混搭服務中使用者參與的偵錯機制

許復凱 [1] 蔡忠潔 [2] 梅興 [3]
輔仁大學資訊工程系 萬維運算研究室, 台北
Email: hsu.shaolin@gmail.com[1], jeffean@gmail.com[2], mei@csie.fju.edu.tw[3]

摘要

自從網路服務進入到 web2.0 的時代後，伴隨而來的混搭應用已經成為一種熱門的技術。在混搭環境下，網路服務可以自由的引用外部資源做各種應用，以提供給使用者或其他網路服務使用。這樣的架構雖然擁有許多便利性，但身為混搭服務的開發者卻很難確保外部資源的可靠度。本論文提出一個錯誤偵測與處理機制，藉由換手的方式以增加混搭站台的可靠度。

本論文嘗試在混搭環境之下換手，以增加混搭站台的可靠度。為了增加對於錯誤偵測的效率，我們也提出了利用使用者回報外部資源狀態的『使用者參與偵錯模式』，並模擬換手的過程來驗證提出的機制。

關鍵詞—Web2.0、混搭、換手、偵錯機制。

I. 緒論

A. 研究背景

自從 2005 年 O'Reilly 提出 Web2.0 的概念之後，使用者開始有開放參與貢獻的概念，越來越多的資源共享行為漸漸在網路上出現，新型態的網站服務不斷應運而生，相關的 Web2.0 技術也持續發展中。

混搭是 Web2.0 其中一項被廣為應用的概念，主要是透過 JavaScript/Flash/等技術的實做，將眾多使用者產生的內容(User Generated Content, UGC)方便聚合(Aggregation)成加值的內容，這些不同來源的程式資料混合成創新的應用，就是混搭。跟以往 Web Service 模式不同，這些資源都是透過外部網站開放的 API 存取使用，所有人都可以輕易的在網站中混搭，在眾多 Web2.0 網站中也能夠看到各式各樣的混搭運用。常見的網路龍頭如 Google, Yahoo!, Amazon, Microsoft 等也都提供了許多強大免費且易於使用的 API，在這樣的環境之下，更大量加速了混搭應用的發展與網路之間的活絡。從 mashup feed(http://www.mashupfeed.com/)統計的註冊資料來看，目前有上以千計的 API 供使用者使用，還有更多來自於世界各地的 API，其數量都足以顯示混搭產業的興盛[4][5][6]。

B. 研究動機

而因為混搭產業的興盛，在加上網路資訊爆炸，一個網站為了增加資料豐富度很難沒有摻雜混搭應用在其中。因此，網路服務和網路服務之間的相依性開始越來越大，要確保一個網站本身的可靠度反而更加困難。一個含有混搭應用的網站可能會因為外部資源的不穩定、或是實體網路出現問題，造成外部資源存取失敗。對於提供混搭服務的網站而言，這樣的情形輕微者導致該應用失去效用，但若沒有進一步妥善處理，甚至會出現網站癱瘓的狀況，這是網站必頇要盡力去避免的問題 [1]。

為了維護網站的可靠度，避免上述因為外部資源讀取不到造成的癱瘓，是目前在混搭環境中迫切需要解決的問題。以往提供混搭服務的網站不能完全解決這樣的問題，只能單方面相信外部資源本身的可靠度。因此，在目前外部資源提供者眾多的前提下，本論文想要藉由換手機制作為錯誤處理方式，以增加混搭網站的可靠度。

C. 研究目的

混搭環境下的換手機制就是當主要外部資源中斷服務時，有另外一個提供同樣服務的次要外部資源可以接手，例如有一個地圖服務發生錯誤，會由另外一個地圖服務接手。

由於混搭的定義非常鬆散，再加上目前混搭的應用非常多元與複雜，所以，要在混搭環境下換手的情形也必然複雜。過去一些文獻[2][3][7]對混搭建立許多模型，但仍無法完整套用在換手機制中。本論文將針對換手機制對混搭進行分類，並提出三種換手方式以符合不同的混搭應用。

另外，在混搭環境加入換手機制後，我們開始思考這樣的機制是否有效率。本論文發現在客戶端進行混搭的應用有一些效率上的問題，對於混搭服務的新使用者而言必頇要負擔很長一段的偵錯時間。於是本論文提出了『使用者參與偵錯模式』，藉由每個使用者回報外部資源的服務品質資訊，來判斷外部資源的狀態以提供適當的服務。

D. 論文架構

本論文的結構分為 6 章：接下來第 2 章為文獻探討，將介紹混搭的分類以及其對本論文的關係。第 3 章為本文核心混搭換手模式，介紹混搭換手機制中的各個模組，並對於混搭做分類以及其對應的換手流程。第 4 章探討在混搭環境下換手機制的效率問題，並提出一個使用者參與偵錯機制以改善問題。第 5 章為模擬，透過模擬混搭環境以驗證使用者參與偵錯機制的效果。第 6 章為結論與未來工作，提出本文之結論與未來相關研究之建議。

II. 背景研究

在過去有非常多領域提出換手機制，其中最類似於混搭環境的是分散式系統領域，因為混搭也擁有分散式的架構。然而，混搭環境卻比過去分散式系統更加複雜，一來

是網路環境的複雜、錯誤類型變多，發生錯誤的時機也多；二來是混搭環境下的外部資源太多變，難以建立模型，因此要在混搭環境下換手也變得困難[10][12][13][14]。

針對多變混搭環境，從過去一些文獻可以對混搭整理出以下分類[8][9]：呈現式混搭(Presentation mashup)、客戶端資料混搭(Client-side data mashup)、客戶端軟體混搭(Client-side software mashup)、伺服器端資料混搭(Server-side data mashup)、伺服器端軟體混搭(Server-side software mashup)。以下為這些分類的說明：

在混搭服務中，對於外部資源只有負責呈現而沒有進行整合的混搭稱為呈現式混搭，常見的呈現式混搭例如 Widgets/Gadgets/Chicklets、單純的 JavaScript、CSS 抑或是 HTML/XHTML。

