第五章

系統設計

在這一章主要是介紹本論文所設計的新信賴模型模擬器。主要分為兩個部分，第一部分我們將概略介紹模擬器的功能；第二部份則是執行簡單的測試，驗證模擬器的正確性。

5.1. 設計目的

本文所描述的新信賴模型，是基於社會學的行動理論與人際關係環境理論所建立起來的新信賴關係，希望可以透過建立節點之間的合作競爭 (信賴路徑與轉手路徑，請詳見本文第三章。)，提昇整體小世界的公共生產力並對節點的詐欺行為出現有效的排擠效應，讓小世界中的每一個節點都能趨良逐劣。為了能得到連續且全面的觀察，我們需要一個可以允許 「動態模擬」 的模擬器。我們定義，所謂的連續且全面的觀察，乃指我們期望觀察到的信賴關係是由一連串不斷合作與競爭建立起來的，也就是每一個加入小世界的個體都會不斷的與其他同樣在小世界中的個體發生互動，接受被評價或是評價他人，而這些評價最後將對整體小世界的公共生產力產生影響。所以如果僅單靠觀察某一個靜止的時間點，或是僅僅單憑初始時的信賴度關係，是無法得知小世界的公共生產力將會提升或是下滑。

但是我們調查目前已經問市的模擬器後發現，不論是其他研究信賴度模型的

論文用以佐證其論點的模擬器，或是市面上類似的商用模擬器 (例如交通大學所研發的 「NCTUns 3.0 Network Simulator and Emulator」)，都是屬於靜態模擬的模擬器，並無法真正符合本論文所需。因此我們需要設計一個可以實現動態模擬的模擬器，來驗證本論文所提出的新信賴模型。

5.2. 設計原理

為了設計一個能夠允許連續性的動態模擬，並且忠實呈現本論文所提新信賴模型的模擬器，我們認為此模擬器必須要達成三個重要的目標才能算是符合條件的模擬器：

1) 事件必須能夠被賦予給指定的節點：本論文所引用的 「社會學行動理論/韋伯方法論」 認為，每一個個體都是一個獨立的行為個體，因為本身的慾望而產生行為，進而與其他個體互動後影響整個小世界。要讓小世界中的任一個體模擬真實的社會狀況，我們就必須 「使其產生行為」 ，也就是賦予事件給個體。故我們將 「賦予個體事件」 的動作獨立出來成為一個重要的控制元件，並稱之為 「事件引擎」 (Events engine) 。這個控制元件的主要任務為接受來自使用者定義的事件、分析事件的目的與觸發時間、描述為模擬器可以了解的資料結構，以及當事件的觸發時間到達時正確地執行事件的目的。我們將在下一節作更進一步地描述這個控制器動作的流程。

2) 節點之間的信賴關係必須有效地被更新：我們想要設計一個動態模擬的環境，能夠隨著 「時間」 經過，忠實地呈現當某一個個體產生某些行為時對其他個體的影響。這裡我們定義的 「時間」 有兩個涵意：模擬器時間 (Time of simulation) 與事件觸發時間 (Active time of events)。當整個小世界在模擬過程中出現了預期或者非預期的情況，基於讓使用者便於觀察的想法，我們會需要設計一個允許 「停止」 (Stop simulation)以及 「啟動」 (Start simulation) (或 「再啟動」) 模擬器行進時間的功能，此時我們將可以停止或啟動的模擬時間定義為 「模擬器時間」 ；事件引擎接受來自使用者定

義的事件後，會將事件的觸發時間設定在使用者預期的發生時間點上，例如：「從現在的模擬器時間開始算起 5 分鐘之後發生」 或者是 「立刻發生」 。伴隨著事件被定義而產生的時間描述，我們便稱之為 「事件觸發時間」 。事件觸發時間與模擬器時間有著相依關係 (Dependence) ，只有在模擬器時間被啟動或者重新啟動的狀況下，事件觸發時間才會繼續累加，直到事件被觸發或者事件被移除。當事件發生後，會立即對被賦予事件的節點產生作用，促使節點作出某些反應。

我們假設，使用者定義一個事件 X 為 「當時間 t_0 時節點 A 由非詐欺節點轉變為詐欺節點」且我們預期詐欺行為的出現將對被賦予的計算任務造成一定程度的傷害，並且降低其他節點對 A 的信賴度。於是當節點 A 在時間 t_0 時成為詐欺節點開始，到時間 t_1 時另一個節點 B 賦予計算任務給 A 為止，我們必須有效率的更新所有認識節點 A 的其他個體對 A 的信賴度，確保動態模擬過程中信賴關係的正確性，否則將會發生信賴度不一的情況。我們依照個體更新信賴度的方法分為兩種方式，「主動」(Initiative) 與「非主動」 (Non-initiative) 。主動與被動更新信賴度的方式差別，在於被更新信賴度的節點，是否在模擬器時間 t_0 時為小世界中任一條信賴路徑或是轉手路徑上的節點。我們以前面提到的節點 A 為例，當事件 X 被觸發時，節點 A 由非詐欺節點轉變為詐欺節點，使得在模擬時間 t_0 時賦予節點 A 的計算任務失敗，造成 「從服務要求節點的好友開始一直到節點 A 之間」 所連成的信賴路徑上的所有節點受到一定比例的懲罰，並降低自身節點 (路徑上受到懲罰的節點) 好友名單上對節點 A 的評價。因此凡是此信賴路徑上節點的任一第一區位的好友節點 B 均會被 「非主動」 地告知節點 A 的評價不高，而降低下次賦予任務給 A 的可能性；反過來說，若是當模擬器時間 t_1 時，另一個節點 C 很有可能仍沒有被更新有關節點 A 的任何消息，此時節點 C 應該 「定期且主動」 地去拜訪自身節點好友名單上的好友，以求更新最新的信賴度關係。

