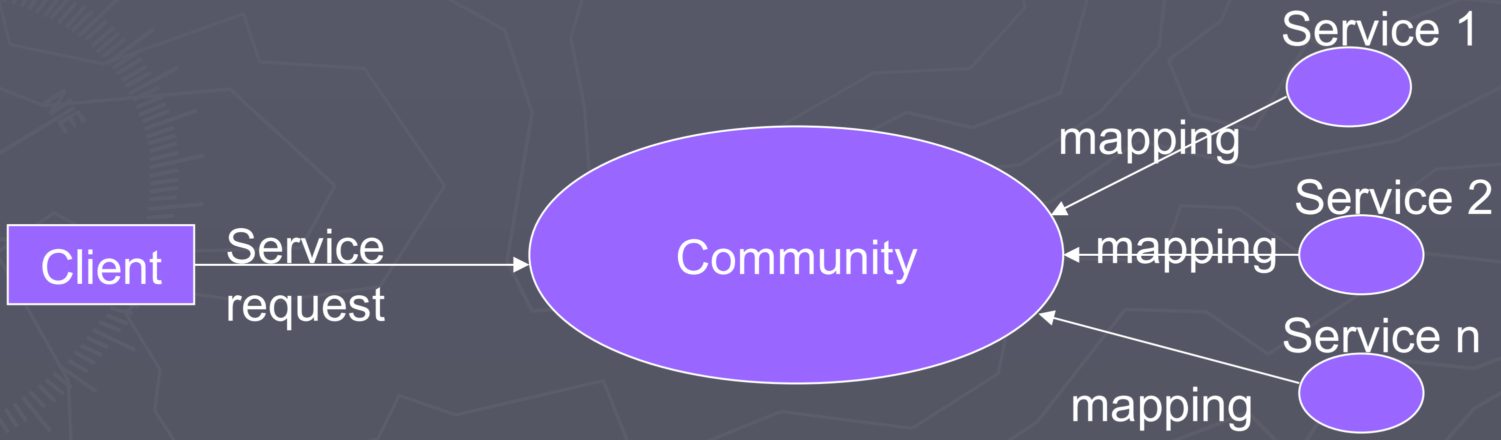
- A 'Web service' is defined by the W3C as "a software system designed to support interoperable machine-to-machine interaction over a network".
- Web services are frequently just Web APIs that can be accessed over a network, such as the Internet, and executed on a remote system hosting the requested services.
- Features:
  - Public interface (WSDL)
  - Standard comunication channels (SOAP, XML, HTTP)
  - Reuse of existing services
  - Easy integration

Web Service Composition – why

- Allows for reuse of existing services
- Different than importing libraries
  - Remote systems may rely on different technologies
  - Libraries require additional learning steps
  - The published APIs offer complete services rather than functionalities
- Distributed services
- Composition transparently creates new services by using the existing ones

Web Service Composition – how

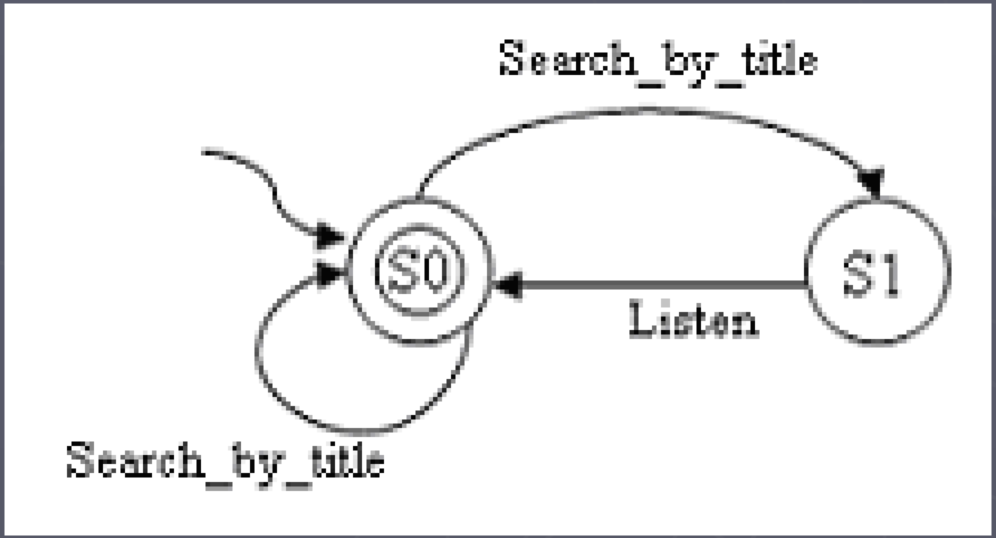
- Naming conventions:
  - Available services / community
  - Target service
- How:
  - The target service is transparently simulated for the client
  - Reuse of the functionality in the available services



Web service description

- Web services describe their functionality within a WSDL file (object types, interaction mode, methods).
- WSDL does not describe behavior
- A transition system describes the behavior of a WS:
  - $\langle A, S, S_0, \delta, F \rangle$
    - A actions
    - S states
    - $S_0 \in S$ initial states
    - $\delta \in S \times A \times S$ transitions
    - $F \in S$ final states