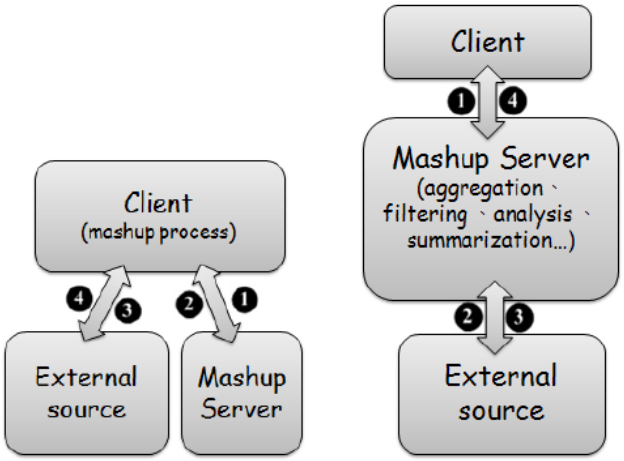

圖. 1. 伺服器端混搭與客戶端混搭

若混搭服務對於外部資源有做整合運算等動作，依照處理外部資源的位置可以再把混搭細分為客戶端混搭及伺服器端混搭。圖 1 左側是客戶端混搭的示意圖，混搭站台將混搭資訊傳給客戶端，由客戶端自行跟外部資源溝通及整合處理。圖 1 右側是伺服器端混搭的示意圖，由混搭站台負責處理外部資源的資料，再將結果呈現給客戶端。在本論文中的偵錯方式會根據這種分類而有所不同，其效果也會不同，在第四章中會有較詳細的比較。

除了依照處理外部資源的位置分類外，尚可以對外部資源的種類做分類，分成資料混搭和軟體混搭。資料混搭單純的利用外部資源的資料，例如 RSS、Atom、AJAX/XML、JSON、SOAP 這種形式的資料，之後的整合完全由混搭處理端(客戶端或伺服器端)處理。軟體混搭是利用外部資源的資料以及功能，這兩者差別在於與外部資源的互動性。這種分類對外部資源的存取方式不同，所以換手的流程也會不同，本論文將在 3-2 介紹不一樣的換手流程。

III. 混搭換手模式

本章節探討的是混搭換手的組成與流程。主要分成兩個部分，前半部會介紹在混搭換手中所需要的模組，以及每個模組的設計方針；後半段會根據不同的混搭應用分成三種換手流程，並對這三種流程做詳細的介紹。

A. 主要模組

混搭站台為了確保自身服務的可靠度，必頇提供一些時間及空間上的負擔。當外部資源出錯時，本論文提出換手模式作為解決方案，但這個換手模式仍有以下二點前提：

- 假設有一個提供相同或相似服務的外部資源(replica)可供換手之用。

根據研究背景的介紹，目前提供的外部資源數以千計，除非是非常稀有的服務，否則我們可以很輕易的完成這項假設。

- 換手模式中的各模組以及換手的對應程式將由混搭站台負責提供。對於替代的外部資源選擇，也由混搭站台所決定。

這是混搭站台確保本身服務可靠度所必頇要付出的負擔，站台必頇要設計符合自己服務需求的程式以對應不一樣的外部呼叫方式。由於服務是混搭站台所提供，也因此只有混搭站台能夠決定哪些外部資源是適合自己的替代外部資源。

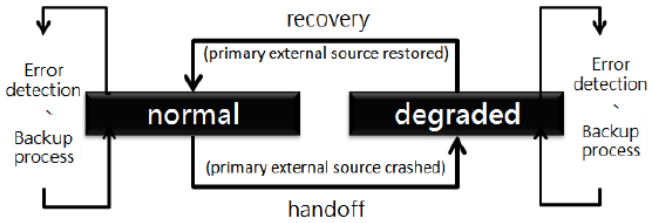

圖. 2. 基本換手模式的狀態轉換

接下來我們將介紹換手模式中的個個模組，並提供各模組下所必頇注意的開發方針以供混搭站台開發者參考。在此之前，必頇要了解換手模式的架構與流程。圖 2 顯示的是基本換手模式中兩個主要換手狀態，一般的換手模式都會有類似圖中的換手流程。正常狀態是指換手前的狀態，意即主要外部資源沒有任何問題，此時會有模組負責隨時的偵錯以及備份使用者使用服務的流程。備援狀態是指換手後的狀態，意即主要外部資源發生錯誤，正在使用替代的外部資源，此時亦會有模組負責偵錯以及備份使用者使用的服務流程。當兩種狀態交替的同時，會有模組將前面一個狀態下的使用流程記錄復原回去，以達到換手而不影響使用流程的目的。根據以上的架構，本論文將換手機制分成『資料儲存模組』、『錯誤偵測模組』、『換手決策模組』和『資料還原模組』四個模組，下面小節將有詳細描述。

1) 資料儲存模組

資料儲存的模組主要在於記錄使用者的使用流程，以方便資料還原模組能將這些流程復原到先前的狀態。而需要儲存的內容，將視混搭應用的內容而定，不同的應用所要儲存的方式及內容都不盡相同，但可以依照以下介紹的方式，以決定儲存的時機。

- 建立檢查點

在使用混搭服務的過程中，可能會多次存取外部資源，通常，資料儲存模組會在每一次存取外部服務的時間建立檢查點，每到一個檢查點，就會對上一個檢查點到這一個檢查點中間的流程做儲存的動作。例如：在使用混搭地圖服務，從一個座標點移動到下一個座標點；或是使用混搭微網誌的加值服務，使用者發表一篇微網誌的動作。這個檢查點，通常也是下面將介紹的錯誤偵測模組偵錯的時間點，定義在同一點的好處在於，當偵測到錯誤發生時，資料儲存模組儲存的內容會跟錯誤發生時一樣，以便完成迅速以及連續的換手。

- 流程記錄

本模組會記錄檢查點與檢查點之間的過程，該過程必頇要注意到過程發生的動作、執行動作的角色、動作發生的時間、是否有跟其他資源（外部、內部）互動、以及動作中產生的資料。例如：一個混搭地圖服務，使用者從一個點移動到下一個點，並觀看該地點的上另外一個外部資源提供的服務。這時需要記錄執行動作之角色為使用者，做了移動的動作，目前的座標位置（資料），移動發生的時間點，與另一個外部資源的位置資料。透過這些資料的儲存，我們能夠確保資料的順序性以及完整性，以達到平滑換手的目的。

