物聯網中介軟體：複雜事件處理引擎
IoT Middleware: Complex Event Processing

李允中	吳佳芷	任哲晨
Jonathan Lee	Chia-Chih Wu	Che-Chen Jen
國立台灣大學資訊工程學系	國立台灣大學資訊工程學系	國立台灣大學資訊工程學系
jlee@csie.ntu.edu.tw	r03944010@csie.ntu.edu.tw	r04944041@ntu.edu.tw

摘　要

近年來，關於物聯網的研究與日俱增、相關應用設備與系統也漸漸出現在市面上，然而，現今物聯網應用系統發展中面臨了幾個問題:缺乏複雜事件之概念,無法有效分析由感測設備來的資料成為人類所能理解*的事件。為解決這些問題，在此多年計劃中我們將提出一個複雜事件處理系統,處理從感測設備收集到的資料，將其轉換為事件，再經由複雜事件處理引擎利用事件模式轉換為複雜事件，最後將複雜事件轉換為服務，使事件成為商業流程的元件,最後以應用情境驗證從資料到服務之流程,此外使用者也能自行定義事件模式並應用於不同情境。

本研究實作複雜事件處理引擎：使用者可定義複雜事件模式，同時複雜事件處理系統將 gateway 收集之感測器資料轉換為事件輸入處理引擎，引擎則採用樹結構方法偵測出複雜事件。

關鍵字：物聯網中介軟體、複雜事件處理系統

一、總論

近年來，關於物聯網的研究與日俱增、相關應用設備與系統也漸漸出現在市面上，然而，現今物聯網應用系統發展中面臨了幾個問題:缺乏複雜事件之概念,無法有效分析由感測設備來的資料成為人類所能理解的事件。為解決這些問題，在此多年計劃中我們將提出一個複雜事件處理系統,處理從感測設備收集到的資料，將其轉換為事件，再經由複雜事件處理引擎利用事件模式轉換為複雜事件，最後將複雜事件轉換為服務,使事件成為商業流程的元件,最後以應用情境驗證從資料到服務之流程,此外使用者也能自行定義事件模式並應用於不同情境。

本研究開發複雜事件處理引擎：使用者可定義複雜事件模式，同時複雜事件處理系統將 gateway 收集之感測器資料轉換為事件輸入處理引擎,引擎則採用樹結構方法偵測出複雜事件。

為了解決目前物聯網應用系統開發所面臨的問題，我們採用複雜事件處理的概念，將龐大的感測數據轉換為人類容易理解的資訊。我們提出一個基於物聯網中介軟體的複雜事件處理系統,以處理分析由下層感測設備收集而來的資訊,將之轉換為事件,傳入複雜事件處理引擎，同時使用者可自行設計複雜事件模式,描述欲偵測的複雜事件。藉由以上功能,除了可對大量感測數據資料進行過濾分析,也可增進物聯網應用系統開發的速度與彈性。

二、相關研究

本章節將介紹國內外相關文獻包括：事件型別與複雜事件處理引擎。在複雜事件處引擎中，包含利用非決定性有限狀態自動機、具緩衝之非決定性有限狀態自動機以及利用樹結構的處理引擎。

2.1 事件型別(event type)

以下為兩份文獻對於事件型別的簡單介紹,[2]對事件型別以及其內容做基本的定義；論文[6]則更深入提出事件型別化的概念和模型,進一步提出屬性型別以及事件型別的繼承關係。

[2]對事件型別的定義:事件型別是對一群擁有相同意義與結構的事件物件的規範，事件物件(event object)可被視為一個事件型別的實體(instance)。一個事件擁有三種屬性:Header、Payload

* 本研究接受科技部編號: MOST 104-2622-E-002-030-CC1 研究計畫經費補助

以及 Open content。Header 屬性包含事件的 meta-information，例如事件發生的時間；payload 屬性記錄該事件特有的資訊，例如網購事件中的訂單編號以及商家名稱；而事件也可用自由格式儲存 open content。

論文[6]介紹結構化與型別化事件的概念和方法，以及提出事件的元模型。其提出的概念為：事件的定義以事件物件型別儲存於事件型別知識庫，事件物件即是該型別的實體。事件物件型別可繼承自其他型別，可包含多種屬性，每一屬性會對應到屬性型別，屬性型別可為基本型別(single-value type)、集合型別(collection type)或是字典型別(dictionary type)。以訂單事件為例，訂單事件包含四種屬性：顧客編號、訂單編號、下單時間以及產品集合，前三者為基本型別，後者則是集合型別，包含一組商品事件的集合。