我們將 「負責接受來自事件引擎觸發的事件，正確地驅動被指定的節點執行事件所描述的動作，並且依照我們定義的信賴度更新原則維護小世界信賴度正確性」 的重要控制元件獨立出來，稱之為 「節點引擎」 (Node

engine ）。節點引擎是本文所設計的模擬器最重要的一環，我們將在下一節詳細描述它的動作流程。

3) 小世界演化的過程必須更容易地被觀察：我們所設計的模擬器與其他模擬器最大的不同處，在於我們可以觀察到信賴關係建立的過程，而非僅僅只有結果。我們期望從觀察到的模擬過程中得知「定義多少比例的信賴係數與懲罰係數」，可以讓優良的節點很快的提升聲望，而詐欺節點可以很快的被小世界所排擠。我們一方面利用人際環境關係圖形理論，顯示自身節點與自身節點第一區位好友的信賴關係，讓使用者可以視覺化地看到人際環境關係的「類聚性」（詳情請見本文第一章），進而觀察類聚性程度的高低與搜尋到優良節點的快慢之間是否存在某種關係；另一方面利用曲線圖來分析自身節點的行為對小世界的公共生產力的影響，讓使用者很容易地發現什麼樣的事件對小世界影響最鉅，甚至成為導致非預期結果的原因。我們將「負責呈現來自節點引擎計算結果，以及提供控制項讓使用者選擇觀察面向」的控制元件獨立出來，稱之為「顯示元件」（View engine），我們亦在下一節作較為詳細的動作流程說明。

5.3. 主要控制元件

事件引擎、節點引擎以及顯示引擎是構成本文所設計的動態模擬器的三個重要控制元件。使用者透過將定義好的事件傳遞給事件引擎分析，來模擬真實世界中可能會發生的狀況，待節點引擎依照本論文所倡議建立的新信賴模型計算出信賴度關係後，再經由顯示引擎將模擬過程顯示在模擬器介面上。圖 5.1 顯示由本論文所定義的三個重要控制元件：事件引擎、節點引擎以及顯示引擎所建構的動態模擬器基本架構圖。

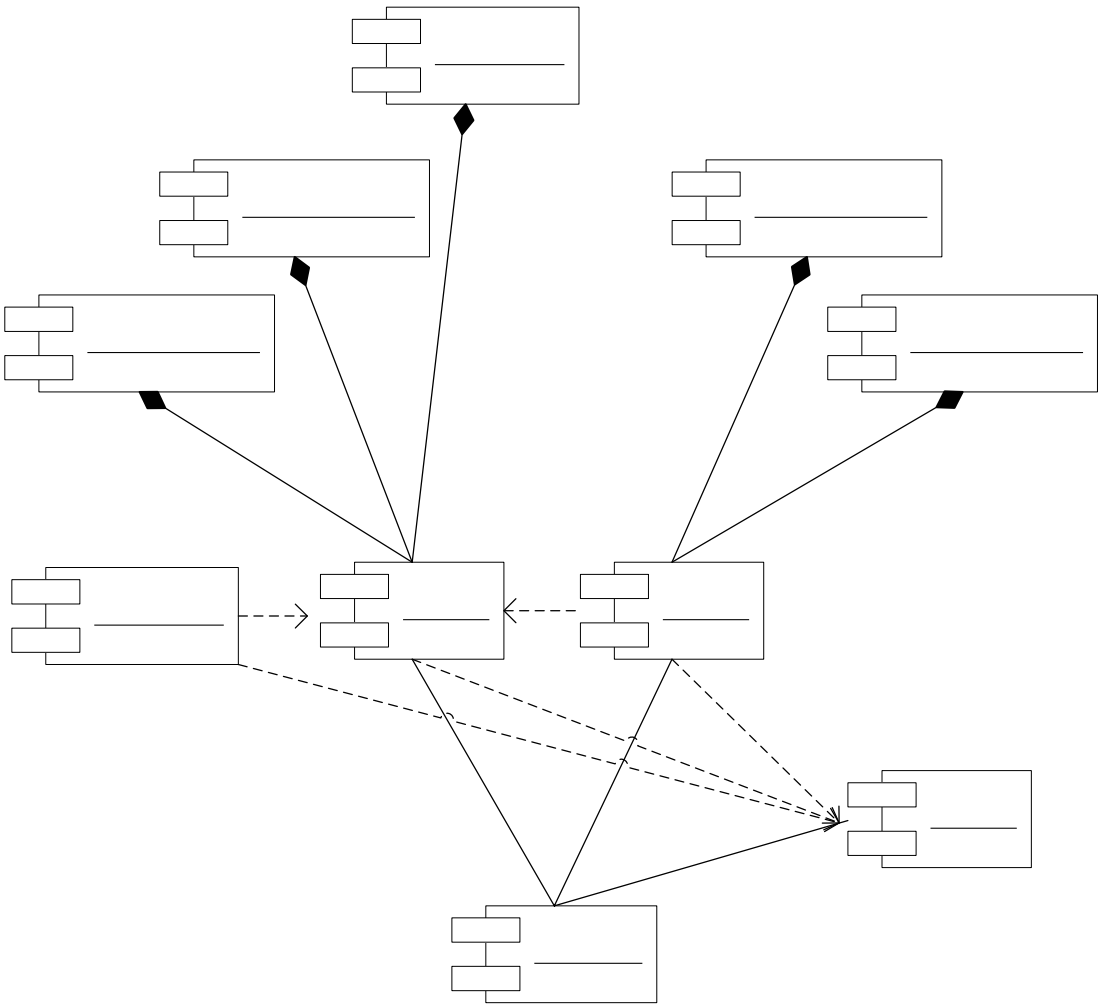

圖 5.1，模擬器主要元件架構圖。

事件引擎分為兩個部份：第一個部份是將使用者指定的事件解析並描述成「事件佇列」(Queues of events) ；第二個部份則是隨著模擬器時間的進行，累加佇列中每一個事件的事件觸發時間，待事件佇列中出現「符合觸發條件者」時，再將符合條件的事件傳遞給節點引擎，讓節點引擎執行事件所指定的動作。

我們定義事件佇列是一個包含執行動作與執行時間的佇列資料結構，事件引擎會不斷地接收來自使用者的事件指定動作，將事件依照模擬器定義的格式拆解成執行動作與執行時間，並加入事件佇列中參與 「排序」 (Sort) ；同時事件引擎也會不斷地從事件佇列中挑選出符合事件觸發條件者，將其送入節點引擎