2) 錯誤偵測模組

錯誤偵測模組的功用在於偵測目前使用的外部資源狀態。

本論文提出的錯誤偵測方式，是屬於被動式偵測，當有需要存取外部資源時才被動對外部資源的回應做判斷，這樣的作法至少能保證錯誤發生時能即時發現以讓接下來要介紹的換手決策模組做決定。本論文將在之後的第四章提出使用者參與模式，加強部分混搭應用下的偵錯效率。

在混搭環境下的錯誤可分成：『網路相關錯誤』、『物理相關錯誤』以及『邏輯相關錯誤』，錯誤偵測模組必頇要能判斷這些錯誤：

- 網路相關錯誤

最常發生的一種錯誤，是與網路連線有關的錯誤。例如：Communication Timeout, Service Unavailable (http 503), Bad Gateway (http 502), Server Error (http 500)。這些錯誤較常發生，但是最容易判斷。

- 物理相關錯誤

存取位置與實際位置不一致所發生的錯誤。例如：Page not found (http 404)。跟網路相關錯誤一樣，這種錯誤很容易判斷。

- 邏輯相關錯誤

邏輯錯誤是外部資源本身傳回錯誤的資料或錯誤的計算結果。這種錯誤比較難去偵測，通常對於這種錯誤的判斷是定義回傳格式，若不符合格式，則可能為錯誤發生。本論文沒有考慮這種錯誤方式，故之後提到的錯誤皆為網路相關錯誤以及物理相關錯誤。

錯誤偵測模組的偵錯技術會根據是伺服器端混搭或是客戶端混搭而不同。在伺服器端混搭的偵錯一般都是用開發服務的語言進行偵錯，例如：Java/J2EE、PHP、Python、C#/.NET、Ruby/RoR。這些語言對於錯誤處理的定義比較嚴謹，也較好控制錯誤發生後的後續處理動作。而客戶端混搭的偵錯都是利用 JavaScript/Flash 這些在客戶端執行的語言，早期這些語言比較少用於偵錯，隨著網路的發展，服務品質越來越被重視，在客戶端的偵錯也漸漸模組化，圖 3 就是 Facebook 和 Gmail 利用 JavaScript 偵錯產生出來的畫面，對於能判斷的錯誤類型也越來越精細。

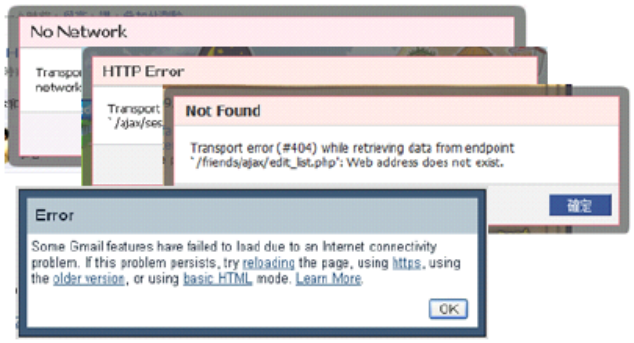

圖. 3. Facebook 和 Gmail 利用 JavaScript 偵錯

3) 換手決策模組

換手決策模組根據錯誤偵測模組判斷的外部資源狀態，而決定是否要換手。然而，對於某些提供較複雜服務的外部資源，比較難找到相同的替代外部資源，如果決定換手到相似的替代外部資源，可能會失去某些功能。例如在混搭地圖服務中，有些地圖供應者提供導航服務，有些則沒有；又例如混搭音樂播放的服務，一些既有的音樂歌曲評價會隨著換手而消失。在這樣的情況下，需不需要馬上換手就變成混搭站台需要好好考慮的問題，若換手為最不得已的選擇，可以在錯誤發生時導入重詴的方針，讓使用者重新對外部資源存取偵測。

4) 資料還原模組

資料還原模組與資料儲存模組是相對應的兩個模組，資料還原模組依照資料儲存的順序復原回去。這個模組需注意處理不同外部資源的呼叫方式，以順利將資料或是流程復原到換手前的狀態。

B. 換手流程

在上一節中介紹在換手模式下的各個模組，這些模組全部都由混搭站台根據混搭服務的種類而設計。這些被設計出來的換手模組會放置於混搭處理端，也就是若混搭服務為伺服器端混搭，則這些模組就會存在於伺服器端；反之，若混搭服務為客戶端混搭，則這些模組就會被送至客戶端執行。

本論文所提出的換手流程能確保使用者「使用混搭服務的過程」順暢，如果外部資源出錯，能夠即時換手以延續服務。「使用混搭服務的過程」之定義，係指一次從混搭站台讀取混搭應用，到結束該次混搭應用為止。即若混搭服務是由瀏覽器呈現，則這段過程為打開瀏覽器使用混搭服務，到離開或關閉瀏覽器中該混搭服務為止。在本論文中考慮的換手流程是從主要外部資源換手到替代外部資

源上，不考慮在使用混搭服務過程中換回主要外部資源。其原因有二，一是使用混搭服務的過程通常很短，在這樣短的時間內通常主要外部資源也很難回復錯誤狀態；二是使用替代外部資源已經可以延續服務，若不信任替代外部資源的可靠度，可以採用多個替代外部資源，以增加整體混搭服務的可靠度。

由於混搭應用非常多元，最初所提及的換手流程並不足以拿來應用到混搭環境下。在不同混搭服務中，對於外部資源的相依度都不盡相同，有些服務只是將外部資源當作一個短暫資訊，將資料整合結束後即不再用到外部資源；有些服務會與外部資源做互動；有些甚至會將資料存放在外部資源上。根據這個特性，本論文依照服務本身對於外部資源的相依度將換手流程區隔，分成『直接換手』、『短期儲存換手』和『長期儲存換手』。此三種換手方式最大的不同在於對於資料儲存的位置，直接換手不需要儲存，短期儲存換手將流程記錄在混搭處理端，長期儲存換手則將流程直接存在替代的外部資源上方便錯誤發生時能夠馬上換手。

在使用混搭服務過程中，會需要換手的狀況只有兩種，其一是剛開始使用混搭服務，偵錯模組發現錯誤進而換手；其二是在使用混搭服務中發現錯誤才換手。第一種狀況較為單純，因為尚未開始使用服務，所以並不需要記錄任何流程，根據本身使用的換手流程，直接進入到換手後的狀態。第二種在使用中才發生的錯誤，就必頇要有記錄流程的動作。底下三小節將更詳細的介紹不同的換手流程。