Web service description



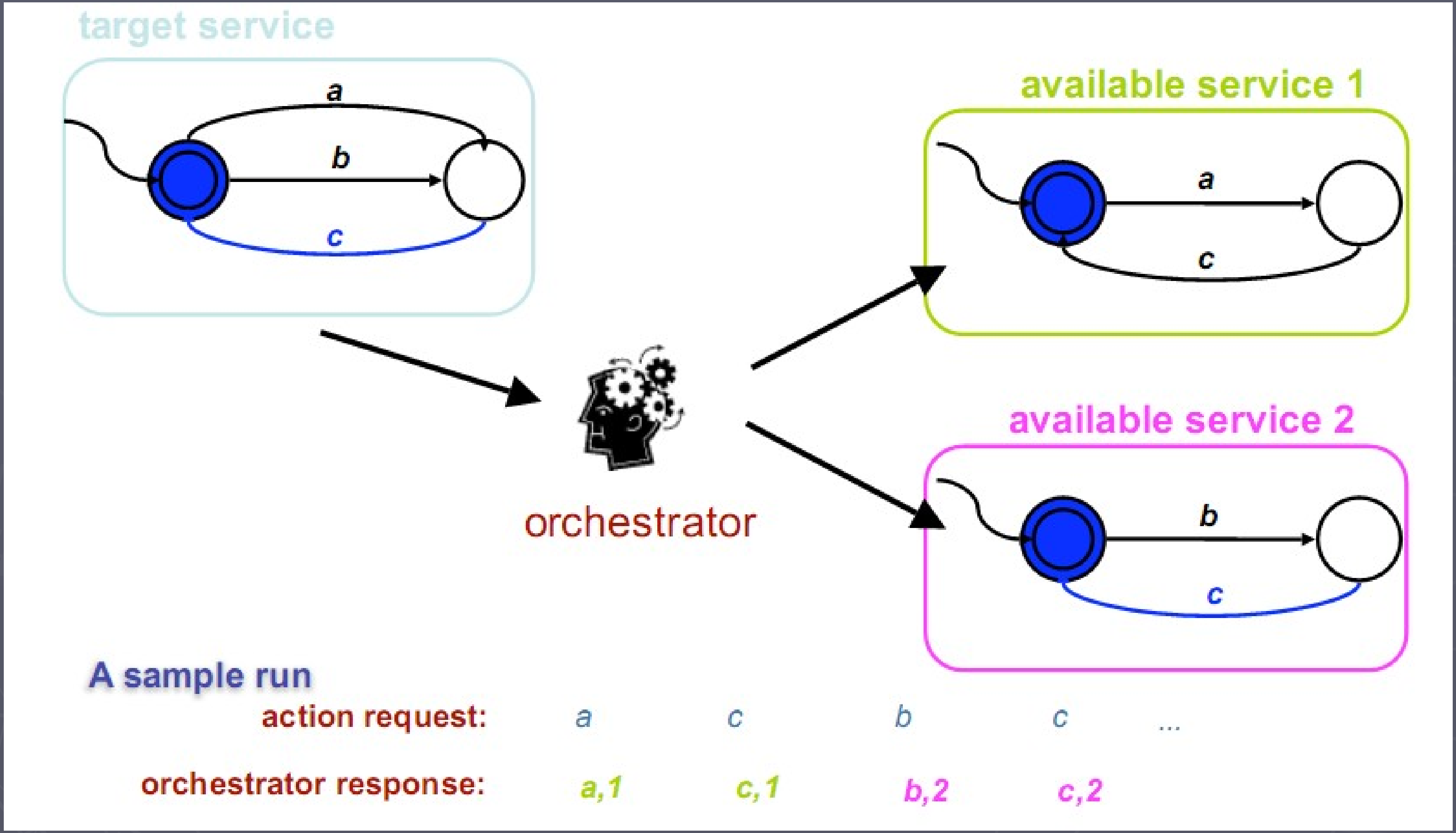
- A = {Search_by_title,Listen}
- S = {S0,S1}
- So={S0}
- δ = {(S0, Search_by_title,S1), (S1,Listen,S0),(S0, Search_by_title,S0)}
- F = {S0}

Target composition

- The target service has the same properties of a service (WSDL, transition system)
- Virtual service (does not rely on backend functionality)
- Requirements:
  - Has to have one initial state S_0
  - Has to have deterministic transitions

- The resulting target service is deterministic, while the available services in the community may also be non deterministic (devilish – don't know, different than and opposed to angelic – don't care)