中，並從事件佇列中移除。我們定義所謂 「符合觸發事件條件者」 的事件，其事件觸發時間必須小於或等於模擬器時間，才有資格被送入節點引擎執行；而「排序」 則是指當事件被加入事件佇列的末端後，事件引擎將會依照事件觸發時間由小到大重新排序，節省事件引擎挑選事件所時花費的時間。圖 5.2 描述事件引擎的概略動作流程，事件引擎會不斷地重複圖 5.2 所描述的動作流程，直到模擬器被停止。

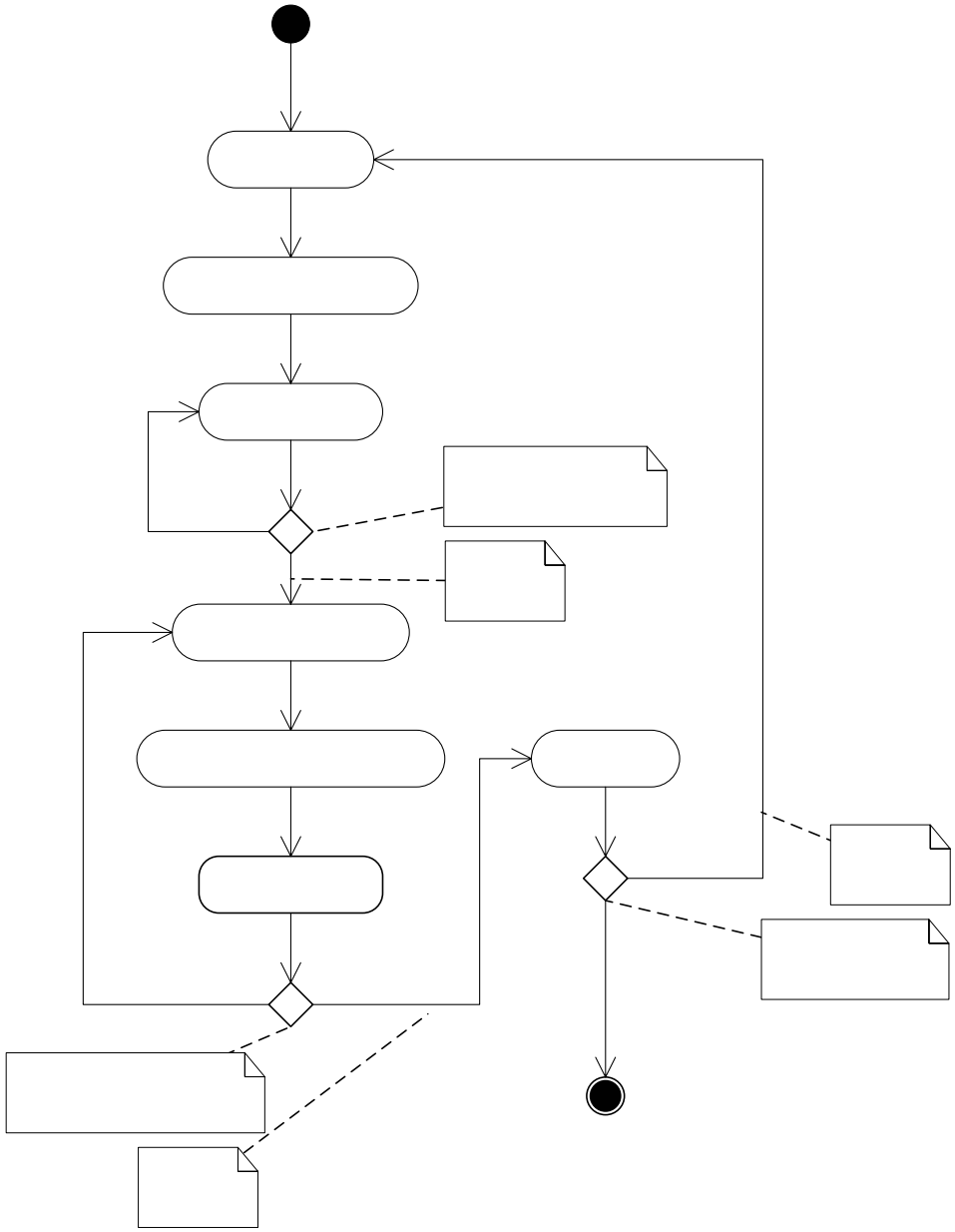

圖 5.2，事件引擎（Events engine）動作流程圖。

節點引擎是我們設計的動態模擬器中最重要的一個控制元件，它負責的工作非常多，包含接受來自事件引擎所描述的動作，並且正確地執行、負責建立信賴路徑與轉手路徑、責處理所有節點「主動」與「非主動」更新信賴度關係的要求、負責將計算完成後的結果傳送給顯示引擎以利使用者觀察以及負責維護小世界模擬過程的正確性。

節點引擎在載入來自模擬器的使用者介面（User interface）所指定的小世界狀態 Θ 後，會將 Θ 中的所有節點描述成「節點矩陣」（Node array），並賦予每一個節點編號。當事件引擎將已分析完畢的事件指定給節點 A，並將這樣的訊息傳遞給節點引擎，此時節點引擎便從節點矩陣中「搜尋」(Sequential search) 節點 A，並依事件指定的行為改變節點 A 的狀態或是驅使 A 執行某些動作。

我們定義節點矩陣是一個固定大小的矩陣，其中每一個「項」(Item) 均包含了「好友名單」(Friend list) 以及「同業公會」(Guild list) 兩個佇列。好友名單被設計成為一個節點佇列，內容僅紀錄所有自身節點人際環境關係中第一區位的好友編號與其提供的服務。當自身節點 A 在時間 t_0 被賦予計算任務 X 時，節點 A 便可以依據服務要求者的服務需求在好友名單中進行信賴度排序，挑選最適合的好友並請求服務，或者開始建立信賴路徑。同樣的我們將同業公會設計也成為一個節點佇列，但與好友名單不同的是，同業公會佇列中紀錄著「曾經認識且擁有相同服務的節點編號與其服務」，這代表著同業公會中的節點不一定是好友，但若是好友名單中的好友且擁有相同的服務者則必然是同業公會中的一員。這樣的設計讓節點引擎可以隨著時間經過，漸漸地加快建立信賴路徑與轉手路徑的速度，理由是優良的候選人將會很快的被這兩個節點佇列所包含。