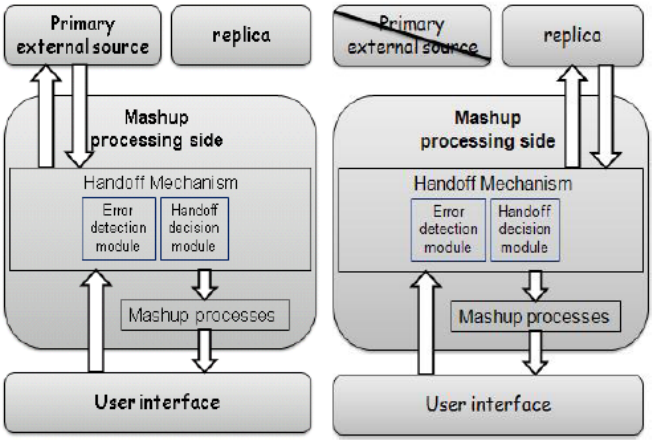

圖. 4. 直接換手-換手前與換手後

1) 直接換手

當混搭服務使用的外部資源，其存取內容是一次性且與下次存取的內容獨立不相關的時候，採用直接換手。這樣的混搭應用因為每次存取之間沒有關聯性，故沒有使用流程備份的問題，在直接換手的流程中，不需要有資料儲存模組，也不需要有資料還原模組。例如使用電影場次與天氣預報的混搭，這樣的應用只會存取一次外部資源，當取得外部資訊後經混搭處理呈現給使用者，若提供電影場次的外部資源發生錯誤，直接換手至另外一個提供電影場次的替代外部資源上即可。

直接換手的流程如圖 4 左側，在還沒有偵測到錯誤的時候，使用者向混搭處理端要求服務，混搭處理端接受到請求後會對主要外部資源存取，同時換手模式中的錯誤偵測模組會監測外部資源的狀態，並將結果交由換手決策模組判斷，若判斷不用換手則將混搭處理過的資訊回傳給使用者；若判斷為需要換手則直接換手到替代的外部資源上，如圖 4 右側。

2) 短期儲存換手

當混搭服務使用的外部資源，其存取內容為連續或有互動性，但互動的資料只是暫時需求的時候，採用短期儲存換手。這樣的混搭應用具有與外部資源互動的特色，必頇將流程記錄下來，在換手後才能延續互動，故在短期儲存換手的流程中，有直接換手所沒有的資料儲存模組以及資料還原模組。例如使用混搭地圖服務，這樣的服務通常會跟地圖供應者做密切的互動，而且中間的使用流程只是暫時的，下次再使用混搭地圖服務也不需要這些資料，所以必頇要記錄流程像是經緯度，若是地圖供應者發生錯誤，則換手到替代的地圖供應者後，需要將經緯度資訊回復。

短期儲存換手的流程如圖 5 左側，基本上運作模式與直接換手一樣，不同點在於短期儲存換手多了資料儲存模組以及資料回復模組，在換手前記錄使用流程，並儲存在混搭處理端。一旦錯誤偵測模組偵測到主要外部資源發生問題，換手決策模組決定要換手，資料回復模組會讀取替代外部資源，並將使用流程恢復至換手前的狀態。換手後的短期儲存換手如圖 5 右側，換手前後的差別只在於存取之外部資源不同。

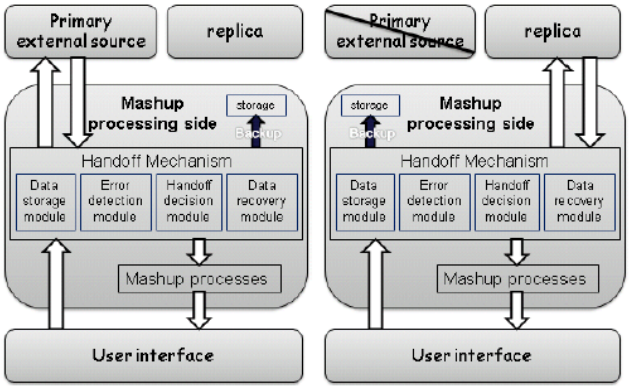

圖. 5. 短期儲存換手-換手前與換手後

3) 長期儲存換手

當混搭服務使用的外部資源，其存取內容為連續或有互動性，但互動的資料是長久性的時候，採用長期儲存換手。這樣的混搭應用與外部資源的相依性很高，甚至部分資料會永久存在於外部資源上。在這樣的環境之下，勢必要記錄所有的使用流程，然而這些流程通常重要性高且份量多，再加上回復這些資料的時間較久，所以長期儲存換手將利用像同步的方式將流程記錄直接放在替代的外部資源上。例如使用混搭微網誌的服務，這樣的服務會跟微網誌的提供者不斷存取資源，使用者的微網誌內容也會存在提供者那邊，若利用同步的方式將使用者每一篇微網誌內容也存到替代的微網誌提供者，則當主要微網誌提供者發

生錯誤，直接換手到替代微網誌提供者即可。

長期儲存換手的流程如圖 6 左側，與前面兩種換手最大的差異，是使用流程直接同步記錄在替代的外部資源上，當錯誤偵測模組偵測到錯誤發生，換手決策模組決定換手，則直接換手到替代的外部資源。另一點與前面兩種換手不同的是，長期儲存換手在換手後，混搭站台必頇要負起流程記錄的責任，將混搭處理端傳來的記錄儲存起來，如圖 6 右側，此時混搭站台的錯誤偵測系統需要對主要外部資源偵測，一旦偵測到主要外部資源回復，則將已記錄的流程回復到主要外部資源上。比對起換手後將使用流程儲存在混搭處理端，這樣做的用意在避免主要外部資源還沒回復，需要回復的使用流程就已經因為使用者離線而刪除，交托給混搭站台儲存不會隨著使用者離開而刪除，且會在主要外部資源回復時將換手後的使用流程復原回去。