2.2 複雜事件處理引擎：利用非決定性有線狀態自動機方法

一個事件模式可以對應至一個非決定性有限狀態自動機(NFA、Nondeterministic Finite State Automata)[4]，假設一個事件模式描述一種事件類型 a 和一種複雜事件類型 E，非決定性有限狀態自動機可以塑模為圖 1 所示，其中*表示萬用字元，可以匹配任何事件類型。

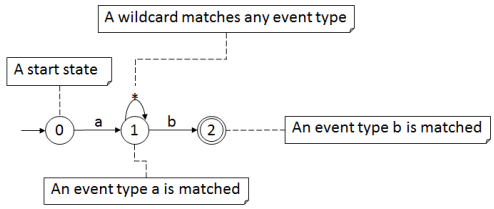


圖 1、非決定性有限狀態自動機

序列掃描與子序列建構主要是從一串事件序列中，根據事件模式找出符合模式的事件子序列，假設輸入事件序列為<a1, c1, a2, b2>，如圖 2，事件模式同樣是描述事件 a 與事件 b，該演算法首先在輸入事件序列的元素之間設置狀態 0，根據狀態轉換表導出狀態 1、狀態 2 與狀態之 z 的轉換。

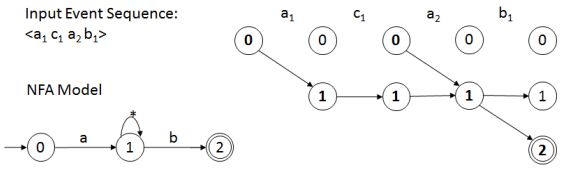


圖 2、序列掃描與子序列構建

我們從接受狀態，即狀態 2，透過深度優先演算法可以找出兩組狀態轉換，如圖 3 所示，我們只收集不同狀態轉換的字母，可以找出<a1, b1>及<a2, b1>兩個子序列。

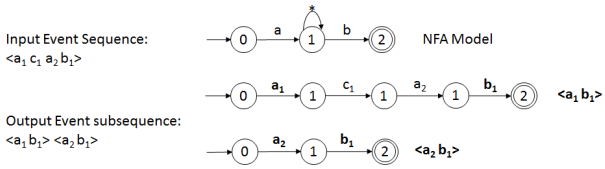


圖 3、子序列之建構

此方法的問題是無法對應至具有條件的事件模式：例如一個事件模式描述事件 a 與事件 b，以及其條件 a.x > 0, b.x > 0, b.x - a.x < 5。如圖 4。

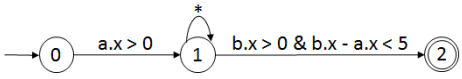


圖 4、加入條件判斷之非決定性有限狀態自動機

假設我們修改 NFA 的轉換函數，使其可以輸入一個帶有條件字母。當我們執行序列掃描與子序列之建構時，先匹配 a1 與 a2 事件，當判斷 b1 是否符合條件時，該演算法無法判斷 b1 應該跟 a1 還是 a2 比對才算符合條件。如圖 5 所示。

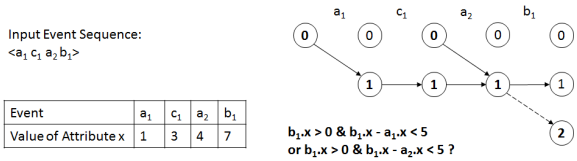


圖 5、無法處理的事件序列

2.3 複雜事件處理引擎：具緩衝之非決定性有限狀態自動機(Buffered NFA)

另一種非決定性有限狀態機 NFAb (Nondeterministic Finite State Automata buffer)，其中 b 意味著緩衝(buffer)，用來記錄事件序列中符合事件模式的事件元素。

一個帶有條件的事件模式可以對應至一個 NFAb，假設一個事件模式描述事件 a 與事件 b，滿足條件：a.x > 0, b.x > 0, b.x - a.x < 5，如圖 6。

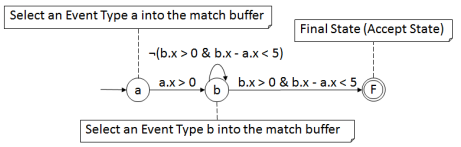
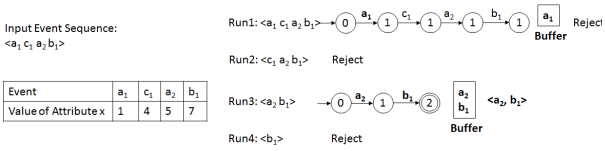


