

國科會自由軟體專案研究計劃系統設計報告  
*System Design Document of NSC Open Source Project*

一個 *Java GUI* 測試工具的設計與製作

NSC 92-2218-E-027-017

陳偉凱 副教授

國立台北科技大學資訊工程研究所

Department of Engineering and Applied Science

National Science Council, Taiwan

2004/7/15

# 目 錄

---

1.	系統模型與架構(System Model/System Architecture)	2
1.1.	系統功能與介面(System Requirement and Interfaces)	3
1.2.	系統解決方案限制(Establish Technical Solution Criteria)	5
1.3.	其他可行方案比較(Describe Alternative Solutions)	6
1.4.	選擇可行方案(Select System Solution)	7
1.5.	介面需求與設計(Interface Requirement and Design)	8
2.	系統與軟體架構(System/Software Architecture)	10
2.1.	使用案例分析 (Use Case Analysis)	10
2.2.	使用者介面分析 (User Interface Analysis)	17
2.3.	類別圖分析(Class Diagram Analysis)	28
2.4.	循序圖分析(Sequential Diagram Analysis)	37
3.	設計方式(Design Issues and Solutions)	39
3.1.	系統特性(Subsystem Characteristics)	39
3.2.	建立技術解決方案選取條件(Establish Technical Solution Criteria)	39
3.3.	選取技術方案(Selected Subsystem Solution)	39
3.4.	偵錯與復原(Error Detection and Recovery)	39
4.	系統細節與介面描述(Detailed of System and Interface Description)	41
4.1.	細部設計(Detailed Design)	41
4.2.	系統介面與設計(System Interface Requirement and Design)	49
4.2.1.	內部介面	49
4.2.2.	外部介面	54
5.	詞彙(Glossary)	57
6.	參考文獻(Reference)	60
附錄一	類別列表(Class List)	62
附錄二	系統模組與類別回朔表	68
附錄三	系統模組與需求回朔表	76

# 1. 系統模型與架構(System Model/System Architecture)

本工具的概觀架構如下圖，GTT 介於 User 與 Java Swing-based Application(簡稱 AP)之間，負責錄影(截取)、編輯、播放(執行)、測試的功能。平時我們使用 AP，與 AP 進行互動時，絕大多數的輸入界面是滑鼠或鍵盤，輸入界面所發出的訊息，會藉由 JVM(Java Virtual Machine)將訊息轉為事件送達 AP。在 GTT 的架構底下，我們提供了一個方法，在不影響使用者與應用程式執行狀態的前提下，將原本 JVM 直接送達 AP 的 Java 輸入事件(Java Input Event，簡稱 JIE)，以有系統的方式截取下來，並且將截取出事件資訊轉換為比較高階的表達方式—抽象化的輸入事件資訊(Abstract Input Event Information，簡稱 AE 資訊)呈現在我們的工具之上。我們可以再利用截取下的 AE 資訊，加以編輯、修改、製作測試流程、儲存或讀取。

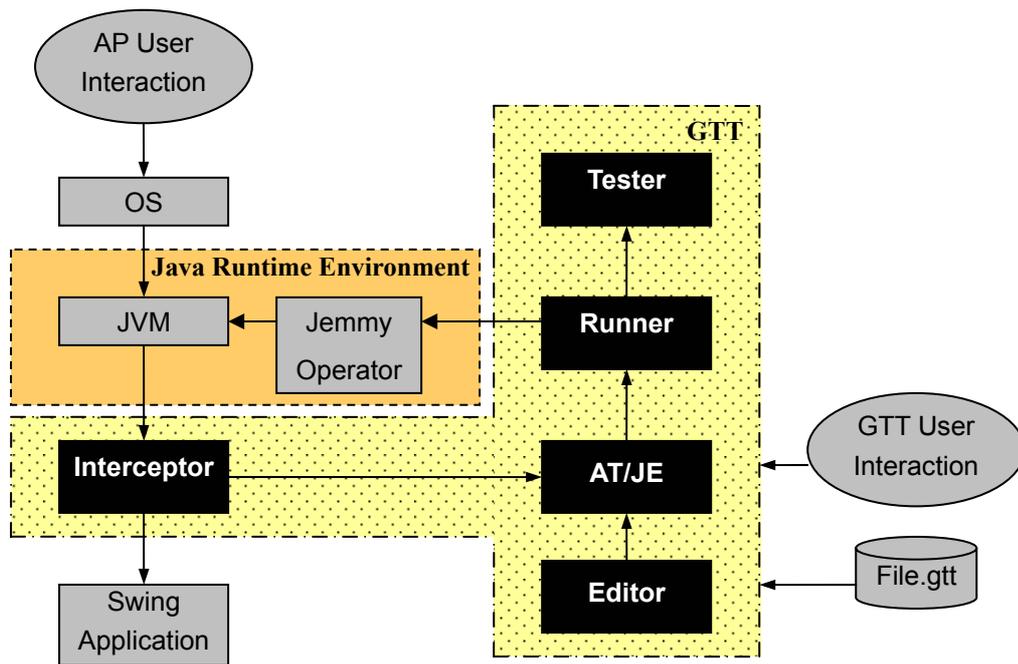


圖 1.1 GTT 模組架構圖

GTT 共分為五個模組：攔截模組(Interceptor)、事件與測試模組(AT/AE)、編輯模組(Editor)、執行模組(Runner)、測試模組(Tester)。這五個模組各司其職，是架構出此工具的關鍵。各模組的功能與特性如下：

- **攔截模組(Interceptor)**：負責使用者輸入動作的截取，以及事件資訊的轉換。
- **事件與測試模組(AT/AE)**：為 GTT 工具存取 AE 資訊與 AT 資訊的核心，存有 AE 資訊與 AT 資訊兩個抽象資訊，是一個樹狀結構，是編輯時資料的處理對象，與播放時測試資料的來源。
- **編輯資訊模組(Editor)**：主要負責呈現 AE 與 AT 等測試資訊的流程架構，以及流程編輯的功能。
- **執行模組(Runner)**：主要負責找尋 GUI 元件、AE 資訊的轉換、執行時間控制、執行時的錯誤回報。
- **測試模組(Tester)**：主要負責「測試的執行」，若有測試執行後產生的錯誤，需將錯誤資訊適時的回報。

### 1.1. 系統功能與介面(System Requirement and Interfaces)

GTT 為一個 GUI 應用程式的測試工具，其中包含了一些測試的方法及觀念，詳細的功能列表如下：

功能編號	優先順序	功能描述
GTT001	1	可在 GTT 的操作環境下，執行以 Java swing 為圖形介面基礎的待測應用程式
GTT002	1	提供自動的擷取(錄製)使用者對於 GUI 操作的動作流程。
GTT003	1	提供自動的播放使用者先前錄製之 GUI 操作流程的機制。
GTT004	1	將滑鼠事件做高階的抽象化(例如將滑鼠「按下按鍵」與「放開按鍵」的事件轉為一個「Click」的事件稱之為高階抽象化)
GTT005	1	將鍵盤資訊做高階抽象化(例如將鍵盤「按下按鍵」與「放開按鍵」的事件轉為輸入一個字元的事件稱之為高階抽象化)
GTT006	1	將抽象化的資訊事件以簡單易懂的樹狀圖表現。
GTT007	1	抽象化事件之樹狀圖可以增加一個新的事件

GTT008	1	抽象化事件之樹狀圖可以刪除一個舊有的事件
GTT009	1	抽象化事件之樹狀圖中已有的事件可以編輯修改
GTT010	1	抽象化事件之樹狀圖可以隨意增加一個新的「測試節點」
GTT011	1	抽象化事件之樹狀圖可以刪除一個舊有的「測試節點」
GTT012	1	抽象化事件之樹狀圖中已有的「測試節點」可以編輯修改
GTT013	1	提供可讓 GTT 使用者檢視抽象化事件的細部內容(細部內容指低階的動作，例如滑鼠之 Click 中就包含了「按下按鍵」與「放開按鍵」的細部動作)。
GTT014	1	可對抽象化事件的細部內容新增一個低階的事件。
GTT015	1	可對抽象化事件的細部內容一個低階的事件作修改。
GTT016	1	可對抽象化事件的細部內容刪除一個低階的事件。
GTT017	1	可儲存整個測試事件的流程
GTT018	1	可開啟一個新的(空白並只包含樹狀流程根節點的)測試事件流程
GTT019	1	可開啟過去所儲存的測試事件流程
GTT020	1	提供工具的延伸點，讓使用者可將自行以 JUnit 格式寫作的外部測試檔案(Class 檔)加入測試中。
GTT021	1	以高階抽象化截取時，GTT 必須將擷取之事件轉換為 Jemmy API 所表達之事件，讓抽象化層次再提昇
GTT022	1	GTT 使用 Jemmy API 播放所表達之高階抽象化事件。
GTT023	2	當待測程式發生位移時(相對於整個桌面之位移)，先前的測試流程必須能夠保持運作。
GTT024	2	進一步的改善程式架構
GTT025	2	操作介面的改善
GTT026	2	使用方式的改善
GTT027	2	執行更穩定(除錯)

GTT028	2	提供更多的執行範例
GTT029	2	完整的英文說明

另外，針對提出來的系統架構，我們定義了內部介面與外部介面兩個部份，

列表如下：

- 內部介面
  - GTT 必須將擷取之事件轉換為 Jemmy API 所表達之事件。
  - GTT 使用 Jemmy API 播放所表達之事件。
- 外部介面
  - 以 Java swing API v1.4 來呈現 GTT 的使用者介面。
  - GTT 使用 JDK v1.4 的檔案 API 儲存或載入測試事件流程的檔案。
  - GTT 能讀取 JUnit 之測試案例檔案作為外部測試節點。

待測應用程式必須有程式起始點之函式(預設值為 main 函式)，作為執行待測應用程式之起點。

## 1.2. 系統解決方案限制(Establish Technical Solution Criteria)

在這個小節中，我們列出幾項關於 GTT 設計架構上的技術解決方案：

- Ease of Development  
程式撰寫與開發的難易度
- Ease of Editing  
編輯錄影事件流程的難易度
- Ease of Testing  
測試節點撰寫的難易度
- Extensibility  
程式的擴充性
- Maintenance  
錄影事件記錄檔的維護

### 1.3. 其他可行方案比較(Describe Alternative Solutions)

系統的設計架構部份，我們主要針對截取訊息視窗元件的訊息來加以分析，提出了兩個可行的設計架構來加以介紹。

- 從 OS 層截取訊息

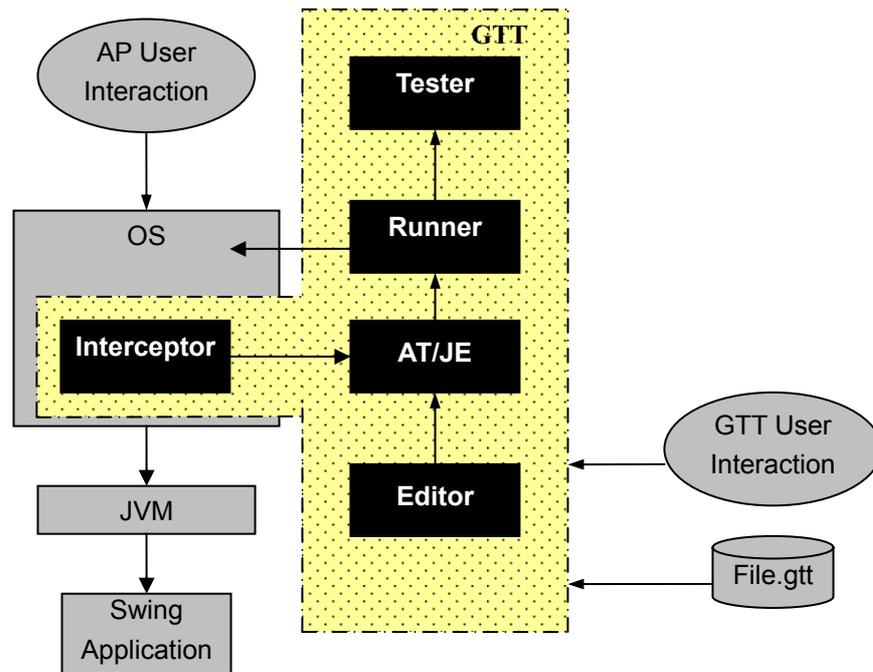


圖 1.2 經由 OS 截取事件訊息架構圖

程式由 OS 層來截取所有的視窗元件訊息，如圖 1.2 所示，再將這些訊息加以分析處理。這樣的做法，必須改寫作業系統中一些相關的 DLL 檔案，經由這些 DLL 檔來輔力我們取得所有事件。好處是待測程式可以是由任何語言所撰寫出來的程式，但是壞處則是改寫相關的 DLL 檔非常不易。另外，對於事件流程方面的處理，事件流程的記錄檔不易維護，而且所有的測試流程中記錄的資料最好與 Layout 是無關的，但透過 OS 來截取事件訊息，無法得到程式視窗元件的資訊。

- 經由 JVM 來截取訊息

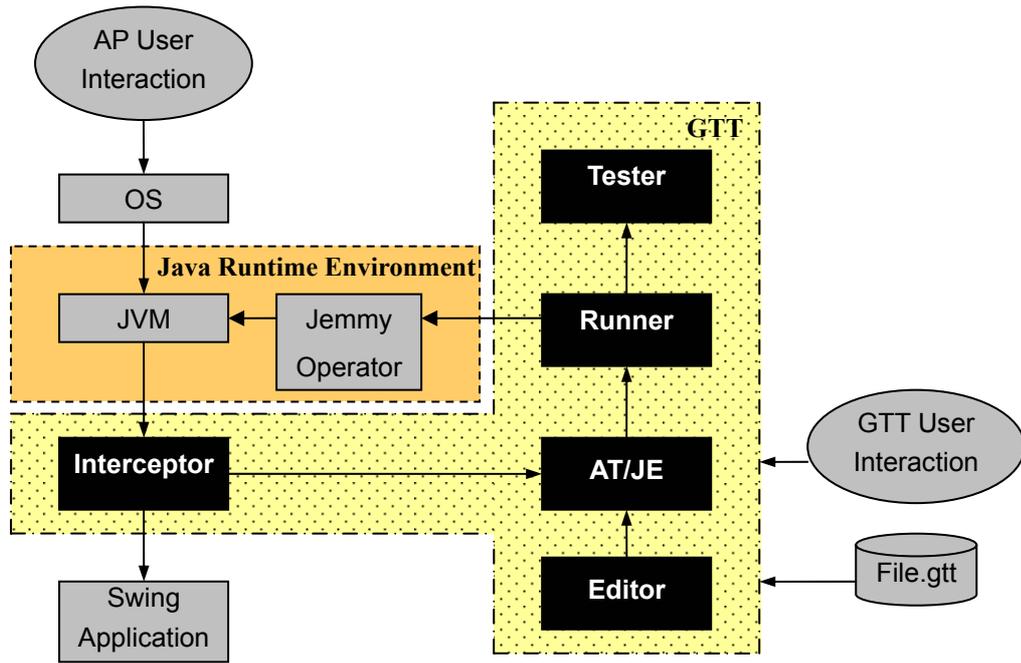


圖 1.3 經由 JVM 截取事件訊息架構圖

程式經由 JVM 來取得待測程式的視窗元件訊息，如圖 1.3 所示，然後再進行處理。這樣的好處在於 Java 提供了 Event Queue 的機制，程式可以透過現有的函式從 JVM 中取得這些在 Event Queue 的抽象化訊息，對於編輯以及維護測試流程上較為方便。但壞處在於待測程式被限制為使用 Java Swing 所開發的程式。

#### 1.4. 選擇可行方案(Select System Solution)

根據我們在 1.2 小節所提出的技術解決方案，將 1.3 小節所提出的方法做一個評估與比較：

	從 OS 截取訊息	從 JVM 截取訊息
<b>Ease of Development</b>	困難	容易
<b>Ease of Editing</b>	困難	容易
<b>Ease of Testing</b>	困難	容易
<b>Extensibility</b>	中	中

<b>Maintenance</b>	低	高
--------------------	---	---

經過比較的結果可以得知，GTT 這個系統的原始目的是針對以 Java 所開發的程式進行測試，而且對於測試流程與事件的處理需要比較完整的功能，所以最後採用從 JVM 來截取事件訊息的方法較為合適。

## 1.5. 介面需求與設計(Interface Requirement and Design)

我們定義了內部介面與外部介面需求兩個部份，列表如下：

- 內部介面

需求編號	優先順序	需求描述
GTT030	1	GTT 以繼承 class java.awt.EventQueue 的方式銜接 JDK v1.4，藉以擷取使用者發出之事件。
GTT031	1	以高階抽象化截取時，GTT 必須將擷取之事件轉換為 Jemmy API 所表達之事件。
GTT032	1	GTT 使用 Jemmy API 播放所表達之事件。

- 外部介面

介面編號	優先順序	介面描述
GTT033	1	以 Java swing API v1.4 來呈現 GTT 的使用者介面。
GTT034	1	GTT 使用 JDK v1.4 的檔案 API 儲存或載入測試事件流程的檔案
GTT035	1	GTT 要能讀取 JUnit 之測試案例檔案作為外部測試節點。
GTT036	1	待測應用程式若要使用 JUnit 之測試案例檔案作為外部測試節點，待測應用程式本身必須提供 getInstance，此函式負責傳回待測應用程式之實體。

GTT037	1	<p>待測應用程式若要使用 JUnit 之測試案例檔案作為外部測試節點，GTT 提供 getInstance、getRoot 與 getComponent 三個函式給 JUnit 之測試案例檔案中使用：</p> <ul style="list-style-type: none"> <li>● getInstance 可傳回待測程式的實體。</li> <li>● getRoot 可傳回待測程式的主視窗。</li> <li>● getComponent 可依據兩個參數傳回待測主視窗上的元件，第一個參數是元件所在的視窗，第二個參數是元件名稱。</li> </ul>
GTT038	1	<p>待測應用程式必須有程式起始點之函式(預設值為 main 函式)，作為執行待測應用程式之起點。</p>