使用矩陣存放節點資訊的好處，也可以在更新顯示引擎資訊上顯現出來。節點引擎在每次的計算之後，只需要將節點矩陣中被更新節點的索引及更新資訊傳遞給顯示引擎，便可以確保使用者可以在很短的時間內，看到事件觸發所造成的影響，而無需等待因為傳遞整個節點矩陣給顯示引擎而造成的時間差，增加信賴度不一的機率。圖 5.3 顯示了節點引擎的概略動作流程。

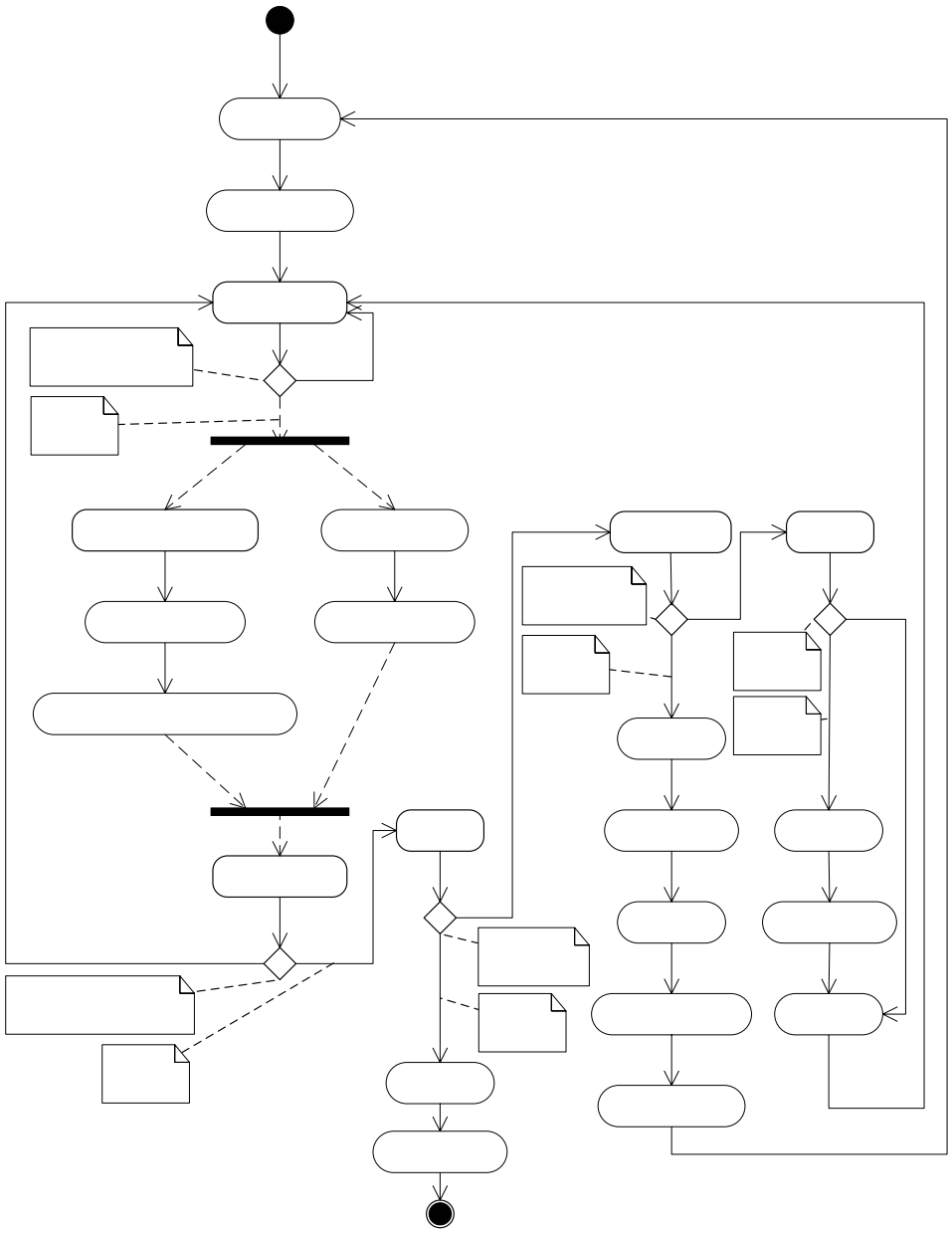

圖 5.3，節點引擎 (Node engine) 動作流程圖。

時脈控制器是否開啓?

顯示引擎是使用者與模擬器溝通的重要元件，它最主要的工作在於接受來自節點引擎計算過後的信賴度關係，加以分析並且繪製成使用者可以一目了然的拓樸網路以及曲線圖。我們依照人際環境關係網路 (請詳見本文第三章，圖 3.1) 設計拓樸網路，並且定義節點與節點之間的連線稱之為「邊」 (Edge) 。邊的長度與兩節點之間的信賴度成反比關係。這使得我們可以透過拓樸網路觀察出人

等待事件觸發引擎將事

際關係網路的類聚性，也就是關係越好的節點會靠得越近，而關係越差的節點便會離得越遠；換句話說，就是聲望越好的節點越容易被其他節點所搜尋到，這與我們現實社會中的情形是相仿的。曲線圖顯示則較為單純，用以顯示本論文對公共生產力評比的四個面向 [25]：目標達成度、社會公平、服務品質，以及服務滿意度。我們以曲線圖來顯示四個面向各自所代表的數據高低，讓使用者可以很容易觀察出整體小世界是否因本論文所建立的新信賴模型帶動了生產力的提升。圖 5.4 描述了顯示引擎簡易的動作流程圖。