圖 6、將事件模式對應至具緩衝之非決定性有限狀態自動機

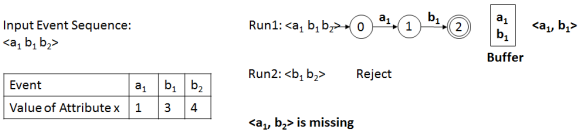
將輸入事件序列拆分為多個子序列，如圖 7 所示，在把子序列丟入 NFAb 分析，並且使用一個緩衝器(Buffer)紀錄符合事件模式之元素。事件序列為<a1, c1, a1, b1>，使用上述事件模式，只有<a2, b1>符合事件模式。



Event	a_1	c_1	a_2	b_1
Value of Attribute x	1	4	5	7

圖 7、具緩衝之非決定性有限狀態自動機與多重運行

NFAb 的問題在於某些情況下無法找出所有的事件子序列，假設輸入事件為<a1, b1, b2>，只有找出<a1, b1>，遺漏<a1, b2>。如圖 8 所示。



Event	a_1	b_1	b_2
Value of Attribute x	1	3	4

圖 8、無法找出所有子序列

2.4 複雜事件處理引擎：利用樹方法

利用樹方法的複雜事件處理引擎 ZStream[3]，可以用來和非決定性有限狀態自動機比較。

論文[3]中提到：複雜事件處理系統搜尋事件序列偵測複雜事件模式的發生，目前大部分的複雜事件處理系統採用順序查詢(Sequential query)的方法，也就是根據事件發生的時間排序，例如：偵測汽車的移動、偵測股票異常、網路監控等。然而，純粹只用順序查詢並不足以表達真實世界的複雜事件，例如 conjunction(同時發生的事件)、disjunction(只有兩個之中的其中一個)以及 negation(排除特定事件)，而這些因素都加深了複雜事件處理的困難。

目前的複雜事件處理系統大多採用非決定性有限狀態自動機方法，然而此種方法存在著一些先天上的問題：

第一項問題為固定順序評估：由於非決定性有限狀態自動機本身的限制，採用有序狀態來表達事件模式，因此非決定性有限狀態自動機的方法只能根據其設計好的自動機採用固定順序評估複雜事件模式。然而，事件模式通常有不只一種順序評估方式，彼此之間效能也有差異，因此，比起採用固定順序評估，如何選擇好的評估方法以達最大效率為複雜事件處理系統必須考量的重點。

第二項問題為排除事件：在非決定性有限狀態自動機中處理排除事件並不直覺，特別是有值的比較時，舉例：定義一個事件模式為「A 事件之後發生 C 事件(不用連續)，而中間不能有 B 事件」，此事件模式雖然能被非決定性有限狀態自動機所偵測，但若在事件模式中加入 B 事件與 C 事件屬性比較(例如：B 的價錢大於 C 的價錢)的條件，則無法利用非決定性有限狀態自動機偵測此事件模式。由於這項限制，目前非決定性有限狀態自動機處理排除事件的方法是加入事後過濾的步驟。

第三項問題為同時發生的事件：由於狀態有序性，非決定性有限狀態自動機難以處理併發事件。

由於非決定性有限狀態自動機的諸多限制，[8]提出以樹為主要結構的複雜事件處理系統—ZStream，此系統嘗試解決上述所提出非決定性有限狀態自動機的問題與限制，其特色包含：以樹結構為主的複雜事件處理系統、尋找事件模式的最佳計畫、可以處理併發事件、可以處理排除事件以及能夠適時改變事件模式的評估計畫。Zstream 之樹結構如圖 9。

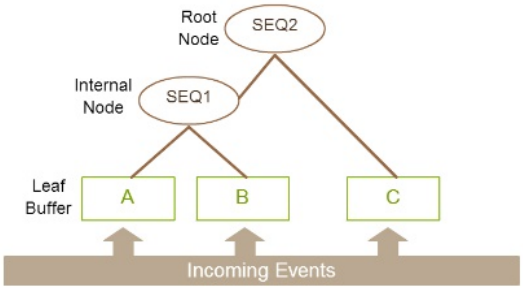


圖 9 、樹結構之複雜事件處理引擎結構

首先將詢問表達式(query expression)轉換為樹形式。葉節點緩衝器(leaf buffer)儲存原始事件，內部節點緩衝器(internal node buffer)儲存從其子樹內部產生的結果，每一個內部節點和運算子以及事件的數值有關。系統假設資料會根據時間從葉節點進