## 2. 系統與軟體架構(System/Software Architecture)

### 2.1. 使用案例分析 (Use Case Analysis)

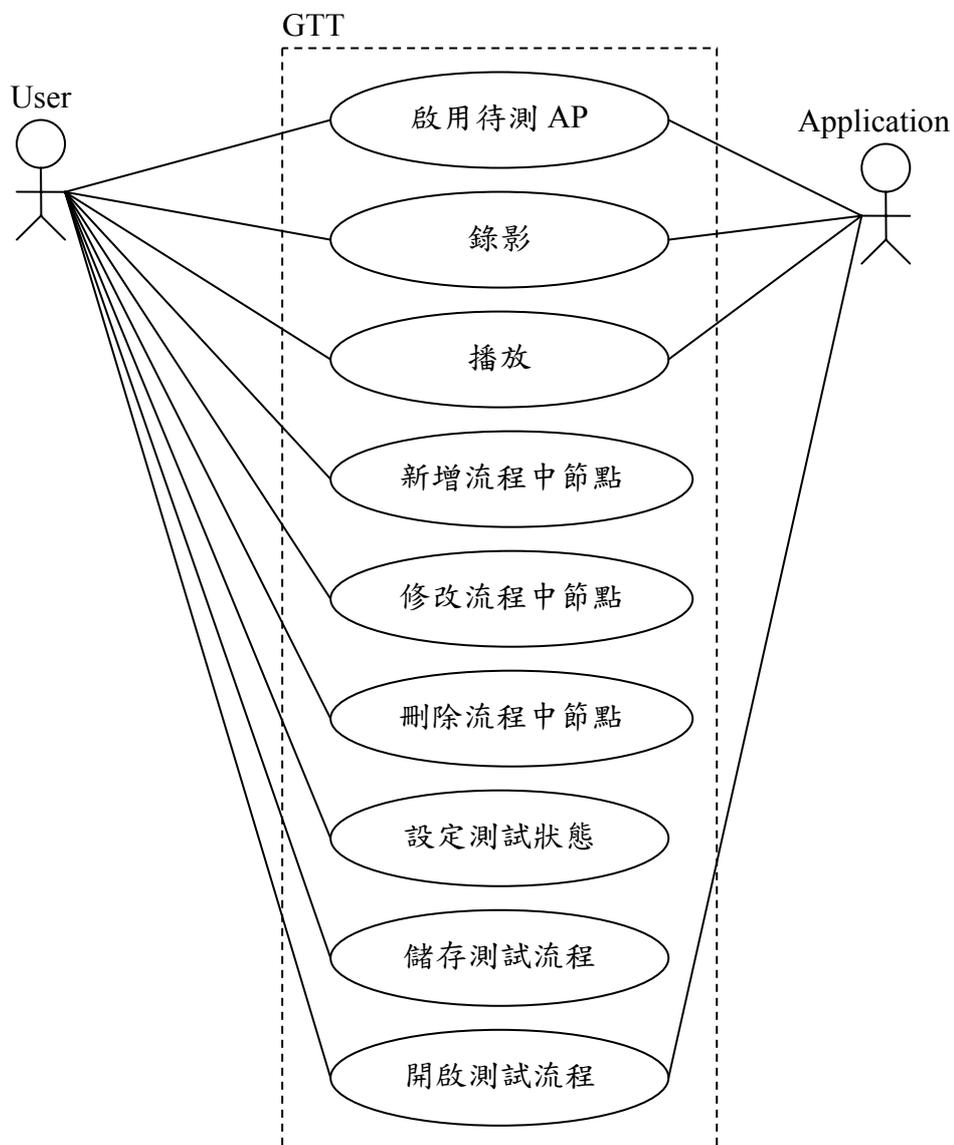


圖 2.1 : Use case diagram

<b>*Use case name</b>	<u>啟用待測AP</u>
<b>Summary</b>	將待測 Application 開啟。
<b>*Actor</b>	User。
<b>*Dependency</b>	
<b>Precondition</b>	已開啟 GTT

<b>*Description</b>	<b>Actor Actions :</b>	<b>System Responses :</b>
	使用者執行開啟待測 AP 功能	GTT 系統開啟選擇 Application 之 class 介面
	使用者選擇所要開啟之待測 Application 之 class 檔，並按下完成	GTT 關閉開啟待測程式介面，並依據使用者選定之 class 檔開啟待測應用程式
<b>Alternatives</b>	使用者在開啟應用程式時按下取消，取消設定	
<b>Post-condition</b>	待測應用程式開啟	
<b>Inputs</b>	Java 之 class 檔	
<b>Outputs</b>		

<b>*Use case name</b>	<b>錄影</b>	
<b>Summary</b>	錄製使用者對 Application 的所有 GUI 操作 event。	
<b>*Actor</b>	User、AP(Application)。	
<b>*Dependency</b>	<b>啟用待測AP</b>	
<b>Precondition</b>	已開啟 GTT，並已開啟待測 Application	
<b>*Description</b>	<b>Actor Actions :</b>	<b>System Responses :</b>
	使用者啟動 GTT 錄影功能	GTT 系統將 Focus 轉移至 Application 上
	使用者與 Application 互動	系統將使用者對 Application 之互動記錄到 Interceptor 模組 <sup>1</sup>
	使用者關閉 GTT 記錄功能。	AT/AE 模組接收 Interceptor 模組暫存的多個 AE 資訊，並傳遞至 Editor。
<b>Alternatives</b>		
<b>Post-condition</b>	Editor 產生事件流程	
<b>Inputs</b>	使用者對 Application 發出事件	
<b>Outputs</b>	產生事件流程	

<sup>1</sup> JVM 接收內部事件並加以解析轉換成為輸入事件(JIE)，並會被放置在事件佇列(Java System Event Queue)內，此時的事件佇列被我們的 Interceptor 模組取代。此時 Interceptor 模組將會跟著接收 JIE，並在 JIE 未被事件發送執行緒(Java Event Dispatch Thread)提取之前，先行將 JIE 截取出，並轉換成 AE 資訊。轉換的 AE 資訊會暫存於 Interceptor 模組內。最後再將事件發送執行緒將事件傳給 AP。AP 接收事件做後續處理。

<b>*Use case name</b>	<b>播放</b>	
<b>Summary</b>	將事件流程之連續事件對待測 AP 做執行。	
<b>*Actor</b>	User。	
<b>*Dependency</b>	<b>啟用待測AP</b>	
<b>Precondition</b>	已開啟 GTT，已啟用待測 AP，並且錄製且編輯好一段操作流程。	
<b>*Description</b>	<b>Actor Actions：</b>	<b>System Responses：</b>
	使用者啟動 播放功能	系統將 Focus 轉移至 Application，並將事件流程之事件依序對 Application 執行 <sup>2</sup> ，系統反映出正確與否
	使用者結束播放	
<b>Alternatives</b>	使用者在播放時按下停止，AP 停止執行，focus 返回 GTT。	
<b>Post-condition</b>	產生結果	
<b>Inputs</b>		
<b>Outputs</b>		

<b>*Use case name</b>	<b>新增流程中節點</b>	
<b>Summary</b>	新增測試流程中的節點	
<b>*Actor</b>	User。	
<b>*Dependency</b>	<b>啟用待測AP、錄影</b>	

<sup>2</sup>系統反應之詳細播放細節：

1. **Runner 接收 AT/AE 所提供的資訊：**AT/AE 元件提供 AE 資訊和 AT 資訊給與 Runner 元件。
2. **若為 AE 資訊執行 3a 若為 AT 資訊執行 3b**
- 3a. **Robot 物件接收 Runner 元件提供的低階資訊：**Runner 物件接 AE 資訊，並將之轉為符合 Robot 發出動作時所需的資訊，提供給與 Robot 物件。
- 3b. **Tester 模組接收 Runner 元件轉交的 AT 資訊：**Runner 元件接收 AT 資訊，並將之轉交 Tester 模組執行測試，繼續 2。
3. **作業系統接收 Robot 物件所發出滑鼠與鍵盤資訊：**Robot 物件所發出滑鼠或鍵盤資訊，首先會由作業系統接收，作業系統接著發出 System native event。
4. **JVM 接收系統內部事件：**內部事件會被 JVM 加以解析轉換成為 JIE，並會被放置在事件佇列。

**AP 接收 JIE：**事件發送執行緒將事件傳給 AP。AP 接收事件做後續處理。繼續 2，直到所有事件與測試資訊結束或錯誤發生。

<b>Precondition</b>	已開啟 GTT，已啟用待測 AP，並且已經錄製好一段操作流程	
<b>*Description</b>	<b>Actor Actions :</b>	<b>System Responses :</b>
	使用者先選取流程中欲增加節點之相鄰節點，並選取增加事件/測試節點之按鈕	GTT 系統開啟待測應用程式
	使用者由待測應用程式中選取一 object	GTT 將使用者選擇之物件細節介面秀出
	使用者設定物件之事件細節，並執行完成	GTT 將節點加入事件流程中
<b>Alternatives</b>	使用者在選擇物件細節介面時按下取消，取消新增	
<b>Post-condition</b>	測試流程增加新節點	
<b>Inputs</b>		
<b>Outputs</b>		

<b>*Use case name</b>	<u>修改流程中節點</u>	
<b>Summary</b>	修改測試流程中的節點	
<b>*Actor</b>	User。	
<b>*Dependency</b>	<u>啟用待測AP、錄影、開啟測試流程</u>	
<b>Precondition</b>	已開啟 GTT，已啟用待測 AP，並且已經錄製好一段操作流程	
<b>*Description</b>	<b>Actor Actions :</b>	<b>System Responses :</b>
	使用者點選欲修改之事件/測試節點，並按下修改按鈕	GTT 系統依據節點特性，開啟專屬之事件對話框
	使用者由對話框中編輯欲修改之項目，並按下完成	GTT 將依據對話框之內容，修改節點資訊
<b>Alternatives</b>	使用者在對話框出現時按下取消，取消修改	

<b>Post-condition</b>	測試流程中某節點之資訊改變
<b>Inputs</b>	
<b>Outputs</b>	

<b>*Use case name</b>	<b>刪除流程中事件</b>	
<b>Summary</b>	刪除測試流程中的節點	
<b>*Actor</b>	User。	
<b>*Dependency</b>	<b>啟用待測AP、錄影、開啟測試流程</b>	
<b>Precondition</b>	已開啟 GTT，已啟用待測 AP，並且已經錄製好一段操作流程	
<b>*Description</b>	<b>Actor Actions：</b>	<b>System Responses：</b>
	使用者點選欲刪除之事件/測試節點，並按下刪除按鈕	GTT 系統開啟對話框詢問是否刪除
	使用者按下確定刪除	GTT 刪除選擇之節點
<b>Alternatives</b>	使用者在對話框出現時按下取消，取消刪除	
<b>Post-condition</b>	測試流程中某節點被刪除	
<b>Inputs</b>		
<b>Outputs</b>		

<b>*Use case name</b>	<b>設定測試狀態</b>	
<b>Summary</b>	設定關於錄影或播放之 detail。	
<b>*Actor</b>	User。	
<b>*Dependency</b>		
<b>Precondition</b>	已開啟 GTT	
<b>*Description</b>	<b>Actor Actions：</b>	<b>System Responses：</b>
	使用者啟動設定測試狀態	GTT 系統開啟設定測試狀態選單
	使用者調整測試狀態，並按下完成	GTT 依據設定之數值調整內部之狀態，並關閉設定測試狀態選單
<b>Alternatives</b>	使用者在調整測試狀態時按下取消，取消設定	

<b>Post-condition</b>	測試狀態改變
<b>Inputs</b>	
<b>Outputs</b>	

<b>*Use case name</b>	<u>儲存測試流程</u>	
<b>Summary</b>	儲存目前之測試流程	
<b>*Actor</b>	User	
<b>*Dependency</b>	<u>錄影、開啟測試流程</u>	
<b>Precondition</b>	已開啟 GTT	
<b>*Description</b>	<b>Actor Actions :</b>	<b>System Responses :</b>
	使用者點選儲存測試流程	GTT 系統開啟對話框要求使用者輸入檔案名稱及選擇儲存位置
	使用者輸入檔案名稱並選擇儲存位置，並按下確定存檔	GTT 將測試流程儲存至檔案中
<b>Alternatives</b>	使用者在 GTT 開啟對話框時按下取消，取消儲存	
<b>Post-condition</b>	產生測試流程檔案(.gtt 檔案)	
<b>Inputs</b>		
<b>Outputs</b>		

<b>*Use case name</b>	<u>開啟測試流程</u>	
<b>Summary</b>	開啟已存在之測試流程	
<b>*Actor</b>	User	
<b>*Dependency</b>		
<b>Precondition</b>	已開啟 GTT，檔案系統中已存在一個測試流程(.gtt 檔案)	
<b>*Description</b>	<b>Actor Actions :</b>	<b>System Responses :</b>
	使用者點選開啟測試流程	GTT 系統開啟對話框要求使用者選擇測試流程檔案
	使用者選擇測試流程檔案，並按下確定	GTT 將依據選取之檔案，載入測試流程，並開啟待測應用程式

<b>Alternatives</b>	使用者在 GTT 開啟對話框時按下取消，取消開啟
<b>Post-condition</b>	載入測試流程，並開啟待測應用程式
<b>Inputs</b>	
<b>Outputs</b>	

## 2.2. 使用者介面分析 (User Interface Analysis)

GTT 的畫面主要分為四個區域(見圖 2.2)，畫面左邊佔據大部份的畫面，這個區域主要是我們流程結構的呈現區域，我們稱之為「測試流程編輯區」，是一個樹狀結構，GTT 上大部份功能都是支援這個結構，這個樹狀結構是整個測試流程的核心，我們可以透過點選編輯區上方 Tab 按鈕來切換選取不同的測試流程。樹狀結構以圖  來表示一個複合節點，以圖  來表示事件節點，以圖  來表示測試節點。跟隨在事件節點  或測試節點  代表圖之後，是目前元件的代表圖形(見圖 2.3)，這個圖形會依不同的元件類別，顯示元件相對應的圖形，方便使用者觀察；隨後是一段以括號包起的文字，第一個括號內是元件上的文字(對一個 button 而言是它的 Label)，第二個括號內是元件的名稱，第三個括號是元件的類別。

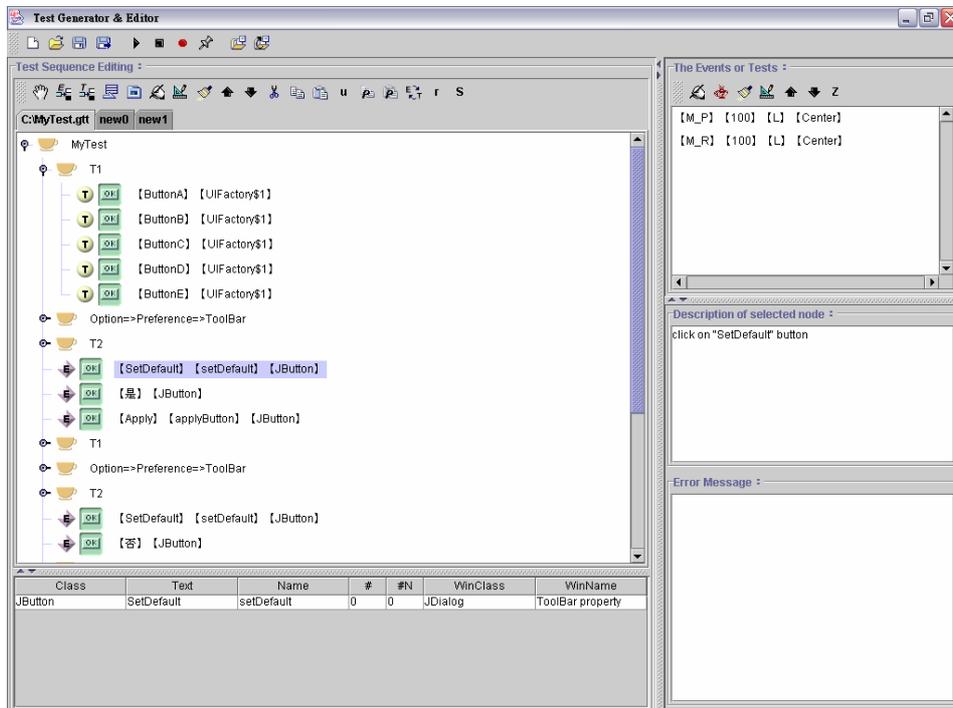


圖 2.2 GTT 主畫面

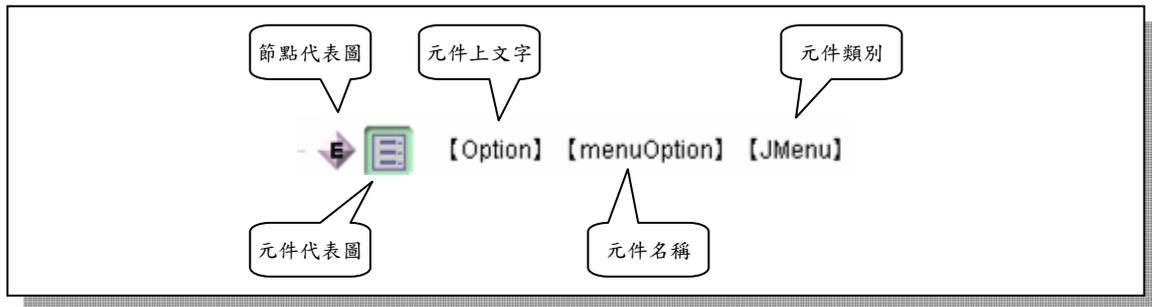


圖 2.3 事件節點資訊

GTT 主畫面左下角是一個「元件抽象資訊表示區」，當我們點選到一個事件節點或測試節點，畫面(見圖 2.4)上的表格就會呈現點選節點的相關抽象資訊，依序為「class(元件類別)」、「text(元件上文字)」、「name(元件名稱)」、「#(視窗中同類別元件索引)」、「#N(視窗中同類別元件名稱索引)」、「WinClass(所在視窗類別)」、「WinName(所在視窗名稱)」。