Who manages the underlying services

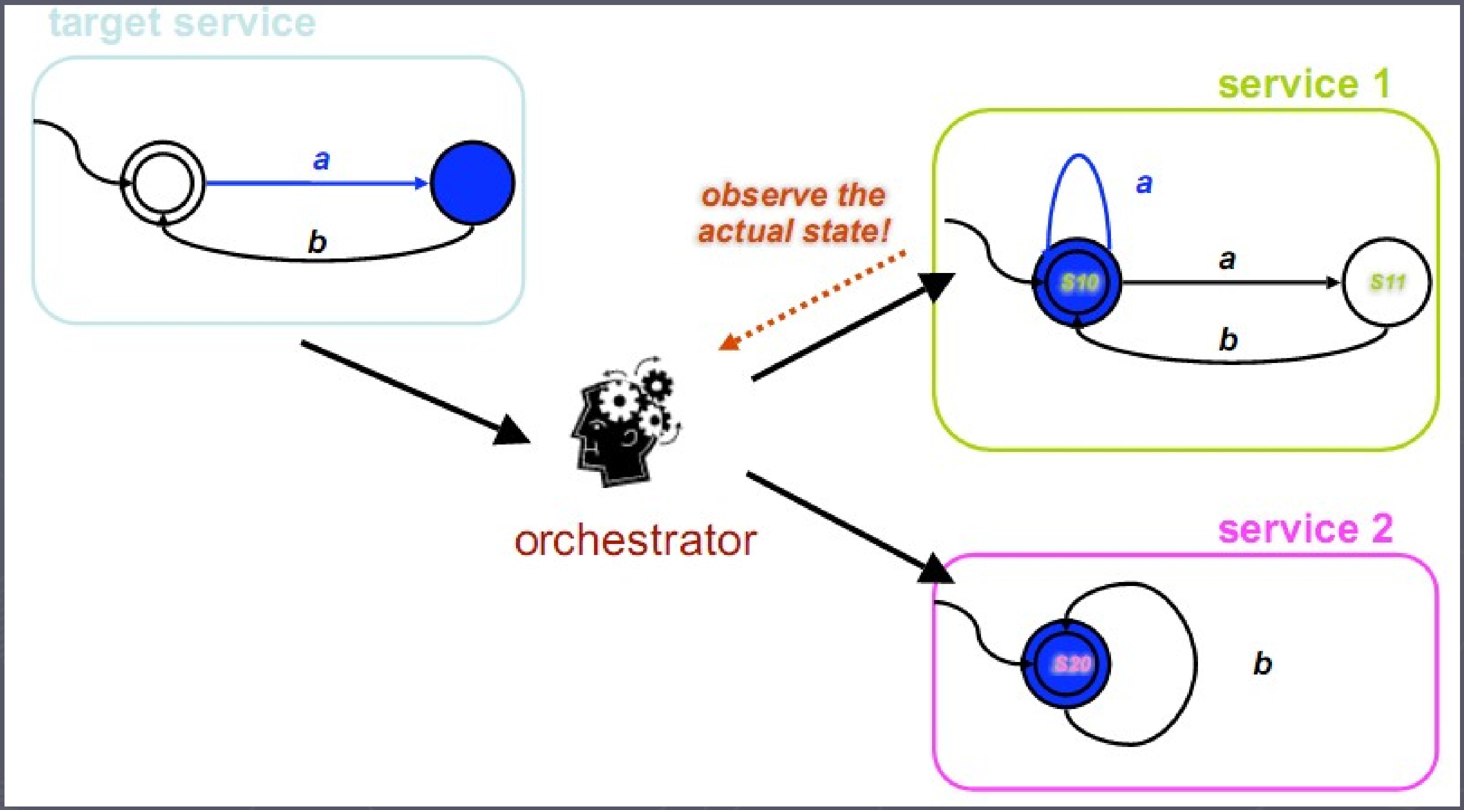


- The orchestrator decides which service does what

The orchestrator

- The orchestrator is a software entity that works in the background, transparently delegating the client requests towards the target service to the underlying services.
- Can be seen as a function: P(h, a) = i   given the history and the current action, decides the next invocation.
- In a deterministic community, the orchestrator can always calculate the state of each service, as it can be described backwards as: <ak, P([a_1 a_2 .. a_{k-1}], a_k)>.
- The orchestrator is unaware of the services implementation

The non deterministic orchestrator



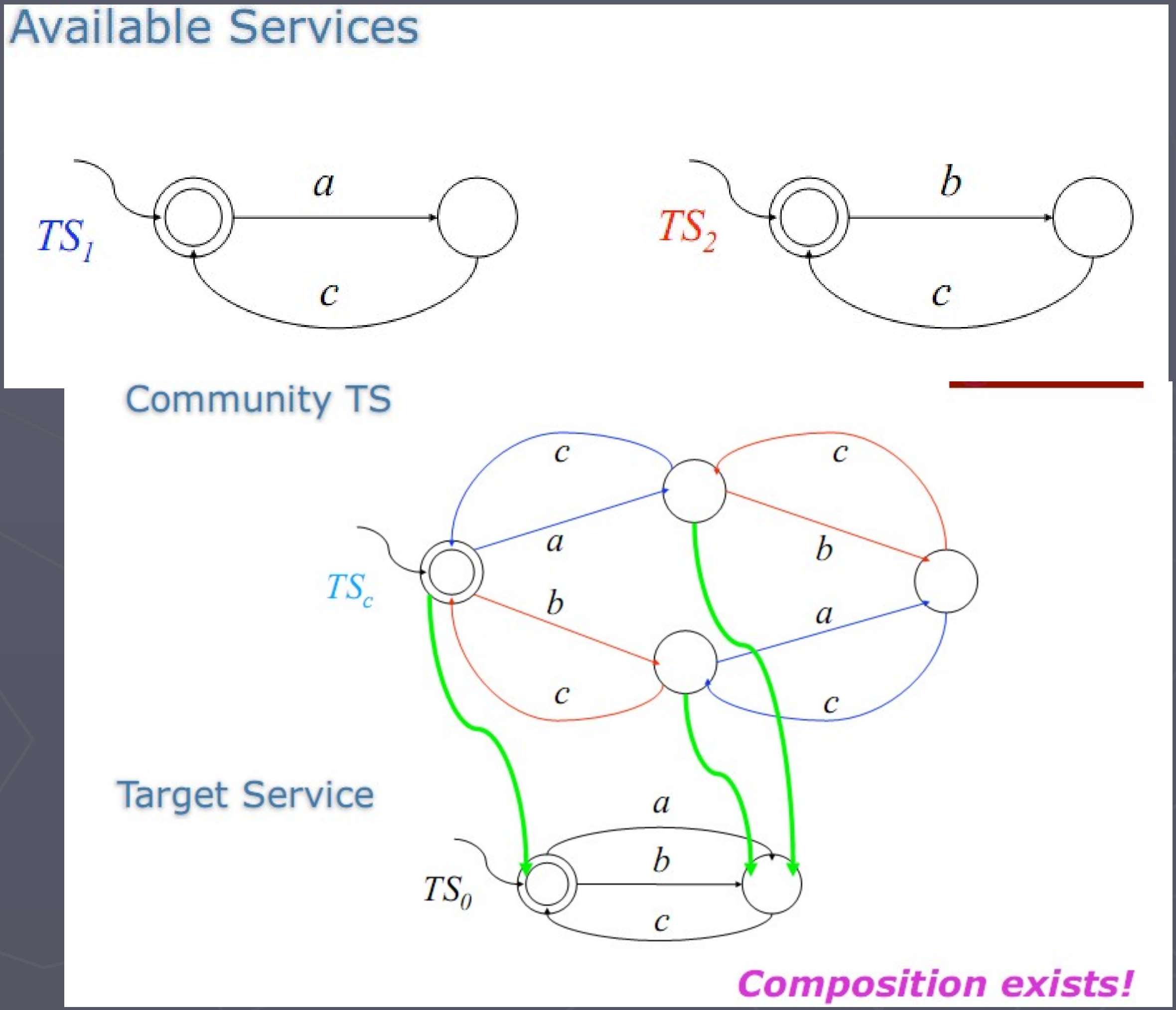
- In a non deterministic environment, the orchestrator must ask the services their current state

Composition techniques

- Problem: create an orchestrator that can actuate the target service by delegating tasks to available services
- Solution:
  - PDL (propositional dynamic logic) – not the chosen path
    - asserts partial correctness
    - PDL formula (problem description) satisfiable in EXPTIME and PSPACE
  - Simulation: the target behavior has to be **reproduced** by a combination of existing services, the community.
    - If the community can reproduce the target's behavior in every moment, it can correctly simulate the target.

