WADE

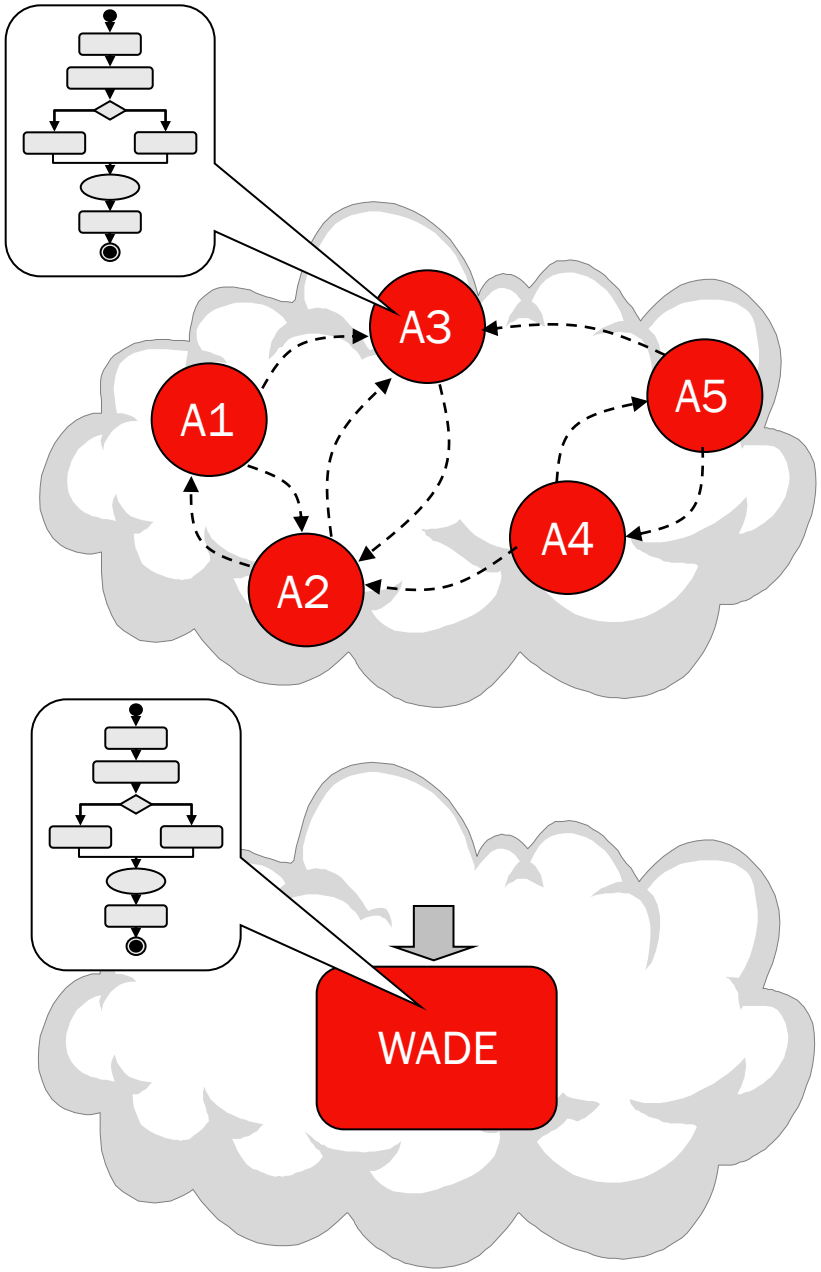
Workflow and Agents Development Environment

WADE

- E` un'estensione di JADE
  - Un'applicazione WADE-based e` anche un'applicazione JADE-based
  - Tutte le feature di JADE sono disponibili anche in WADE
- Aggiunge a JADE
  - La possibilita` di definire i taks eseguibili dagli agenti secondo la metafora dei workflow
    - Non un unico workflow engine, ma tanti "*micro-workflow engines*" embedded negli agenti
  - Componenti e meccanismi che facilitano l'amministrazione di una applicazione WADE- based
  - I due aspetti possono essere usati separatamente
- Open Source da Maggio 2008
  - http://jade.tilab.com/wade
  - Versione corrente: 3.0
  - Internamente 3.0.18 (http://nnem-tracall.cselt.it/trac/jade)

Utilizzi

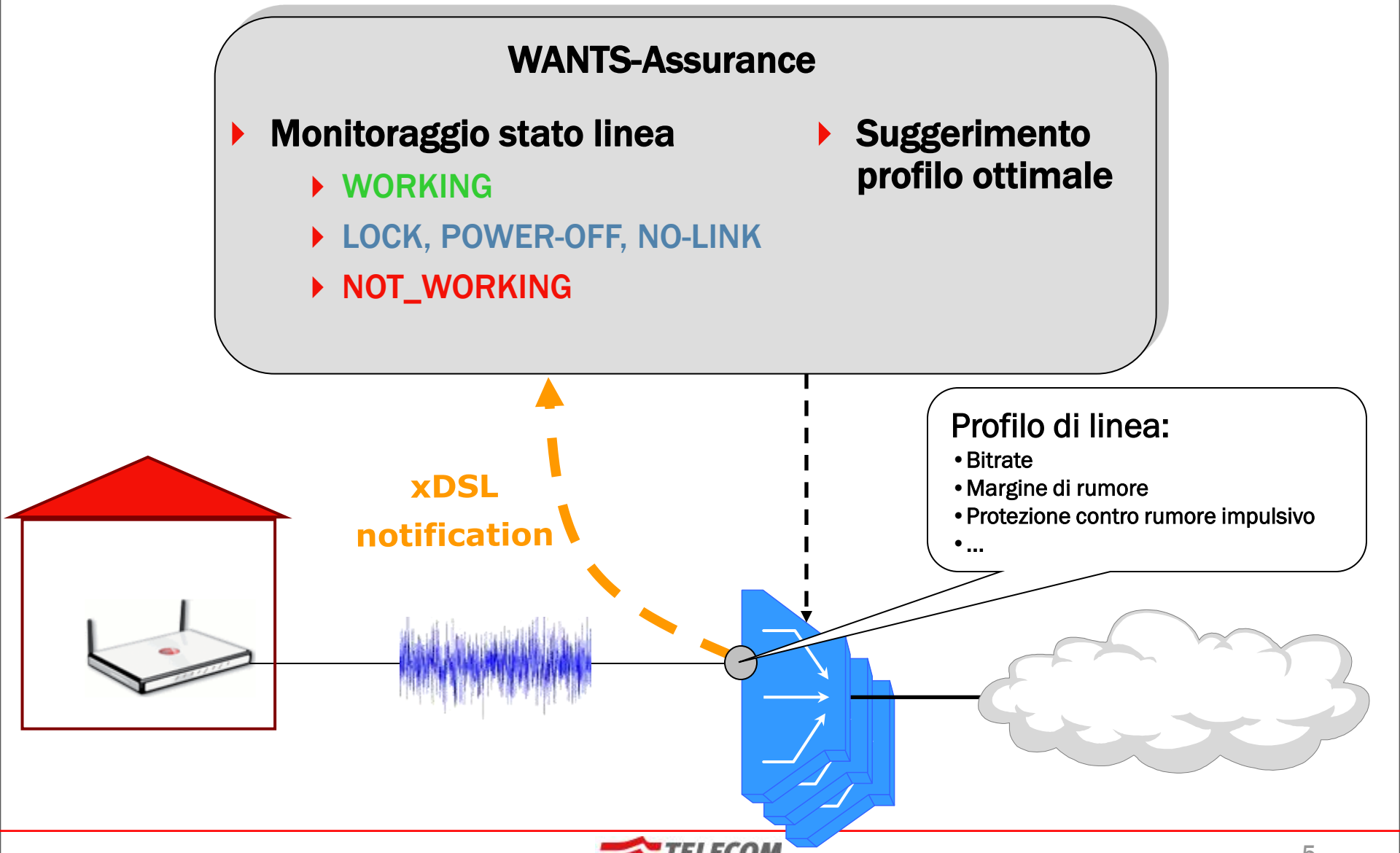
- **Framework di sviluppo**
  - Creo una applicazione completa utilizzando WADE come framework di sviluppo
  - Le componenti dell'applicazione sono agenti domain-specific
  - Parte delle logiche di business dell'applicazione sono implementate come workflow
- **Workflow Engine**
  - Uso WADE come motore per eseguire workflow all'interno di un'applicazione esistente
  - Non vedo gli agenti