入，並採用批次迭代模型處理新來的原始事件並執行內部節點的運算，最後在根節點產生所求之複雜事件。

在樹結構的每一個節點都有一個緩衝器，對於葉節點用於儲存新來的事件；對於內部節點則儲存中途產生的結果。每一個緩衝器儲存數筆紀錄，每筆紀錄包含事件指標向量以及事件起始時間，並以結束時間為順序儲存。

在事件的處理上提出批次迭代模型，可分為兩部分：閒置回合(idle round)與組合回合(assembly round)。閒置回合負責累積新進事件到樹節點，當最後一個事件緩衝器至少有一項事件時，進入組合回合，組合回合負責運算填入內部節點緩衝器並產生結果。

Zstream 能夠處理五項運算子，包含序列(sequence)、排除(negation)、同時(conjunction)、擇一(disjunction)與 Kleene closure。每一項運算子皆有其對應的演算法，對左子樹事件緩衝與右子樹事件緩衝進行運算，並將結果儲存至該運算子的內部節點緩衝區。

三、研究方法

圖 10 為系統架構圖，上方部分為使用者端：事件模式規範以及事件模式設計工具；下方部分為後端複雜事件處理系統，包含資料到事件之轉換、複雜事件處理引擎以及事件到服務轉換。在此研究中我們定義基礎複雜事件模式規範、將感測數據轉換為事件以及實作複雜事件處理引擎，接下來針對各模組進行說明。

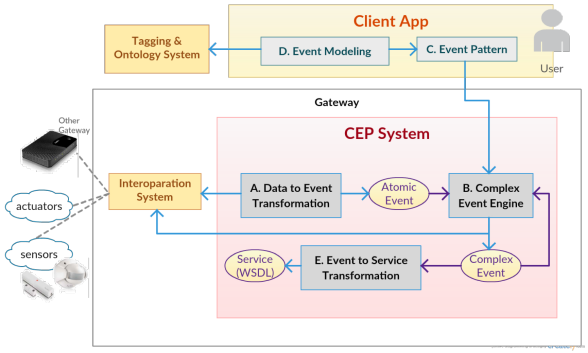


圖 10、系統架構圖

3.1 定義與設計複雜事件模式

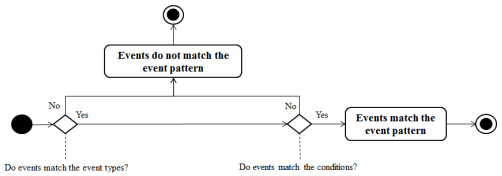


圖 11 、如何用事件模式找出特定事件集合

對於一個複雜事件模式，必須定義其參數，常見的有事件模式結構、事件屬性比較、事件屬性限制、複雜事件時間限制以及輸出結構等。藉由這些參數的設定，描述欲偵測的複雜事件，並交由複雜事件處理引擎，最後複雜事件處理引擎根據輸出結構產生該複雜事件，一個簡單的流程如圖 11。

定義複雜事件的屬性和方法，開發者可利用事件模式設計應用所需的複雜事件模式，再將複雜事件模式交由複雜事件引擎偵測複雜事件的發生。事件模式的能力仰賴複雜事件引擎所支援的功能，因此複雜事件模式與複雜事件處理引擎息息相關。

由於複雜事件模式的設計與複雜事件處理引擎有關：有定義好的複雜事件模式才能交由複雜事件處理引擎，而複雜事件模式的能力將依賴複雜事件處理引擎的強度，兩者相輔相成。因此此步驟的重點為：根據複雜事件處理引擎的能力定義複雜事件模式能夠處理的情況。另一個重點為參考複雜事件處理引擎相關研究的複雜事件模式，找出一個易用好學或被廣泛所知的表達方法，直接採用或適度修改。以下舉例兩種複雜事件模式的表達方法：

在[5]中的事件模式表達方法為：

[FROM <input stream>]
PATTERN <pattern structure>
[WHERE <pattern machine structure>]
[WITHIN<time constraints>]
[HAVING <pattern filtering condition>]
RETURN <output specification>

在[3]中的事件模式表達方法為：

PATTERN Composite Event Expressions
WHERE Value Constraints
WITHIN Time Constraints
RETURN Output Expression