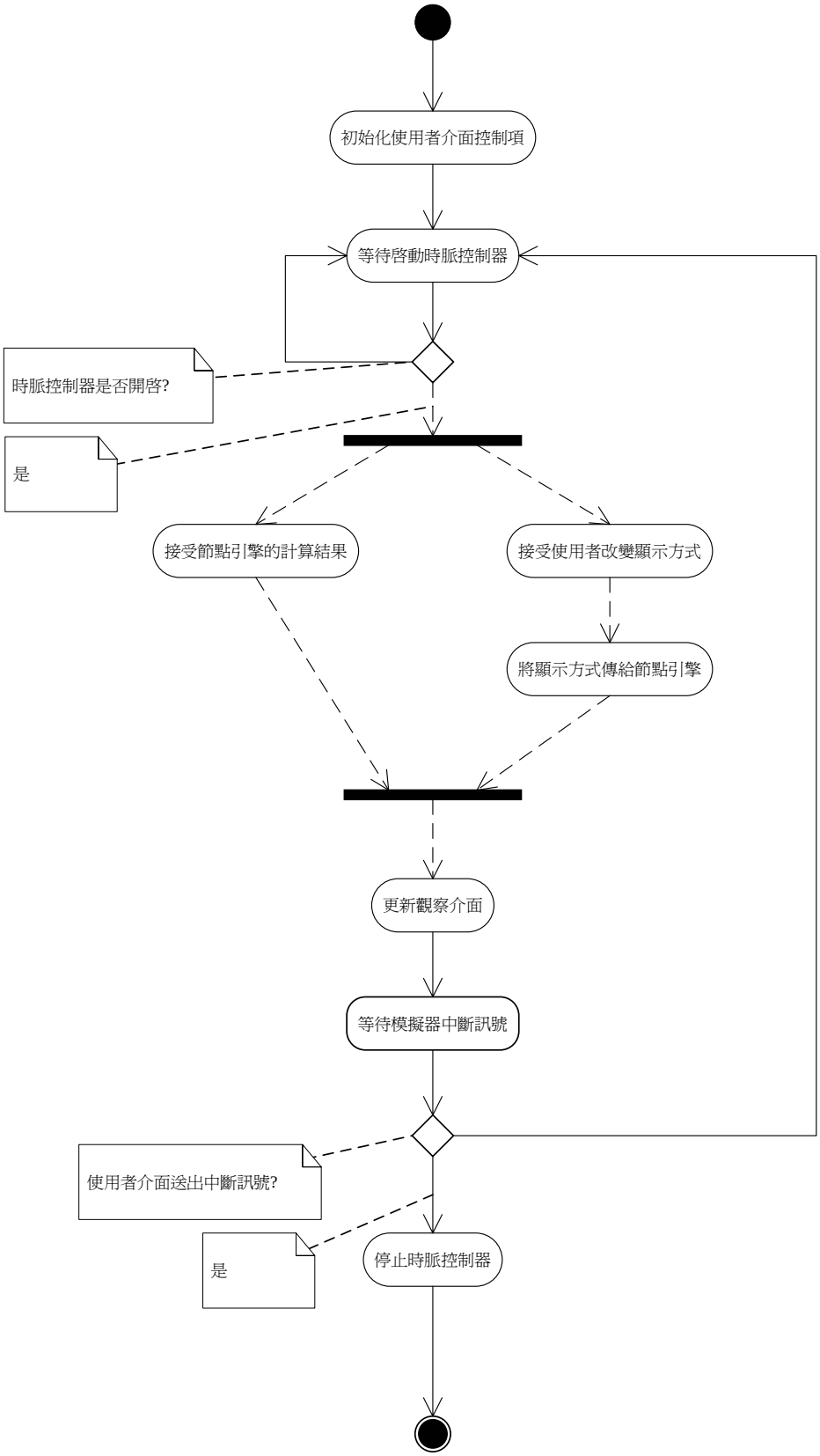

圖 5.4，顯示引擎（View engine）動作流程圖。

我們設計出一個能夠忠實呈現本論文提出之新信賴模型的動態模擬器，讓使用者可以透過觀察模擬器連續且即時反應出信賴度關係的模擬過程，找出適合的信賴度係數以及懲罰係數；並且觀察到新信賴模型對於詐欺節點產生排擠效應的過程。我們將在下一節簡單地描述系統實做的結果。

5.4. 系統實做的結果

我們將透過一些簡單的操作流程，來觀察系統實做的結果。圖 5.5 顯示我們所設計的模擬器初步的使用者介面 (User interface) ，隨著實驗的進行，我們仍會針對部分的使用者介面做些微的調整。