Simulating composition

- The community can at least reproduce the target's behavior



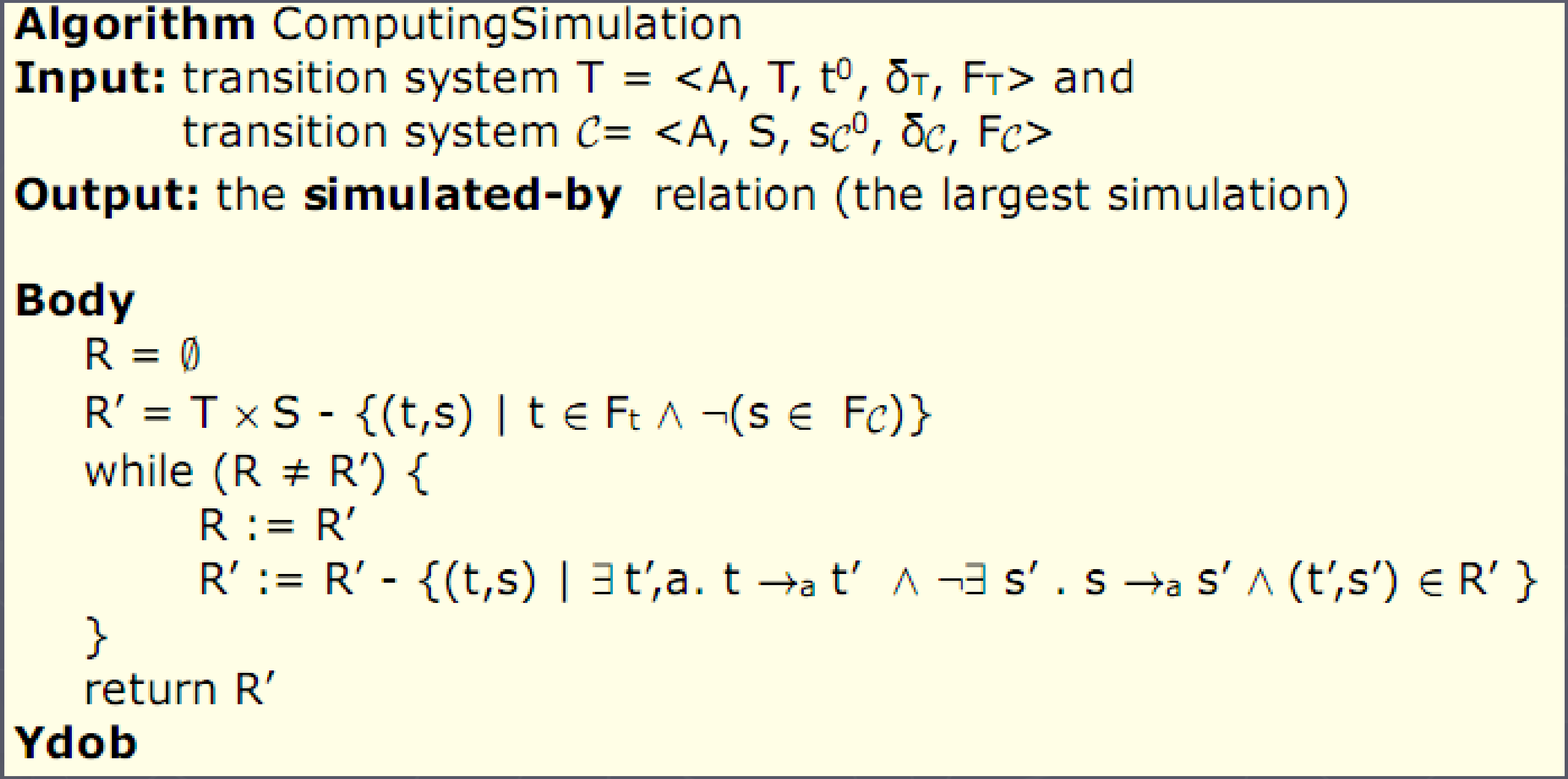
Simulation – background

- The community and the target service can be described with transition systems:
  - T = <A, ST, S_0T, δT, FT>
  - C = <A, SC, S0C, δC, FC>
- There is a **simulation relation** R, a binary relation that associates each state of T to a state of C s.t.

  if <sT, sC>$\in$R then sT final implies that sC is final and, for each action a, if sT $\rightarrow_a$ sT' then $\exists$ sC' | sC $\rightarrow_a$ sC' and

  <sT', sC'> $\in$ R.
- In details
  - If sT is final, so is sC
  - An action a brings sT and sC in two states, which pair is also in R

Simulation – how



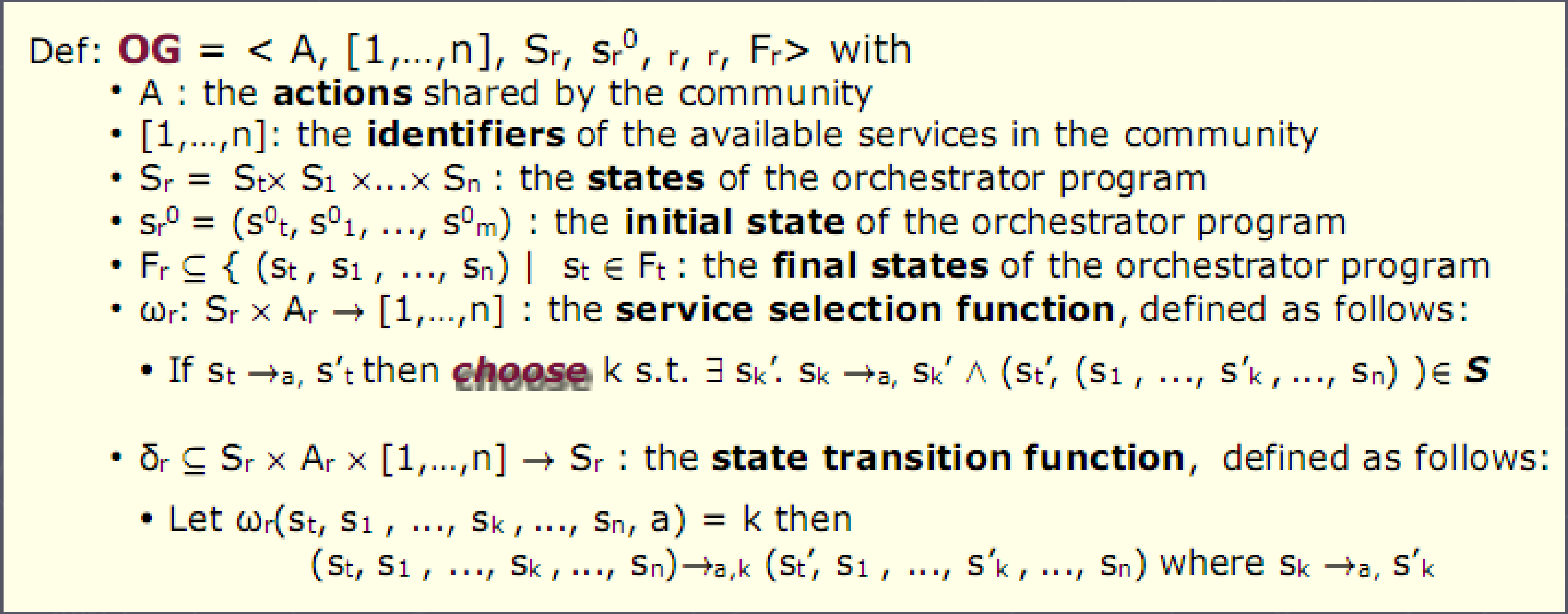
```
Algorithm ComputingSimulation
Input: transition system T = <A, T, $t^0$, $\delta_T$, $F_T$> and
       transition system $\mathcal{C}$= <A, S, $s_{\mathcal{C}}^0$, $\delta_{\mathcal{C}}$, $F_{\mathcal{C}}$>
Output: the simulated-by relation (the largest simulation)

Body
    R = $\emptyset$
    R' = $T \times S$ - $\{(t,s) \mid t \in F_t \wedge \neg(s \in F_{\mathcal{C}})\}$
    while (R $\neq$ R') {
        R := R'
        R' := R' - $\{(t,s) \mid \exists t',a.\ t \rightarrow_a t' \wedge \neg\exists s'.\ s \rightarrow_a s' \wedge (t',s') \in R'\}$
    }
    return R'
Ydob
```