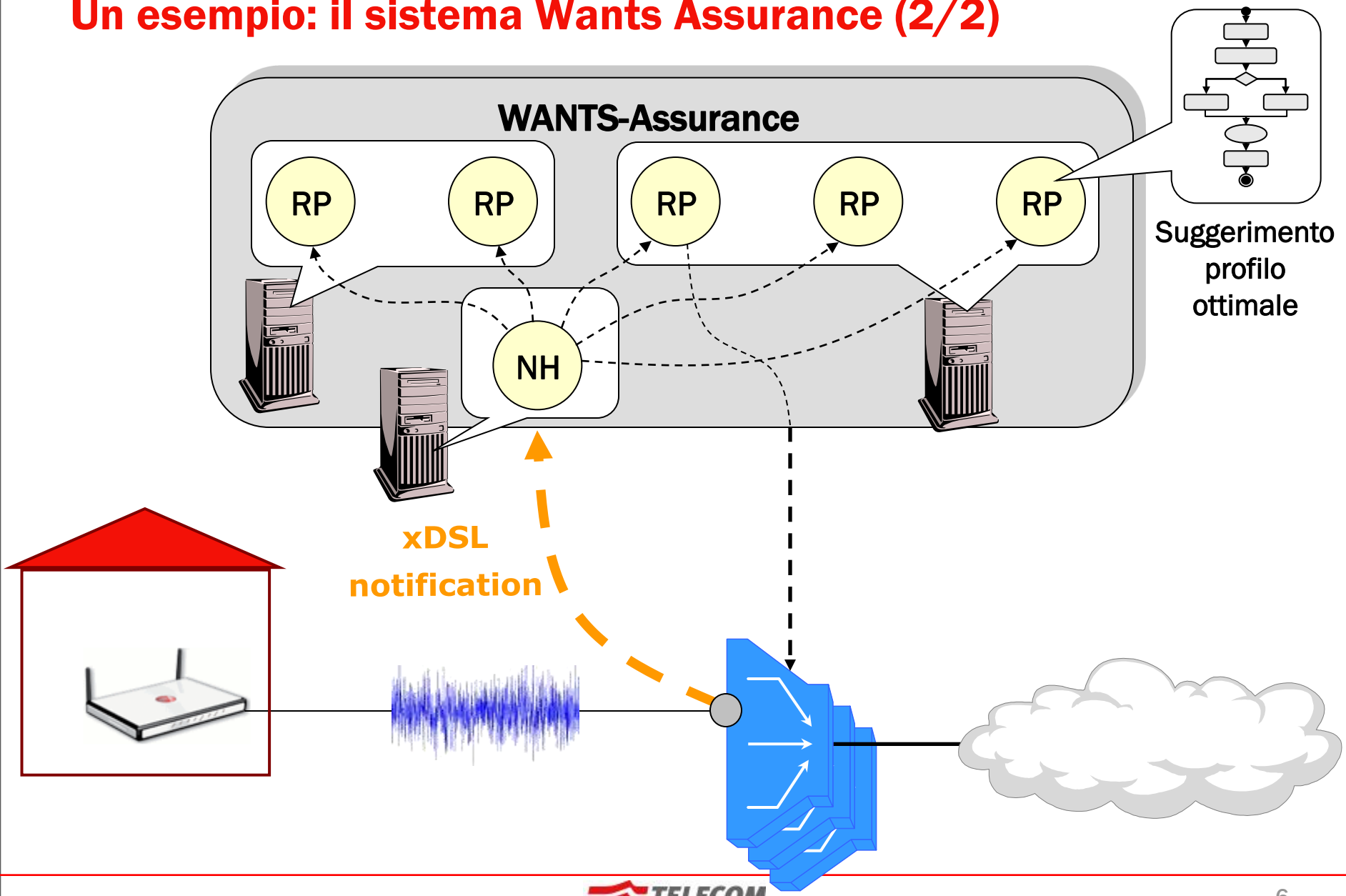
Applicazioni TI basate su WADE

- WANTS – Attivatore servizi broadband con accesso fisso
- WIZARD/WeFlow – Sistema di guida assistita ai tecnici in field e in back office in interventi di installazione e manutenzione
- WANTS-Assurance – Sistema di monitoraggio linee XDSL
- WeMash – Ambiente di creazione on-the-fly di semplici applicazioni (mash-up) di supporto all'operativita`
- Crowd-Testing – Sistema di gestione di campagne di test di servizi VAS su terminali mobili
- Piattaforma di GAS Management

Un esempio: il sistema Wants Assurance (1/2)