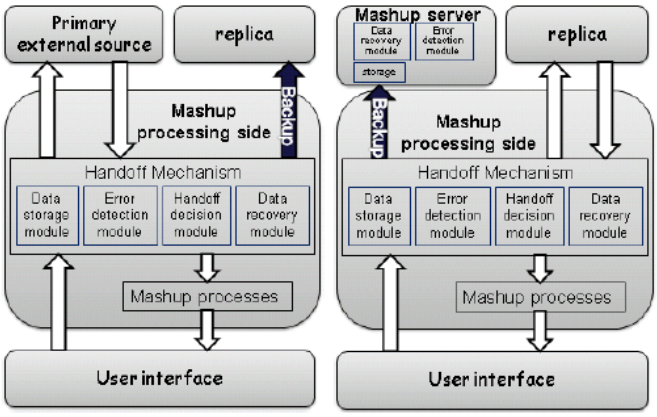

圖. 6. 長期儲存換手-換手前與換手後

表 1 混搭服務對外部資源的相依程度

Handoff type	The rate of handoff	Storage location	Recovery to replica	Recovery to primary external source
Direct	Fast	X	X	X
short-term	Slow (should recovery first)	Mashup processing side	From backup storage	X
long-term	Fast	Replica	sync	From backup storage (mashup server)

C. 小結

在本論文提出的混搭換手模式中，有『資料儲存模組』、『錯誤偵測模組』、『換手決策模組』以及『資料還原模組』四種模組，本章介紹了它們的功用與設計方針。此外，由於各種外部資源所提供的服務及混搭應用類型都相當不一致，導致換手的方式也不盡相同，為了方便建立模型，本論文依照混搭服務對外部資源的相依程度，分成『直接換手』、『短期儲存換手』和『長期儲存換手』三種換手方式。這三種方式的比較如表 1，比較了換手的速度、儲存的位置以及復原時取得的備份資料位置。

本章節介紹的換手模式能確保使用者在使用混搭服務期間，遇到外部資源錯誤能藉由換手到另外一個替代外部資源的方式延續服務，並在換手過程中，回復使用流程，以減少換手之間的差異。混搭站台的開發者可以依照本章提出的模組內容開發，並依據模型建立換手流程，增加自身混搭服務的可靠性。

IV. 使用者參與偵錯機制

前一章描述了混搭換手模式下的架構與流程。本章節將延續混搭環境下的換手模式，深入探討換手機制下的偵錯效率。

A. 換手機制中的偵錯

在第三章介紹的錯誤偵測模組中，會在每次讀取外部資源時進行偵錯，這樣的作法可以保證每次存取外部資源時的可靠度，即使外部資源發生錯誤，也能在第一時間發現並且錯誤處理。然而，在已知外部資源有問題的狀況下，新加入的混搭服務使用者仍然要等待一段偵錯時間，接著才換手，這樣的方式並不是一個有效率的方式。

為了解決這樣的問題，讓使用者不需要等待無意義的偵錯時間，我們依照錯誤偵測模組所在的位置分類成伺服器端混搭偵錯和客戶端混搭偵錯。在伺服器端混搭中，又可以按混搭應用的類型將對外部資源的存取分成預先存取和被動存取。預先存取外部資源是指混搭站台先收集外部資源並且整合好，當使用者存取混搭站台時，混搭站台將整合好的資訊傳給使用者，在這種情況下不會發生上述的問題，由於是預先存取，所以也很方便的可以預先做快取或預先換手。被動存取是當使用者有所請求，混搭站台才會對外部資源做存取，雖然仍是屬於被動，但偵錯方都是混搭站台，混搭站台可以根據之前的偵測結果預先決定是否要直接換手。回到客戶端偵錯，由於客戶端的混搭資訊、換手模組全都由混搭站台送出，送後不理，所以對於客戶端的服務品質，混搭站台不知道，也難以去避免上述的問題。因此，建立一個在客戶端混搭有效率的偵錯方式是本論文第二個目的。.

B. 使用者參與偵錯機制

在前一節中我們發現在客戶端的混搭應用中，因為執行環境是在難以掌控的客戶端，所以形成各自為政的偵錯換手環境。因此，在外部資源出問題的情況下，很多客戶端的偵錯行為都是耗時且多餘的。若能夠在混搭站台加入錯誤偵測模組及換手決策模組，透過頻率式主動偵測外部資源狀態，再根據狀態判斷是否需要在接下來的使用者請求中回傳換手過後的資訊，將可以省去部分客戶端偵錯換手的時間。

然而這樣的頻率式主動偵錯，並不是一個有效率的方式，可能還會有誤判的情形發生，因此本論文想藉由使用者回報外部資源狀態的方式來增加偵錯效率，此方式稱為使用者參與偵錯機制(user participation error detection mechanism, UPEDM)。

使用者參與偵錯機制的架構如圖 7。在客戶端的機制仍跟第三章所述一樣，在每次存取外部資源的時候偵測錯

誤，並交由決策者決定是否換手。此外，客戶端還多了一個狀態回報器，用以回報每一次的偵測結果，其回報內容描述了外部資源的網路品質狀態，包含 RTT(round trip time)以及是否正確讀取的訊息。而混搭站台也多了偵錯者，透過頻率式的偵測外部資源狀態，並將資料傳回資料收集器。資料收集器在整合客戶端以及混搭站台的資料後交由決策者判斷是否要做出換手的決定。