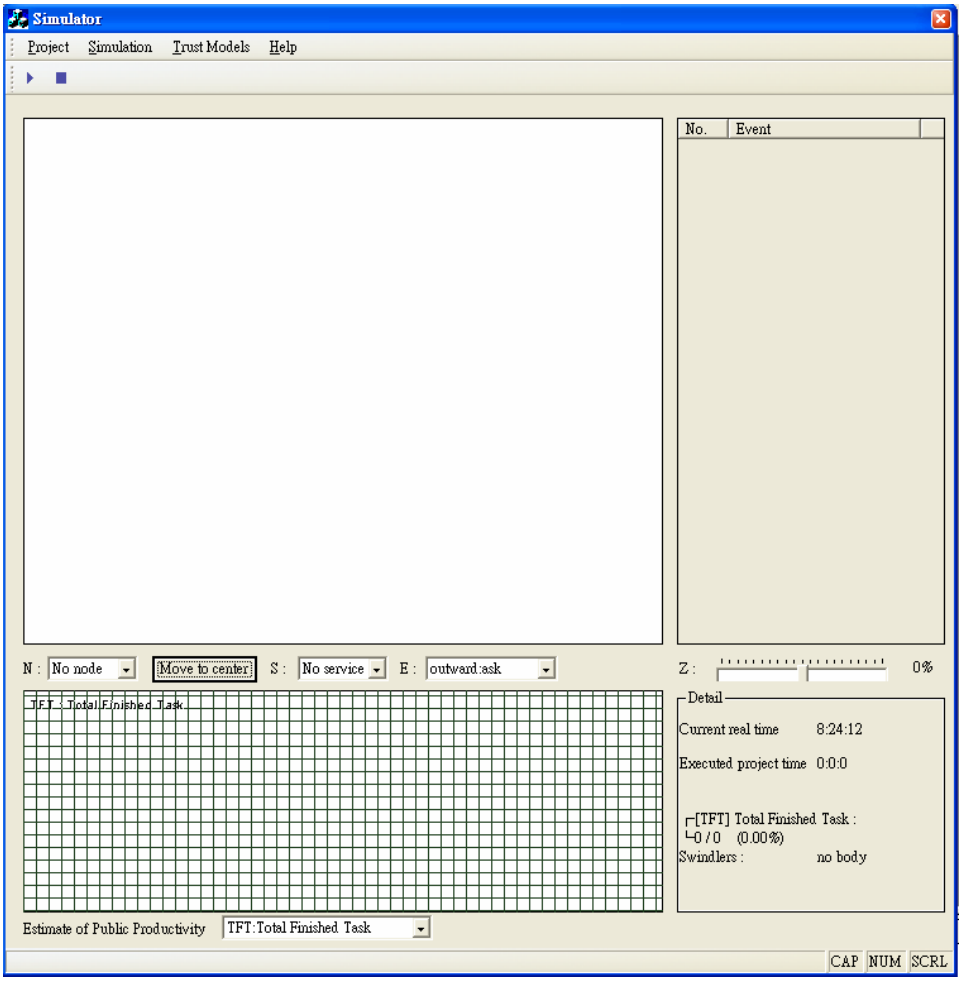

圖 5.5，模擬器初步的使用者介面。

透過使用者介面，我們產生了一個已經包含了三個節點的狀態檔 Θ ，並且指定三個節點都擁有相同的服務資源 ε 。模擬器載入 Θ 之後會將 Θ 交給節點引擎作分析描述並開始計算 Θ 所代表的小世界裡每一個節點相互之間的信賴度關係。依照我們對模擬器的設計，節點引擎在接受使用者所指定的狀態檔 Θ 之後，會將 Θ 所描述的三個節點依序賦予節點編號並且存放進節點矩陣之中，計算節點相互之間的信賴度，最後將計算的結果傳遞給顯示引擎顯示在模擬器介面上。由於我們預期在剛載入新建立的小世界時，節點之間互相並不認識，所以圖 5.6 在拓樸圖形顯示區上正確地顯示三個各自獨立的節點。

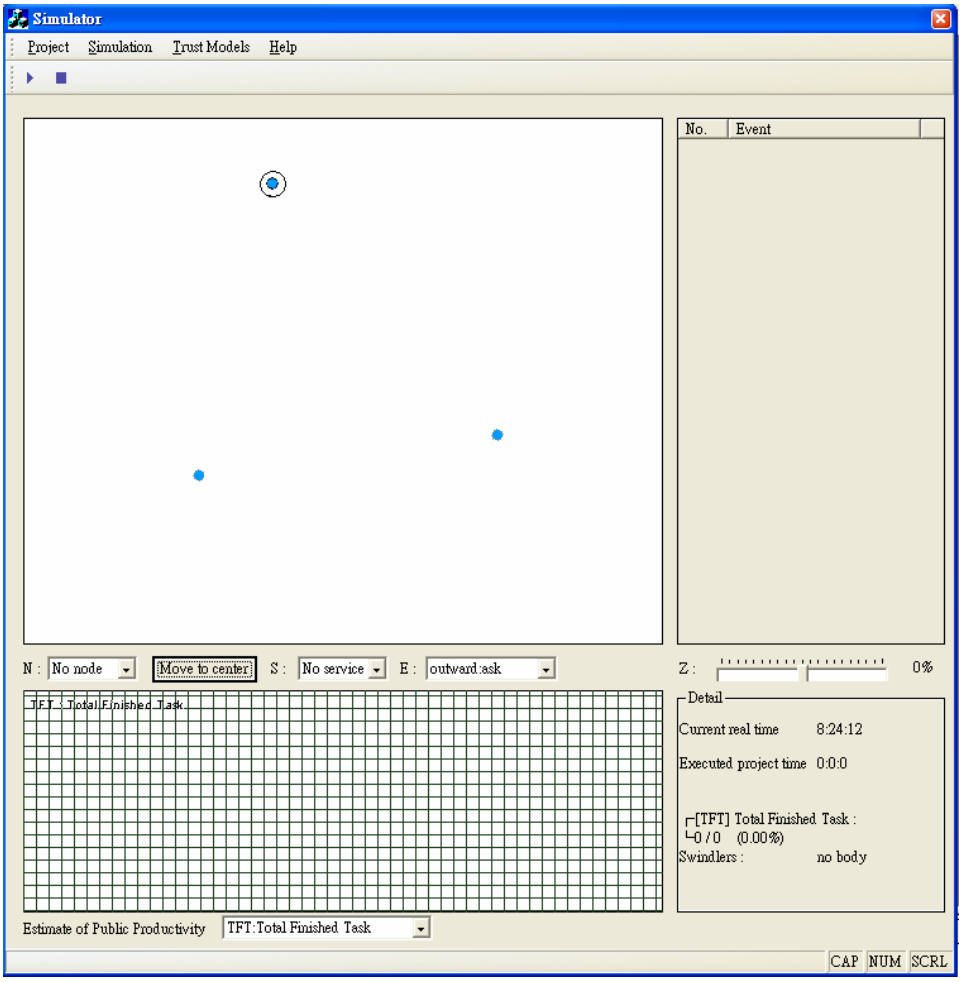

圖 5.6，模擬器載入還有三個節點的狀態檔，此時時脈控制器尚未被啟動，模擬還未進行。

模擬器初始化完成並載入預設或是使用者指定的狀態檔後，會自動地將節點編號最小的節點標示為「被觀察節點」(Observed node)，如上述圖 5.6 中所示，圖中被一圓圈圈起來者即為被觀察節點。我們將「被觀察節點」定義為「預設或是目前使用者所選擇的個體，且我們正透過它觀察它與整體小世界之間的互動關係」。我們以被觀察節點作為拓樸圖形樹根，被觀察節點的第一區位好友為第一層葉節點，並以此類推向外延伸，我們可以畫出一棵「自由樹」(Free tree)，而這棵自由樹便形成的我們拓樸圖形。圖 5.7 顯示預設的被觀察節點為編號 1 的節點，當模擬器時間由 t_0 推移到 t_1 時，小世界中三個節點在我們預