- Largest simulation algorithm (remove from the Cartesian product T x C those pairs that do not match the rules)
  - EXPTIME in PSPACE

Who creates the orchestrator

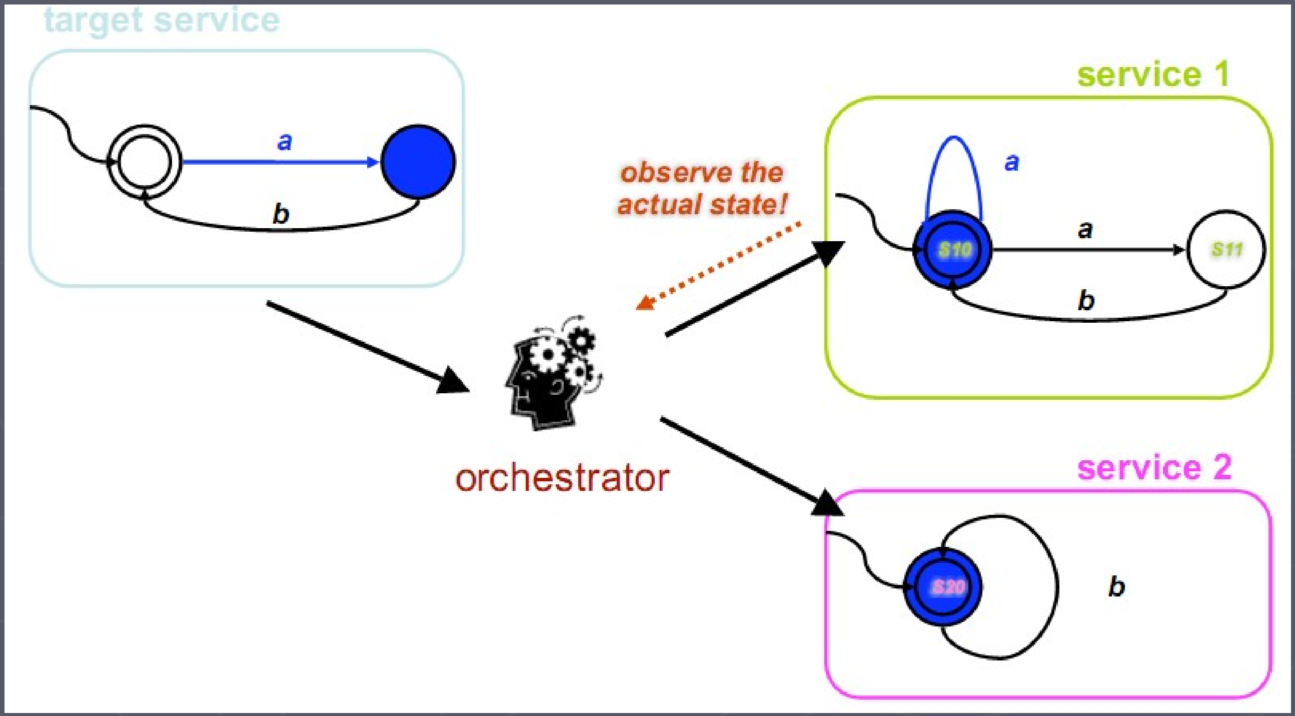
- Given the largest simulation, the orchestrator can be automatically built by an orchestrator generator



Def: **OG** = $< A, [1,\ldots,n], S_r, s_r^0, {}_r, {}_r, F_r>$ with

- A : the **actions** shared by the community
- $[1,\ldots,n]$: the **identifiers** of the available services in the community
- $S_r = S_t \times S_1 \times \ldots \times S_n$: the **states** of the orchestrator program
- $s_r^0 = (s^0{}_t, s^0{}_1, \ldots, s^0{}_m)$: the **initial state** of the orchestrator program
- $F_r \subseteq \{ (s_t, s_1, \ldots, s_n) \mid s_t \in F_t$: the **final states** of the orchestrator program
- ω_r: $S_r \times A_r \rightarrow [1,\ldots,n]$: the **service selection function**, defined as follows:
  - If $s_t \rightarrow_{a,} s'_t$ then *choose* k s.t. $\exists\, s_k'.\ s_k \rightarrow_{a,} s_k' \wedge (s_t', (s_1, \ldots, s'_k, \ldots, s_n)) \in \boldsymbol{S}$
- $\delta_r \subseteq S_r \times A_r \times [1,\ldots,n] \rightarrow S_r$: the **state transition function**, defined as follows:
  - Let $\omega_r(s_t, s_1, \ldots, s_k, \ldots, s_n, a) = k$ then
    $(s_t, s_1, \ldots, s_k, \ldots, s_n) \rightarrow_{a,k} (s_t', s_1, \ldots, s'_k, \ldots, s_n)$ where $s_k \rightarrow_{a,} s'_k$