以上兩個例子皆是 SQL-like 的事件模式語言，廣泛應用在複雜事件處理系統中，本系統參考與擷取其模式結構，整理成表 1。

表 1、複雜事件模式結構

Pattern Name	對複雜事件命名
Pattern Structure	事件的組合結構
Value Constraints	事件屬性數值限制
Time Constraint	複雜事件時間限制

Pattern name 描述使用者對該複雜事件的命名；pattern structure 描述事件間的組合結構，表達方式為將事件用運算子連結，例如甲事件需發生在乙事件之後；value constraints 描述事件的屬性數值限制，例如溫度事件數值大於 20 度；time constraints 描述複雜事件的時間限制，限制以上條件必須在幾秒鐘之內發生。

表 2 為一簡單的範例。該範例描述一個嬰兒起床事件：當五秒內 motion sensor 事件的值大於零且 sound sensor 事件的值大於 100 時，即代表發生嬰兒起床事件，可以解釋為睡覺的嬰兒起床後翻身且哭鬧。事件模式結構中的符號"&"是"同時運算子(conjunction operator)"，其意義為 motion sensor 事件和 sound sensor 事件都要發生但無順序限制。

表 2、複雜事件模式範例

Pattern Name	Baby wake up
Pattern Structure	Motion Sensor & Sound Sensor
Value Constraints	Motion Sensor>0 AND Sound Sensor>100
Time Constraints	5 secs

事件模式結構中，必須藉由運算子將事件組織成事件模式結構，表 3 列出本系統支援的五種運算子。

藉由以上規範，使用者或開發者在開發相關物聯網應用時，可以依規範自行設計複雜事件模式，而這些複雜事件模式之後會傳入本系統的複雜事件處理引擎進行偵測。

表 3、運算子列表

運算子	符號	描述
Sequence(序列)	A ; B	B 事件在 A 事件後發生
Negation(排除)	!A	A 事件未發生
Conjunction(同時)	A & B	A 事件和 B 事件都發生（順序不限）
Disjunction(擇一)	A | B	A 事件或 B 事件發生
Kleene closure(多個)	A	A 事件發生多次

3.2 感測數據獲取與轉換

本小節旨在將底層數據資料轉換為事件，例如將感測器所讀到的數值轉換成事件，傳入複雜事件處理引擎。

為了收集感測數據，我們利用物聯網中介軟體的 Interoperation 系統獲取感測器資料，interoperation 系統位於 smart gateway，interoperation 系統可與不同廠牌以及不同通訊協定的感測設備連接，並將不同的感測數據轉換成一致的資料格式，包含感測器編號、感測器名稱、資料型態、資料單位和資料數值，表 4 為一由 Phigets 設備中溫度感測器獲取的感測器資料例子。

表 4、感測數據資料格式範例

設備編號	Fc03cd40-2052-4ebe-9325-c4d2b6709d63
設備名稱	Phigets_11250_HUM_TEMP
感測資料型態	Temperature
感測資料單位	Celsius
感測資料數值	25.7

在數據轉換步驟，我們根據事件的元模型，將感測數據轉換為事件，圖 12 為元事件、感測數據事件以及複雜事件類別圖。元事件儲存事件基礎資訊；而事件類別是根據不同事件類型延伸出不同的事件類別，例如感測事件類別根據從 smart gateway 獲取的感測數據格式做設計；複雜事件類別則採用 composite design pattern 設計。在複雜事件處理系統中，每秒會向 interoperation 系統獲取最新的感測數據，並將之轉換成事件，作為複雜事件處理引擎的新進事件。

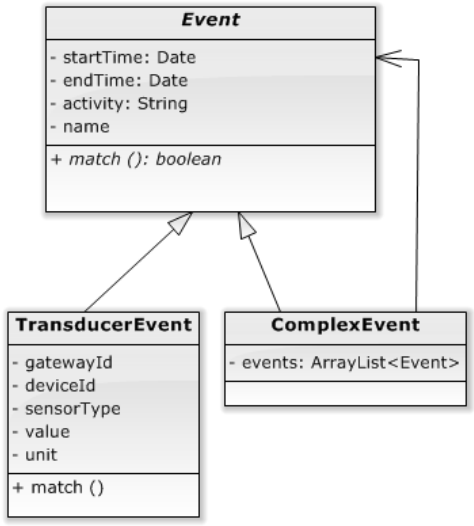


圖 12、事件類別圖

3.3 複雜事件處理引擎：利用樹方法

為了處理使用者設計之事件模式，複雜事件處理引擎首先將事件模式的結構(pattern structure)轉換為樹形式。葉節點緩衝器(leaf node buffer)儲存原始事件，內部節點緩衝器(internal node buffer)儲存從其子樹內部產生的結果，每一個內部節點和運算子以及事件的數值有關。