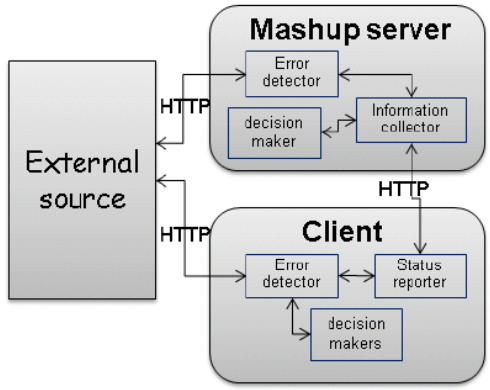

圖. 7. 使用者參與偵錯模型之架構

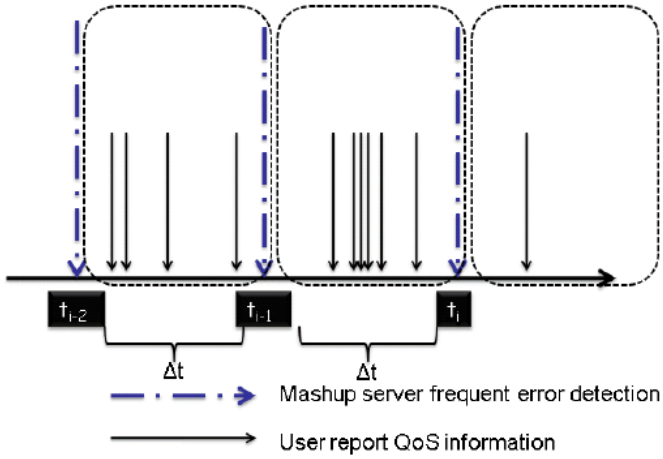

圖. 8. 在時間軸上顯示混搭站台和使用者偵錯的狀況

1) 判斷不穩定度

外部資源除了當掉不能提供服務外，還有一種狀態會影響混搭服務的服務品質，就是當外部資源忙碌的時候。在忙碌的狀態下，每位使用者使用到的服務品質將不盡相同。在使用者參與偵錯機制中，我們得到了來自使用者關於外部資源的服務品質狀態，這些狀態資訊可以做為判斷外部資源是否穩定的依據，為了避免外部資源因為忙碌造成的不穩定，本論文也在使用者參與偵錯機制中加入了外部資源不穩定度的判斷。

圖 8 是混搭站台與使用者回報的模擬狀態，可以看到混搭站台頻率式的偵錯以及使用者像亂數的回報。為了要算出外部資源的穩定度，我們先定義混搭服務可接受的 RTT (RTTacc)，以混搭站台的偵錯頻率當做基準(Δt)，計算在Δt 中所有回報高於 RTTacc 的比率，算出來的結果為混搭站台不能接受的錯誤比率(F)，即對於混搭站台而言外部資源的不穩定度。其計算公式為：

$$F(ti) = \frac{R_{CE}(ti)count + R(ti)}{R_{CE}(ti)count + R_{CS}(ti)count + 1} \times 100\%$$

- t_i , (i≥0): The i-th time mashup server frequent error detection.
- $\Delta t = t_i - t_{i-1}$:the time between two continue detection
- RTT_{acc}: Acceptable RTT of the server
- $R(t_i)$: the result of i-th server detection
  (0: result< RTT_{acc} 1:result≥ RTT_{acc})
- $R_{CS}(t_i)_{count}$: the sum of client return value < RTT_{acc} in $t_i \sim t_{i-1}$
- $R_{CE}(t_i)_{count}$: the sum of client return value ≥ RTT_{acc} in $t_i \sim t_{i-1}$
- $F(t_i)$: Mashup server unacceptable error rate in $t_i \sim t_{i-1}$

2) 使用者參與偵錯流程

根據以上的架構，混搭站台可以透過自己和使用者的回傳資訊，做偵測時機和外部資源狀態判斷 ，本節將介紹這種使用者參與偵錯機制的運作方式以及好處。

使用者參與偵錯機制在客戶端的部分，還是保有原來的換手機制，依然在使用者對外部資源存取的時候偵錯，並由程式決策是否要換手，只是在決策的同時將偵測到的訊息回報給伺服器端，此機制並沒有改變原先的換手架構。

表 2 伺服器端與客戶端於使用者參與偵錯機制下的動作

	Detection	Handoff to replica	Handoff to primary external source.
Client	• each time user access external source	• client detected an error	X
Mashup server	• frequently • user report an error message	• server detected an error • current F(ti) greater than acceptable value	• server detected the primary external source is back • current F(ti) less than acceptable value

而在伺服器端，錯誤偵測為頻率式偵測，混搭站台依照所使用外部資源的狀況定義頻率時間。當使用者回報錯誤訊息時也會被動的偵測以做確認，這種作法可以在外部資源可能發生錯誤的第一時間就收到訊息並且處理。伺服器端決定換手的時機有二，一是伺服器端偵測到外部資源出現問題，二是當下的外部資源不穩定度 F(ti)大於混搭站台的可接受值。當上述兩種狀況回復到一般狀態時，也就是外部資源恢復服務或不穩定度進入到可接受範圍，則換手回原始的外部資源。如表 2，比較了伺服器端與客戶端在使用者參與偵錯機制下的各種動作。

C. 小結

為了避免在客戶端混搭架構下，可能出現無效率的錯誤偵測等待時間，本論文提出了使用者參與偵錯機制，此機制的功用有兩點，一是讓混搭站台能在外部資源可能出錯的第一時間收到通知，以方便混搭站台做後續的處理動作，增加整體效率。二是能夠透過使用者回報的服務品質資訊，了解使用者使用外部服務的狀態，以作為是否要換手的依據。

在這樣的機制下，使用者失去了回傳服務品質的頻寬，伺服器多花效能在偵錯以及計算外部資源不穩定度，但是這些消耗都是微小的，相較之下，這個機制能夠在外部資源發生錯誤時，省去使用者偵錯的時間和效能，並且能夠在外部資源不穩定的狀態，讓混搭站台提供最穩定的服務，以確保其可靠度。

V. 模擬

本章節將模擬一個客戶端混搭的環境，其中包含一個外部資源、一個混搭站台以及多個使用者。混搭站台負責提供混搭服務給使用者，使用者再對外部資源做存取。本論文將模擬外部資源錯誤的發生，並導入使用者參與偵錯機制，以驗證其效果。

A. 模擬環境

參與模擬的機器有三台，分別為外部資源、混搭站台以及使用者。

外部資源提供內容讓人混搭，其服務內容由單一窗口(URL)存取，以降低模擬的複雜度。在外部資源中，尚有錯誤製造器，在每 10 分鐘內會製造連續 3 分鐘的忙碌狀態接著 3 分鐘的當機狀態，以模擬外部資源可能出現的狀態。

而一個使用者會去存取外部資源，計算存取時所耗費的 round trip time，並回報給混搭站台。為了增加使用者的擬真性，我們設定了每個使用者使用服務的總時間（10%：0 到 1 分鐘，10%：10 到 20 分鐘，80%：3 到 5 分鐘），以及在使用服務的過程中，多久會做一個動作以存取外部資源（30%：2 到 3 秒，50%：8 到 10 秒，20%：15 到 30 秒）。此情境有點類似於使用混搭地圖服務，大部分人使用總時間為 3 到 5 分鐘，每 2 到 10 秒會移動一次地圖以擷取圖資。另外，我們設定使用者瀏覽器的 timeout 為 10 秒，超過 10 秒讀取不到外部資源將視為讀取失敗。

