Base Architecture of Perspectives

Joop Ringelberg 23-05-20 Version: 1

Introduction

The purpose served by this text is to provide a clear view of

- the functional components that make up the Perspectives software stack;
- the connections between them
- the required and / or desired non-functional characteristics of these connections.

The suitability of concrete technologies for the Perspectives Stack can be judged by comparing them with these requirements.

This text is informal (and imprecise) in the sense that the protocols and message formats between components is not given in detail[1].

Components

The software stack is conventionally built from four high level components:

1. the Perspectives Runtime (PR);
2. a persistent store of information (Store);
3. a user interface (UI);
4. an authentication provider (AP).

Perspectives runtime

The runtime serves a number of functions:

1. It provides answers to requests sent in by (a single instance of) the User Interface. There is a limited number of requests that is accepted, answered in terms of role instances, context instances and property values.
2. It handles commands sent in by the UI. Again, the number of commands is very limited and allows the end user to add or remove role instances to contexts, to create and remove context instances, and to modify, add or remove property values. It also allows the user to *bind* some role to another.
3. When we say the *PR handles commands* we mean that it ensures that subsequent requests reflect the changes ordained before. In practice, this means it *stores* contexts, roles and property values (or rather *causes* them to be stored, see *Persistent store of information*).

[1] A precise definition is not yet available in text.

4. For each elementary change effected by a command, it constructs one or more Deltas. A Delta is like a remote procedure call. It contains enough information for another PR to reproduce the change. Deltas are collected in Transactions. Transactions are sent to other PR installations.
5. It handles incoming Transactions. After checking whether the sender is authorised to effect the change, each Delta is executed. This allows PR's to synchronise their stored contexts, roles and property values.

All these are governed by one or more *models* that authorise user roles to perform some actions (and that includes requesting contexts, roles and property values through the ui). Models are written in the Perspectives Language.

A PR is not expected to be active at all times. Assuming so would simplify some aspects of the architecture.

Persistent store of information

The Store can be thought of as a list of all elementary changes sent in by a single end user, including endorsed changes sent to it by other end users. In practice, it will contain an accurate accumulation of all those changes. It does not allow 'time travel', that is, one cannot ask it to revert to a previous state.

It is important to realise that each Store will be updated by exactly one PR, just as a PR will be used by exactly one end user.

We consider the structure of the persistent form of contexts and roles to be *internal*, even though they are exchanged between PR and Store. This is because the Purescript data definitions are leading. The serialised forms derive from those data structures. The type of serialisation deployed is an implementation choice. Consequently, we only stipulate that these persistent forms are JSON.

Concretely, with respect to the PR version 0.4.0, the store should

1. Provide the ability to create named compartments to store items in;
2. Be able to store and produce on request JSON documents in such a compartment;
3. Be able to provide a list of the documents in a compartment

There is a number of self-evident desired non-functionals, such as reliability, lots of capacity, etc.

Performance is important in the sense that the PR will very frequently request documents and update them. To some extent this requirement is relaxed because the PR *caches* each document it has requested. This cache is transparent with respect to updates.

User Interface

There are hardly requirements for the user interface other than in the most general of terms: it should enable a single end user to

1. Enter requests and inspect the answers sent by the PR;
2. Enter commands;
3. Identify him- or herself to the PR.

A UI will often be a *graphical* UI presenting screens and forms. As of version v0.4.0 of Perspectives, a JavaScript proxy library is available through which a process can interface with the PR. Also, built on top of that, a ReactJS library of data containers is available that provides higher level abstractions to deal with the PR.

Authentication provider

The end user authenticates him- or herself through the UI to the PR.

Currently, we have taken a very simple approach to authentication. The authentication provider is not a separate component. The required functionality is provided by the PR. User credentials are kept in the Store.

In version v0.4.0 the Store (being Couchdb) also doubles as a kind of post office for communication between PRs (see *Connections between components*, *PR – PR*). Accordingly, each Couchdb installation can be accessed by many PRs. Consequently authentication at the Store is important. Again, this is a consequence of the current implementation choices. One can easily envision an architecture where the Store is only accessed by a single PR and both are deployed on the same node, relaxing authentication requirements.

