Fluidtime

The architecture

Introduction

Increasingly, people live and work with a new set of habits regarding time, such as the increased use of the mobile phone to quickly schedule or change appointments. However, aside from the phone, few tools or services exist that support this new way of life, especially when people interact with public or private services.

Fluidtime [FT] supports flexible planning by providing people with personalised, accurate time-based information directly from the real-time databases of the services they are seeking.

Currently, individuals have limited access to timely information about public services or even private appointments, and are left wondering when their bus will arrive or whether their doctor is on time.

The Fluidtime team created 2 services and their interface prototypes to that show how using Fluidtime can be simple, effective and enjoyable.

The first of these services is aimed at public transport riders in Turin. While on the move, travellers can find dynamic information on mobile screen-based devices (a phone, a watch); while at home or at the office, people can find the same information on mechanical display units.

The other service is a personalized and flexible scheduling system to help Interaction-Ivrea students organize shared laundry facilities; mobile and stationary tools give them constant updates about the progress of their laundry cycle.

This documents describes the architecture of the Fluidtime system. Its design is based on the concepts described in [ARCH].

The architecture

Figure 1 describes the main component of the Fluidtime system:

- The red boxes on the left end-side represent the clients that connect through different channels (either in a push or pull form) to Fluidtime.
- The big white box in the middle represents the Fluidtime server itself, it is made by:
    - a component for each supported channel;
    - a DB, that stores the user and application information;
    - a rule engine, used to personalized the behavior of Fluitime, according with the time personality of the users;
    - a scheduler, that drives the push channels;
- The green boxes on the right end-side represent the content providers.

The Fluidtime system can be accessed through the following channels:

- **web PULL**, a web application is the default interface to interact with the fluidtime services;
- **SMS PUSH and PULL**, a protocol of human readable messages has been defined to held the interaction;
- **email PUSH**, a protocol of human readable messages has been defined to held the interaction;
- **HTTP PULL**, common HTTP GET are used to interact with fluidtime (require or subscribe services).

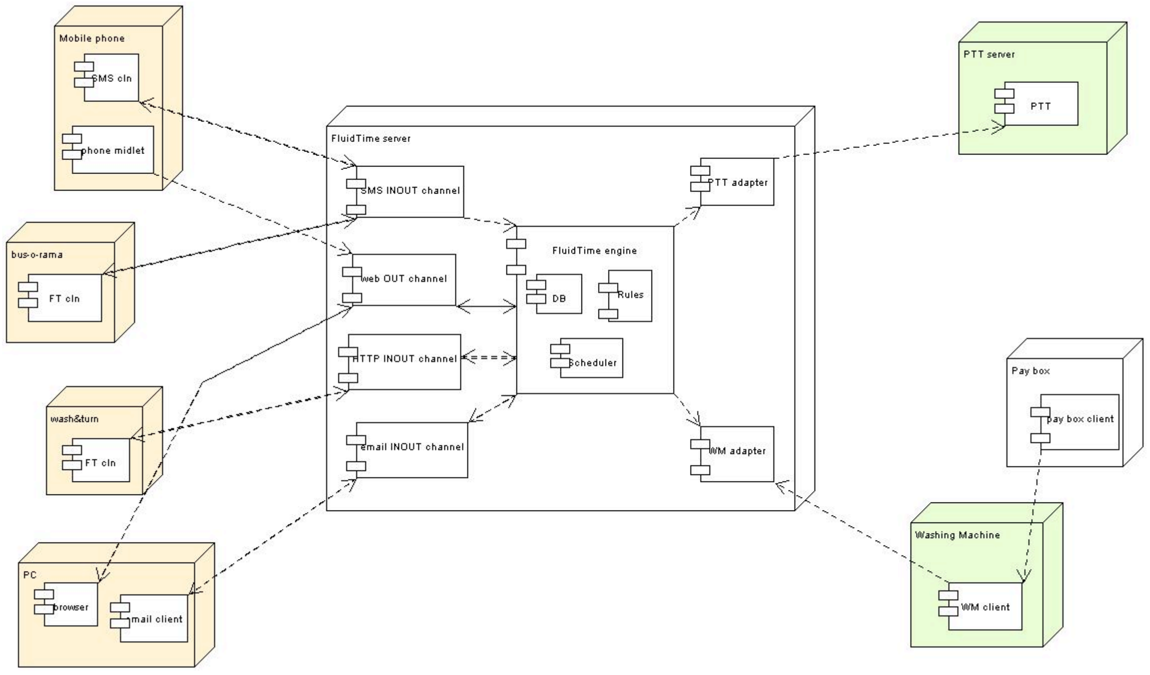

Figure 1: The main Fluidtime components.