混搭站台提供窗口讓使用者回報資訊，並且收集這些資訊。此外，混搭站台每一分鐘會去對外部資源偵錯，並依據使用者參與偵錯機制運作：在偵測到錯誤發生時，進入換手階段，讓接下來要使用混搭服務的使用者直接拿到換手後的資訊，存取替代的外部資源。

在這些前提之下，可以依照需求模擬出一定量不同行為的使用者去使用混搭服務，藉此模擬出使用者參與偵錯機制的效果。

B. 模擬結果

為了驗證使用者參與回報機制的效率，我們分別收集了三種不同狀態下混搭站台接收到使用者回報的趨勢，此三種狀態分別是：1. 最單純的混搭狀態 2. 使用者端加入了換手機制 3. 使用者參與偵錯機制。模擬是在 20 分鐘隨機安插 100 位不同的使用者使用混搭服務，而外部資源如前一節所介紹每 10 分鐘會有 3 分鐘忙碌 3 分鐘當掉，所得到在混搭站台收集的數據如圖 9、10、11。從圖中可以看到所有使用者回傳的 RTT，因為 timeout 設定為 10 秒，所以當外部資源是當掉的情況，使用者回傳的 RTT 會落在 10 秒附近。

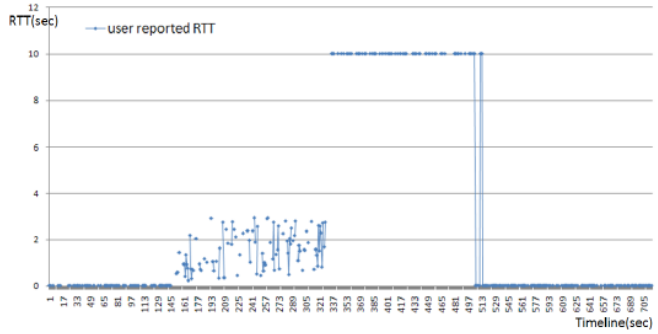

圖. 9. 模擬客戶端混搭環境中使用者回報的狀態(無換手)

圖 9 呈現出最基本混搭環境會發生的問題，使用者會隨著外部資源的狀態而得到不一樣的服務品質。若外部資源癱瘓，除了不斷重新嘗詴連線等待外部資源恢復服務外，並沒有保證混搭服務品質的機制在裡面。

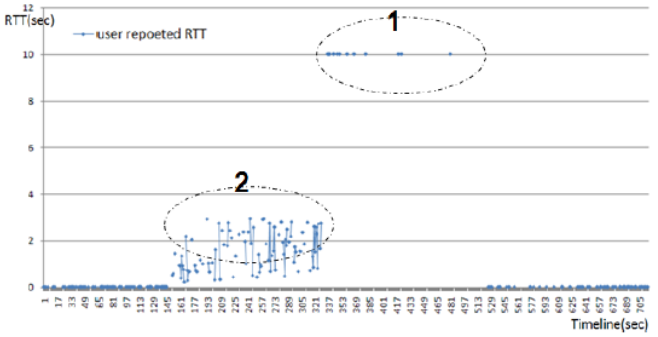

圖. 10. 模擬客戶端混搭環境中使用者回報的狀態(有換手)

圖 10 呈現的是在客戶端使用了換手機制，在外部資源發生錯誤時，可自行讀取替代外部資源的資料。其中在第 401 秒附近從密集的偵測失敗，漸漸變為分散的偵測失敗，這意味著密集回報的使用者因為偵測到錯誤已經在客戶端直接換手了，因此不再回報這個外部資源的訊息。接著使用者們從第 529 秒開始回傳服務正常的資訊，隨著使用外部資源的使用者越來越多，其回傳狀態的時間也變得越來越頻繁。

值得注意的是，在圖 10 中標記為 1 的範圍，是本論文欲解決的第一個問題，這些使用者在外部資源當掉的情況下，仍然需要等待 10 秒的時間偵錯失敗才能正確換手。透過使用者參與偵錯機制，在可能發生錯誤的第一時間點偵錯並換手處理，可以避免上述情形發生。圖 10 中標記為 2 的範圍，是外部資源不穩定的狀態，混搭站台為了自身的服務品質，希望在外部資源不穩定時換手，藉由使用者參與偵錯機制，可計算外部資源的不穩定度，以作為決定換手的依據。

接著，在相同狀況下，同樣在 20 分鐘內隨機安插 100 個不同的使用者使用混搭服務，但是導入使用者參與偵錯機制。模擬的數據如圖 11，在約 353 秒的時候，混搭站台因為使用者回報的錯誤而再去確認偵測，當確認外

部資源出問題時，混搭站台開始傳送換手過後的資訊給使用者，因此，再也沒有使用者因為不必要的偵錯而等待(353 秒~561 秒)，直到混搭站台再度偵測到外部資源回復的狀態，使用者陸續使用外部資源並回報狀態。此外，假設混搭站台定義的可接受 RTT(RTTacc)為 2 秒，其所同步計算的外部資源不穩定度如圖，此可接受數值可根據混搭站台的應用做調配，並在需要的時候決定換手。

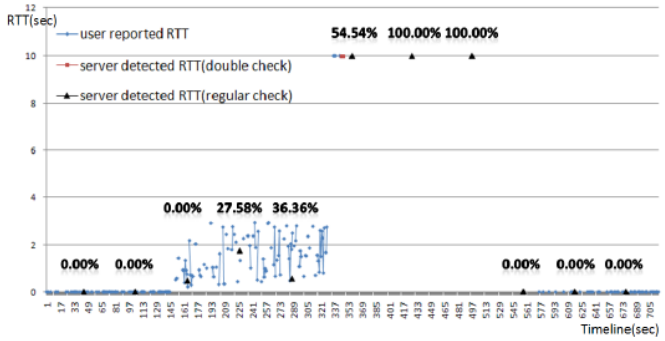

圖. 11. 導入使用者參與偵錯機制後混搭站台接收之數據

從模擬的過程中，可以呈現在加入使用者參與偵錯機制後，混搭站台能在外部資源錯誤發生之初偵測到錯誤，並且成功幫助了之後的使用者直接換手，節省這些使用者多餘的偵錯時間。另外，利用使用者參與偵錯機制所計算的外部資源不穩定度也能反應外部資源的狀態，讓混搭站台有更多的資訊能為本身的服務品質可靠度做把關。