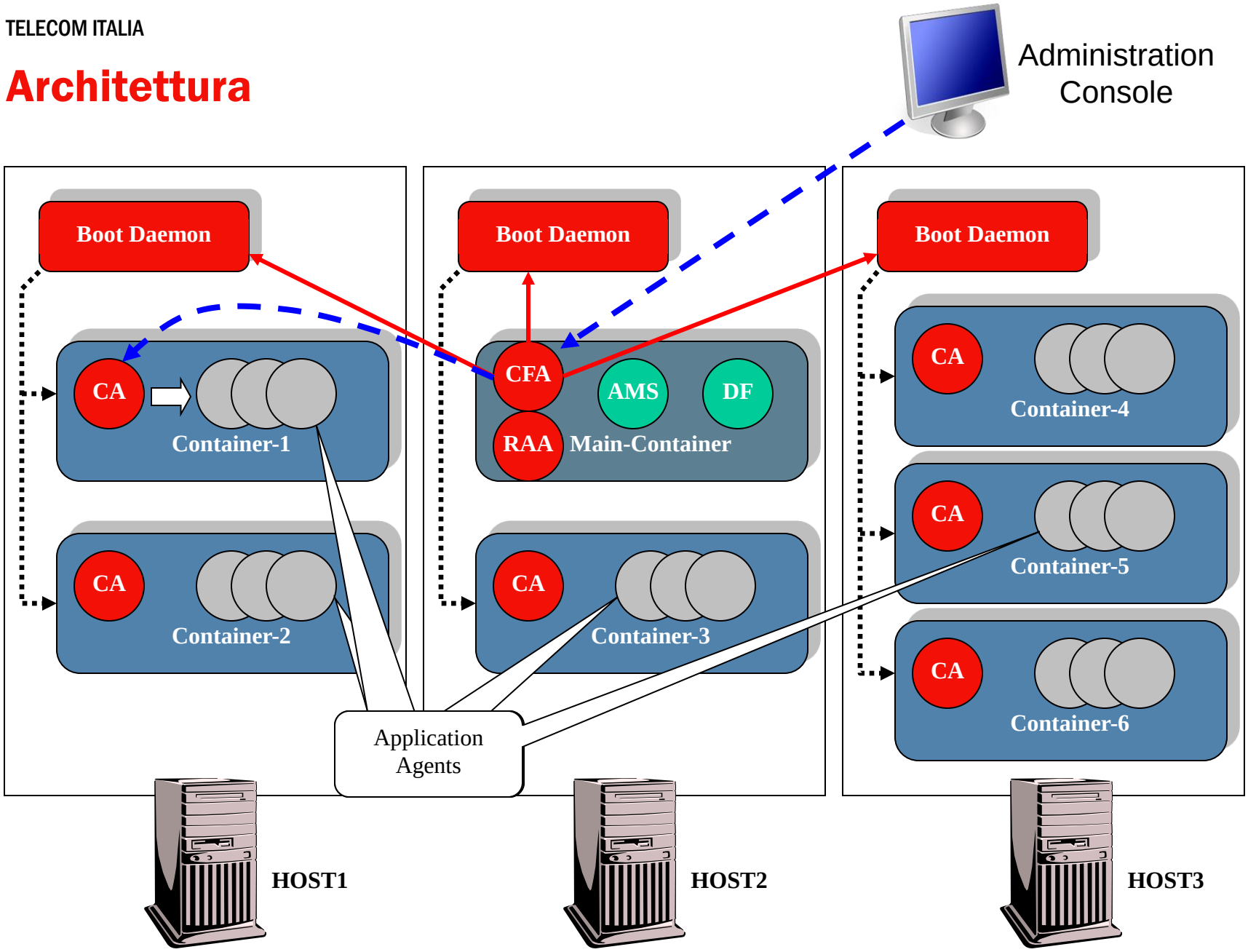
Un esempio: il sistema Wants Assurance (2/2)



Componenti specifiche di WADE

- **Boot Daemon (1 x host)**
  - Processo (non-agent) responsabile dell'attivazione dei container su un dato host
- **Configuration Agent – CFA (1 sul Main Container)**
  - L'agente responsabile di gestire startup e shutdown dell'applicazione WADE-based interagendo con i Boot Daemons
- **Controller Agent – CA (1 x container)**
  - Gli agenti responsabili di controllare le risorse dei singoli container e di gestire i meccanismi di fault tolerance
- **Runtime Allocator Agent – RAA (1 sul Main Container)**
- **Workflow Engine Agent – WEA (n)**
- **Workflow Status Manager Agent (1)**
  - L'agente che traccia tutte le esecuzioni di WF
- **Event System Agent (1)**
  - L'agente che implementa il meccanismo ad eventi su cui possono bloccarsi i WF