Non deterministic orchestration



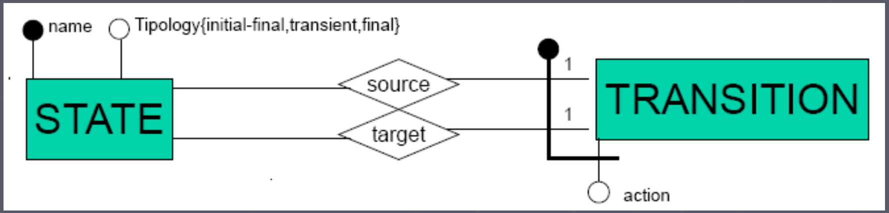
- Aside from the existential choice of the deterministic service, adds an universal choice for the non deterministic parts
- The idea is the same, but the orchestrator needs to ask available services their status

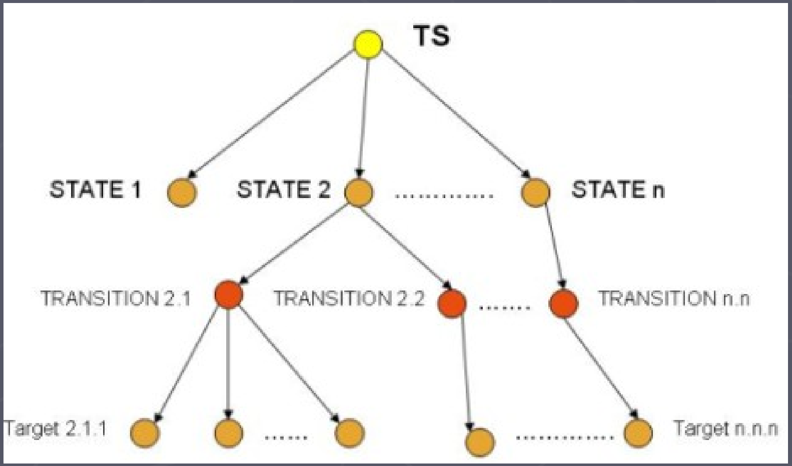
Automatic orchestration

- The orchestrator can be generated using a set of three standalone projects: TLV, WSCE, Devilish Web Services Composition and Orchestration Engine.
- The generation is mainly handled by TLV (Temporal Logic Verifier), which is a tool that can decide if a composition cam be accomplished, given a description in LTL (extension of SMV).
- WSCE generates the input and parses the output of TLV and provides useful information for the orchestrator generator.

A web service temporal description

- As seen before, TLV requires a temporal description of the web service to work with
- The LTL description is automatically written by WSCE, which requires a temporal description of the web service to be written in a language defined ad-hoc, **WSTSL**
- WSTSL is XML based and uses nodes to show relationships between the states and their transitions





- The structure can be visited as a tree and therefore as an XML tree

Project working environment

- This project uses the web services implementation proposed by the Apache Foundation, Axis
- Axis is a Java EE web application, that can be deployed in a Java EE server such as Tomcat
- Axis is a SOAP library and a SOAP client, provides a web services implementation
- Has two services deploy methods:
  - Renaming simple .java files into .jws files
  - Deploys Java libraries with WSDD description interfaces: the WSDL is generated and published

Automatic orchestration steps

- Thus these three tools accomplish the following steps:
  - Read the WSTSL descriptions of the available services and target service
  - Build the SMV file that describes the system
  - Decide if the composition can be achieved (TLV)
  - Interpret TLV result, generate the orchestrator by using an SQL table to save its status
  - The orchestrator is now ready to handle the requests of the target service, by using SQL tables and generated logic

Additional tools

- Another tool from the previous project (TargetGenerator) allows to create and deploy the target service
  - It requires in input: the target service name, the SQL generated by the OrchestratorGenerator, the WSDL of the community
  - Produces in output: completes the additional SQL required for the orchestrator, generates and builds the java class for the target service (which uses the orchestrator) and also those of any other objects described in the WSDL, creates and deploys the target WSDL

Session management

- As a web application, multiple clients can require various instances of the generated services, and each one has to have it's own data for the duration of the session
- The session management is based on the web application session management, as configured in the WSDD:
  - <parameter name="scope" value="session" />
- The session is also maintained by the orchestrator with the various available services that have to be, therefore, available with a session scope
- Obviously the end client has to be able to maintain the server session