Class	Text	Name	#	#N	WinClass	WinName
JButton	setDefault	setDefault	0	0	JDialog	ToolBar property

圖 2.4 節點元件抽象化資訊

GTT 主畫面右上角是「單元輸入事件清單」或「測試方法清單」。若一個清單元素為滑鼠單元輸入事件(見圖 2.5)，則元素上顯示依序為「動作類型」、「間歇時間」、「按鍵別」、「座標方法」；若一個清單元素為鍵盤單元輸入事件(見圖 2.6)，則元素上顯示依序為「動作類型」、「間歇時間」、「按鍵別」；若一個清單元素為測試方法(見圖 2.7)，則元素上顯示依序為「測試方法」、「預期回傳值」、「錯誤回傳訊息」、「參數型態」、「參數值」；若一個清單元素為測試類別(見圖 2.8)，則元素上顯示依序為「測試類別」、「測試方法」、「參數型態」。

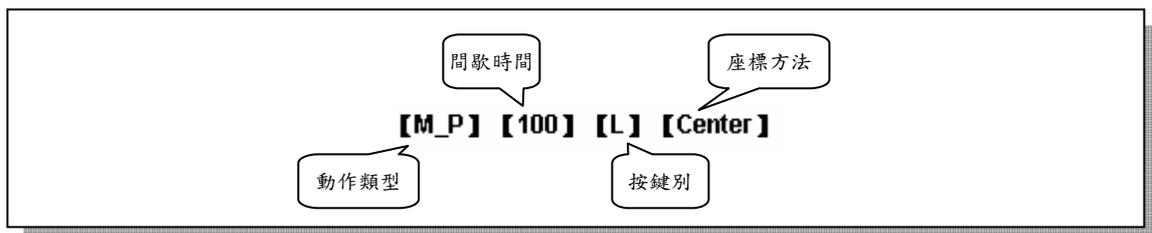


圖 2.5 滑鼠單元輸入事件

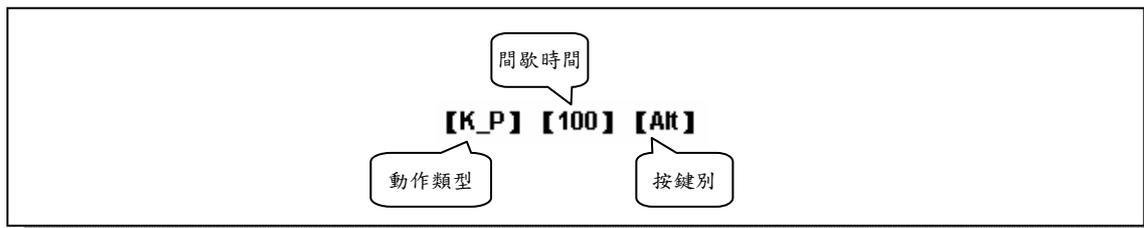


圖 2.6 鍵盤單元輸入事件

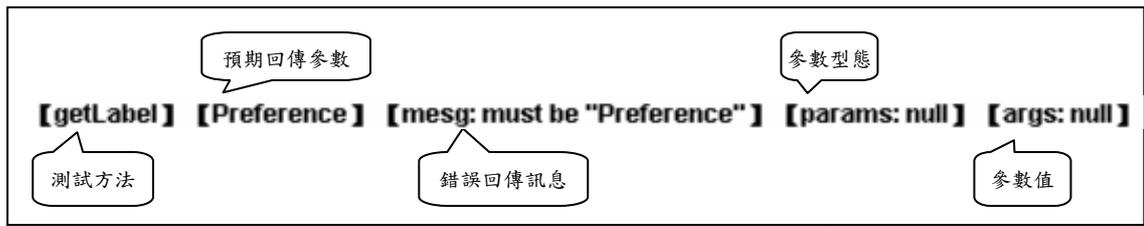


圖 2.7 測試方法



圖 2.8 測試類別

GTT 主畫面的右下角包含兩個文字欄位，在上面的欄位是用來描述每個樹狀結構節點的資訊，使用者可以在選取節點後，在欄位鍵入所需的描述文字；在下面的是一個錯誤訊息回應欄位，當執行期間樹狀節點發生執行錯誤，則錯誤訊息會附加到該節點，可以透過選取錯誤節點來觀看發生錯誤的情況。我們以節點代表圖形的顏色，來表示執行結果，綠色    表示正確，紅色    表示錯誤。

圖 2.9 是一個 GTT 主要工具列，由左到右分別為「開啟新文件」、「開啟舊文件」、「儲存」、「另存新檔」、「啟動自動測試」、「停止錄製」、「開啟錄製」、「設定執行停止節點」、「開啟被測應用程式」、「重置被測應用程式」。詳細使用功能與使用時機，見表 4.1。



圖 2.9 GTT 主要工具列

圖案	按鍵名稱	按鍵功能	使用時機
	開啟新文件	新增文件到編區內	使用者欲新增一個文件，利於編輯新的測試流程。執行後會在文件編輯區內新增一個文件，同時會產生代表這個文件的 Tab 按鈕。
	開啟舊文件	開啟舊文件到編區內	使用者欲開啟一個儲存的文件。執行後會載入儲存的文件並置入編輯區內，同時會產生代表這個文件的 Tab 按鈕。
	儲存文件	儲存目前選取的文件	使用者欲儲存目前選取的文件，若文件尚未命名存檔，則跳出檔案管理員，否則直接存檔。
	另存新檔	另存目前選取的文件	使用者欲儲存目前選取的文件，跳出檔案管理員，儲存目前選取的文件。
	啟動自動測試	行執自動測試	使用者欲自動執行選取的文件。執行後重新啟動被測應用程式，利用文件上呈現的架構作出適當的執行流程。
	停止錄製	停止錄製功能	使用者欲停止錄製功能，並決定是否將錄下的事件流程，以序列的方式加入文件中選取的節點。
	開啟錄製	啟動錄製功能	使用者欲以錄製功能將流程加入文件中。執行後啟動錄製功能，隨後使用者與被測應用程式的對話，將會被記錄下來，直到停止錄製為止。
	設定執行停止節點	設一個節點為執行停止節點	使用者欲將執行流程只執行到所設定的停止節點為止。當自動執行到此節點時，執行便會在前一個節點結束。
	開啟被測應用程式	選擇開啟被測應用程式	使用者欲載入執行被測應用程式。利用檔案管理員，選取被測應用程式並開啟執行。

	重置被測應用程式	重新啟動被測應用程式	使用者欲重新啟動被測應用程式。
---	----------	------------	-----------------

表 2.1 GTT 主要工具列功能說明與使用時機

圖 2.10 是編輯文件流程主要的工具列，由左到右分別為「停止選取插入動作」、「插入事件節點」、「插入測試節點」、「選取所有元件」、「新增資料夾」、「手動新增節點」、「編輯節點」、「刪除節點」、「上移節點」、「下移節點」、「剪下節點」、「複製節點」、「貼上節點」、「解除 sharing」、「sharing 方式複製節點」、「sharing 方式貼上節點」、「節點角色互換」、「新增重新啟動節點」、「重設節點狀態」。詳細使用功能與使用時機，見表 2.2。



圖 2.10 編輯文件流程主要工具列

圖案	按鍵名稱	按鍵功能	使用時機
	停止選取插入動作	關閉事件攔截	選擇插入輸入事件或是插入測試節點功能鍵，會啟動事件攔截，當完成動作後需使用此功能鍵，以便關閉事件攔截。
	插入事件節點	啟動事件攔截，並開啟元件選取對話框	使用者要以點選的方式得到元件抽象資訊，並加入單元輸入事件。使用者點選所要區域範圍，利用對話框的方式，選取所需元件並選取所要單元輸入事件。
	插入測試節點	啟動事件攔截，並開啟元件選取對話框	使用者要以點選的方式得到元件抽象資訊，並加入測試節點。使用者點選所要區域範圍，利用對話框的方式，選取所需元件並選取所要測試方法。

	選取所有元件	選取視窗上所有元件，並開啟元件選取對話框	類似插入輸入事件節點或插入測試節點，不同點在於提供被測程式顯示的所有元件，不必點選所要區域範圍。
	新增資料夾	新增一個資料夾	使用者想利用資料夾來整理測試流程。
	手動新增節點	開啟新增節點對話框	利用對話框的方式，直接填入代表一個元件所需要的抽象資料。
	編輯節點	開啟編輯節點對話框	利用對話框的方式，編輯選取的節點上的元件抽象資料。
	刪除節點	刪除選取的所有節點	使用者欲刪除選取的多個節點。
	上移節點	向上移動一個節點	使用者欲在同一個資料夾內，向上移動目前選取的節點。
	下移節點	向下移動一個節點	使用者欲在同一個資料夾內，向下移動目前選取的節點。
	剪下節點	剪下目前選取的所有節點	使用者欲搬動目前選取的所有節點，隨後將之貼到所要的區域內。
	複製節點	複製目前選取的所有節點	使用者欲複製目前選取的所有節點，隨後將之貼到所要的區域內。
	貼上節點	貼上經由複製或剪下的所有節點	使用者欲將複製或剪下的所有節點，貼到所要的區域內。
	解除 sharing	解除 sharing 關係	解除選取節點與其它 sharing 節點的相依性關係。
	sharing 方式複製節點	sharing 方式複製節點	使用者欲使節點有相依性，以 sharing 方式複製節點，複製的節點與原始節點存在相依性，更改 sharing 中一個元素，其它元素會跟著改變。
	sharing 方式貼上節點	sharing 方式貼上節點	貼上以 sharing 方式複製的節點。
	節點角色互換	事件節點與測試節點角色互換	使用者欲利用角色互換的方式新增事件節點或測試節點。選取的是事件節點，執行此功能會轉為測試節點。相反的，選取的是測試節點，執行此功能會轉為事件節點。

r	新增重新啟動節點	新增重新啟動節點	使用者欲在測試期間重新啟動被測程式。新增一個重新啟動節點，當自動測試每遇此節點，便會重新開啟被測程式。
S	重設節點狀態	重新設定每個節點的狀態	自動測試執行，會因為執行結果狀態，將代表事件或測試的圖形顏色改變，當使用者欲恢復原始狀態與顏色時，可以使用此功能。

表 2.2 編輯文件流程工具列功能說明與使用時機

圖 2.11 是編輯清單主要的工具列，由左到右分別為「手動新增元素」、「新增測試類別方法元素」、「刪除元素」、「編輯元素」、「上移元素」、「下移元素」、「新增間歇時間元素」。詳細使用功能與使用時機，見表 2.3。



圖 2.11 編輯清單工具列

圖 2.12 是手動新增或編輯「滑鼠單元輸入事件對話框」，第一個項目為「ID(事件的類別)」：代表當此事件被執行時，應該以那種事件類型來反應發出動作。第二個項目為「Button(滑鼠按鍵類別)」：代表當此事件發生時，應該以按下那個按鍵。第三個項目為「SleepTime(間歇時間)」：代表當此事件送出後，應停止多少時間讓事件能被消耗，元件能夠重繪。接下來的項目是「座標方法」：代表當此事件發生時，應該以什麼方式來取得座標。對話框會依使用者選取節點元件類型的不同，呈現不同的座標方法欄位，這些不同的記錄方法欄位會動態依元件類型產生，目前 GTT 上有六種座標方法(見圖 2.13)。

圖案	按鍵名稱	按鍵功能	使用時機
	手動新增元素	手動新增單元事件，開啟新增事件對話框；或手動新增測試方法元素，開啟新增測試對話框	使用者欲新增單元事件元素或測試方法元素。利用對話框填選的方式，加入所需的單元事件元素或測試方法元素。以插入的方式加入清單，若尚未選取清單元素，則加入插入第一筆；若選取一清單元素，則插入此元素之後。
	新增外部測試類別方法元素	開啟新增外部測試類別方法元素對話框	使用者欲新增外部類別測試方法。利用對話框填選的方式，加入所需的測試類別方法元素。
	刪除元素	刪除選取元素	使用者欲刪除目前選取的元素。
	編輯元素	開啟編輯對話框，編輯選取元素	使用者欲編輯目前選取的元素。依選取元素開啟不同的對話框。
	上移元素	向上移動一個元素	使用者欲在清單內，向上移動目前選取的元素。
	下移元素	向下移動一個元素	使用者欲在清單內，向下移動目前選取的元素。
	新增間歇時間元素	開啟新增間歇時間元素對話框	使用者欲在事件之間，新增間歇時間元素。

表 2.3 編輯清單工具列功能說明與使用時機

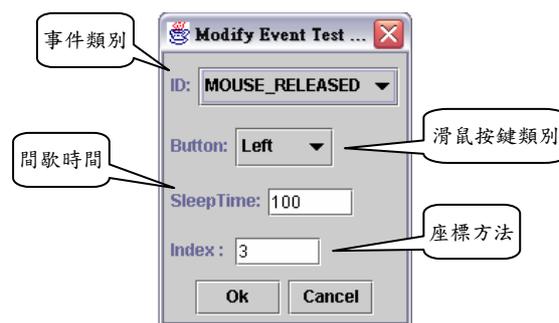


圖 2.12 滑鼠單元輸入事件對話框

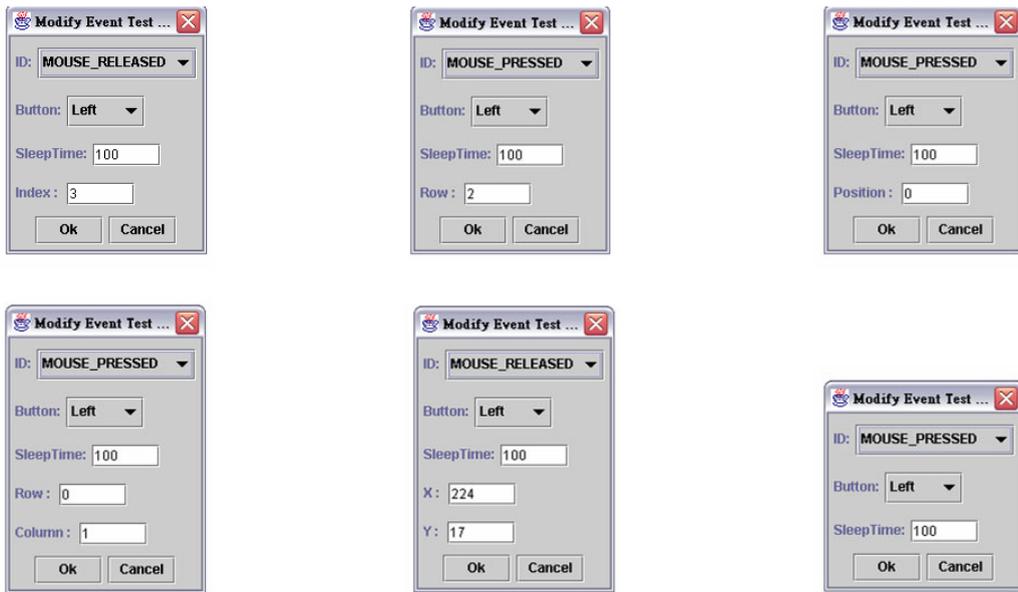


圖 2.13 滑鼠單元輸入事件對話框的六種座標方法

圖 2.14 是手動新增或編輯「鍵盤單元輸入事件對話框」，第一個項目為「ID(事件的類別)」：代表當此事件被執行時，應該以那種事件類型來反應發出動作。第二個項目為「SleepTime(間歇時間)」：代表當此事件送出後，必需停止多少時間讓事件能被消耗，元件能夠重繪。第三個項目為「Key(按鍵類別)」：代表當此事件發生時，應該按下那個鍵。第四項目是「Select Key(選取按鍵類別)」：使用者可以用選取的方式，選取無法鍵入的類別型態，代表當此事件發生時，應該按下那個鍵。

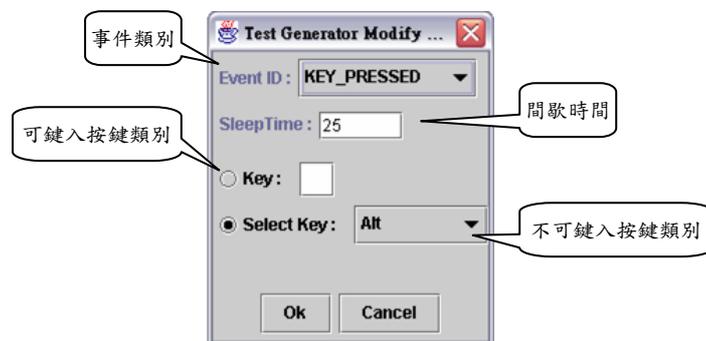


圖 2.14 鍵盤單元輸入事件對話框

圖 2.15 是手動新增或編輯「元件抽象資訊對話框」，第一個項目為「元件所在的視窗類別」：代表當尋找視窗時，應在那以什麼視窗類別尋找。第二個項目為「元件所在的視窗標題」：代表當尋找視窗時，應以什麼視窗標

題尋找。第三個項目為「元件的類別」：代表當在視窗中尋找元件時依據的類別。第四個項目為「元件的名稱」：代表當在視窗中尋找元件時依據的名稱。第五個項目為「元件類別的索引」：代表當在視窗中尋找元件時，元件類別在相同元件類別的索引順序。第六個項目為「元件名稱的索引」：代表當在視窗中尋找元件時，元件名稱在相同元件類別的索引順序。