在樹結構的每一個節點都有一個緩衝器，對於葉節點用於儲存新來的事件；對於內部節點則儲存中途產生的結果。每一個緩衝器儲存數筆紀錄，每筆紀錄包含事件指標向量以及事件起始時間，並以結束時間為順序儲存。

從事件模式之詢問轉化為事件模式之樹結構，一個簡單的例子如圖 13。而根據不同的詢問，會產生不同的節點，種類共有：SEQ 節點、NSEQ 節點、CONJ 節點、DISJ 節點、以及 KSEQ 節點。每一個節點均有其特定之演算法。例如 SEQ 節點的演算法參考圖 13：

演算法解釋：輸入為左右緩衝區與最早可允許時間，並將結果輸出至結果緩衝區。針對每一個右緩衝區的事件，都要對每一個左緩衝區的事件進行屬性數值比較，如果符合，將此二事件加入結果緩衝區，並將右緩衝區清除。注意在提取緩衝區內的事件時，會先和 EAT(最早可允許時間)比較，若事件的時間早於最早可允許時間，代表不符合時間限制，則移除該事件，不必再繼續執行。

Input: right and left child buffers *RBuf* and *LBuf*, *EAT*
Output: result buffer *Buf*
foreach *Rr* ∈ *RBuf* **do**
if *Rr.start-ts* < *EAT* **then** remove *Rr*; continue;
for *Lr* = *LBuf*[0]; *Lr.end-ts* < *Rr.start-ts*;*Lr*++ **do**
if *Lr.start-ts* < *EAT* **then** remove *Lr*; continue;
if *Lr and Rr satisfy value constraints* **then**
Combine *Lr* and *Rr* and insert into *Buf*
Clear RBuf

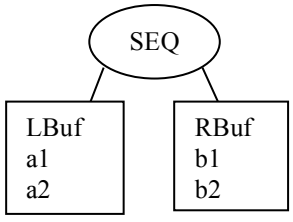


圖 13、樹結構 SEQ 節點之演算法

當樹建構好之後，接下來將進行一般事件輸入與事件樹的走訪。以下為較詳細的說明：

系統假設資料會根據時間從葉節點進入，並採用批次迭代模型處理新來的原始事件並執行內部節點的運算，最後在根節點產生所求之複雜事件。

採用批次迭代模型進行樹的走訪。批次迭代模型分為兩部分：閒置回合(idle round)與組合回合(assembly round)。閒置回合負責累積新進事件到樹節點，當最後一個事件緩衝器至少有一項事件時，進入組合回合，組合回合負責運算填入內部節點緩衝器並產生結果。批次迭代模型有四個步驟：步驟 1. 批次讀入一般事件至葉節點；步驟 2. 如果最後一個葉節點是空的(代表沒有事件)，繼續執行步驟 1，否則執行步驟 3；步驟 3. 計算最早允許時間，並從根節點傳到每一個緩衝器；步驟 4. 根據不同運算子的實作方式，將事件從葉節點組合至根節點，中途會儲存中間結果至相對應之節點緩衝器，以及去除掉不符合時間限制(早於最早允許時間)的事件。

藉由批次迭代模型，能夠從葉節點開始走訪整棵樹直到根節點並產生最終複雜事件的結果。此外，採用批次迭代模型與提早去除不符時間限制的事件，能夠有效率的使用記憶體。

複雜事件處理引擎之系統架構可分為三大模組：事件模式分析模組；樹結構模組與複雜事件偵測模組。

事件模式分析模組負責分析使用者所設計之複雜事件模式，包含事件模式結構、事件屬性限制、時間限制，建構方式為將事件結構轉換為樹結構並設定運算子，之後設定事件屬性數值和時間限制。

樹結構模組描述樹結構的內部設計，包含根節點(root node)、中間節點(internal node)、葉節點(leaf node)和其相對應的記錄緩衝器(record buffer)與屬性數值限制判斷(predicate)，以及實作五種演算子演算法的運算子類別。圖 14 為樹結構模組的部分類別圖，葉節點與中間節點繼承節點類別，而根節點繼承中間節點並記錄於樹類別中，每個節點有自己的記錄緩衝器(record buffer)以及屬性判斷類別(predicate)，在屬性判斷類別中，我們利用