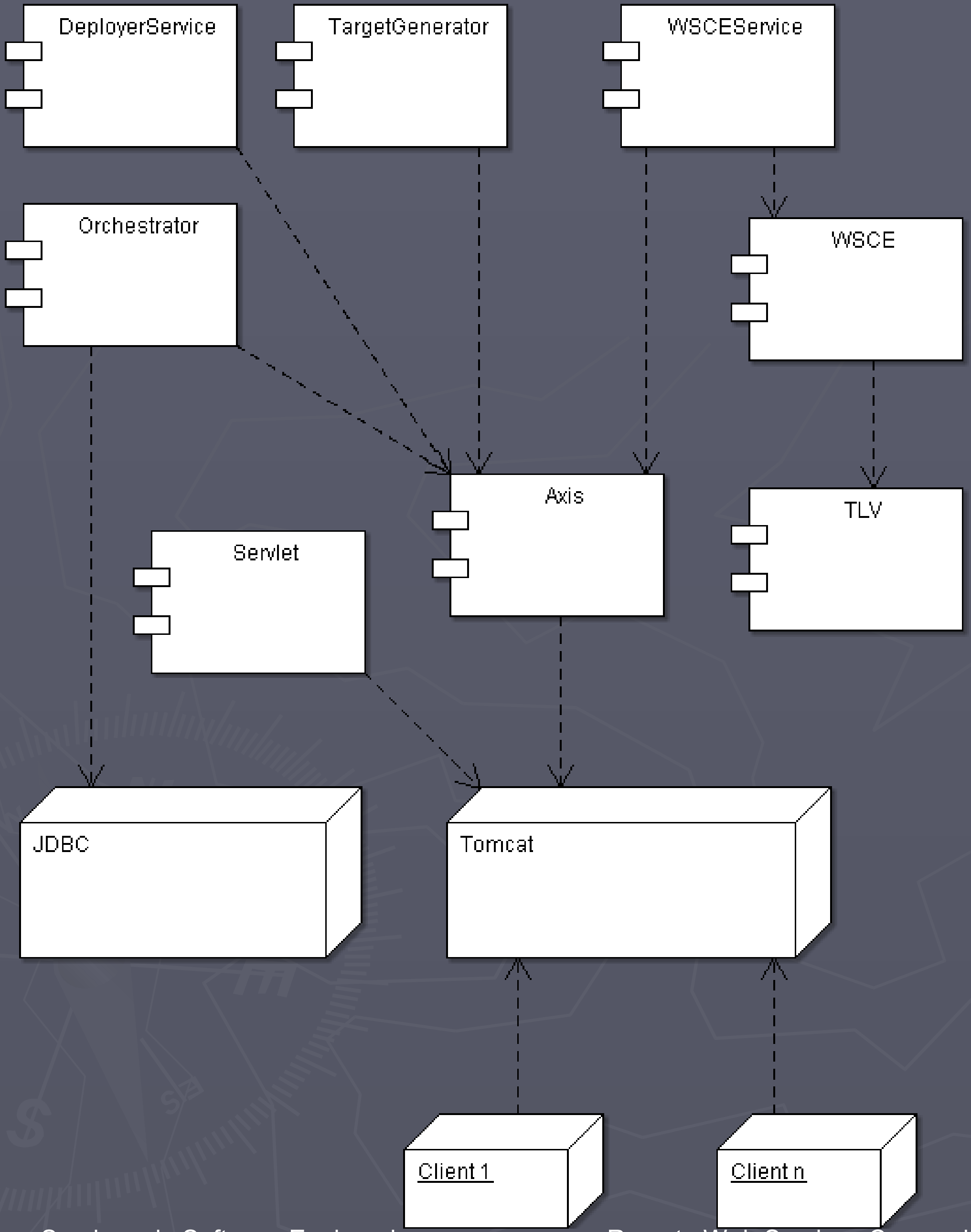
Available services constraints

- The available web services in the community have to abide the following constraints:
  - They have to be stateful (session scope)
  - They have to express their status through an "int getStatus()" method
  - They have to use RPC services (and not RMI services)
- There are no constraints on the deterministic state of the available services

Project challenges

- These tools are rather loosely coupled and require additional work to get them functional
- Some modifications were required in order to create and distribute a **single application** that offers integrated access and hides much of the details to the user that wants to perform a composition and orchestration
- Various optimizations and improvements were needed
- New services (to complete functional gaps) and graphical tools enhance the user's experience

Component diagram



- With a careful redesign we modified the way the various components interact with eachother

New services

- Two new web services handle that part of the composition flow that was not automatized:
  - **DeployerService** – given a service's working directory and its WSDD, it can deploy the new service in the Axis container
  - **WSCEService** – allows for WSCE to be called from the web services interface, from remote users as well, offering descriptive messages; this service also has a method for listing all the target services on the server that have not yet been generated

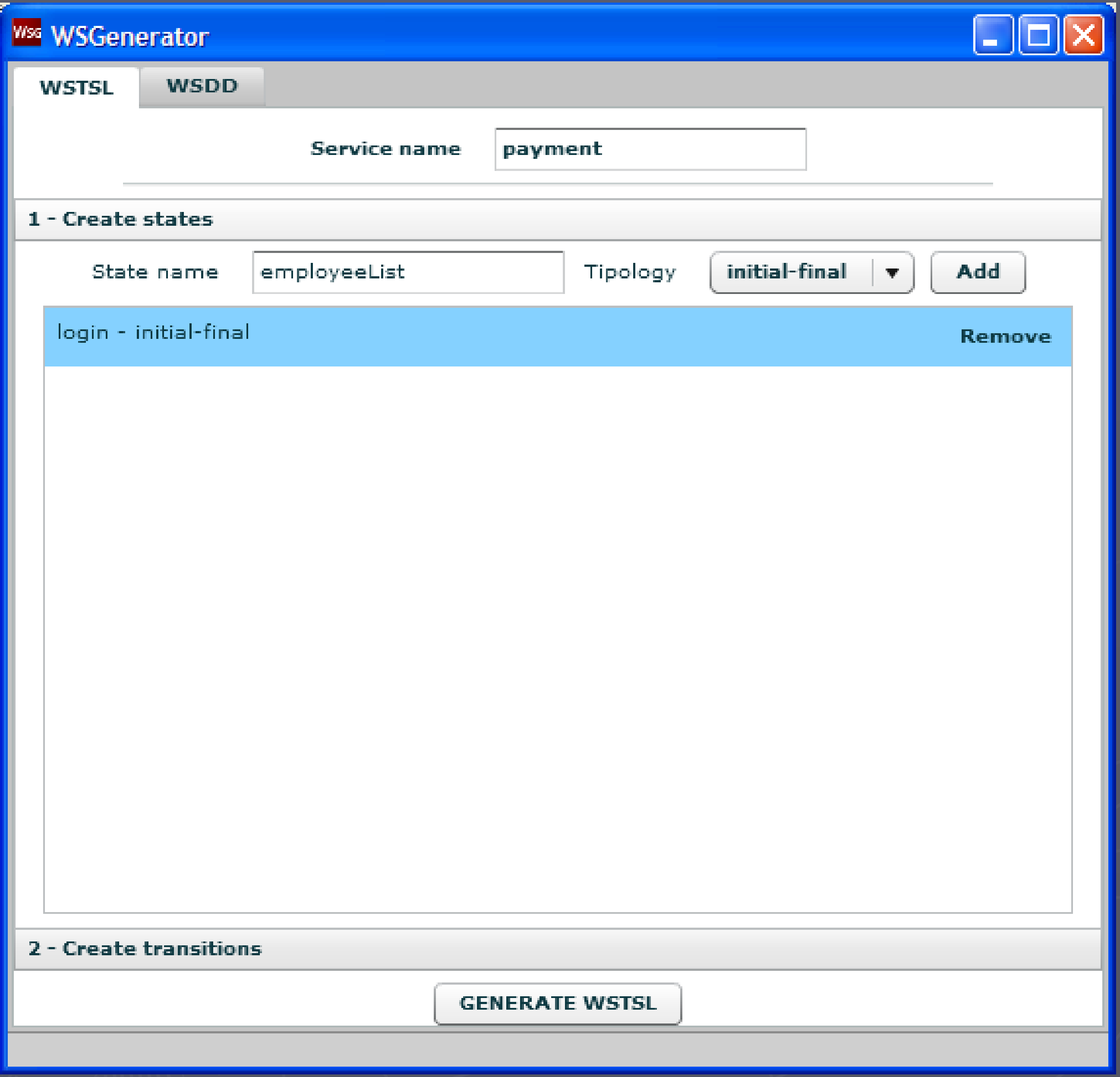
New servlets

- Aside the new web services, two Java EE servlets have been added, to complete the handling of the remote operations:
  - **UploadServlet**: allows a remote user to upload WSDD, WSDL, WSTSL and JAR libraries into the system, that are then ready to be deployed or orchestrated
  - **CompositionDirectoryCreatorServlet**: receives in input the working directory and creates on the server the complete directory structure necessary for WSCE to work