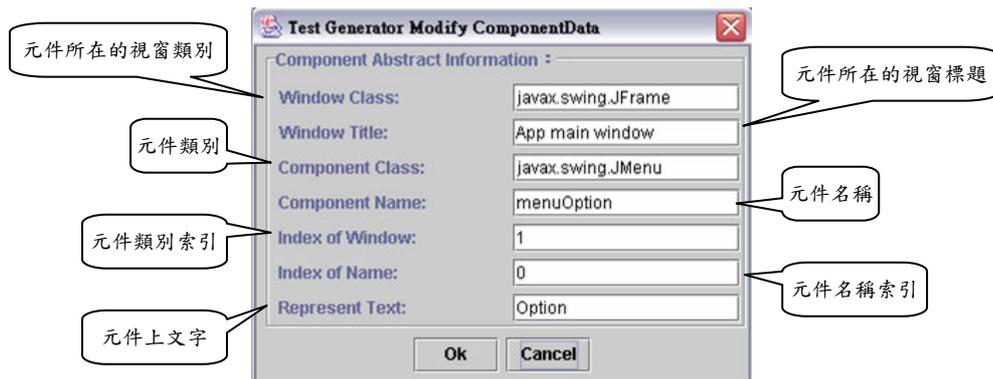


圖 2.15 鍵盤單元輸入事件對話框

圖 2.16 是「元件選取對話框」，畫面最左邊區域為樹狀結構，畫面中間區域為清單結構，畫面右邊區域為表格結構。樹狀結構依元件類別分類為不同的資料夾，可在葉子節點按兩下或按下  加入清單元件上；清單結構為將要加入流程的元件集合，清單的順序即為加入流程後的順序，可以利用   鍵來調整順序；表格結構為事件或測試的集合，當點上一個清單元素後，可以利用  開啟單元輸入事件或測試方法對話框，來加入的事件或測試，加入後以表格的方式呈現，可以選取表格列用  刪除。畫面左上角的 ShowComponent 按鈕，執行後可以顯示所選擇的元件在被測程式中的所在。

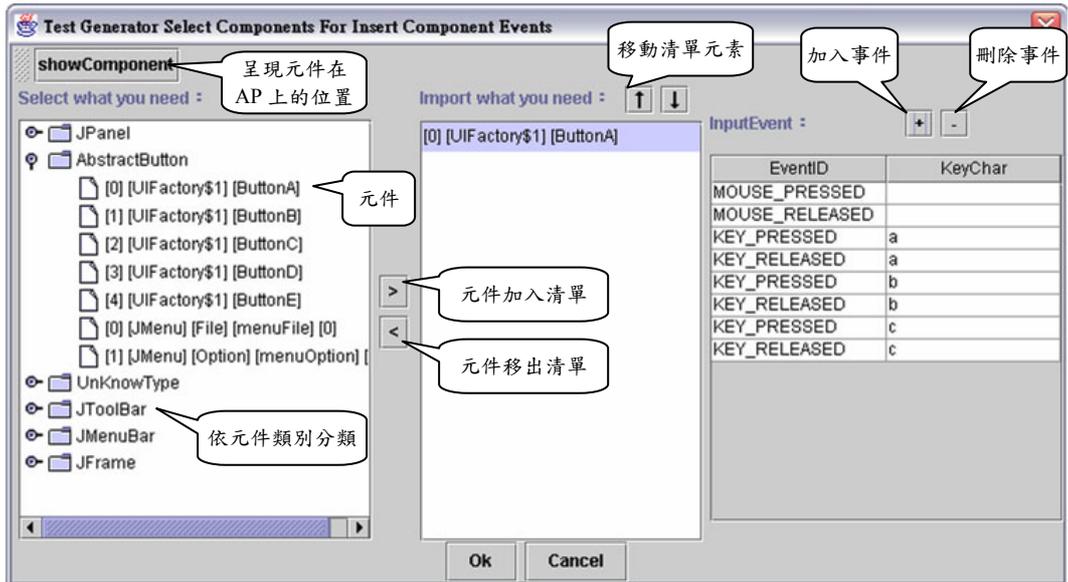


圖 2.16 元件選取對話框

### 2.3. 類別圖分析(Class Diagram Analysis)

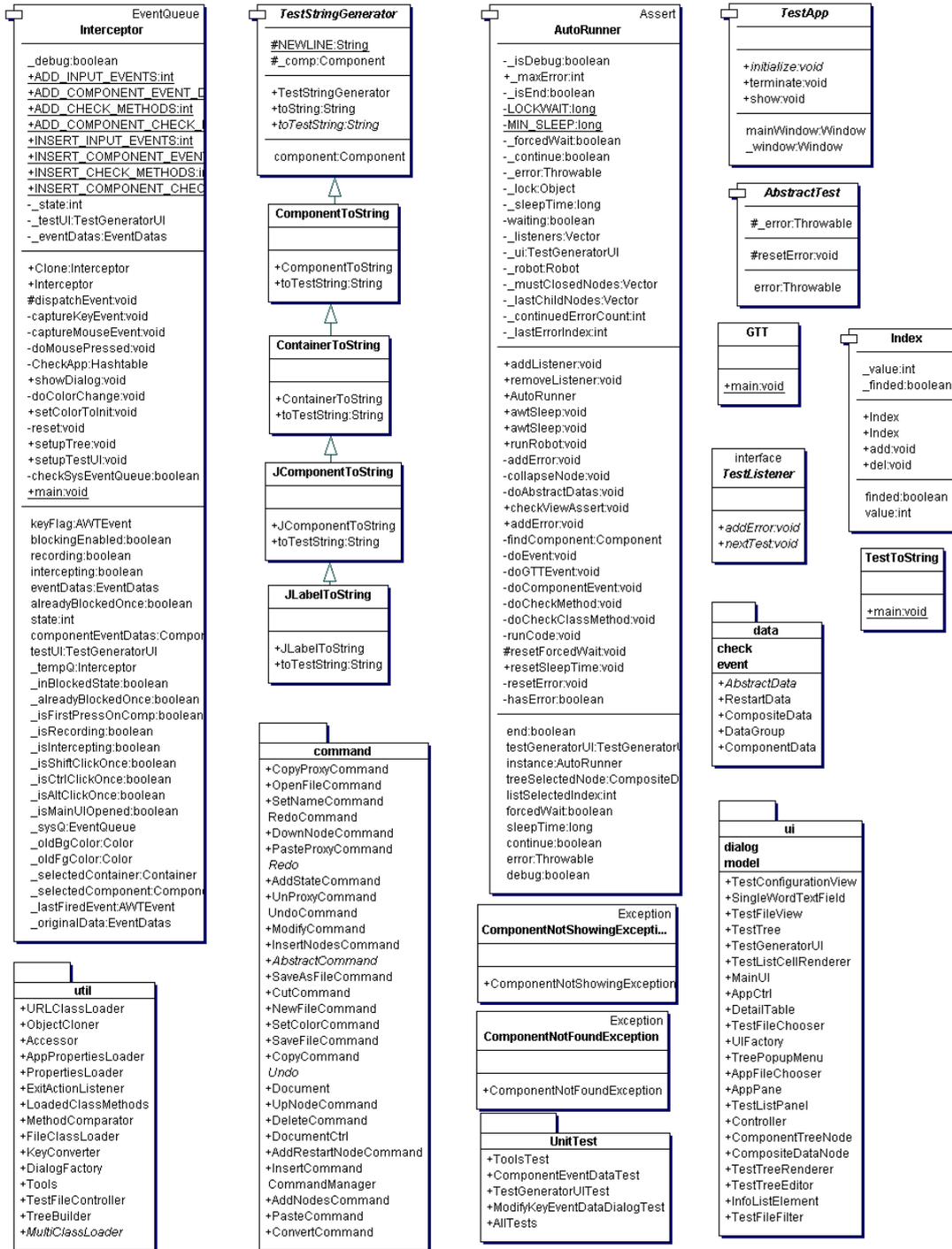


圖 2.17: “default” Package Class Diagram



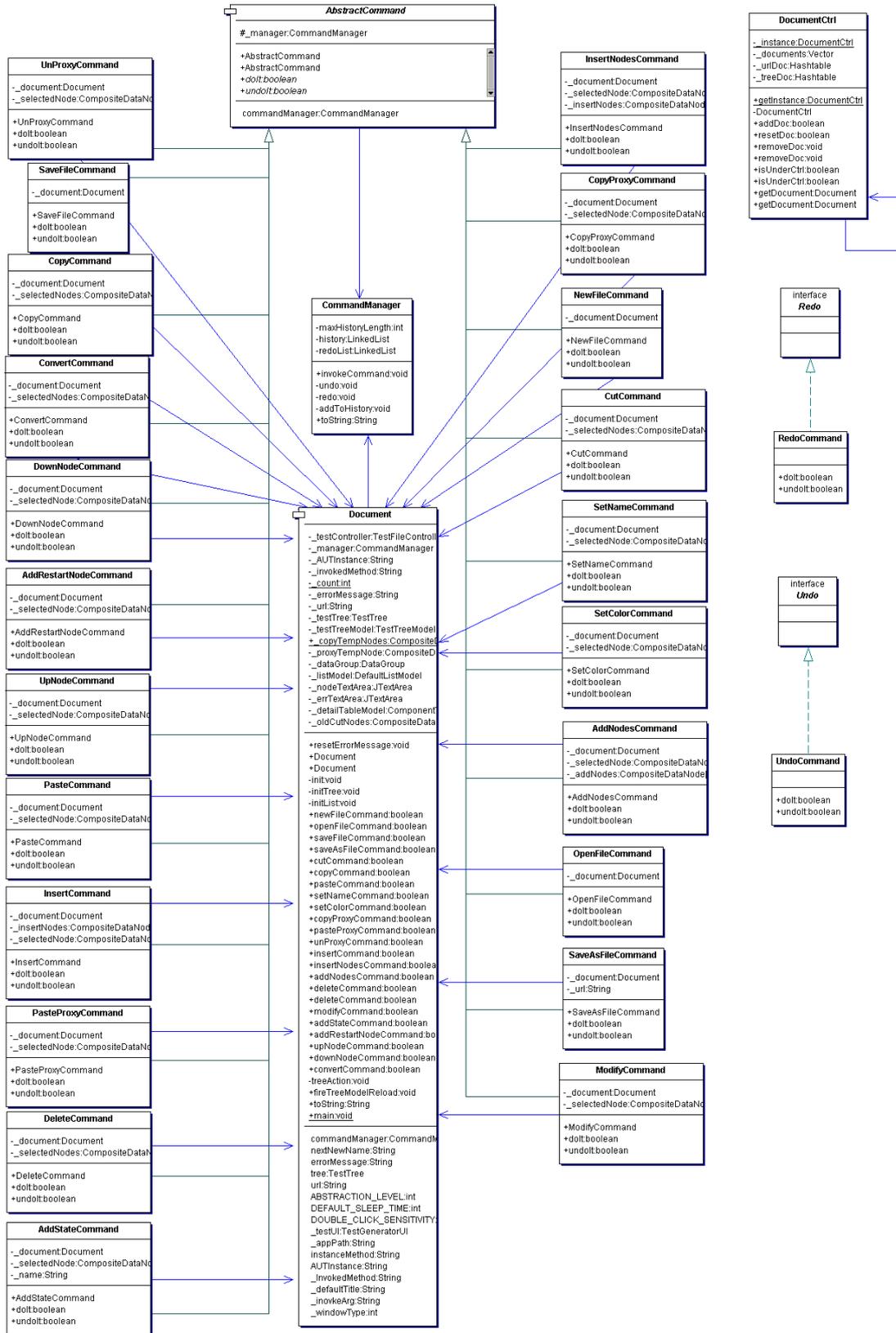


圖 2.19 : “command” Package Class Diagram



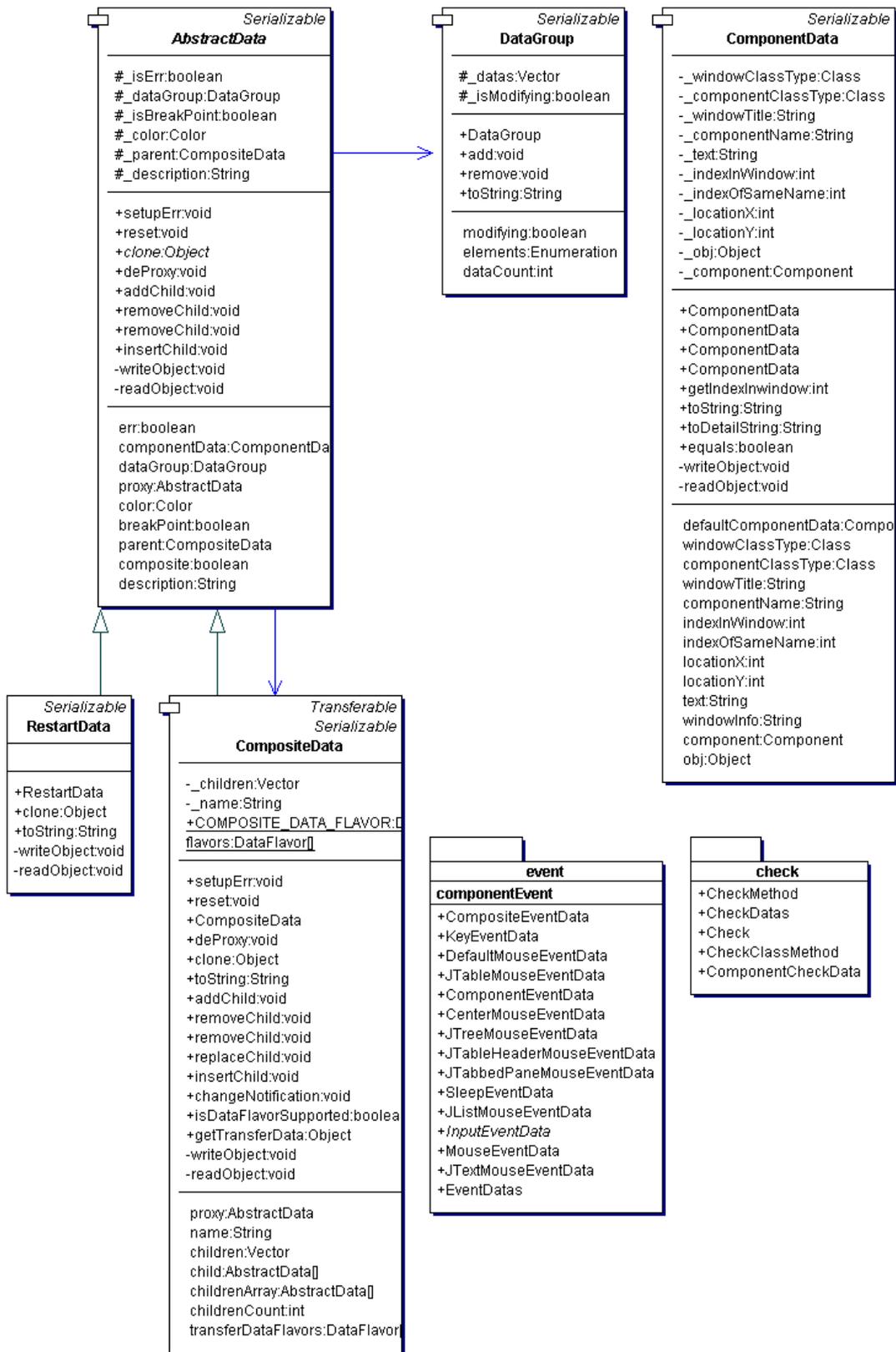


圖 2.21 : “data” Package Class Diagram

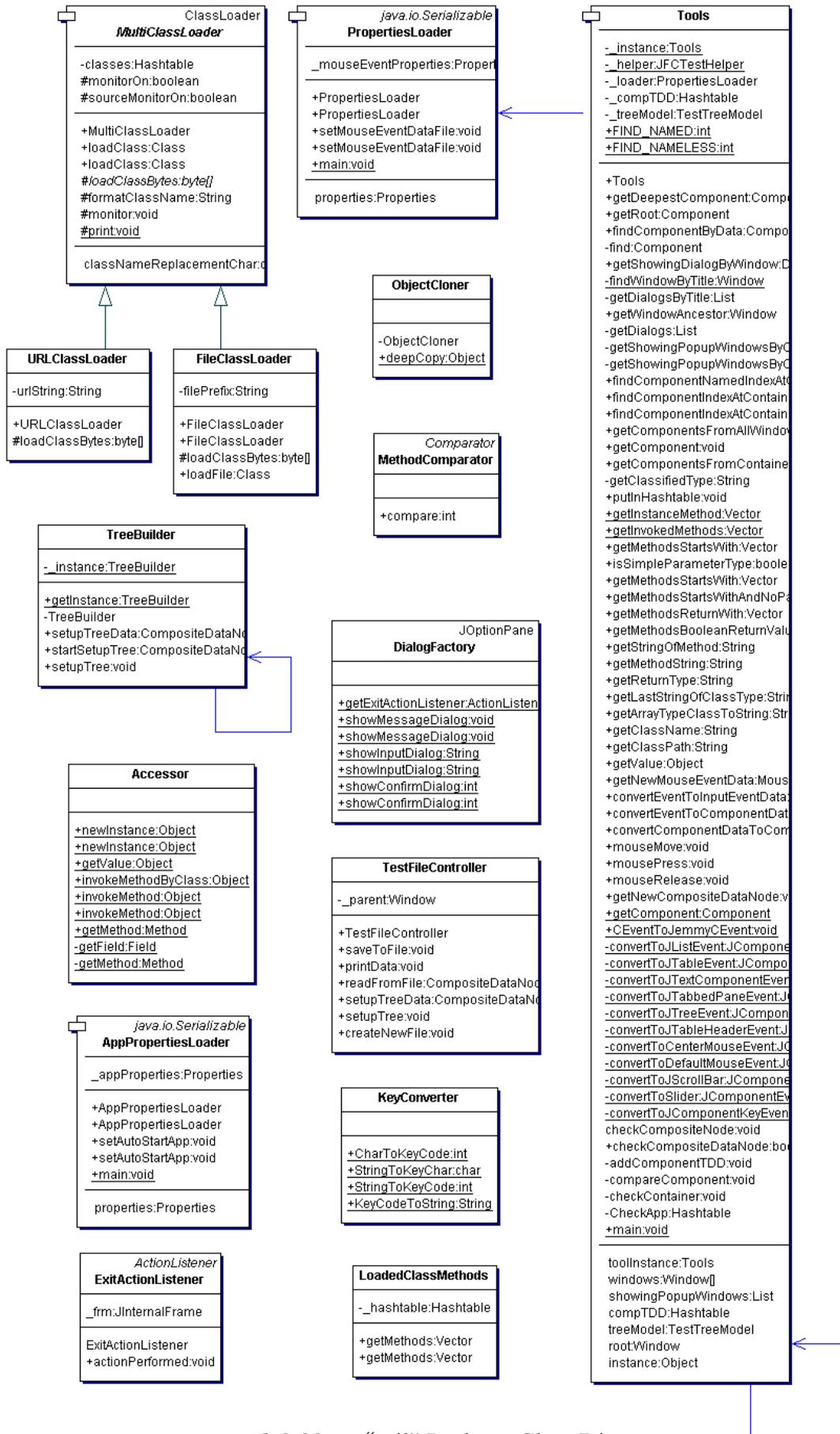


圖 2.22 : “util” Package Class Diagram

以下說明程式中主要之類別：

<b>Class Name</b>	ui.TestGeneratorUI
<b>Description</b>	main 函式所在之類別，為整個 GTT 之主使用者介面類別。
<b>Relationship with other classes</b>	Interceptor、Controller、TestListPanel、DetailTable、TestFileChooser、AppFileChooser、TreePopupMenu、TestConfigurationView、TestTree、CompositeDataNode、AbstractAction、DataGroup
<b>Traceability with other use case</b>	啟用待測 AP、錄影、播放、新增流程中節點、修改流程中節點、刪除流程中節點、設定測試狀態、儲存測試流程、開啟測試流程