JEval[7]程式庫進行變數代換並進行數值比對，JEval 程式庫可對數學、布林以及函數表達式進行解析和評估。

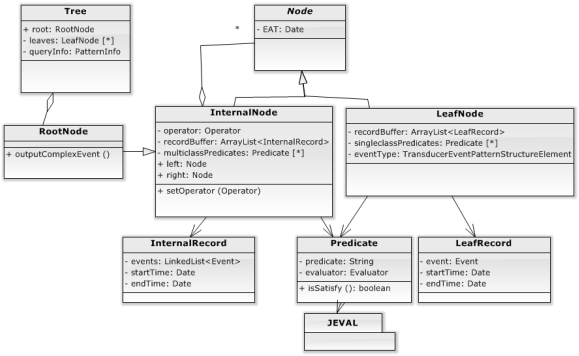


圖 14、樹結構模組之類別圖(局部)

複雜事件偵測模組負責偵測複雜事件的發生，一執行緒獲取數據資料，將之轉換為事件並傳入批次迭代模型的閒置回合進行累積，同時另一執行緒執行批次迭代模型（Batch iterator）演算法偵測複雜事件。圖 15 為複雜事件偵測模組部分類別圖，當複雜事件偵測類別呼叫開始偵測方法時，複雜事件偵測器執行緒(ComplexEventDetectorThread)與事件獲取執行緒(EventRetrieverThread)皆會被執行，複雜事件偵測器執行緒進行批次迭代模型演算法，而事件獲取執行緒則藉由 interoperation 系統獲取 gateway 上的感測數據事件，獲取的事件會被放入事件序列(eventQueue)中，同時複雜事件偵測器執行緒會從同一事件序列拿取事件，該事件序列以阻塞隊列(ArrayBlockingQueue)實作，阻塞隊列可用來解決多執行緒下的生產者消費者問題，對應到本系統，生產者為事件獲取執行緒，消費者為事件偵測器執行緒，放置的物件為事件而放置物件的空間為阻塞隊列。

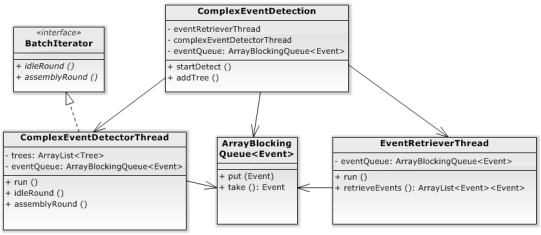


圖 15、複雜事件偵測模組類別圖(局部)

四、研究結果

4.1 系統測試

在實作複雜事件處理引擎的過程中，我們針對主要功能及元件使用 JUnit 撰寫測試，用以確認程式的運作結果是否符合預期，執行測試時使用 JaCoCo[7]工具計算代碼涵蓋率。測試的結果在主要負責複雜事件偵測的樹結構模組代碼涵蓋率為 92%，未來將以代碼涵蓋率 100%為目標，提升系統之穩定性與品質。

4.2 情境驗證

本步驟除了進一步探討複雜事件處理系統的應用情境，同時也將針對這些應用情境進行系統驗證。每一個應用情境，皆有其對應的感測設備以及複雜事件模式，因此在驗證時將實際裝置感測設備，讀取感測設備資料及複雜事件模式至複雜事件處理系統，最後將產生的複雜事件結果與預期結果比較，包含欲偵測之複雜事件是否有確實被偵測，其數量、時間、屬性數值是否正確等等。

以表 2 所舉之嬰兒起床事件為例，佈置一空間並放置 smart gateway 與所需之設備和感測器，使用者根據複雜事件模式規範設計嬰兒起床之事件模式傳送至 gateway 的複雜事件處理引擎，此時 gateway 上的複雜事件處理引擎開始獲取傳感器資訊，之後一名人員模擬嬰兒起床之情境（移動以及發出聲音）以達嬰兒起床事件之設定條件，此時複雜事件處理引擎成功偵測出嬰兒起床之複雜事件，在異地的使用者接收到該事件發生，可觀看事件記錄和即時畫面，示意圖如圖 16。