New graphical tools

- Two graphical tools have been added:
  - **WSGenerator**: generates and saves on the local computer WSTSL and WSDD files;
  - **AxisAdmin**: acts as an interface to the Axis services, providing simple means of communication with the default components;
- The tools have been developed using Adobe Flex and Adobe Air

WS Generator


WSGenerator

Service name: payment

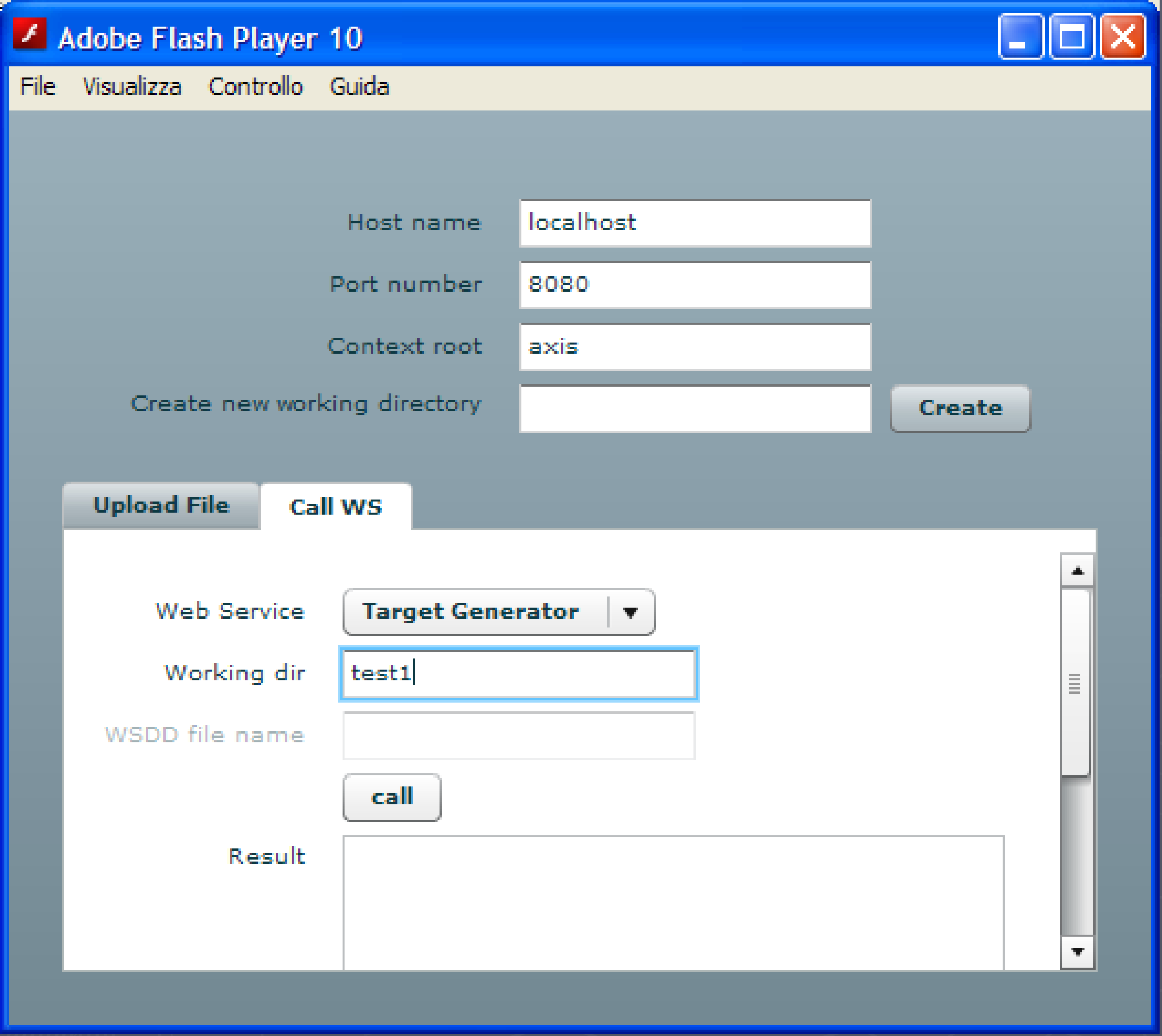
1 - Create states

State name: employeeList
Tipology: initial-final

login - initial-final

2 - Create transitions

Axis Admin



Host name: localhost
Port number: 8080
Context root: axis
Create new working directory:

Web Service: Target Generator
Working dir: test1
WSDD file name:
Result:

WSCE Integration

- As to pursue the goal of distributing one, functional web application, we decided to distribute WSCE within the application
- WSCE had to be partially modified, as its previous design didn’t allow it to accept a different composition directory (whose name however was embedded in the source code) and didn’t provide a specific entry point
- The main modifications are:
  - The composition directory is now an absolute path, received as parameter
  - A static method has been added, to access its functionality without having to call it into a separate process

Optimizations and improvements

- Various areas of the components have been modified for improvements and remove those that could have become deadlocks or bottle-necks.
- The application now does not use external processes to call other Java components, which are all available through their entry interfaces.
- The compilation process is also performed within the application process, by invoking special Java methods (tools.jar) for on-the-run compilation (no more deadlocks).

Optimizations and improvements

- The TargetGenerator and Orchestrator now share the same JDBC connection manager, ensuring faster access and better session management.
- The application uses commons-logging as a logging interface
- The configuration system has been re-written: most of the configuration parameters have been removed and they reside in a properties file, that can be edited without the need for a recompilation

Conclusions

- Web Service composition is an extremely interesting means for reusing existing services and creating new ones transparently, that can be source for composition as well.
- The available services will have to implement a method that returns their current status, feature that is not implemented in the existing systems.