<b>Class Name</b>	Interceptor
<b>Description</b>	繼承 JVM 之 EventQueue，負責擷取 JVM 上發生之使用者互動事件
<b>Relationship with other classes</b>	TestGeneratorUI、EventDatas
<b>Traceability with other use case</b>	啟用待測 AP、錄影、新增流程中節點、開啟測試流程

<b>Class Name</b>	AbstractCommand
<b>Description</b>	所有按鍵命令所構成 Command Pattern 之 Interface。
<b>Relationship with other classes</b>	AbstractCommand、AddNodesCommand、AddRestartNodeCommand、AddStateCommand、CommandManager、ConvertCommand、CopyCommand、CopyProxyCommand、CutCommand、DeleteCommand、DocumentCtrl、DownNodeCommand、InsertCommand、InsertNodesCommand、ModifyCommand、NewFileCommand、OpenFileCommand、PasteCommand、PasteProxyCommand、RedoCommand、SaveAsFileCommand、SaveFileCommand、SetColorCommand、SetNameCommand、UndoCommand、UnProxyCommand、UpNodeCommand
<b>Traceability with</b>	啟用待測 AP、錄影、播放、新增流程中節點、修改

<b>other use case</b>	流程中節點、刪除流程中節點、設定測試狀態、儲存測試流程、開啟測試流程
-----------------------	------------------------------------

<b>Class Name</b>	command.Document
<b>Description</b>	GTT 負責存放 Model 的類別，管理存檔及接受其他 Command 之動作
<b>Relationship with other classes</b>	CommandManager、TestFileController、AppFileChooser、TestTree、CompositeDataNode、AbstractAction、DataGroup
<b>Traceability with other use case</b>	啟用待測 AP、錄影、播放、新增流程中節點、修改流程中節點、刪除流程中節點、設定測試狀態、儲存測試流程、開啟測試流程

<b>Class Name</b>	<p>XXXDialog 類別，包含</p> <p>AddCheckClassMethodDialogAddCheckDialog、</p> <p>AddEventDialog、AddKeyEventDialog、</p> <p>AddSleepEventDialog、</p> <p>ModifyAbstractButtonEventDialog、</p> <p>ModifyCheckClassMethodDialog、</p> <p>ModifyCheckMethodDialog、</p> <p>ModifyComponentDataDialog、</p> <p>ModifyEventDataDialog、</p> <p>ModifyJColorChooserEventDialog、</p> <p>ModifyJComboBoxEventDialog、</p> <p>ModifyJComponentEventDialog、</p> <p>ModifyJDialogEventDialog、</p> <p>ModifyJFileChooserEventDialog、</p> <p>ModifyJFrameEventDialog、</p> <p>ModifyJInternalFrameEventDialog、</p> <p>ModifyJListEventDialog、</p> <p>ModifyJMenuBarEventDialog、</p> <p>ModifyJMenuEventDialog、</p> <p>ModifyJPopupMenuEventDialog、</p> <p>ModifyJScrollBarEventDialog、</p> <p>ModifyJScrollPaneEventDialog、</p> <p>ModifyJSliderEventDialog、</p>
-------------------	---

	ModifyJSpinnerEventDialog、 ModifyJSplitPaneEventDialog、 ModifyJTabbedPaneEventDialog、 ModifyJTableEventDialog、 ModifyJTableHeaderEventDialog、 ModifyJTextAreaEventDialog、 ModifyJTextComponentEventDialog、 ModifyJTreeEventDialog、 ModifyKeyEventDataDialog、 ModifyMouseEventDataDialog、 ModifySleepEventDialog 及 SelectComponentsDialog 三十六個類別
<b>Description</b>	分別表達不同的 Swing 元件之可修改細節
<b>Relationship with other classes</b>	個別相依於所屬之 Swing class
<b>Traceability with other use case</b>	新增流程中節點、修改流程中節點

<b>Class Name</b>	CompositeData
<b>Description</b>	以 Composite Pattern 實作之測試流程樹狀圖中，表達複合節點(Composite node)之類別
<b>Relationship with other classes</b>	AbstractData、ComponentData
<b>Traceability with other use case</b>	啟用待測 AP、錄影、播放、新增流程中節點、修改流程中節點、刪除流程中節點、儲存測試流程、開啟測試流程

## 2.4. 循序圖分析(Sequential Diagram Analysis)

以流程圖 2.23 來說明播放流程，當執行測試時會執行五個 Thread，MainThread 是 GTT 程式本身，AutoRunThread 負責達成自動執行，RunCodeThread 負責一個段落的測試執行開始與結束，EventDispatchThread 是在 Swing 應用程式中唯一執行事件 Thread，RunTestThread 負責 GTT 事件與測試的發送。

1. 當我們按下播放鈕，此時 EventDispatchThread 中執行按鈕中的動作，利用前序走法(preorder)將測試流程中的事件與測試資料拿出分為多個集合。播放鈕同時啟動 AutoRunThread。
2. AutoRunThread 拿出一個資料集合，啟動 RunCodeThread，並利用 invokeAndWait 通知 RunCodeThread 可以開始執行此資料集合的測試。
3. RunCodeThread 負責初始化 EventDispatchThread 與 RunTestThread 之間的同步，並啟動 RunTestThread。
4. RunTestThread 發出一個事件後，會執行間歇時間(awtSleep())讓 AutoRunThread 間歇，並且進一步通知 EventDispatchThread 可以處理事件，重新繪製與呼叫方法。AutoRunThread 完成間歇後，利用 invokeAndWait 靠著 EventDispatchThread 間接通知 RunTestThread 執行下一個動作。
5. 重複 4 直到資料集合為空。
6. 若尚有其它資料集合執行 2。

GTT 的播放架構能夠掌控一般 Java GUI 的寫作方式(非 multithread)，若使用者使用 multithread 的方式來處理事件發生後的動作，那麼使用者需要額外增加間歇時間來等待該事件的處理，否則會造成播放時的錯誤判斷。

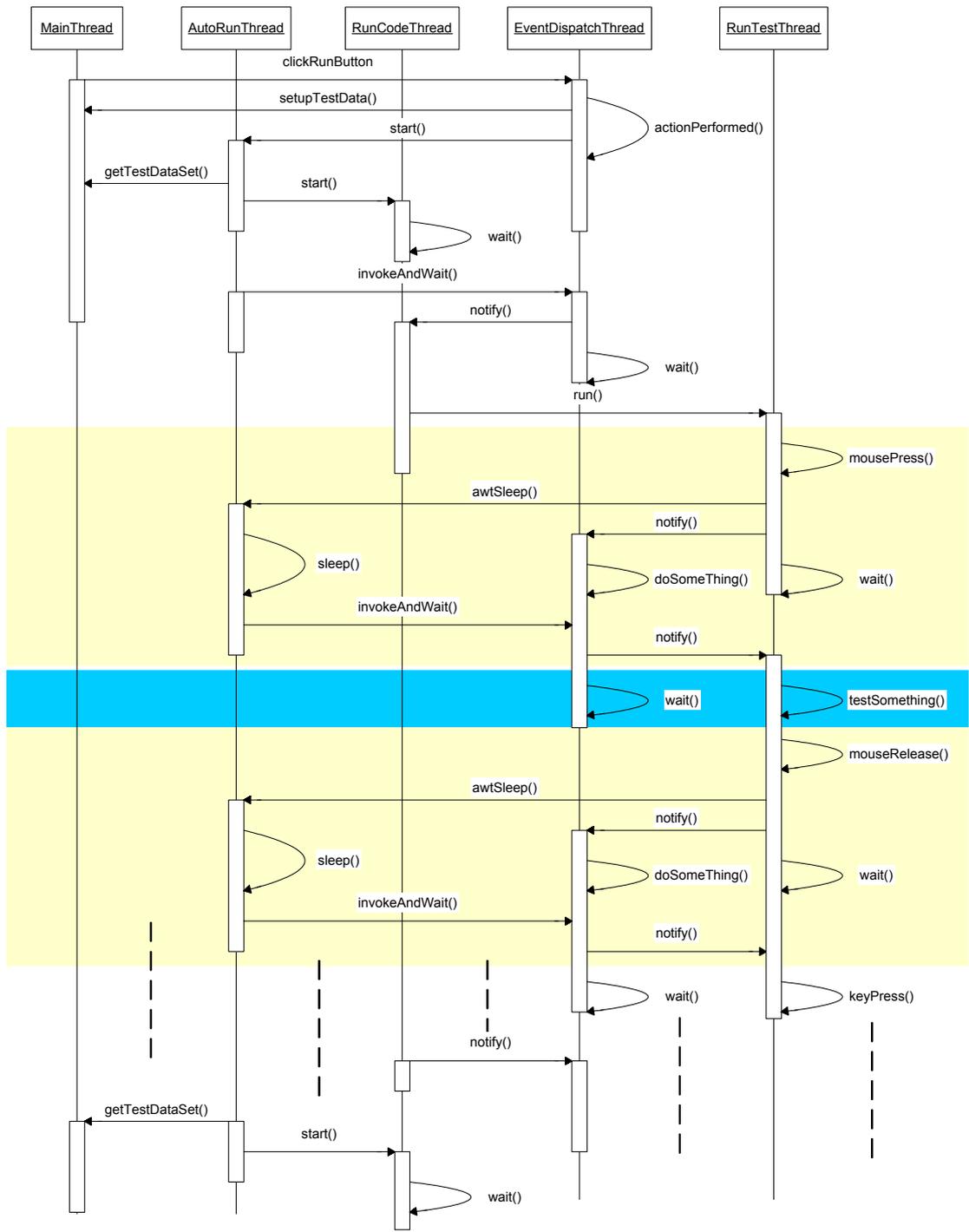


圖 2.23：錄製與播放循序圖

### 3. 設計方式(Design Issues and Solutions)

#### 3.1. 系統特性(Subsystem Characteristics)

本系統並沒有採用分割子系統之設計，關於主系統的功能與特性，請參照 1.1 小節。

#### 3.2. 建立技術解決方案選取條件(Establish Technical Solution Criteria)

本系統並沒有採用分割子系統之設計，關於主系統的建立技術解決方案選取條件，請參照 1.2 小節。

#### 3.3. 選取技術方案(Selected Subsystem Solution)

本系統並沒有採用分割子系統之設計，關於主系統的建立技術方案，請參照 1.3 與 1.4 小節。

#### 3.4. 偵錯與復原(Error Detection and Recovery)

- 待測應用程式未包含程式起始點之函式  
GTT 會偵測出並告知使用者待測程式不合規格。
- 未開啟待測程式即執行測試流程  
GTT 不會進行測試，且告知使用者必須先開啟待測程式。
- 另外，針對播放時所產生的錯誤部份，整理如下表：

錯誤發生情形	錯誤可能原因	錯誤訊息設定	錯誤情況處理
無法找到 AC 資訊所描述的「GUI 元件」。	<ul style="list-style-type: none"><li>● 元件被刪除</li><li>● 焦點視窗並非元件所在視窗</li></ul>	<ul style="list-style-type: none"><li>● 無法設定</li></ul>	<ul style="list-style-type: none"><li>● GTT 回應錯誤訊息給與元件節點。</li><li>● 將「元件節點」設為錯誤的燈號(紅燈)。</li><li>● 繼續下一個節點的執行。</li></ul>

<p>無法找到 AE 資訊所描述的「座標方法之相對元素」。</p>	<ul style="list-style-type: none"> <li>● 元素個數改變</li> <li>● 無法展開元素</li> </ul>	<ul style="list-style-type: none"> <li>● 使用者「延伸滑鼠抽象資訊類別」時設定。</li> <li>● GTT 內建之延伸滑鼠抽象資訊類別無法設定。</li> </ul>	<ul style="list-style-type: none"> <li>● GTT 回應錯誤訊息給與元件節點。</li> <li>● 將「清單上事件錯誤位置」與「元件節點」設為錯誤的燈號。</li> <li>● 繼續下一個節點的執行。</li> </ul>
<p>執行 AT 資訊指定之元件中方法呼叫，「結果與預期回傳值不符」，丟出例外。</p>	<ul style="list-style-type: none"> <li>● AP 本身錯誤</li> <li>● 預期回傳值錯誤</li> </ul>	<ul style="list-style-type: none"> <li>● 錯誤訊息由「測試方法加入對話框」的錯誤訊息欄位設定。</li> </ul>	<ul style="list-style-type: none"> <li>● GTT 回應錯誤訊息給與元件節點。</li> <li>● 將「清單上測試錯誤位置」與「元件節點」設為錯誤的燈號。</li> <li>● 繼續下一個節點的執行。</li> </ul>
<p>執行 AT 資訊指定之外部物件中方法呼叫，「方法中丟出例外」或「找不到這個方法」。</p>	<ul style="list-style-type: none"> <li>● AP 本身錯誤</li> <li>● 外部物件本身錯誤</li> </ul>	<ul style="list-style-type: none"> <li>● 錯誤訊息由「外部物件測試方法」利用 Assert 機制寫作。</li> </ul>	<ul style="list-style-type: none"> <li>● GTT 回應錯誤訊息給與元件節點。</li> <li>● 將「清單上測試錯誤位置」與「元件節點」設為錯誤的燈號。</li> <li>● 繼續下一個節點的執行。</li> </ul>

## 4. 系統細節與介面描述(Detailed of System and Interface Description)

### 4.1. 細部設計(Detailed Design)

GTT 細部分為五個模組(如圖 4.1)：攔截模組(Interceptor)、事件與測試模組(JE/AT)、編輯模組(Editor)、執行模組(Runner)、測試模組(Tester)。但是各個模組的功能卻有所變更，以下我們將對各個模組在 GTT 作詳細的說明與介紹：

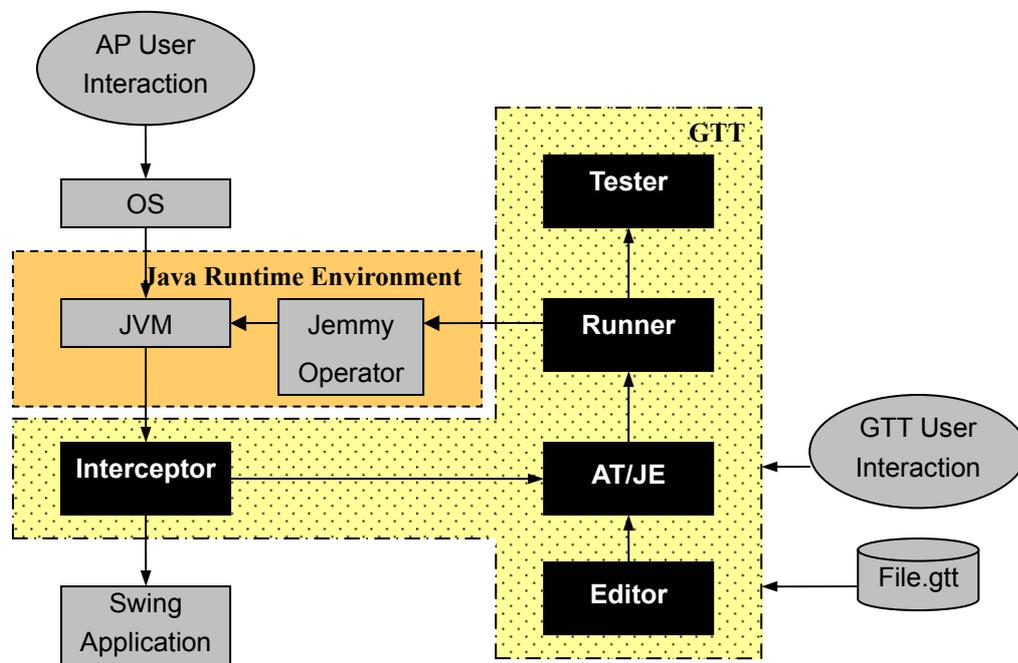


圖 4.1 GTT 模組架構圖

#### ➤ 攔截模組(Interceptor)：

攔截模組主要可以分為四個部分：「事件截取機制」、「事件抽象化轉換機制」、「事件層級轉換機制」、「事件類別轉換機制」，當使用者啟動錄影或攔截模式時，攔截模組便會啟動，錄影工作流程會如圖 4.2。

