GT2003HackathonFlow

Raw notes taken collaboratively during the talk.

Carsten on Flow

Breaking up Flowscripts in smaller pieces

Explains how Flow basically works. His view of Flow is that you write a big javascript file for your application, which is not good.

Carsten would like to split the big javascript into several parts, smaller pieces ("flow units") of javascript which do their work and pass control to the next "Flow unit". Wouldn't need a different implementation of Flow, only a different way of doing things.

Aggregating several Flow applications in a single page

Is it possible to handle several Flow applications in coplets on a portal page?

According to Sylvain this is possible, each coplet can have its own continuation. The only problem currently is that continuations which are not used might expire. Making a distinction between "refresh" (used to refresh pages only) and "active" continuations might help. And in addition, if only one coplet "continues", the other coplets should not execute any script code and perhaps even not the pipeline.

Marc on continuations and Apples

Starts by saying that he's not going to talk about Apples (Yoko Ono wouldn't like it anyway 😉 Who did believe him?

Possible race conditions in continuations cleanup?

Currently in Cocoon there is the "web continuation" distinct from the "native continuation".

The native continuation holds all the stack information, where you're at in the currently executing script. Technically it is like an opaque container, handled by the ContinuationManager.

Every sendPageAndWait creates a new continuation - the forest of continuations is managed by the ContinuationManager. Destroying continuations is different from invalidating them.

ToDo: are there race conditions in continuation management? He's worried about complexity in the removal/cleanup of expired continuations. Today all continuations are handled in the scope of one session/one user, so potential problems might not have been exposed yet.

Marc would like to create a test case today, for possible race conditions. (sorry, to say the 'test' didn't happen. However we shouldn't panic on this: I really think it's a very rare case that would get us into the race conditions with current flowscript stuff and as Stefano was saying: if real time applications are not providing the tests that let the possible race conditions surface, then making the test probably only has academic value)

Apples

Original text:

Subtrees of continuation stored in Java objects

He's been experimenting with "subtrees" of continuations, where a subtree stores all intermediate states of your "use case". Stefano asks what exactly a "use case" is in this context. Storing a tree of continuations into a java object would allow REST-like access to every step of the use case (are we still following this? (I am getting lost here) 😉 (it's about whether you want magic managing your application state, or you would like to know what is really going on 😉

Update: (mpo) Well, reading the above I even don't understand it, so I guess I was quite succesfull at being completely misunderstood 🙂 The word 'use-case' indicates that the bigger goal behind things is to enable a 1-1 technical coding of the use case emerged from the analysis fase. So it really is the UML use-case as being `<< the written down set of pre-conditions/post-conditions and intermediate interaction-steps with external users and systems >>` but written down in a formal programming language so it can really be executed. I like this 1-1 mapping back to how analysts perceive the system to be broken down in pieces of atomic behaviour. IMHO the real goal of a system that declares itself as MVC would be that it allows more then anything else to cater for a (one) "controller" which is really just that (and thus the use case written down as an atomic executable piece of the system, in our world logically an avalon component)

Now where is the sub-tree confusion comming from? Well, basically I did a clumsy job at trying to explain what is different between apples and flowscript in respect to maintaining the actual 'state' you 'use-case' (or controller) is in. See: in both cases the controller is guiding the user through the different interactions that make up the use case... we could look at each interaction-moment as some kind of wait-point (waiting for the user to 'complete the form, folow a link, in general: 'to continue the use case') Now these 'wait-points' are actually to be seen as finite states of your controller IMHO. They comprise 'where you are in the flow, what has been entered up to know, intermediate results achieved,... Now the (only) (technical) difference between flowscript-interpreting and apple-processing is that the first is remembering all the different 'wait-points' and the path of how you got from the one to the other: you get a tree of continuations, each with their own unique id. The complete set of them is holding the knowledge of how the end-user stepped through the use-case. That led me to say: "The complete state of the use-case (or put different: the knowledge we have about the user stepped through all interactions) is inside the tree of continuations."

While in the case of apples we are remembering a lot less: only the last wait-point, or the current state. (comparing back to flowscript this is as if you would invalidate every continuation upon use, and since you only have one continuation at each time would decide to have them all use the same continuation-id rather then allocate new ones, which must sound horrific to flowscript peeps but really explains the difference on a technical level)

Too me, there is no magic either way 😉

Original text:

Marc thinks that, based on his analysis, some name changes in the continuations stuff might be of order.

Update: (mpo) actually the argumentation behind the proposed name changes is based on the contract between the sitemap and the flow. Putting up the hat of the URI-focus, and what the sitemap sees is two kinds of URI's: fixed ones, known upfront that point to a use case to start, and then the dynamic or temporary ones, carrying the continuation-id to point to the 'temporary resources' that store the wait-point info that allows to continue the use case.

So in fact may name change proposal indicates the sitemap should carry semantics which are in his URI-focussed realm. I consider the current ones as being 'biassed' by the interpreter-implementation. It's a somewhat purist way of pointing out possible 'mixing of concerns' Admittedly more at the conceptual and 'how we think about it' level then anything else.

Using URLs to access continuations which represent steps in a use-case (form-filling for example) could make it easier to:

- Fill several forms at the same time, for example when filling a form and getting a phone call which makes you start filling another instance of the same form.
- Sharing form filling between multiple users simultaneously (help line, etc) this is possible because that 'form' (actually every step of it IIUC) has it's own dynamic URL

Someone asks: Why? We'll let Marc fill you up on this one later 😉

Update: (mpo) The response to this really is asking it the other way around, so I jokedly replied 'why not' but honestly the return question is "Why would I save all wait-points if my app doesn't need that?"

Continuations management

According to Marc the only thing missing is the ability to gracefully end continuations: letting the user work on it if currently active, but do not allow the starting of new use cases.

Example: big deployment, need to let users finish their work before stopping a system.

In this state, new continuations could be created for "call continuation" but not for "call function" to prevent the starting of new use cases.

Should this be handled at the application or system level? It could be simple enough at the application level.

Update: (mpo) agreed, it's just a different aspect that gets added over and over again to multiple applications, so from that perspective there remains some stress to think about factoring the concern out so it becomes reusable across apps. This is however not an argument to just put in into the continuations-manager.

Concurrency

Today it is not possible to have two users work on the same continuation simultaneously.

The question seems to be more about sharing application data state than sharing continuations per se. We need to find a balance between making it easy yet not breaking concepts like continuations and sessions, which are not meant to be shared.

Flow Interpreter Interface:

```
    callFunction (func, args, environment)
    handleContinuation (id, args, environment)
```

These are all you need to implement to write your own Flow Engine.

Reinhard on interceptions in Flowscript

Scratchpad prototype

AOP/interceptions in Flowscript

Reinhard has written a prototype in the Cocoon scratchpad offering five interception points:

Event-driven approach

Has been implemented by Reinhard using an Observer pattern: his code registers with the interpreter to observe function calls.

- before
- after
- around (replaces a function by another)
- stop execution
- continue execution

Is static interception enough?

Use-cases

Reinhard sees the possibility of customizing apps without modifying them.

Authentication could be another use.

Wrapping Approach

Decorator Pattern - actually intercept function calls by delegating them to the original code after "pre-processing".

Stefano: Wrap a flowscript?

- Call my starting code
    - Excecute the wrapped FlowScript
- Call my ending code

Decorator vs. Observer pattern discussion.