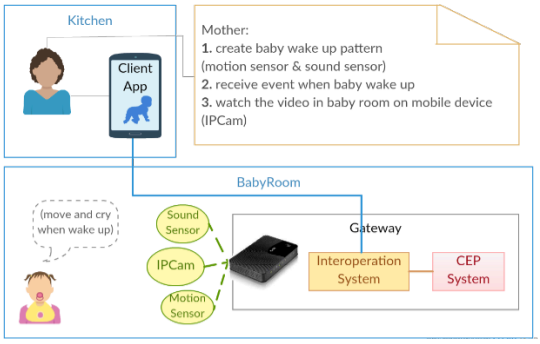


圖 16、嬰兒起床事件之情境驗證

4.3 效能分析

在此部分我們針對複雜事件處理引擎進行效能分析。

4.3.2 複雜事件處理引擎

複雜事件處理引擎的效能指的是該引擎每一秒能夠處理多少筆事件，事件模式的多寡以及複雜度皆會影響能夠處理的事件數，表 5 為本效能測試實驗中之事件模式，以真實歷史感測資訊做修改模擬獲取之感測事件，實驗方法為讀入十萬筆模擬感測資料並轉換為事件，存入複雜事件處理引擎之事件序列，實驗結果為一秒鐘能處理約 40,000 筆的事件(計算的方法為將處理的事件數量除以經過時間)，足以處理由 interoperation 子系統中獲取之數據；使用 java 內建 runtime 類別與 JConsole 計算記憶體用量，讀完感測數據十萬筆時使用 80MB，複雜事件處理引擎運行時記憶體用量則逐步減少最後降至 5MB 左右；實驗環境為雙核心 CPU 2.7GHz Intel Core i5 與 8GB RAM 的設備。

表 5、效能實驗測試之事件模式

事件模式名稱	測試事件模式 1	測試事件模式 2
模式結構	Motion Sensor & Sound Sensor	Temperature; Humidity; Motion Sensor
屬性限制	Motion Sensor>0 Sound Sensor>90	Temperature<21 Humidity>40 Motion Sensor>0
時間限制	1 sec	2 secs

五、結論

本研究為開發物聯網中介軟體之複雜事件處理系統，主要專注於複雜事件處理引擎之設計與實作，並對複雜事件模式進行規範以及獲取 gateway 上之感測數據。複雜事件處理引擎採用樹結構方法，並實作五種運算子進行運算；複雜事件模式規範包含事件名稱、模式結構、屬性限制和時間限制；另外利用物聯網中介軟體之 interoperation 層之技術獲取 gateway 上的感測數據資料，轉換為事件作為複雜事件處理引擎之事件來源。

實驗結果可分為情境驗證與效能分析兩部分。情境驗證以一智慧家庭的例子，模擬與偵測嬰兒起床事件；而效能分析則計算感測數據獲取與複雜事件處理引擎的吞吐量。

參考資料

[1] D. C. Luckham, The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems, Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA ©2001, 2001.

[2] O. Etzion and P. Niblett, “Defining the events,” in *Event Processing in Action*, 2010, pp. 61-85.

[3] Y. Mei and S. Madden, "ZStream: a cost-based query processor for adaptively detecting composite events," presented at the International Conference on Management of Data, 2009.

[4] E. Wu, Y. Diao, and S. Rizvi, "High-performance complex event processing over streams," in SIGMOD, 2006.

[5] J. Agrawal, Y. Diao, D. Gyllstrom, and N. Immerman, "Efficient pattern matching over event streams," in SIGMOD, 2008.

[6] S. Rozsnyai, J. Schiefer, and A. Schatten, "Concepts and models for typing events for event-based systems," in *Inaugural international conference on distributed event-based systems*, 2007.

[7] *JaCoCo.* EclEmma, 2015.

[8] *JEval.* Robert Breidecker. 2013.

[9] C. Alexander, S. Ishikawa, and M. Silverstein, A pattern language: towns, buildings, construction vol. 2, Oxford University Press, 1977.

[10] E. Gamma, Design patterns: elements of reusable object-oriented software: Addison-Wesley Professional, 1995.

[11] A. R. Graves and C. Czarnecki, “Design patterns for behavior-based robotics,” IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans, pp. 36-41, 2000.

[12] J. Zhu and P. Jossman, “Application of design patterns for object-oriented modeling of power systems,” IEEE Transactions on Power Systems, pp. 532-537, 1999.

[13] C. Blilie, “Patterns in scientific software: an introduction,” Computing in Science & Engineering, vol. 4, pp. 48-53, 2002.

[14] B. U.-L. M. P. a. W. F. T. L. Prechelt, “Two controlled experiments assessing the usefulness of design pattern documentation in program maintenance,” IEEE Transactions on Software Engineering vol. 28, pp. 595-606, 2002.

[15] B. Huston, “The effects of design pattern application on metric scores,” Journal of Systems and Software, vol. 58, pp. 261-269, 2001.