Architettura

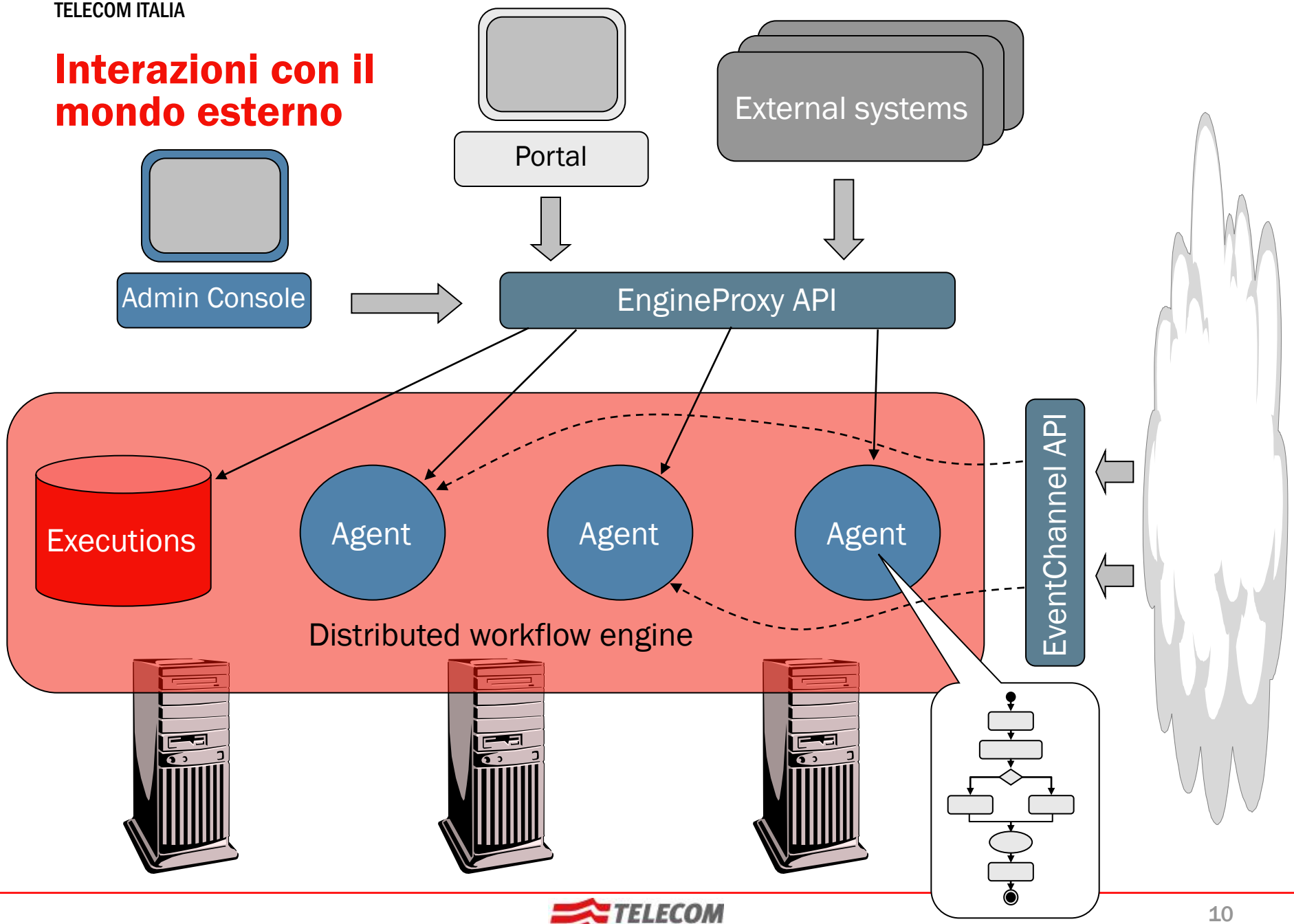


Application configuration file

```
<platform description="This is a sample configuration" name="Sample">
  <hosts>
    <host name="localhost">
      <containers>
        <container name="Execution-Node">
          <agents>
            <agent name="performer1" type="Workflow Engine Agent"/>
            <agent name="performer2" type="Workflow Engine Agent"/>
          </agents>
        </container>
        <container name="Administration-Node">
          <agents>
            <agent name="wsma" type="Workflow Status Manager Agent"/>
            <agent name="esa" type="Event System Agent"/>
          </agents>
        </container>
      </containers>
    </host>
  </hosts>

  <agentPools>
    <agentPool name="foo" type="My Type" size="100"/>
  <agentPools>
</platform>
```

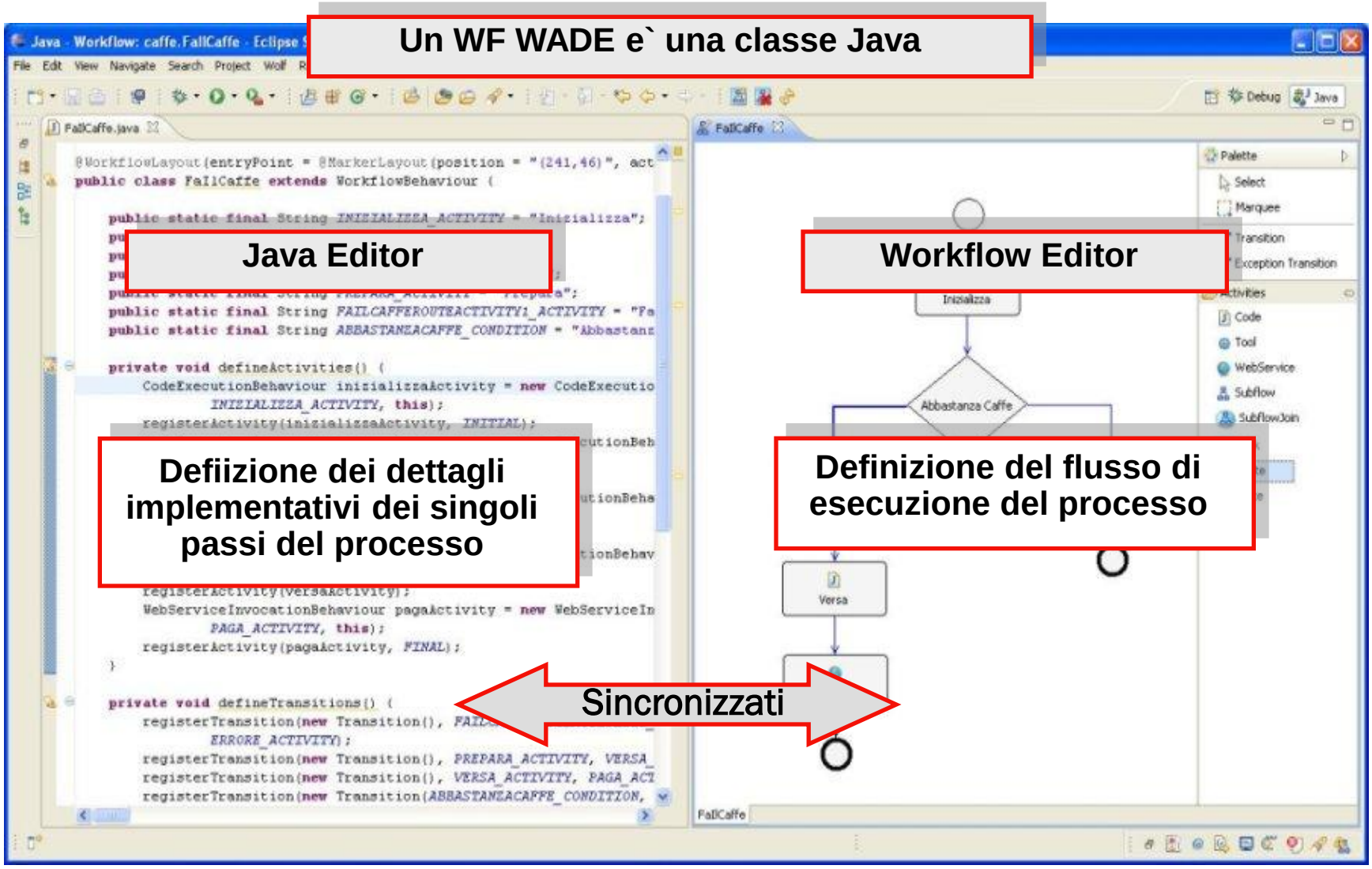
Interazioni con il mondo esterno



Workflow

- **Un workflow e` la definizione formale di un processo in termini di attivita` da eseguire, relazioni tra queste, criteri che ne specificano l'attivazione e la terminazione e informazioni addizionali quali eventuali input e output, i tool da invocare, i dati che devono essere manipolati durante l'esecuzione, I partecipanti…**
- **Un Workflow Engine e` uno strumento capace di eseguire processi definiti come workflow automaticamente.**
- **Vantaggi dell'approccio a Workflow**
  - Consentono una rappresentazione grafica del flusso di esecuzione
  - Auto-documentati
  - Tracciabili
  - Modificabili a caldo
  - Passi di esecuzione espliciti → possibilita` di realizzare meccanismi di rollback (semi)automatici