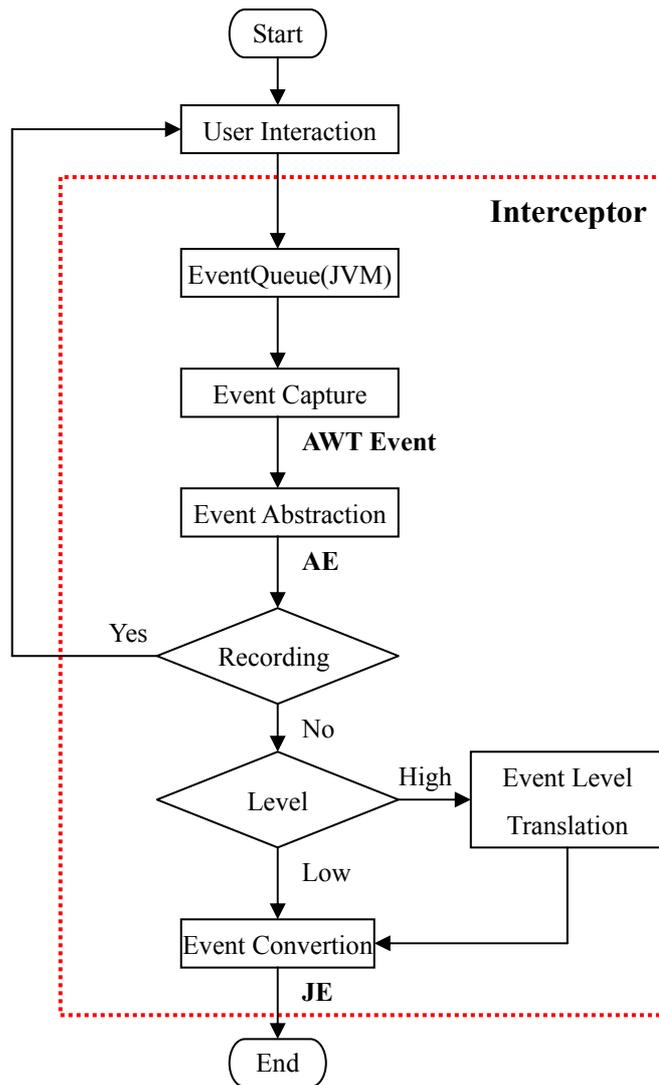


圖 4.2 攔截模組 — 錄影工作流程圖

1. **事件截取機制(Event Capture Mechanism)**：將原本使用者對待測程式所發出的事件流程截取下來。
2. **事件抽象化轉換機制(Event Abstraction Mechanism)**：將截取下來的 AWT 事件資訊轉換成為 AE 抽象資訊，並暫存這些資訊。
3. **事件層級轉換機制(Event Level Translation Mechanism)**：將儲存的 AE 事件作組合，將低階的連續 AE 事件轉換為高階的 AE 事件。
4. **事件類別轉換機制(Event Conversion Mechanism)**：將 AE 資訊轉換為相對應的 JE 事件。(詳見 3.3.2)

➤ 事件與測試模組(JE/AT)：

GTT<sup>2</sup> 的資料儲存核心，GTT<sup>2</sup> 的編輯對象，主要為「元件資訊」(表 4.1)、  
「Jemmy 事件資訊」(表 4.2)、「外掛測試資訊」(表 4.3)、「內部測試資訊」(表  
4.4)、「Configuration 資訊」(表 4.5)。

表 4.1 元件抽象資訊表

欄位說明	格式	例子
視窗類別	String	Javax.swing.JFrame
視窗標題	String	MainWindows
元件類別	String	Javax.swing.JButton
元件名稱	String	myButton
元件索引	Int	2
元件文字	String	OpenFile

表 4.2 Jemmy 事件抽象資訊表

欄位說明	例子
事件類別	MOUUSE_CLICK
事件閒歇時間	100 ms
事件座標	(10,20)
事件作用元件	見表 3.1 之元件資訊
事件模組	見表 3.8~3.10 之 Event Model

表 4.3 測試抽象資訊表－外掛測試點

欄位說明	例子
------	----

外掛測試類別	“testModel.CalculatorTest”
測試函數名稱	“testAddButton”

表 4.4 測試抽象資訊表－內部測試點

欄位說明	例子
測試元件	Javax.swing.JTextField，見表 3.x 之元件資訊
測試函數名稱	“getText”
測試函數參數型態	{Java.lang.Integer}
測試函數參數值	{new Integer(2)}
測試預期執行結果	“Hello world”
錯誤回傳訊息	“JTextField Display Error”

表 4.5 TC 抽象資訊表

欄位說明	例子
滑鼠雙擊靈敏度	500 ms
測試流程撥放速度	500 ms
待測程式視窗標題	“MyWindows”
待測程式視窗類別	Javax.swing.JFrame
待測程式啟動點	main(String args[])
啟動點傳入參數值	(預設為 null)
待測程式實體位置	main(String args[])

➤ **編輯資訊模組(Editor)：**

GTT 主要負責呈現與編輯測試流程的結構為「事件與測試模組」的資

訊，在編輯資訊模組中我們利用「樹狀結構」配合「清單結構」來表達一個測試流程的完整資訊，圖 4.3 表示在測試樹中的節點種類及其儲存的資訊。

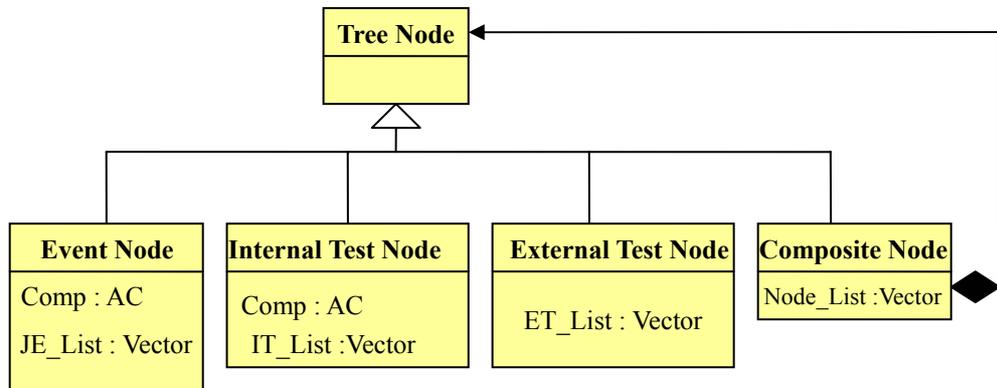


圖 4.3 GTT 的樹狀結構及事件與測試節點的說明

1. **樹狀結構編輯機制：**以樹狀的方式儲存、呈現及編輯測試流程，測試樹上的節點上主要可分為四類：
  - ◆ **事件節點：**儲存會儲存相關的 AC 資訊及 JE 事件清單。
  - ◆ **測試節點：**可分為兩類，一類用會來做內部測試，測試節點，會儲存 AC 資訊及內部測試點測試清單。
  - ◆ **複合節點：**相當於資料夾的，可包裝數個事件節點及測試節點。
2. **事件清單結構編輯：**GTT 利用清單結構的方式，來編輯在同一個事件節點上的「JE 資訊序列」或「AT 資訊序列」。GTT 的測試流程會以圖 4.4 的架構呈現。

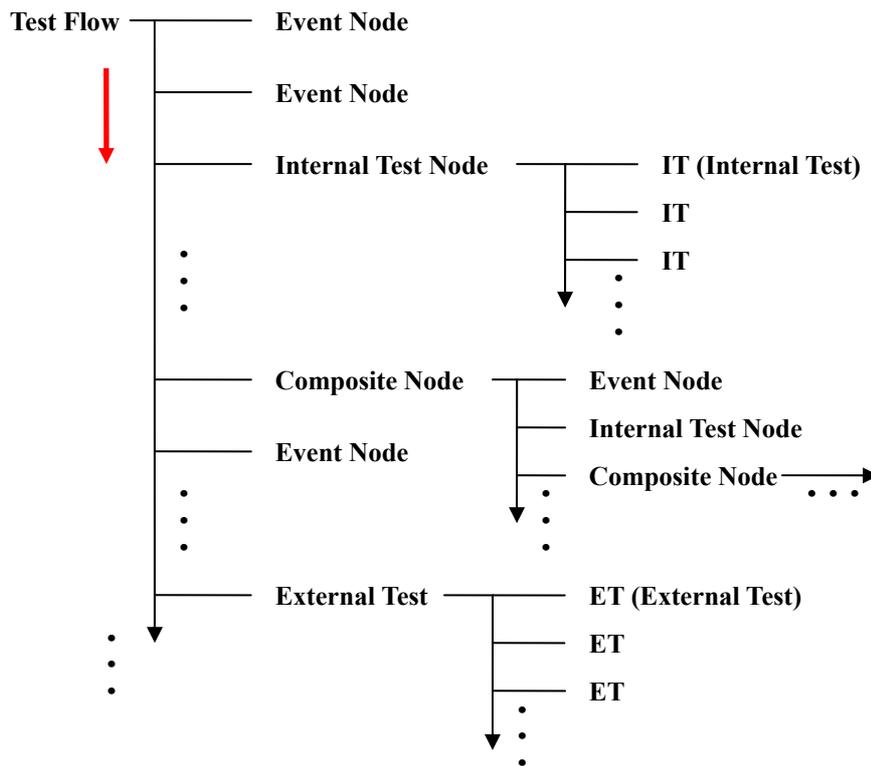


圖 4.4 GTT 中的測試流程表現方式

➤ **執行模組(Runner)：**

一個編輯好的測試流程，在 GTT 會以樹狀結構顯示，而執行測試時會以前序的方式執行測試流程中的每個節點。每個節點的執行包括四個步驟：找尋 GUI 元件、JE 資訊的發送、執行時間控制、執行時的錯誤回報的工作。由四個機制構成：「GUI 元件搜尋機制」、「事件執行機制」、「執行時間控制機制」、「執行錯誤回報機制」，執行測試的工作流程圖如圖 4.5：

1. **GUI 元件搜尋機制(Component Search Mechanism)：**利用樹狀結構上節點所包含的 GUI 元件抽象資訊 AC，在執行中的待測程式視窗畫面中找尋該 AC 資訊代表的 GUI 元件。
2. **節點執行機制(Node Execute Mechanism)：**依清單結構中的每個元素，利用元素所包含的 JE 資訊，轉換成 Jemmy Operator 所能接受的資訊，將資訊傳送給 Jemmy Operator，由 Jemmy Operator 執行發出使用者事件給待測程式。

3. **閒歇時間執行機制(Sleeptime Execute Mechanism)**：依測試流程中的群體閒歇時間加上每個 JE 事件所儲存的個別閒歇時間，來間隔每個 JE 事件。
4. **執行錯誤回報機制(Error Return Mechanism)**：當執行錯誤發生時，主動將錯誤的訊息回報給 GTT 主畫面。當錯誤個數大於使用者設定的連續錯誤個數時，此段落執行終止並執行下一個段落。

➤ **測試模組(Tester)：**

主要負責「測試的執行」，若有測試執行後產生的錯誤，需將錯誤資訊適時的回報。

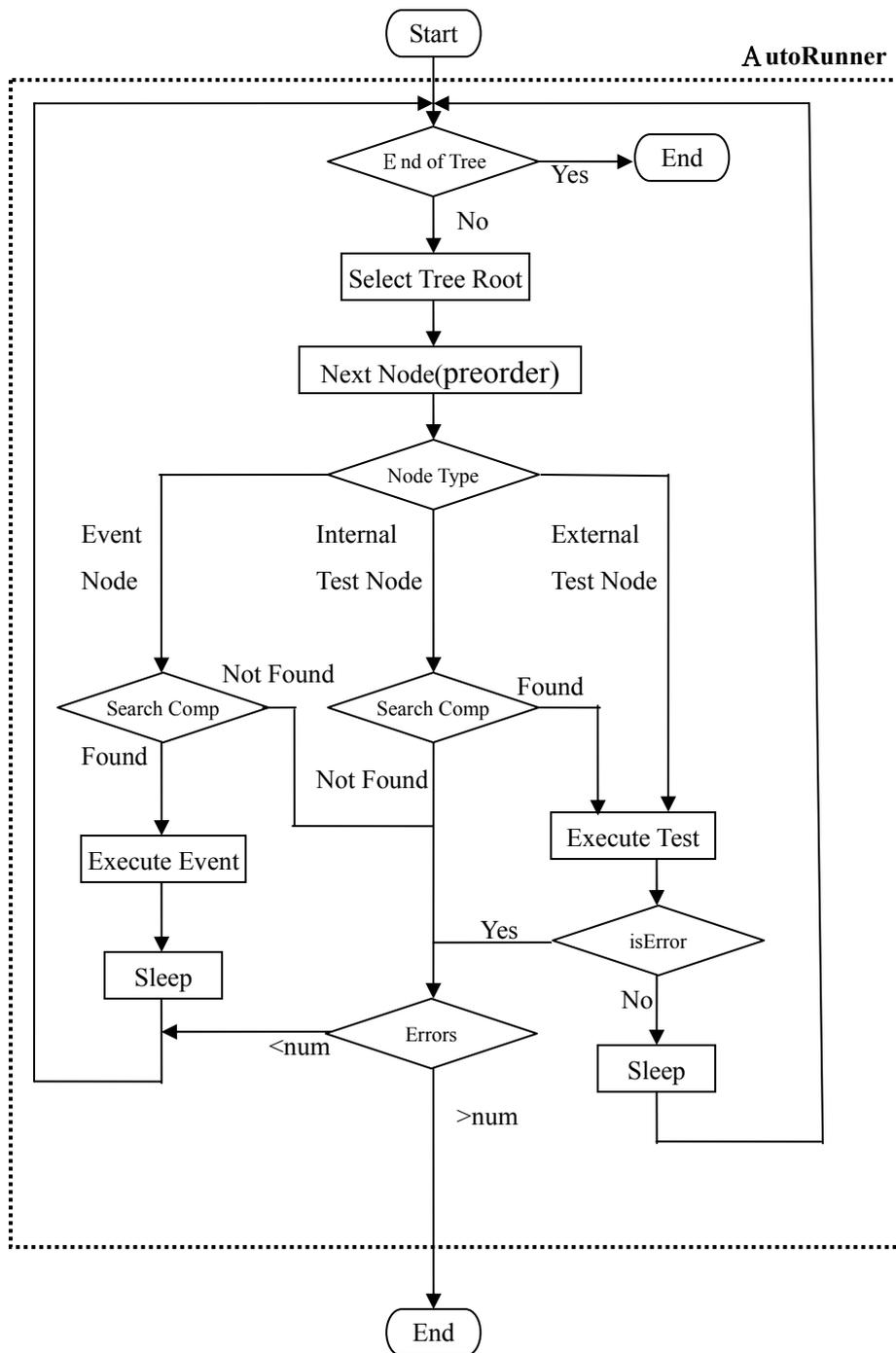


圖 4.5 GTT 執行模組工作流程

## 4.2. 系統介面與設計(System Interface Requirement and Design)

### 4.2.1. 內部介面

**內部介面 1** GTT 以繼承 class java.awt.EventQueue 的方式銜接 JDK v1.4，

藉以擷取使用者發出之事件：

**內部介面 2** 以高階抽象化截取時，GTT 必須將擷取之事件轉換為 Jemmy

API 所表達之事件：

在 GTT 中對滑鼠事件座標的抽象化，會針對 8 類元件做資訊的轉換，因此當我們在做事件的轉換時也須根據事件作用到的元件做轉換的處理，如表 4.6 由於 GTT 中的低階元件事件有保留 GTT 原先提供的抽象化事件，因此在轉換過程中我們不須對基本事件(M\_P、M\_R、K\_P、K\_R)做轉換的工作，只需要針對我們處理過的高階事件做轉換，而滑鼠中的雙擊事件我們只針對了 JList、JTable、JTree 及 AbstractButton 的事件做轉換，轉換的方式上基本與表 4.14 相同，只是在 JE 資訊中的滑鼠按鍵次數設為 2。

表 4.6 GTT<sub>2</sub> 的滑鼠事件轉換表

AE 事件	作用元件	JE 事件
CLICK_MOUSE	JTextComponent	JTextComponentEvent.CHANGE_CARET
CLICK_MOUSE	JList	JListEvent.CLICK_ON_ITEM
CLICK_MOUSE	JScrollBar	JScrollBar.SCROLL_TO_PERCENT
CLICK_MOUSE	JSlider	JScrollBar.SCROLL_TO_PERCENT
CLICK_MOUSE	JTable	JTableEvent.CLICK_ON_CELL
CLICK_MOUSE	JTabbedPane	JTabbedPaneEvent.SELECT_PAGE
CLICK_MOUSE	JTableHeader	JTableHeaderEvent.SELECT_COLUMN
CLICK_MOUSE	JTree	JTreeEvent.CLICK_ON_PATH
CLICK_MOUSE	AbstractButton 或 其他元件	AbstractButtonEvent.PUSH_NO_BLOCK

在 GTT 中鍵盤的抽象化只會針對事件發送對象屬於在 JTextComponent

的實體類別元件而做，因此我們所做的只需對發在 JTextComponent 實體類別元件的鍵盤事件做轉換即可，如表 4.7。

表 4.7 GTT 的鍵盤事件轉換表

AE 事件	作用元件	JE 事件
TYPE_KEY	JTextComponent	JComponentEvent.typeKey
TYPE_TEXT	JTextComponent	JTextComponentEvent.typeText