VI. 結論及未來研究

A. *結論*

在混搭的環境中，由於外部資源的不可靠因素造成提供混搭服務的混搭站台不可靠，本論文提出在混搭環境下的換手機制改善現況。然而，由於混搭的定義鬆散，外部資源提供的服務及混搭應用的類型都相當不一致，這些不一樣的混搭服務會影響到換手的方式。為了方便建立模型，本論文依照混搭服務對外部資源的相依程度，將換手流程分為直接換手、短期儲存換手、長期儲存換手。這些換手模式都能確保使用者在使用流程中發生外部資源錯誤的情況下，藉由換到另外一個替代外部資源的方式延續服務，並在換手過程中，回復使用流程，減少換手之間的差異。混搭站台的開發者可以依照本論文提出的模組內容開發，並依據模型建立換手流程，增加自身混搭服務的可靠性。

接著本論文開始探討在混搭環境下換手機制的效率問題，我們發現在客戶端的混搭應用因為執行環境是在難以掌控的客戶端，所以形成各自為政的偵錯換手環境，若能夠將這些偵錯資訊統整，統一提醒，將可以省去部分客戶端偵錯換手的時間。本論文提出的使用者參與偵錯機制即是以這樣的概念所設計，該機制除了節省客戶端偵錯的時間，其所計算的外部資源不穩定度也可以供混搭站台做應用，在外部資源狀態不穩定時換手，確保混搭服務的品質。

在混搭應用越來越普及的 web2.0 時代，本論文站在混搭服務提供者的角度，將混搭分類並建立換手模型，以建立服務本身可靠度；稍後提出的使用者參與偵錯機制，也幫助換手機制下的偵錯能更有效率，並且能確保混搭服務的品質。期望在之後使用者都能夠擁有一個可靠的混搭環境。

B. *未來研究*

本論文提出了在混搭環境下的換手機制，在這個機制下目前只考慮到主要外部資源的錯誤，並沒有考慮到替代外部資源的狀態，意即我們無法保證換手過去的服務是可靠的。在本論文中所提及的解決方案是採用多個替代外部資源。雖然同時發生主要外部資源當機，替代外部資源也當機的機率較低，但仍有這種可能發生，未來我們將考慮到替代外部資源的狀態，建立一個有效率的模式，在眾多替代資源當中選取一個最可靠的服務。

在使用者參與偵錯機制中，混搭站台藉由使用者回傳的服務品質資訊，提高偵錯的效率，並增加混搭服務品質。然而這樣的機制不能防止使用者有誤或惡意的回報，雖然有再次判斷的驗證機制，但若這類的訊息多了會增加伺服器的負擔以及計算不穩定度的準確性。這是機制需要改進的地方。

此外，我們要對錯誤做進一步的分類與實驗分析，讓本論文中的使用者參與偵錯機制能夠處理不同的錯誤，並比較這些錯誤對於伺服器端混搭與客戶端混搭的差異，以建立更好的換手模式。

VII. 參考文獻

[1] E.M. Maximilien and S. Tai, “Mashups'07: First International Workshop on Web APIs and Services Mashups,” Service-Oriented Computing - ICSOC 2007 Workshops: ICSOC 2007, International Workshops, Vienna, Austria, September 17, 2007, Revised Selected Papers, Springer-Verlag, 2009, pp. 1-2.

[2] J. Warner and S.A. Chun, “A citizen privacy protection model for e-government mashup services,” Proceedings of the 2008 international conference on Digital government research, Montreal, Canada: Digital Government Society of North America, 2008, pp. 188-196.

[3] E. Zahoor, O. Perrin, and C. Godart, “Mashup Model and Verification using Mashup Processing Network,” The 4th International Conference on Collaborative Computing: Networking, Applications and Worksharing - CollaborateCom 2008 .

[4] J. Tatemura, A. Sawires, O. Po, S. Chen, K.S. Candan, D. Agrawal, 及 M. Goveas, “Mashup Feeds: continuous queries over web services,” Proceedings of the 2007 ACM SIGMOD international conference on Management of data, Beijing, China: ACM, 2007, pp. 1128-1130.

[5] J. Wong and J. Hong, “What do we "mashup" when we make mashups?,” Proceedings of the 4th international workshop on End-user software engineering, Leipzig, Germany: ACM, 2008, pp. 35-39.

[6] G. Wang, S. Yang, and Y. Han, “Mashroom: end-user mashup programming using nested tables,” Proceedings of the 18th international conference on World wide web, Madrid, Spain: ACM, 2009, pp. 861-870.

[7] Grammel, L. and Storey, M.-A., "An End User Perspective on Mashup Makers," Technical Report DCS-324-IR, Department of Computer Science, University of Victoria, September 2008.

[8] J.J. Hanson, Mashups: Strategies for the Modern Enterprise, Addison-Wesley Professional, 2009.

[9] “Is IBM making enterprise mashups respectable? | Enterprise Web 2.0 | ZDNet.com.”, http://blogs.zdnet.com/Hinchcliffe/?p=49

[10] M. Treaster, “A Survey of Fault-Tolerance and Fault-Recovery Techniques in Parallel Systems,” cs/0501002, Dec. 2004.

[11] E.N.(. Elnozahy, L. Alvisi, Y. Wang, and D.B. Johnson, “A survey of rollback-recovery protocols in message-passing systems,” ACM Comput. Surv., vol. 34, 2002, pp. 375-408.

[12] F. Cristian, “Understanding fault-tolerant distributed systems,” Commun. ACM, vol. 34, 1991, pp. 56-78.

[13] M. Larrea, “Understanding perfect failure detectors,” Proceedings of the twenty-first annual symposium on Principles of distributed computing, Monterey, California: ACM, 2002, pp. 257-257.

[14] T.D. Chandra and S. Toueg, “Unreliable failure detectors for reliable distributed systems,” J. ACM, vol. 43, 1996, pp. 225-267.