L'editor grafico WOLF

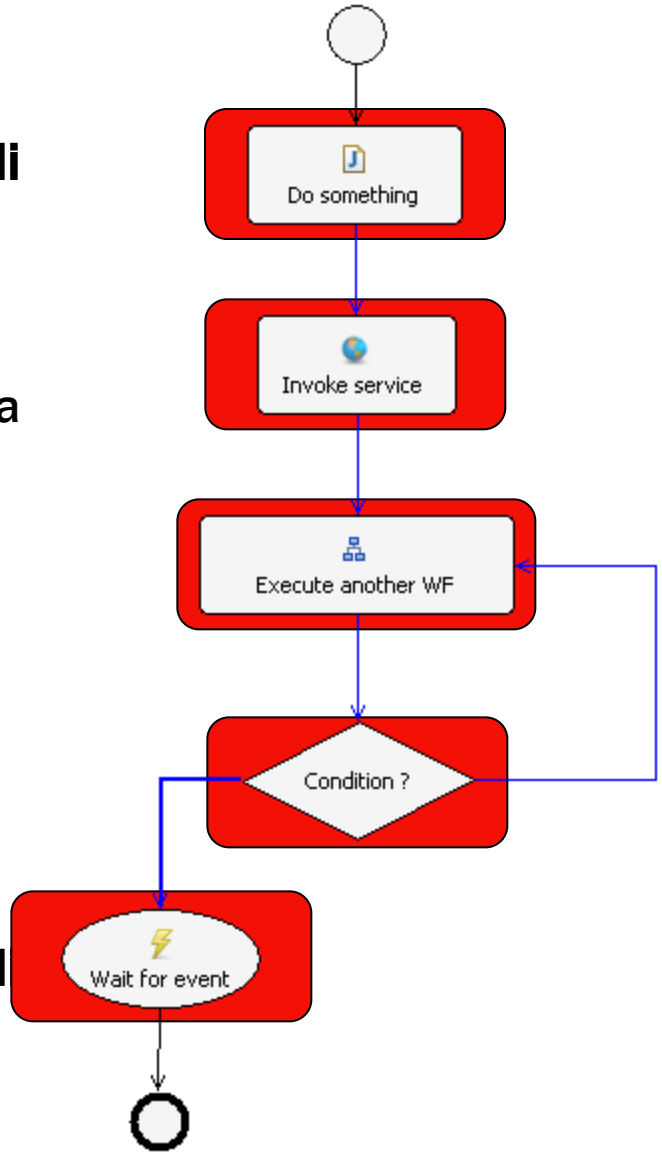


Funzionalità di Wolf

- Creazione ed editing grafico di Workflow
- Setting di un progetto per un'applicazione WADE-based
  - Impostazione classpath, configurazioni…
- Quick-launch/debug di un Workflow
- Connessione ad una piattaforma WADE remota
- Deploy di workflow

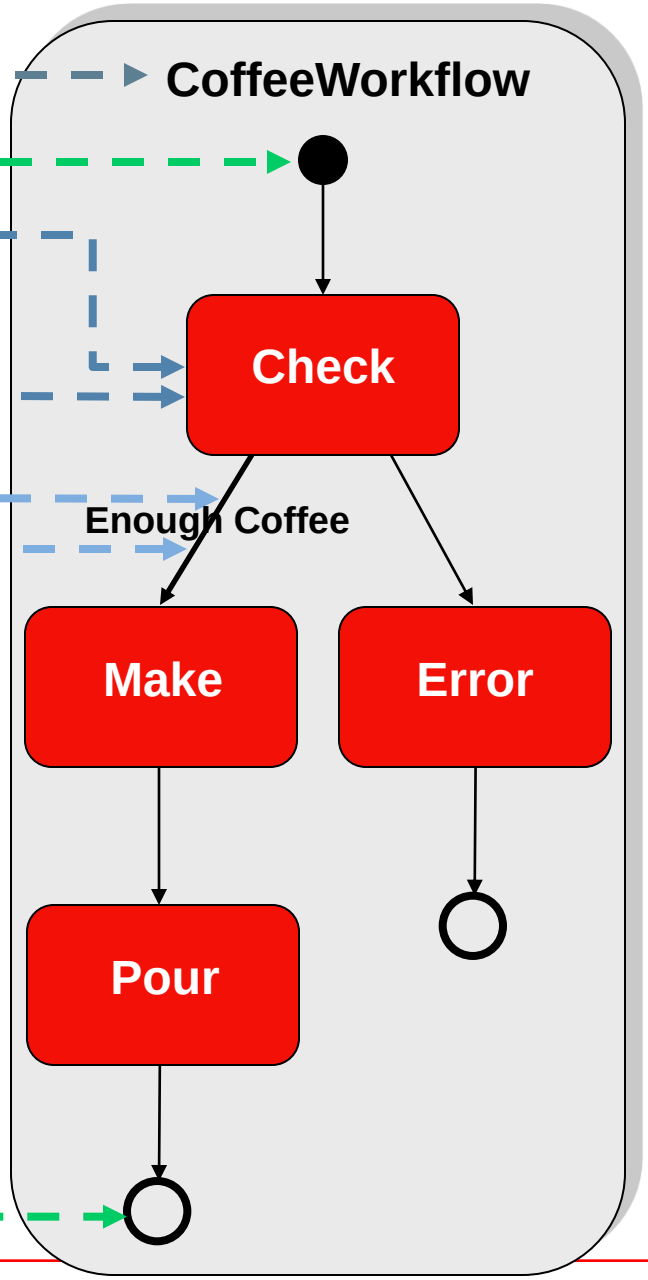
Principali tipi di activity

- **CodeActivity - Invocazione di blocchi di codice user defined**
- **WebServiceActivity - Invocazione di WS**
  - Supporto completo alla gestione della security nelle invocazioni
- **SubflowActivity - Esecuzione di altri WF**
  - Inline
  - Sincroni
  - Asincroni
  - Indipendenti
- **WaitEventActivity - Sospensione in attesa d eventi asincroni esterni al sistema**
- **Blocchi condizionali**



Workflow class

```
public class CoffeeWorkflow extends WorkflowBehaviour {
    ...
    private void defineActivities() {
        registerActivity(new CodeExecutionBehaviour(this, “Check”) , INITIAL);
        registerActivity(new CodeExecutionBehaviour(this, “Make”));
        registerActivity(new CodeExecutionBehaviour(this, “Error”), FINAL);
        registerActivity(new CodeExecutionBehaviour(this, “Pour”) , FINAL);
    }
    ....
    private void defineTransitions() {
        registerTransition(new Transition(this, “EnoughCoffee”), “Check”, “Make”);
        registerTransition(new Transition(), “Check”, “Error”);
        registerTransition(new Transition(), “Make”, “Pour”);
    }
    ...
    protected void executeCheck() {
    }
    ...
    protected void executeMake() {
    }
    ...
    protected void executePour() {
    }
    ....
    protected void executeError() {
    }
    ...
    protected boolean checkEnoughCoffee() {
    }
}
```