期之下，透過主動且隨機地（詳情請見本文第三章）挑選小世界中另一個節點互相認識之後所構成的簡單的拓樸圖形。

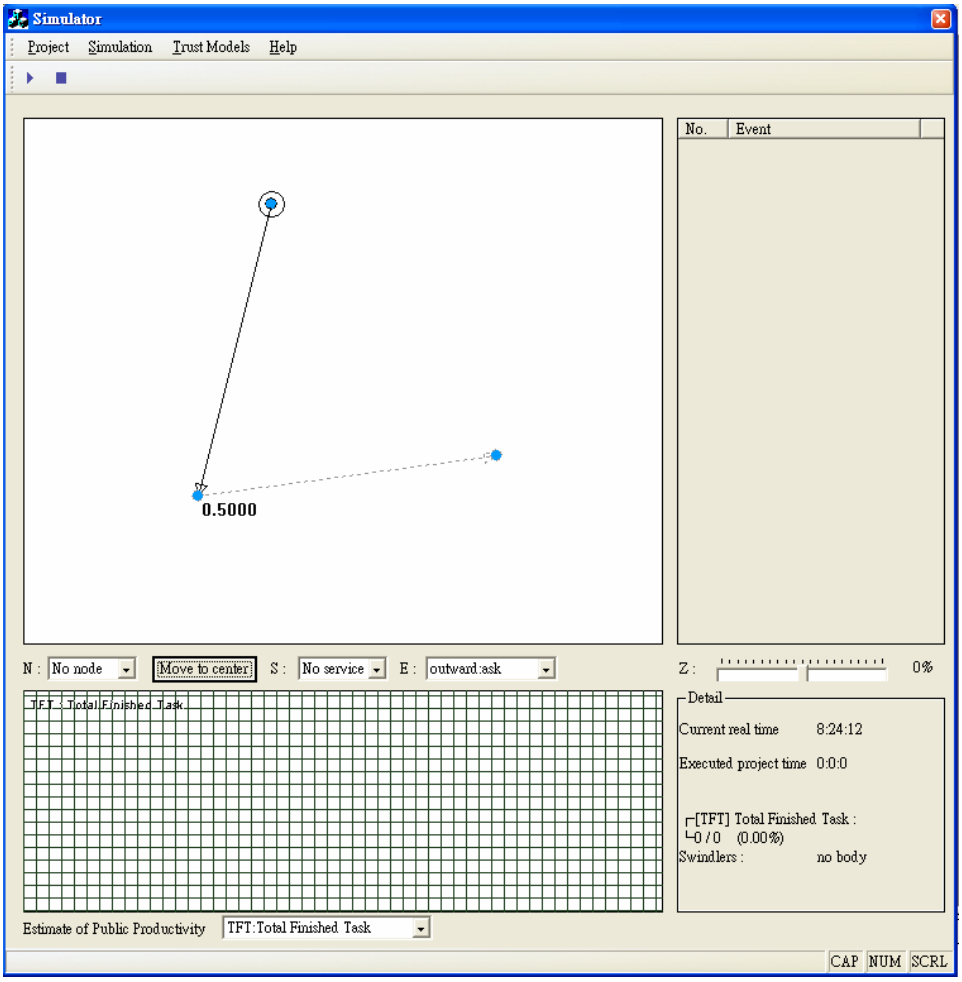

圖 5.7，模擬器啟動時脈控制器開始模擬之後的情形，此時因為還未合作過，所以基礎的信賴度為 0.5000。

模擬器完成載入狀態檔與部署後，假設如上圖 5.7 所示，當模擬器時間遞移到 t_1 時，節點 1 與節點 3 因為節點 1 的拜訪而相識，且節點 1 對節點 3 的基礎信賴度為 0.5000；同樣的節點 3 也因為隨機拜訪而認識節點 2，基礎信賴度亦為 0.5000；節點 1 與節點 2 之間相互並不認識。此時我們指定一個計算任務的服務需求 ε 給被觀察節點 1，且設定在模擬器時間 t_1 時觸發一

個事件 X ：節點 3 在模擬器時間大於或等於 t_1 時轉變成高負載節點，也就是無法再處理來自其他節點的服務請求。這裡我們定義服務 ε 為服務編號為服務編號第 3 號的服務。於是我們可以觀察到在模擬器右下角的 「事件列表」 (Events list) 框框中，因為節點引擎將來自事件引擎指定事件的事件觸發訊息告知顯示引擎，而出現了 「Need service No.3」 以及 「Node 3 High loading」 的事件訊息：