在 GTT 現有的結構中，具備著兩種不同層級的錄影方式，低階的錄影模式只是將錄影下來的 AWT 事件轉換為 JE 的低階元件事件，我們所看到的仍是比較低階的測試流程。高階的錄影模式在錄影結束時，會先經過抽象化處理，組合可以合併的連續事件之後，在將抽象化之後 AE 事件轉換為 Jemmy 的高階元件事件。

### 內部介面 3 GTT 使用 Jemmy API 播放所表達之事件：

在 GTT 中包含一組高階事件，這組高階事件以 Jemmy Operator 為基準，並將所有的事件依據發送的對象分為「基礎事件」與「元件事件」，說明如下：

- **基礎事件(Base Event)**：可以發送到所有 Swing 元件的事件。
- **元件事件(Component Event)**：針對各個元件的特性發展而出的事件，元件事件只能作用在特定的元件之上，如：JTable 存在著 row 與 column 的屬性，因此我們有所謂的 selectCell(row, col)的事件，但是在其他的元件並沒有 row 與 column，因此我們不能對其他元件發出 selectRow 的事件。在 GTT 中所提供的元件事件中可分為低階與高階兩類：
  - ◆ **低階元件事件**：凡是事件的發送對象專屬於特定元件且不可拆解的事件，這部分的事件主要是以 GTT 的 AE 事件作為參考所制定而

成。

- ◆ **高階元件事件**: 凡是事件的發送對象專屬於特定元件且可以拆解成低階事件的事件，這部分的事件主要是以 Jemmy Operator 為參考所制定而成。

表 4.8 是我們在 GTT 中所提供的基礎事件，在事件模組(Event Model)中的資訊表示為 JE 事件發送時所需要的資訊，也就是使用者在編輯事件時所需輸入的資料，在事件(Event)欄位中的參數即表示這個事件會使用的資料。

表 4.8 GTT 的基礎事件

Swing Component	Event	Event Model
JComponent	clickForPopup(x,y,button) clickMouse(x,y,count,button) dragMouse(x,y,button,modifier) enterMouse()、exitMouse() pressKey(keyCode,modifier) pressMouse(x,y)、releaseMouse(x,y) releaseKey(keyCode, modifier) typeKey(Char, Modifier)、 setTimeOut ( int )	int id int count int button int modifier int x, y int timeout int sleepTime char keyChar

在 Java Swing 當中，存在著某些抽象類別：如 JTextComponent，所有跟文字輸入相關的元件會繼承此類別。而我們對此抽象類別所能發送的所有元件事件都可以發送在繼承此抽象類別的元件之上。在表 4.9 中，整理了 GTT 所提供的抽象類別元件事件。(在 Swing Component 欄位中括號內的元件類別繼承自該抽象類別)

表 4.9 GTT 的抽象類別元件事件

Swing Component	Event	Event Model
<b>AbstractionButton</b> ( JButton、JCheckBox、 JMenuItem、	press() push() pushNoBlock()	

JToggleButton )	release()	
<b>JTextComponent</b> (JTextArea、JTextField、 JTextPane、JEditorPane、 JPasswordField)	changerCaretPosition(pos) clearText() enterText(text) scrollToPosition( pos) selectText(text,pos) typeText(text,pos)	<b>String</b> text <b>int</b> pos
<b>Window</b> (JDialog、JFrame)	activate() close() move(x,y) resize(width,height)	<b>int</b> width <b>int</b> height

在表 4.10 中所列的元件事件為 GTT 所提供的其他的元件事件，這些元件事件除了可以發送至各自對應的 Swing 元件外，同樣也保留了繼承的特性，亦可發送自繼承自該元件的使用者自訂元件。

表 4.10 GTT 的元件事件

Swing Component	Event	Event Model
<b>JFileChooser</b>	approve()、cancel() chooseFile(fileName) clickOnFile(index,count) clickOnFile(fileName) enterSubDir(dir). selectFile(fileName) selectFileType(filter) selectPathDirectory(dir)	<b>String</b> fileName <b>String</b> dir <b>String</b> filter
<b>JFrame</b>	maximize()、demaximize().	
<b>JTree</b>	callPopupOnPath(paths) clickForEdit(path) clickOnPath(path) doCollapsePath(path) doCollapseRow(row) doExpandPath(path) doExpandRow(row) scrollToPath(path)、scrollToRow(row) selectPaths(paths)、selectRow(row)	<b>Vector</b> paths

<b>JColorChooser</b>	enterColor(color) enterColor(R,G,B)	<b>Color</b> color <b>int</b> R,G,B
<b>JComboBox</b>	ClearText() 、 enterText(text) typeText(text) 、 selectItem(text) selectIndex(index)	<b>String</b> text
<b>JInternalFrame</b>	activate() 、 close() deiconify() 、 demaximize() maximize() 、 move(x,y) resize(width,height) scrollToFrame()	<b>int</b> width <b>int</b> height
<b>JList</b>	selectItem(items) clickOnItem(items,count) scrollToItem(items)	<b>Vector</b> items
<b>JMenuBar</b>	closeSubmenus() 、 pushMenu(path) pushMenuNoBlock(path) showMenuItem(path)	<b>String</b> path
<b>JPopupMenu</b>	callPopup(comp) 、 pushMenu(path) pushMenuNoBlock(path) showMenuItem(path)	<b>String</b> compName <b>String</b> path
<b>JScrollBar</b> <b>JSlider</b> <b>JSpinner</b>	scrollToMaximum() scrollToMin() scrollToValue(value)	<b>double</b> value
<b>JScrollPane</b>	scrollToButtom() scrollToComponent(comp) scrollToHorizontalValue(valueX) scrollToLeft() 、 scrollToRight() scrollToTop() scrollToVerticalValue(valueY) scrollToValue(valueX,valueY)	<b>double</b> valueX <b>double</b> valueY <b>compoment</b> comp
<b>JSplitPane</b>	expandLeft() 、 expandRight() moveDivider(location) moveToMaximum() moveToMinimum()	<b>double</b> location
<b>JTabbedPane</b>	selectPage(title)	<b>String</b> title
<b>JTable</b>	callPopupOnCell(row,col) clickForEdit(row,col) clickOnCell(row,col) scrollToCell(row,col)	<b>int</b> row <b>int</b> col

	selectCell(row,col)	
<b>JTableHeader</b>	moveColumn(moveFrom,moveTo) selectColumns(cols )	<b>Vector</b> cols
<b>JTextArea</b>	changerCaretPosition(row,col) changeCaretRow(row) selectText(sRow,sCol,eRow,eCol) typeText(text,row,col )	<b>int</b> sRow <b>int</b> sCol <b>int</b> eRow <b>int</b> eCol

#### 4.2.2. 外部介面

**外部介面 1** 以 Java swing API v1.4 來呈現 GTT 的使用者介面：

GTT 之使用者介面是使用 Swing 所設計。

**外部介面 2** GTT 使用 JDK v1.4 的檔案 API 儲存或載入測試事件流程檔案：

所有待儲存之元件類別皆繼承 Serializable 類別，並以 FileOutputStream 輸出成檔案。

**外部介面 3** GTT 要能讀取 JUnit 之測試案例檔案作為外部測試節點。

GTT 採用了 JUnit 單元測試的方式來撰寫外部測試節點(如圖 4.6)，但是有兩點與一般的單元測試可能有所不同：

- ✓ **SetUp 及 TearDown**：一般在寫單元測試時，我們必須自行將待測程式的實體(instance)在 SetUp 設定好，在 TearDown 釋放。但是由於 GTT<sub>2</sub> 會負責控管待測程式的實體，因此我們在撰寫外掛測試時，只需透過 GTT<sub>2</sub> 所提供的方法(表 4.11)取得目前執行中的待測程式即可，而不需在 SetUp 及 TearDown 中管控待測程式的實體。
- ✓ **測試內容**：在一般的單元測試中，會先設定待測程式輸入資訊，再執行預做測試的模組，等待執行過後再做驗證結果的動作。但是使用 GTT 作 GUI 測試時，待測程式的輸入就相當於事件流程，而事件流程的建立及執行在 GTT 中便已經被處理過了，因此我們在寫 GTT 的外掛測試時只需專注的做結果的比對，而不須撰寫程式產生事件流程。

表 4.11 GTT 提供外掛測試使用的靜態方法

靜態方法	功能
getRoot()	取得目前執行中的待測程式主視窗
getInstance()	取得目前執行中的待測程式實體
getComponent(Window, name)	Window 表示元件所在視窗，name 表示元件名稱，傳入這兩項資訊取得目前執行中的視窗元件

```

public class CalculatorTest extends TestCase
{
    private Window mainFrame = null ;
    private JTextField result = null ;
    private JLabel memory_Label = null ;
    public CalculatorTest(String name)
    {
        super (name);
    }
    public void testMValue ()
    {
        mainFrame = Tools.getRoot ();
        memory_Label = (JLabel)Tools.getComponent (mainFrame,"memoryLabel");
        assertEquals ("M Label Error!!", "M",memory_Label.getText ());
        assertEquals ("M Error!!", 12, ((Calculator)mainFrame).getValueMemory ());
    }
}

```

圖 4.6 GTT 的外掛測試—撰寫方式

GTT 根據單元測試的特性提供了兩類外掛測試的執行方式：「測試案例 (TestCase)」及「測試套件(TestSuite)」，如圖 4.7。「測試套件」是將數個單一測試合併，執行時會檢查測試套件內的所有單一測試，將單一測試加入「測試套件」的方式與 JUnit 單元測試的標準用法相同，在此不詳加介紹

```

//testing Memory Value
public void testMValue ()
{
    mainFrame = Tools.getRoot ();
    memory_Label = (JLabel) Tools.getComponent (mainFrame,"memoryLabel");
    assertEquals ("M Label Error!!", "M", memory_Label.getText ());
}

```

```

    assertEquals("M Error!!", 12, ((Calculator)mainFrame).getValueMemory());
}
public static TestSuite suite()
{
    TestSuite suite = new TestSuite("Test for Calculator");
    suite.addTest(new TestSuite(CalculatorTest.class));
    return suite;
}

```

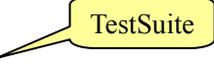


圖 4.7 GTT 的外掛測試種類

**外部介面 4** 待測應用程式若要使用 JUnit 之測試案例檔案作為外部測試節點，待測應用程式本身必須提供 `getInstance`，此函式負責傳回待測應用程式之實體：

設計方式同**外部介面 3**所述。

**外部介面 5** 待測應用程式若要使用 JUnit 之測試案例檔案作為外部測試節點，GTT 提供 `getInstance`、`getRoot` 與 `getComponent` 三個函式給 JUnit 之測試案例檔案中使用：

設計方式同**外部介面 3**所述。

**外部介面 6** 待測應用程式必須有程式起始點之函式：

待測程式必須有程式起點之函式，預設值為 `main` 函式。

## 5. 詞彙(Glossary)

1. **錄影/撥放工具(Capture/Replay Tool, 簡稱 CR)**: GUI 測試中一種普遍的測試方法, 利用錄影將使用者對應用程式的操作紀錄下來; 再利用撥放功能將錄影下來的使用者操作過程重現在應用程式上。
2. **單元測試**: 這邊所指的單元測試代表為使用 JUnit 所撰寫的測試案例 (TestCase) 或測試套件 (TestSuite)。
  - **測試案例(TestCase)**: 一個測試案例表示為對一個單一函式(Method) 所做的測試, 詳見[12]。
  - **測試套件(TestSuite)**: 一個測試套件表示為多個測試案例或測試套件的組合, 詳見[12]。
3. **事件(Event, 簡稱 e)**: 在本論文的事件指的是使用者對待測程式發出的滑鼠及鍵盤事件。在本論文中將事件分為高階及低階:
  - 高階事件: 事件可以在經過拆解的統稱之。
  - 低階事件: 不可拆解的事件統稱之。
4. **事件(Event, 簡稱 e)**: 在本論文的事件指的是使用者對待測程式發出的滑鼠及鍵盤事件, 在本論文中將事件分為高階事件及低階事件。
  - **低階事件**: 不可拆解的事件(例如 PRESS\_KEY, RELEASE\_KEY)。
  - **高階事件**: 由低階事件組合而成的事件(例如 TYPE\_KEY, TYPE\_KEY 為 1 次 PRESS\_KEY 加上一次 RELEASE\_KEY)。
5. **待測程式(Application Under Test)**: 在本論文中泛指 GTT 的測試對象—使用 Java Swing 元件所開發的 GUI 應用程式。
6. **事件流程(Event Flow)**: 一連串的連續事件  $E = \{ e_1, e_2, \dots, e_n \}$ 。
7. **測試點(Test Point, 簡稱 t)**: 在適當的事件  $e_i$  之後可插入測試點 t 以驗證執行結果。每個測試點儲存預期執行結果與待測程式相關資訊, 執行時比對預期結果與待測程式執行結果, 若不符則產生錯誤訊息。

8. **測試流程(Test Flow)**: 在本論文中所提到的測試流程指的是包含測試點的事件流程，

$$\text{Test Flow} = \{ E_1, t_1, E_2, t_2, \dots, E_n, t_n \}$$

9. **Jemmy**: Jemmy Module, 由 Sun 所開發的一套 API, 利用 Java Language 來開發 GUI 測試程式。具有強大的事件發送能力與可擴充的元件搜尋能力
10. **Jemmy Operator**: 屬於 Jemmy API 中的一個族系, 不同的 Operator 可以針對各自對應的 Swing 元件發出相對應的 Jemmy 事件。
11. **Robot**: 屬於 Java AWT 內的一個類別物件, 可以用來產生事件。
12. **EventQueue**: 與 Robot 同屬於 Java AWT 內, 使用者所有的事件在作用至待測程式前都會先被送至 EventQueue, 並由 EventQueue 在適當的時機發送。
13. **AWT 事件**: 在 EventQueue 中所收到的事件都會以 AWT 事件的方式送至待測程式。
14. **M\_P、M\_R、K\_P、K\_R**: 分別為 MOUSE\_PRESS、MOUSE\_RELEASE、KEY\_PRESS、KEY\_RELEASE 的縮寫。
15. **視窗(Window)**: 在 GTT 中所提到的視窗代表為繼承自 JFrame 或 JDialog 之類別, 亦包含 JFrame 及 JDialog 本身。
16. **外掛測試點(External Test Point, 簡稱 ET)**: 以 JUnit 單元測試的方式撰寫測試點, 經過 Java 編譯器編譯過後, 可以由 GTT 編輯器載入, 在撥放過程中執行, 這種以 JUnit 方式撰寫的測試節點稱為外掛測試點。
17. **內部測試點(Internal Test Point, 簡稱 IT)**: 透過 GTT 的編輯器編輯而成的測試點, 驗證的範圍只限於待測程式的 GUI 元件。
18. **元件資訊(Abstraction Component Information, 簡稱 AC)**: GTT 儲存一個 GUI 元件所使用的抽象資訊, 供事件發送與執行內部測試時使用。
19. **事件資訊(Abstract Event Information, 簡稱 AE)**: GTT 在錄影模式時

暫存的事件抽象化資訊。

20. **Jemmy 事件資訊(Jemmy Event Information, 簡稱 JE)**：在 GTT 中儲存一個 JE 事件的所使用的抽象資訊。
21. **Configuartion 資訊(Test Flow Configuration Information, 簡稱 TC)**：在 GTT 中用來執行測試流程所需的相關資訊(如：待測程式的類別、測試流程撥放速度)。
22. **測試資訊(Abstraction Test Information, 簡稱 AT)**：在 GTT 中儲存測試點所需要的抽象資訊。
23. **編輯模式**：當待測程式不存在時，使用者可以直接輸入元件資訊、Jemmy 事件資訊，建立事件流程。
24. **攔截模式**：當待測程式存在時，使用者透過元件選取機制選取待測程式的元件，自動擷取元件資訊，再選取並輸入 Jemmy 事件資訊。建立事件流程。
25. **錄影模式**：當待測程式存在時，使用者透過錄影機制擷取對待測程式發出的所有事件，並在錄影結束時轉換為事件流程。

## 6. 參考文獻(Reference)

1. Johan Andersson, Geoff Bache, The Video Store Revisited Yet Again: "Adventures in GUI Acceptance Testing, Lecture Notes in Computer Science", Springer-Verlag Heidelberg, Vol. 3092, 2004.
2. Kent Beck, Test-Driven Development By Example, New York: Addison Wesley, 2002, November 08
3. R.V Binder, "Design for Testability in Object-Oriented Systems", Communications of the ACM, Volume 37, No 9, 1994, pp. 87-101.
4. Jessica Chen, Subramaniam, Suganthan, "A GUI Environment to Manipulate FSMs for Testing GUI-based Application in Java," Proc. of the 34th Hawaii International Conference on System Sciences, 2001.
5. Jessica Chen, "Expressing Graphical User's Input for Test Specifications", Lecture Notes in Computer Science, Springer-Verlag Heidelberg, Vol. 2480, pp. 347, 2002.
6. Satadip Duta, "Abbot-A Friendly JUnit Extension for GUI Testing", Java Developer Journal, April 2003, pp 8~12.
7. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, Design Patterns, New York: Addison Wesley, 1995, January 15
8. GUI Testing Tool– Abbot: <http://abbot.sourceforge.net>
9. GUI Testing Tool– GUITAR: <http://www.cs.umd.edu/~atif/guitar.html>
10. IEEE Standard for Software Test Documentation, IEEE Std 829, 09/1998
11. Jemmy Module, <http://jemmy.netbeans.org>
12. JUnit Home, <http://www.junit.org/news/extension/gui/index.html>
13. Charles Lowell, "Jeremy Stell-Smith Successful Automation of GUI Driven Acceptance Testing", Lecture Notes in Computer Science, Springer-Verlag Heidelberg, Vol. 2675, pp. 331-333, 2003.
14. Atif M. Memon, A Comprehensive Framework for Testing Graphical User Interface, Ph.D. Thesis, University of Pittsburgh, PA, July 2001
15. Atif M. Memon, "GUI Testing : Pitfalls and Process", IEEE Computer, Volume: 35, Issue: 8, Aug. 2002, pp. 87-88
16. Atif.M. Memon, Pollack, M.E.; Soffa, M.L, "Hierarchical GUI test case generation using automated planning", IEEE Transactions on Software Engineering, Vol. 27, No. 2 , pp 144-155, 2001.
17. Atif M. Memon, Mary Lou, Soffa Martha, and E. Pollack, Coverage criteria for GUI testing, Proceedings of the 8th European software engineering conference held jointly with 9th ACM SIGSOFT international symposium

- on Foundations of software engineering, vol. 26, pp. 256 - 267, 2001
18. J. Newmarch, , "Testing Java Swing-based applications", In the 31st International Conference on Technology on Object-Oriented Language and Systems, Nanjing, China, September 1999.
  19. Ostrand, Thomas, Anodidi, Aaron, Foster, Herber; Goradia, and Tarak, "A Visual Test Development Environment for GUI Systems", Proceedings of ACM SIGSOFT international symposium on Software testing and analysis (ISSTA), Vol. 23, 1998.
  20. J.E Payne, R.T Alexander and C.D. Hutchinson, "Design-for-Testability for Object-Oriented Software", Object Magazine, 1997, 7(5): 34 -43
  21. John Steven, "jRapture:A Capture/Replay Tool for Observation-Based testing", August 2000 ACM SIGSOFT Software Engineering Notes, Proceedings of the International Symposium on Software Analysis, Volume 25 Issue 5.
  22. Yanhong Sun and Edward L. Jones, "Specification-Driven Automated Testing", Proceedings of the 42nd annual Southeast regional conference, Alabama,USA. 2004 April 2-3, pp 140-145
  23. Hong Zhu, "Software Unit Test Coverage and Adequacy", ACM Computing Surveys, Vol. 29, No.4, December 1997.