Elementi della classe di un Workflow

- **Workflow → Classe Java che estende** `WorkflowBehaviour`
- **Activity →**
  - **`Metodo void`** della classe del workflow
  - **`Behaviour`** registrato sull'oggetto Workflow responsabile di invocare il metodo corrispondente all'activity con opportuni parametri a seconda del tipo di activity
- **Transition →**
  - **`Metodo boolean`** della classe del workflow che implementa la condizione (se presente)
  - **`Transition`** registrato sull'oggetto Workflow responsabile di invocare il metodo della condiziona quando questa deve essere valutata
- **Formal Parameters → Field della classe del Workflow annotati con l'annotation** `@FormalParameter`
- **Tutte le informazioni di appearance (e.g. posizioni, routing points…) definite con l'annotation** `@WorkflowLayout`

Principali caratteristiche del Workflow Engine di WADE

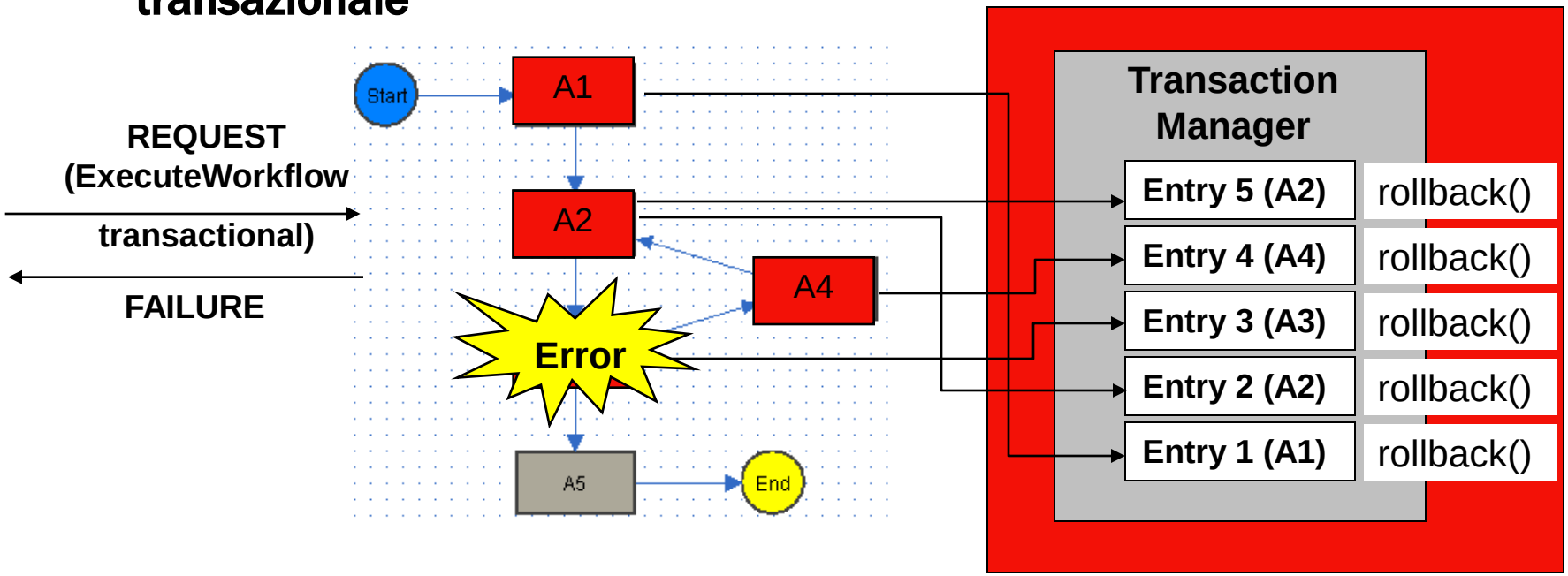
- **Un WF WADE e` implementato da una classe Java**
  - Estrema efficienza di esecuzione (no interpretazione di formalismi XML o simili)
  - Massima liberta`/controllo nell'implementazione delle logiche di business
  - Vista grafica e vista codice tenute sincronizzate
- **Scalabilita`**
  - Engine nativamente distribuito basato sul paradigma ad agenti
- **Deploy/modifiche a caldo dei WF**
  - Ottenuto mediante l'uso di un ClassLoader Java ad hoc
- **Ereditarieta`**
  - Possibilita` di definire nuovi WF a partire da WF esistenti, specificando solo le differenze

WF Short-Running e Long Running

- Short-running – **stateless: Non sopravvivono allo shutdown del sistema**
  - Tempo di esecuzione tipicamente breve (minuti)
  - CPU-consuming → Massima efficienza
- Long running – **statefull: Sopravvivono allo shutdown/restart del sistema**
  - Tempo di esecuzione tipicamente lungo (ore/giorni/mesi)
  - Salvano lo stato di esecuzione ad ogni passo
- **NOTA: Anche i WF Short-Running possono bloccarsi in attesa di eventi, ma se il sistema fa shutdown vengono abortiti**