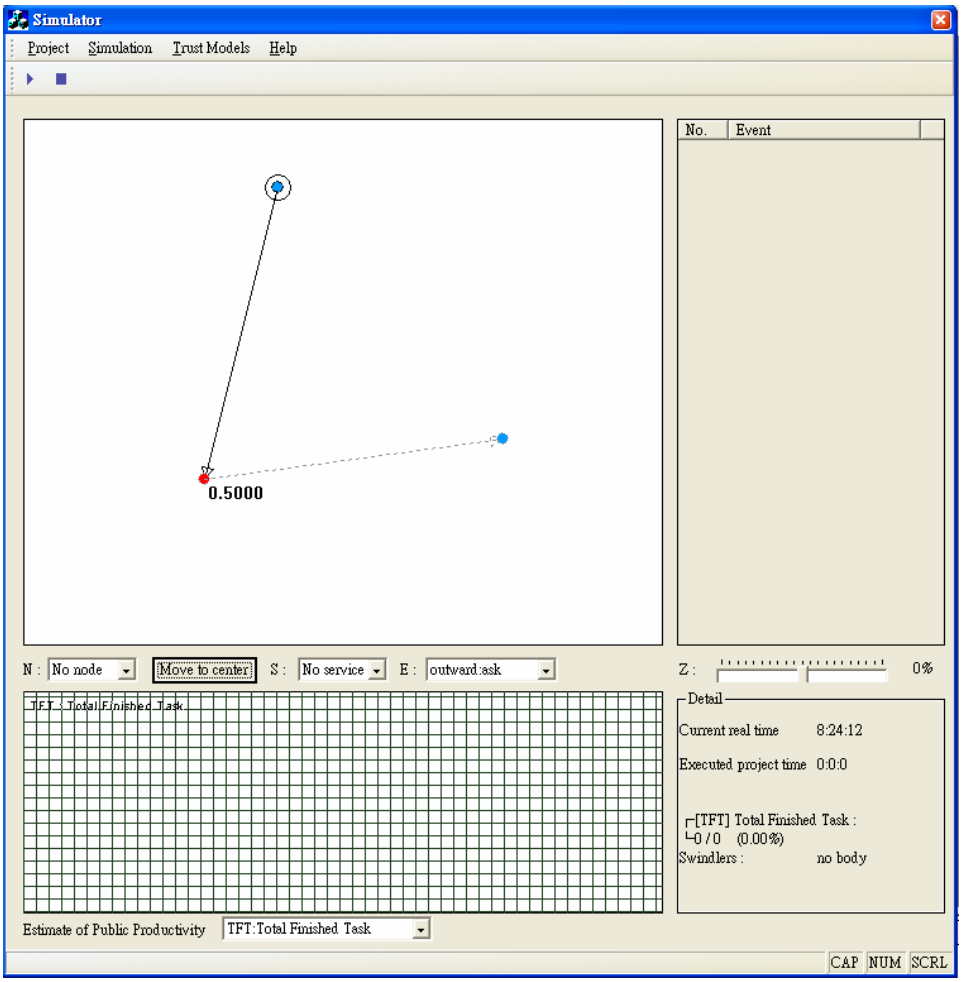

圖 5.8，時脈控制器啟動後，在模擬器時間 t_1 時觸發事件。

我們發現模擬過程如我們所預期的，節點 1 在訪問好友名單中的好友節點 3 之後，發現節點 3 提供服務 ε ，且只有節點 3 可以托付此計算任務 (因

為節點 1 的好友名單中只認識節點 3)，所以節點 1 將計算任務 X 指定給節點 3 。節點 3 為了不受到信賴度懲罰，在訪問好友名單中的好友且加以評估後，決定將計算任務轉手給節點 2 。於是模擬器時間推移到 t_2 後，我們可以發現模擬器正確地建立起信賴路徑 { 節點 1 , 節點 3 } 以及轉手路徑 { 集點 3 , 節點 2 } 。因為計算任務的完成，讓信賴路徑以及轉手路徑上的節點受到信賴度增加的獎勵，並讓節點 1 認識了節點 2 ：

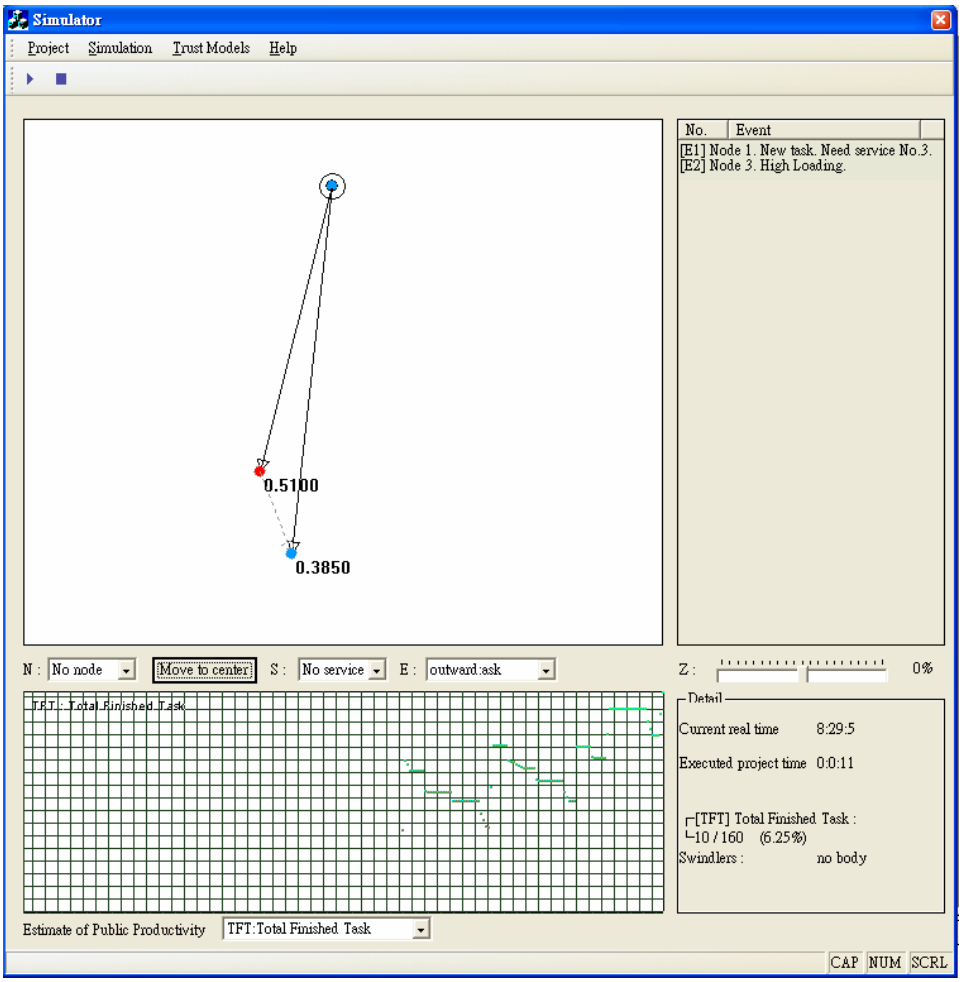

圖 5.9，節點 1 因計算任務認識節點 2 後，三者的拓樸關係。

我們可以從上圖看到，節點 3 因為無法完成計算任務，所以將任務轉手給節點 2 。我們在這個模擬中定義任務完成的信賴度增量為 0.1 ，所以當任務完

成後，節點 3 跟節點 2 同時依比例增加信賴度增量。透過計算任務的完成，節點 1 因為使用了節點 2 的服務進而認識節點 2 ，所以圖中三者的信賴度關係以被觀察節點的角度來看，節點 1 對節點 3 的信賴度增加到 0.51 ，對節點 2 的信賴度則從 0.25 (詳情請參見本文公式 1 ，第四章) 增加到 0.385 (詳情請參見本文公式 2 ，第四章)。

我們也可以同時觀察到圖 5.9 中顯示了整體小世界的目標達成率不斷地往上拉高 (圖 5.9 中下方最大的曲線圖)，理由是我們交付了依個計算任務給小世界，而小世界也在模擬時間 t_2 時將其完成。於是模擬器的三個重要的控制元件相互之間連動的正確性已經可以透過簡單的操作獲得證實，我們將在未來的實驗中，藉此模擬器針對本論文所提出的新信賴模型做更進一步的模擬。