The web channel is implemented by a series of web pages, while the other channels are java clients that connect through HTTP to the fluidtime server. In particular the system has the following clients:

- the **email client**, which polls the fluidtime mail account, parses the incoming mail and translates them in HTTP calls to the server;
- the **scheduler client**, which has a db of scheduled HTTP calls to the server, to have it send PUSH messages to the users;
- the **sms client,** which polls the fluidtime GSM modem, parses the incoming SMSes and translates them in HTTP calls to the server;
- the **washing machine client**, which is connected to the washing machine of the Laundry Service Ivrea and sends to the server the machine status updates.

Fluidtime can be accessed through the following devices:

- a mobile phone, through SMS push and pull or HTTP pull (via a midlet developed by the Fluidtime team);
- the busorama, through SMS push;
- the wash and turn, through SMS push;
- a common PC, through HTTP pull or email push.

Deep inside the architecture

Fluidtime architecture can be divided in tiers and layers (see Figure 2).

Layers

A **layer** contains components with similar concerns and allows to model the interaction between the application being developed and the technical platform (in our case java + the servlet engine) that hosts the application itself.

The lowest layer is the **Infrastructure layer**, which provides the underlying technical and communications capabilities. Generally, the functionality in this layer is purchased from a middleware vendor; frequently, the purchased middleware will be customized.

The **Services layer** contains common utility functions that are useful in multiple tiers and by multiple classes of applications. Services include capabilities such as XML parsing and persistence.

The **Application layer** is where application and business functionality is implemented and where we actually apply the roles and responsibilities of four architectural tiers.

Tiers

A **tier** is a collection of software components, with similar concerns; it's a logical distribution unit, that can be installed on a single node. Different tiers can be installed on the same node.

The underlying layers take care of the transport between the different tiers.

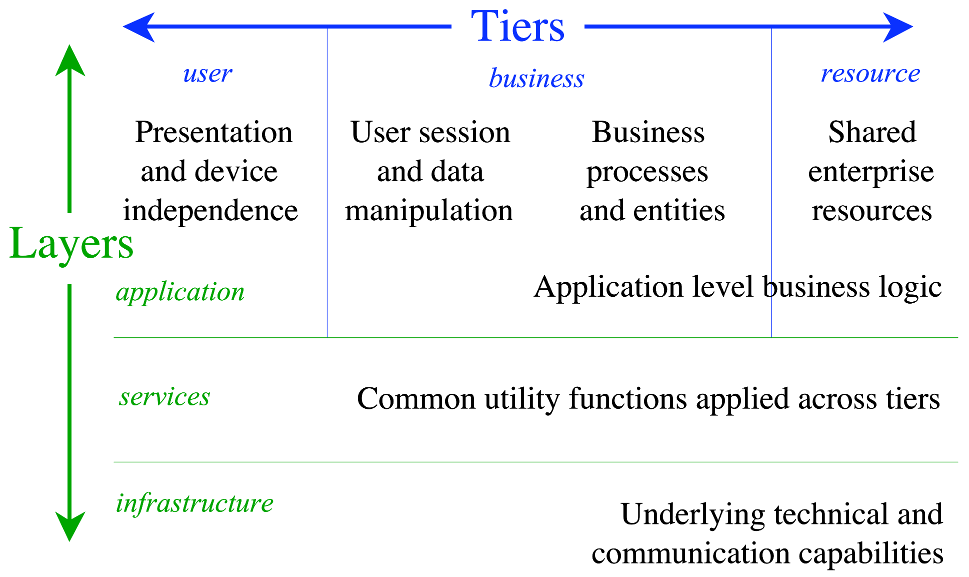
Tiers
user
business
resource
Presentation and device independence
User session and data manipulation
Business processes and entities
Shared enterprise resources
Layers
application
Application level business logic
services
Common utility functions applied across tiers
infrastructure
Underlying technical and communication capabilities

Figure 2: Layers and Tiers of the logical architecture.

The **User Tier** performs security authentication and authorization, establishes identity and does the initial message processing of the web pages or client calls.

The **Business Tier** is responsible for implementing business processes and entities and making their functions available via service-oriented interfaces. The Business Tier is responsible for providing business functionality and managing the integrity of enterprise resources. The business function in the Business Tier may itself be composed of finer-grain functions – in other words, it may be a business composition..

The **Resource Tier** is responsible for managing two basic types of resources; data and existing application services (information provider systems).

The architecture at work

Figure 3 is a UML model that describes how a call from a client or a web page is carried on by the Fluidtime system.