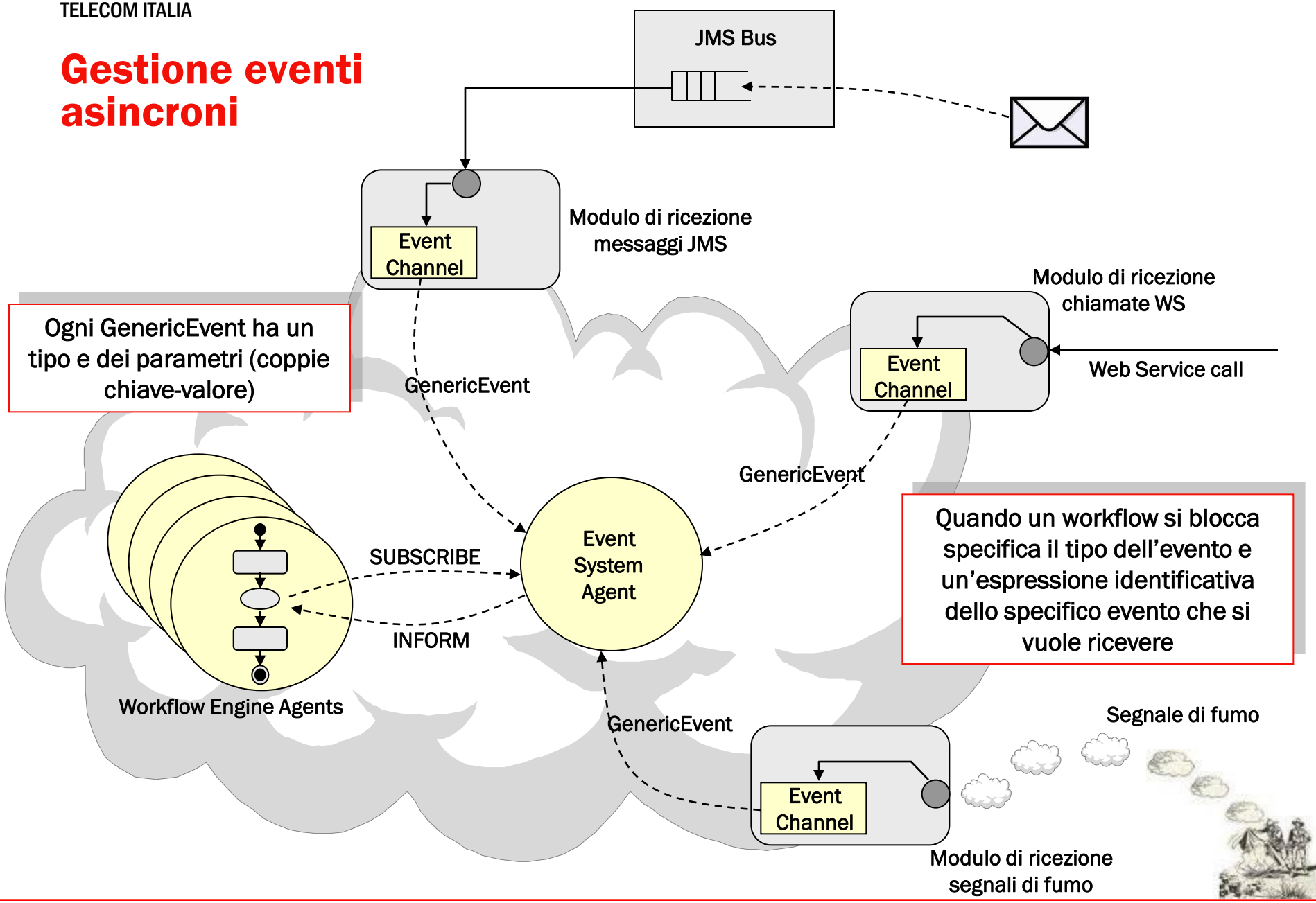
Workflow Transazionali

- Un workflow short-running puo` essere eseguito in modalita` transazionale



- 1 transaction entry per ogni passo del workflow
- E` possibile fornire un workflow di rollback user-defined invece del meccanismo di default

Gestione eventi asincroni



Risveglio di WF sospesi

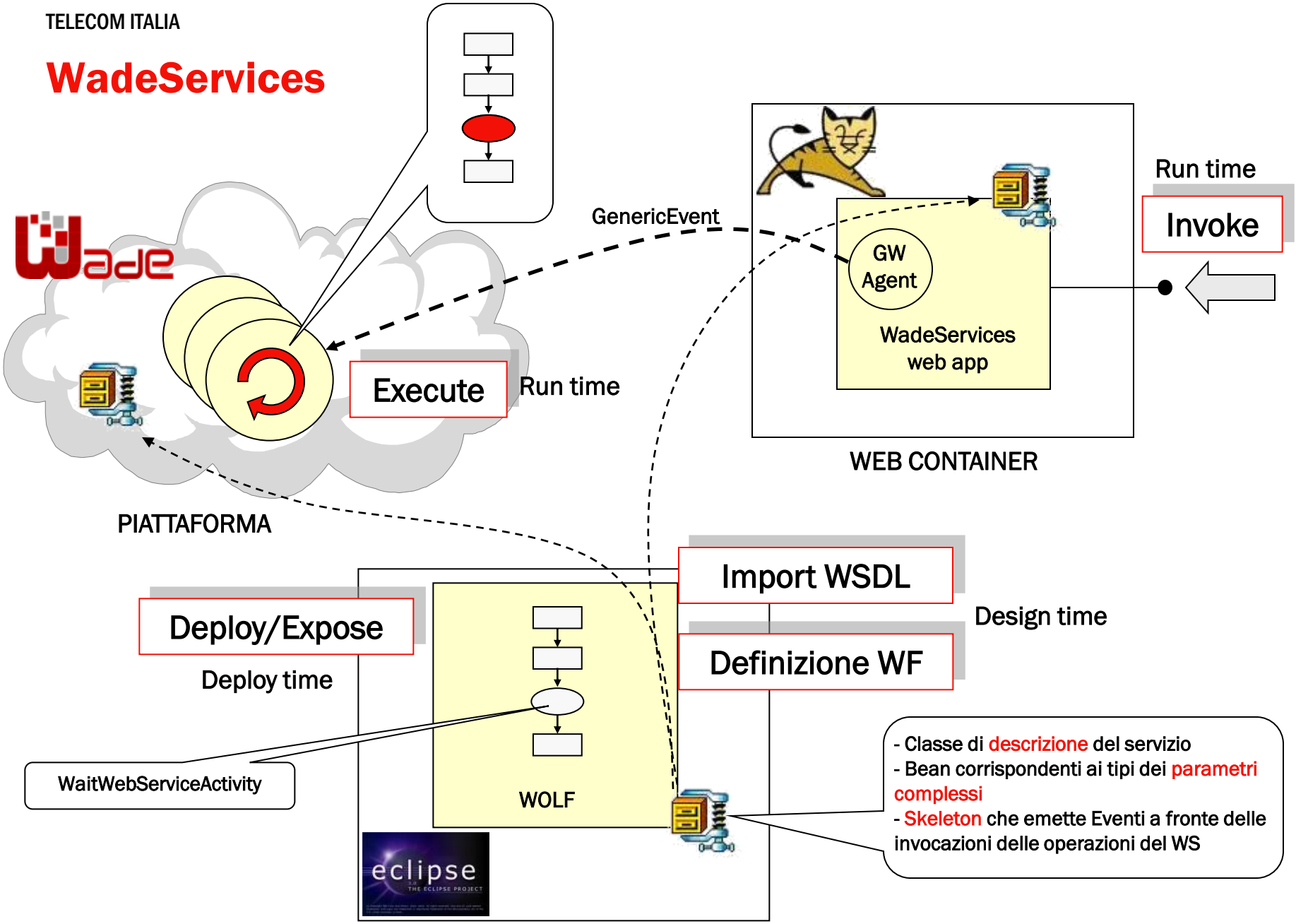
- Ogni volta che l'ESA riceve un evento confronta l'evento con le registrazioni di tutti i WF sospesi e risveglia tutti quelli la cui registrazione matcha con l'evento ricevuto
- Gli eventi hanno un Time-to-live per il quale vengono conservati dall'ESA
- Moduli di ricezione
  - Se ad esempio si vuole poter bloccare dei WF in attesa della ricezione di un SMS, è responsabilità dello sviluppatore creare il componente SW che riceve gli SMS e li immette nel sistema in forma di GenericEvent tramite l'EventChannel API
  - Nel caso di WebService WADE mette a disposizione un componente predefinito, WadeServices, che consente di esporre un servizio definito da un WSDL. Per ogni invocazione alle operazioni del WS, tale componente immette automaticamente un evento nel sistema.