Connections between components

UI – PR

A single end user interacts with a single instance of the PR through a UI. The connection between them should be *confidential*. Transport of information between them should be fast enough that it does not stand in the way of a smooth user experience (this includes all aspects of transport, such as setting up a connection, applying measures to ensure confidentiality, etc).

The information items passing through the connection are usually quite small in terms of bytes when compared to current network bandwidth. The Deltas consist of alphanumeric information. Each Delta is the result of an end user action. There are no actions that lead to massive numbers of Deltas. Files can be handled as *claim data*: that is, the PDR is concerned with identities, not the actual items themselves.

The connection should also be reliable: whenever the end user fires up his UI, it should be able to connect to its PR.

Because of the nature of the UI (to enable an end user to access a PR) we assume that both components are active at the same time.

The connection should not only allow the UI to approach the PR; the PR should be able to initiate a contact, too. We need this to alert the end user to changes initiated by other end users.

PR – Store

A single PR interacts with a single Store (conceptually). We have not yet worked out an architecture where an end user deploys multiple devices. The simplest of architectures would be one where the Stores attached (through a PR) to UI's on multiple devices, synchronise between them. This, however, will not provide a limited user experience when the user *simultaneously uses multiple devices*.

So for the time being we assume the unique association between a PR and a Store. The connection between them should be

- Confidential
- Reliable
- Fast enough to handle the traffic resulting from several humans interacting through their UI (in the order of the number of relations a single person has).

Again, like with the connection between UI and PR, the required bandwidth is quite limited.

Because of the nature of the Store (to persist information shed by the PR) we expect it to be active at the same time as the Store.

PR – PR

PR's send Transactions to each other. However, as we do not require that PR is always available as an active process, the connection between them should handle this.

Consequently, this connection should have the characteristics of a mailbox.

We require the following non-functionals:

1. The connection should be reliable;
2. The connection should be confidential
3. The connection should be restricted to two PRs.
4. The connection should have a push-character: that is, the receiver should be notified after the sender has sent a Transaction.

5. The connection should be reasonably fast, ideally fast enough to allow for a chat-like experience (i.e. time delay introduced by the channel should be low enough to provide a good user experience). This, however, is not a hard requirement.
6. The connection should be able to handle the fact that end users will interact through mobile devices *and do move around*.

Current situation

As a matter of fact, in version v4.0.0 of Perspectives we assume that the connection between two PRs is formed by a shared store[2] – a *channel store*. This channel should offer the following functionality:

1. Both PRs can save Transactions in it (being JSON documents);
2. Each PR will be notified of changes in the channel.

Privacy, scalability and distribution

Perspectives was created as a distributed platform. One of the main functions of the PR is to ensure that each PR installation has exactly the information that the end user it serves, is entitled to, *but not more*. Thereby we ensure a form of privacy.

By eschewing central components, we also strive for scalability.

Obviously, whatever technology we consider to create UI's, Stores, Authentication Providers and the connections between them, the scalability and privacy we achieve with the PR should not be compromised.

Protocols and message formats that need to be published

The exchange of information between the various components is highly structured and, in the current phase of development of Perspectives, for the most part stable (small additions may be expected).

These exchanges are formalised in the form of separate modules in the Purescript source codes. An example is the `Perspectives.ApiTypes` module, that holds data types for, amongst others, types for requests between a UI and the PR.

However, we still need to publish these formats in text form. Similarly, the protocol for exchanging information needs to be formalised. Below we list the documents that have to be produced:

1. Perspectives Request Format (between UI and PR)
2. Perspectives Command Format (similar)

[2] This store has taken the form of a Couchdb database, as we also use Couchdb to persist information in. Couchdb has a built-in mechanism to synchronize duplicates of a database.

3. Perspectives Delta and Transaction Format (between PRs)