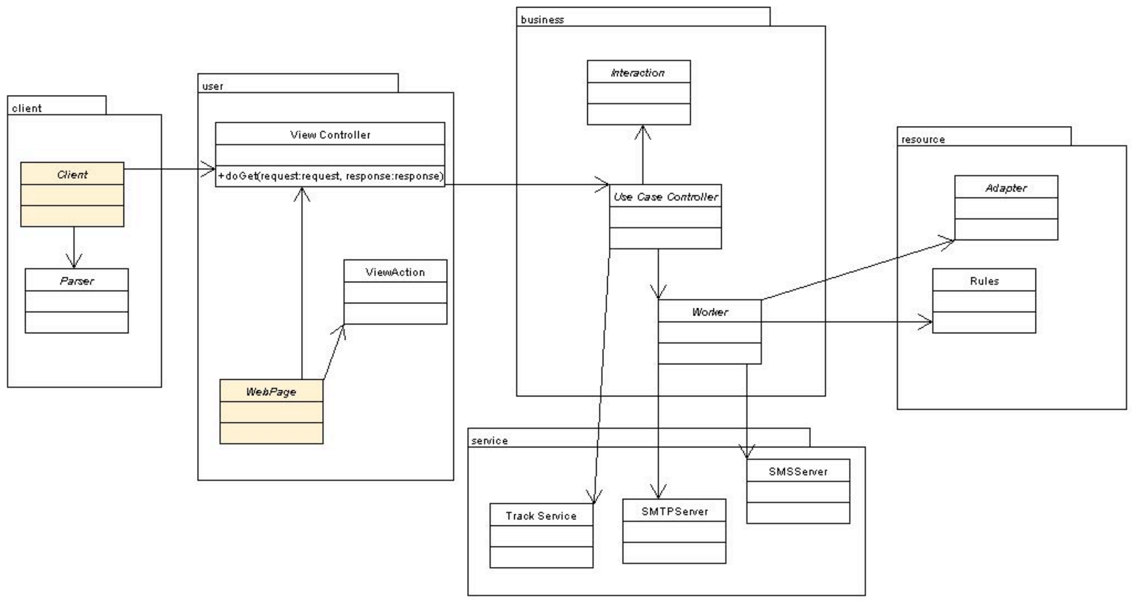

Figure 3: Fluidtime serving a request.

The interaction among the different tiers is as follows:

1. a PULL call comes from a Client or a WebPage to the ViewController;
2. through the "App" parameter, the ViewController forwards the call to the appropriate UseCaseController (either the FTController, the PTTController or the WMController); there's a UseCaseController for each service available (plus one for the general management of the system)
3. the proper UseCaseController has an Interaction class that resolves constant values and some Workers that perform the business logic. Different Worker can act at a different granularity of the business logic.
   Every interaction with the UseCaseController represents a service request, it is tough traced by the TrackingService.
4. Each Worker can interact with Services or Adapters to get or set information (eventually from an information provider), to use a channel different than the web one.

If the interaction is a PUSH, it is stimulated by one of the clients and is converted in a PUSH call to the system by the client itself.

The enabling technology

Fluidtime is a j2ee web application [J2EE]. It uses the Jakarta Tomcat [TMC] as a servlet engine and an XML file as persistent storage. It was developed on the Java JDK 1.4.0 [JDK], using the NetBeans IDE [NB].

The following open source libraries make up the service tier:

- log4j, to support the logging and tracing facilities [LOG4J];
- mailapi and smtp, to implement the mail channel [MAIL];
- mx4j-jmx, to integrate with the tomcat user DB [JMX];
- struts, as web application framework [STRUTS];
- javacomm, to connect to the serial port of the washing machine and GSM modem [JCOMM];

- xalan and xerces, to support the XML parsing [XML];
- javacc, to implement the SMS message parser [JAVACC];

References

[FT]	The Fluidtime website, http://www.fluidtime.net
[ARCH]	Paul Harmon, Michael Rosen, Michael Guttman - Developing E-Business Systems and Architectures: A Manager's Guide.
[JAVA]	Java Development Kit version 1.4.0, http://java.sun.com/j2se/1.4/index.html
[J2EE]	Designing Enterprise Applications with the J2EE Platform http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/index.html
[NB]	Netbeans java IDE, http://www.netbeans.org/
[TMC]	Tomcat servlet engine, http://jakarta.apache.org/tomcat/index.html
[XML]	Xalan XSL parser, http://xml.apache.org/xalan-j/index.html and Xerces XML parser, http://xml.apache.org/xerces-j/
[LOG4J]	Log4j framework, http://jakarta.apache.org/log4j
[MAIL]	javamail framework, http://java.sun.com/products/javamail
[JMX]	mx4j framework, http://mx4j.sourceforge.net
[STRUTS]	struts framework, http://jakarta.apache.org/struts/
[JCOMM]	javacomm framework, http://java.sun.com/products/javacomm
[JAVACC]	javacc parser generator, http://javacc.dev.java.net/