Definizione di eventi custom

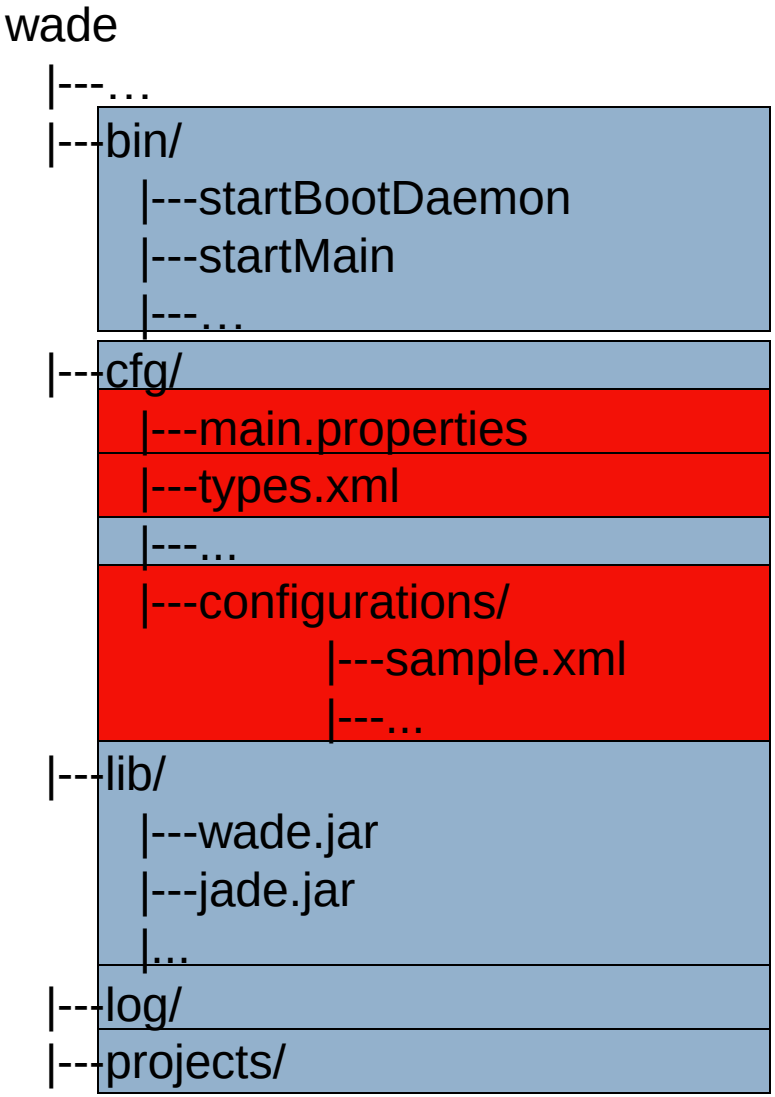
```
<platform>
  ...
  <agentTypes>
    ...
  </agentTypes>

  <customEventTypes>
    <EventType description="WorkRequestCompletata">
      <parameters>
        <Parameter name="wrId" type="java.lang.String"/>
        <Parameter name="esito"
              type="com.tilab.wfm.EsitoWR"/>
      </parameters>
    </EventType>
  </customEventTypes>
</platform>
```

WadeServices



WADE distribution structure



WADE Project

```
C/
|---wade/
|     |---bin/
|          |---startMain.bat
|          |---startBootDaemon.bat
|     |---cfg/
|     |---lib/
|          |---wade.jar
|          |---jade.jar
|          |---...
|     |---log/
|     |---projects/
|          |---...
|          |---booktrading.properties
```

WADE libraries only

```
project-home=C:\bookTrading
...
```

```
|---...
|---bookTrading/
      |---src/
      |---...
      |---classes/
      |---cfg/
           |---main.properties
           |---types.xml
           |---...
           |---configuration/
                |---bookTrading.xml
                |---...
      |---lib/
           |---bookTrading.jar
           |---...
      |---deploy/
           |---...
```

Book Trading project configurations

Book Trading project libraries

Book Trading project workflows

```
...
<agentType description="Configuration Agent" ...>
  <properties>
    <property name="configurationsPath"
        value="${project-home}/cfg/configuration"/>
  </properties>
</agentType>

<agentType description="Control Agent" ...>
  <properties>
    ...
    <property name="class-loader-root"
        value="${project-home}/deploy"/>
  </properties>
</agentType>
...
```

```
C:\wade>bin\startBootDaemon

C:\wade>bin\startMain booktrading
```

La Suite WADE

Plug-in di eclipse per la creazione di workflow e il setting di un progetto WADE-based

Wade Installer

Tool di installazione di applicazioni WADE-based

WOLF

Framework di componenti ZK con funzionalita` di amministrazione direttamente riutilizzabili in una web application (basata su ZK)

Consente di esporre WebServices generando eventi (sui cui possono bloccarsi I WF) in corrispondenza delle invocazioni .
In futuro consentira' anche di esporre WF come WebServices

Web Application per l'amministrazione di applicazioni WADE-based

Visualizzatore di interazioni controllate da workflow interattivi basato su ZK e Android

Interactivity ZK Front-End

Administration Console

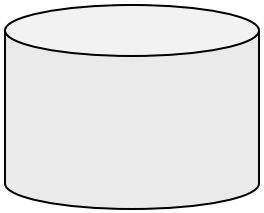
Wade Services

Interactivity Android Front-End

WAM
(WADE Administration)

Consente di persistere lo stato di esecuzione dei WF su un DB relazionale tramite Hibernate

WADE Persistence



Supporto

- **Wiki: http://nnem-tracall.cselt.it/trac/jade**
  - Download di tutte le ultime versioni delle componenti della WADE Suite
- **Mailing list wade-develop@avalon.tilab.com**
  - Per discussione on-line su problemi, indicazioni, commenti, suggerimenti, bachi
  - Iscrizione: https://avalon.tilab.com/mailman/listinfo/wade-develop
- **Supporto diretto**
  - Giovanni Caire
  - Elena Quarantotto
  - Giovanna Sacchi
- **Sito JADE/WADE**
  - http://jade.tilab.com (sezione Papers & Resources → Online Documentation)
  - http://jade.tilab.com/wade