## 附錄一 類別列表(Class List)

Package	Class Name
(default package)	AbstractTest
(default package)	AutoRunner
(default package)	ComponentNotFoundException
(default package)	ComponentNotShowingException
(default package)	ComponentToString
(default package)	ContainerToString
(default package)	GTT
(default package)	Index
(default package)	Interceptor
(default package)	JComponentToString
(default package)	JLabelToString
(default package)	TestApp
(default package)	TestListener
(default package)	TestStringGenerator
(default package)	TestToString
command	AbstractCommand
command	AddNodesCommand
command	AddRestartNodeCommand
command	AddStateCommand
command	CommandManager
command	ConvertCommand
command	CopyCommand
command	CopyProxyCommand
command	CutCommand
command	DeleteCommand
command	Document
command	DocumentCtrl
command	DownNodeCommand
command	InsertCommand
command	InsertNodesCommand
command	ModifyCommand
command	NewFileCommand
command	OpenFileCommand
command	PasteCommand

command	PasteProxyCommand
command	Redo
command	RedoCommand
command	SaveAsFileCommand
command	SaveFileCommand
command	SetColorCommand
command	SetNameCommand
command	Undo
command	UndoCommand
command	UnProxyCommand
command	UpNodeCommand
data	AbstractData
data	ComponentData
data	CompositeData
data	DataGroup
data	RestartData
data.check	Check
data.check	CheckClassMethod
data.check	CheckDatas
data.check	CheckMethod
data.check	ComponentCheckData
data.check	CenterMouseEventData
data.check	ComponentEventData
data.check	CompositeEventData
data.check	DefaultMouseEventData
data.check	EventDatas
data.check	InputEventData
data.check	JListMouseEventData
data.check	JTabbedPaneMouseEventData
data.check	JTableHeaderMouseEventData
data.check	JTableMouseEventData
data.check	JTextMouseEventData
data.check	JTreeMouseEventData
data.check	KeyEventData
data.check	MouseEventData
data.check	SleepEventData
data.check.componentEvent	AbstractButtonComponentEventData

data.check.componentEvent	JButtonComponentEventData
data.check.componentEvent	JCheckBoxComponentEventData
data.check.componentEvent	JColorChooserComponentEventData
data.check.componentEvent	JComboBoxComponentEventData
data.check.componentEvent	JComponentEventData
data.check.componentEvent	JDialogComponentEventData
data.check.componentEvent	JEditPaneComponentEventData
data.check.componentEvent	JFileChooserComponentEventData
data.check.componentEvent	JFrameComponentEventData
data.check.componentEvent	JInternalFrameComponentEventData
data.check.componentEvent	JLabelComponentEventData
data.check.componentEvent	JListComponentEventData
data.check.componentEvent	JMenuBarComponentEventData
data.check.componentEvent	JMenuComponentEventData
data.check.componentEvent	JMenuItemComponentEventData
data.check.componentEvent	JPasswordFieldComponentEventData
data.check.componentEvent	JPopupMenuComponentEventData
data.check.componentEvent	JProgressBarComponentEventData
data.check.componentEvent	JRadioButtonComponentEventData
data.check.componentEvent	JScrollBarComponentEventData
data.check.componentEvent	JScrollPaneComponentEventData
data.check.componentEvent	JSliderComponentEventData
data.check.componentEvent	JSpinnerComponentEventData
data.check.componentEvent	JSplitPaneComponentEventData
data.check.componentEvent	JTabbedPaneComponentEventData
data.check.componentEvent	JTableComponentEventData
data.check.componentEvent	JTableHeaderComponentEventData
data.check.componentEvent	JTextAreaComponentEventData
data.check.componentEvent	JTextComponentEventData
data.check.componentEvent	JTextFieldComponentEventData
data.check.componentEvent	JTextPaneComponentEventData
data.check.componentEvent	JToggleButtonComponentEventData
data.check.componentEvent	JTreeComponentEventData
data.check.componentEvent	WindowComponentEventData
ui	AppCtrl
ui	AppFileChooser
ui	AppPane

ui	ComponentTreeNode
ui	CompositeDataNode
ui	Controller
ui	DetailTable
ui	InfoListElement
ui	MainUI
ui	SingleWordTextField
ui	TestConfigurationView
ui	TestFileChooser
ui	TestFileFilter
ui	TestFileView
ui	TestGeneratorUI
ui	TestListCellRenderer
ui	TestListPanel
ui	TestTree
ui	TestTreeEditor
ui	TestTreeRenderer
ui	TreePopupMenu
ui	UIFactory
ui.dialog	AddCheckClassMethodDialog
ui.dialog	AddCheckDialog
ui.dialog	AddEventDialog
ui.dialog	AddKeyEventDialog
ui.dialog	AddSleepEventDialog
ui.dialog	ModifyAbstractButtonEventDialog
ui.dialog	ModifyCheckClassMethodDialog
ui.dialog	ModifyCheckMethodDialog
ui.dialog	ModifyComponentDataDialog
ui.dialog	ModifyEventDataDialog
ui.dialog	ModifyJColorChooserEventDialog
ui.dialog	ModifyJComboBoxEventDialog
ui.dialog	ModifyJComponentEventDialog
ui.dialog	ModifyJDialogEventDialog
ui.dialog	ModifyJFileChooserEventDialog
ui.dialog	ModifyJFrameEventDialog
ui.dialog	ModifyJInternalFrameEventDialog
ui.dialog	ModifyJListEventDialog

ui.dialog	ModifyJMenuBarEventDialog
ui.dialog	ModifyJMenuEventDialog
ui.dialog	ModifyJPopupMenuEventDialog
ui.dialog	ModifyJScrollBarEventDialog
ui.dialog	ModifyJScrollPaneEventDialog
ui.dialog	ModifyJSliderEventDialog
ui.dialog	ModifyJSpinnerEventDialog
ui.dialog	ModifyJSplitPaneEventDialog
ui.dialog	ModifyJTabbedPaneEventDialog
ui.dialog	ModifyJTableEventDialog
ui.dialog	ModifyJTableHeaderEventDialog
ui.dialog	ModifyJTextAreaEventDialog
ui.dialog	ModifyJTextComponentEventDialog
ui.dialog	ModifyJTreeEventDialog
ui.dialog	ModifyKeyEventDataDialog
ui.dialog	ModifyMouseEventDataDialog
ui.dialog	ModifySleepEventDialog
ui.dialog	SelectComponentsDialog
ui.model	CheckTableModel
ui.model	ComponentTableModel
ui.model	EventTableModel
ui.model	MyMethod
ui.model	TestTreeModel
ui.model	TypeTableModel
UnitTest	AllTests
UnitTest	ComponentEventDataTest
UnitTest	ModifyKeyEventDataDialogTest
UnitTest	TestGeneratorUITest
UnitTest	ToolsTest
util	Accessor
util	AppPropertiesLoader
util	DialogFactory
util	ExitActionListener
util	FileClassLoader
util	KeyConverter
util	LoadedClassMethods
util	MethodComparator

util	MultiClassLoader
util	MyRobot
util	ObjectCloner
util	PropertiesLoader
util	TestFileController
util	Tools
util	TreeBuilder
util	URLClassLoader

## 附錄二 系統模組與類別回溯表

	Interceptor	AT/AE	Editor	Runner	Tester
AbstractTest				↙	↙
AutoRunner				↙	↙
ComponentNotFoundException			↙	↙	↙
ComponentNotShowingException			↙	↙	↙
ComponentToString			↙		
ContainerToString			↙		
GTT	↙	↙	↙	↙	↙
Index		↙	↙		
Interceptor	↙				
JComponentToString		↙			
JLabelToString		↙			
TestApp					↙
TestListener					↙
TestStringGenerator					↙
TestToString					↙
AbstractCommand			↙		
AddNodesCommand			↙		
AddRestartNodeCommand			↙		
AddStateCommand			↙		
CommandManager			↙		
ConvertCommand			↙		
CopyCommand			↙		
CopyProxyCommand			↙		

CutCommand			↳		
DeleteCommand			↳		
Document			↳		
DocumentCtrl			↳		
DownNodeCommand			↳		
InsertCommand			↳		
InsertNodesCommand			↳		
ModifyCommand			↳		
NewFileCommand			↳		
OpenFileCommand			↳		
PasteCommand			↳		
PasteProxyCommand			↳		
Redo			↳		
RedoCommand			↳		
SaveAsFileCommand			↳		
SaveFileCommand			↳		
SetColorCommand			↳		
SetNameCommand			↳		
Undo			↳		
UndoCommand			↳		
UnProxyCommand			↳		
UpNodeCommand			↳		
AbstractData		↳	↳	↳	↳
ComponentData		↳	↳	↳	↳
CompositeData		↳	↳	↳	↳

DataGroup		↵	↵	↵	↵
RestartData			↵	↵	
Check				↵	↵
CheckClassMethod				↵	↵
CheckDatas				↵	↵
CheckMethod				↵	↵
ComponentCheckData				↵	↵
CenterMouseEventData	↵	↵		↵	
ComponentEventData	↵	↵		↵	
CompositeEventData	↵	↵		↵	
DefaultMouseEventData	↵	↵		↵	
EventDatas	↵	↵		↵	
InputEventData	↵	↵		↵	
JListMouseEventData	↵	↵		↵	
JTabbedPaneMouseEventData	↵	↵		↵	
JTableHeaderMouseEventData	↵	↵		↵	
JTableMouseEventData	↵	↵		↵	
JTextMouseEventData	↵	↵		↵	
JTreeMouseEventData	↵	↵		↵	
KeyEventData	↵	↵		↵	
MouseEventData	↵	↵		↵	
SleepEventData	↵	↵		↵	
AbstractButtonComponentEventData	↵	↵		↵	
JButtonComponentEventData	↵	↵		↵	
JCheckBoxComponentEventData	↵	↵		↵	

JColorChooserComponentEventData	↵	↵		↵	
JComboBoxComponentEventData	↵	↵		↵	
JComponentEventData	↵	↵		↵	
JDialogComponentEventData	↵	↵		↵	
JEditPaneComponentEventData	↵	↵		↵	
JFileChooserComponentEventData	↵	↵		↵	
JFrameComponentEventData	↵	↵		↵	
JInternalFrameComponentEventData	↵	↵		↵	
JLabelComponentEventData	↵	↵		↵	
JListComponentEventData	↵	↵		↵	
JMenuBarComponentEventData	↵	↵		↵	
JMenuComponentEventData	↵	↵		↵	
JMenuItemComponentEventData	↵	↵		↵	
JPasswordFieldComponentEventData	↵	↵		↵	
JPopupMenuComponentEventData	↵	↵		↵	
JProgressBarComponentEventData	↵	↵		↵	
JRadioButtonComponentEventData	↵	↵		↵	
JScrollBarComponentEventData	↵	↵		↵	
JScrollPaneComponentEventData	↵	↵		↵	
JSliderComponentEventData	↵	↵		↵	
JSpinnerComponentEventData	↵	↵		↵	
JSplitPaneComponentEventData	↵	↵		↵	
JTabbedPaneComponentEventData	↵	↵		↵	
JTableComponentEventData	↵	↵		↵	

JTableHeaderComponentEventData	↵	↵		↵	
JTextAreaComponentEventData	↵	↵		↵	
JTextComponentEventData	↵	↵		↵	
JTextFieldComponentEventData	↵	↵		↵	
JTextPaneComponentEventData	↵	↵		↵	
JToggleButtonComponentEventData	↵	↵		↵	
JTreeComponentEventData	↵	↵		↵	
WindowComponentEventData	↵	↵		↵	
AppCtrl			↵		
AppFileChooser			↵		
AppPane			↵		
ComponentTreeNode		↵			
CompositeDataNode		↵			
Controller	↵	↵	↵		
DetailTable			↵		
InfoListElement			↵		
MainUI			↵		
SingleWordTextField			↵		
TestConfigurationView			↵		
TestFileChooser			↵		
TestFileFilter			↵		
TestFileView			↵		
TestGeneratorUI	↵		↵		
TestListCellRenderder			↵		

TestListPanel			←		
TestTree			←		
TestTreeEditor			←		
TestTreeRenderer			←		
TreePopupMenu			←		
UIFactory			←		
AddCheckClassMethodDialog			←		
AddCheckDialog			←		
AddEventDialog			←		
AddKeyEventDialog			←		
AddSleepEventDialog			←		
ModifyAbstractButtonEventDialog			←		
ModifyCheckClassMethodDialog			←		
ModifyCheckMethodDialog			←		
ModifyComponentDataDialog			←		
ModifyEventDataDialog			←		
ModifyJColorChooserEventDialog			←		
ModifyJComboBoxEventDialog			←		
ModifyJComponentEventDialog			←		
ModifyJDialogEventDialog			←		
ModifyJFileChooserEventDialog			←		
ModifyJFrameEventDialog			←		
ModifyJInternalFrameEventDialog			←		
ModifyJListEventDialog			←		
ModifyJMenuBarEventDialog			←		

ModifyJMenuEventDialog			↵		
ModifyJPopupMenuEventDialog			↵		
ModifyJScrollBarEventDialog			↵		
ModifyJScrollPaneEventDialog			↵		
ModifyJSliderEventDialog			↵		
ModifyJSpinnerEventDialog			↵		
ModifyJSplitPaneEventDialog			↵		
ModifyJTabbedPaneEventDialog			↵		
ModifyJTableEventDialog			↵		
ModifyJTableHeaderEventDialog			↵		
ModifyJTextAreaEventDialog			↵		
ModifyJTextComponentEventDialog			↵		
ModifyJTreeEventDialog			↵		
ModifyKeyEventDataDialog			↵		
ModifyMouseEventDataDialog			↵		
ModifySleepEventDialog			↵		
SelectComponentsDialog			↵		
CheckTableModel			↵		
ComponentTableModel			↵		
EventTableModel			↵		
MyMethod			↵		
TestTreeModel			↵		
TypeTableModel			↵		
AllTests			↵		↵
ComponentEventDataTest			↵		↵

ModifyKeyEventDataDialogTest			↵		↵
TestGeneratorUITest			↵		↵
ToolsTest			↵		↵
Accessor			↵		
AppPropertiesLoader			↵		
DialogFactory			↵		
ExitActionListener			↵		
FileClassLoader			↵		
KeyConverter		↵	↵		
LoadedClassMethods			↵	↵	↵
MethodComparator			↵	↵	↵
MultiClassLoader			↵	↵	↵
MyRobot				↵	↵
ObjectCloner				↵	↵
PropertiesLoader				↵	↵
TestFileController				↵	↵
Tools			↵		
TreeBuilder			↵		
URLClassLoader				↵	↵

### 附錄三 系統模組與需求回溯表

	Interceptor	AT/AE	Editor	Runner	Tester
GTT001				↙	↙
GTT002	↙	↙	↙		
GTT003				↙	
GTT004		↙			
GTT005		↙			
GTT006			↙		
GTT007			↙		
GTT008			↙		
GTT009			↙		
GTT010			↙		
GTT011			↙		
GTT012			↙		
GTT013			↙		
GTT014			↙		
GTT015			↙		
GTT016			↙		
GTT017			↙		
GTT018			↙		
GTT019			↙		
GTT020			↙	↙	
GTT021	↙	↙			
GTT022		↙			↙
GTT023				↙	↙
GTT024			↙		
GTT025			↙		
GTT026			↙		
GTT027	↙	↙	↙	↙	↙
GTT028	-	-	-	-	-
GTT029	-	-	-	-	-
GTT030	↙				
GTT031	↙	↙			
GTT032		↙		↙	
GTT033			↙		
GTT034			↙		

GTT035			↵	↵	
GTT036				↵	↵
GTT037				↵	↵
GTT038				↵	↵
GTT039				↵	↵
GTT040	↵	↵	↵	↵	↵
GTT041			↵	↵	↵
GTT042			↵	↵	↵
GTT043			↵		
GTT044	↵	↵	↵	↵	↵
GTT045					↵
GTT046	↵	↵			
GTT047	↵	↵			
GTT048			↵	↵	
GTT049			↵	↵	