國 立 交 通 大 學

資訊科學與工程研究所

碩 士 論 文

建構一個角色扮演互動式學習內容模型

Constructing an RPG-like Interactive Learning Content Model

研 究 生：吳東權

指導教授：曾憲雄 博士

中 華 民 國 九 十 六 年 六 月

建構一個角色扮演互動式學習內容模型

研究生：吳東權　　　　　　　　指導教授：曾憲雄博士

國立交通大學資訊學院

資訊科學與工程研究所

摘　要

在過去數位學習的環境中，學習內容通常是由純 Html 語法寫成的，屬於靜態的內容呈現，然而，隨著網頁技術的快速發展，互動式的學習內容變得愈來愈普遍，因為互動的特性會使得學習和教學變得較有趣，可增加學習動機。而本篇論文主要著重在於角色扮演型態的學習內容，但要去建構這種類型的學習內容對一般人而言是困難的。因此，我們提出了一個物件導向的互動式學習內容模型 **(OOICM)**，此模型呈現高階的遊戲知識給學習內容編輯者，並且編輯者可透過它所提供的界面，不需要自己寫程式，就可以容易地建構一個角色扮演型態的學習內容。 OOICM 由三個成分所構成，分別是 **Story Control Flow (SCF)**, **Activity**, 和 **Scene Object (SO)**。在 OOICM 的系統層中，因為我們用派翠網路去描述 SCF，並且用框架知識去描述 Activiy 和 SO，所以我們的系統環境需要派翠網路引擎和框架引擎。此外，我們基於 OOICM 實做了一個產生器用來協助編輯者建構學習內容，並和其它的編輯工具比較，而實驗結果也顯示了我們的產生器在學習內容的重複利用上有較優異的能力。

關鍵字：派翠網路、框架、角色扮演遊戲、知識技術方法

Constructing an RPG-like Interactive Learning Content Model

Student: Dung-Chiuan Wu **Advisor: Dr. Shian-Shyong Tseng**

Institute of Computer Science and Engineering
Nation Chiao Tung University

Abstract

In early years, the e-Learning contents are static and uninteresting because they are only written by HTML. However, with the growth of the web technology, interactive learning contents are getting more and more popular because the interactive feature makes learning as well as teaching more interesting. In this thesis, we focus on RPG-like Flash interactive learning content. But it is hard and costly to create this kind of learning content. Therefore, we propose an **Object Oriented Interactive Content Model (OOICM)** which represents the high level game knowledge for authors and the interface of OOICM can assist authors in constructing an RPG-like learning content easily without writing low level codes. OOICM is composed of three components which are **Story Control Flow (SCF)**, **Activity**, and **Scene Object (SO)**. In the system layer of OOICM, because we apply Petri Net to model SCF and apply frame knowledge representation to model Activity and SO, the system environment must contain a Petri Net engine and a frame engine. In addition, we also implement a generator based on OOICM to help authors construct the learning content. Final, we compare the generator with other authoring tools and the experimental result shows the generator has better reusability.

Keywords: Petri Net, Frame, RPG, Knowledge Based Approach.

致 謝

這篇論文的完成，首先要感謝我的指導教授，曾憲雄老師。在研究所兩年的歲月裡，無論是在學術研究或是為人處世方面，皆讓我受益匪淺，尤其是我學到了對一個知識領域的研究方法、邏輯思考及表達能力的訓練，這將使我終生受用不盡。同時也感謝我的口試委員，楊鎮華教授、孫春在教授和黃國禎教授，他們給予了我相當多的寶貴意見，讓本論文更有意義與價值。

再來要感謝的是蘇俊銘學長、翁瑞峰學長和林喚宇學長，這段期間內，即使他們很忙，還是會騰出時間與我討論並給我建議、想法，協助我修改論文。此外我也從他們身上學習了不少生活態度及為人處事的方法，在此深表感激。還有實驗室的同窗們，昂叡、信男、雨杰、芙民、曉涵、嘉妮，在這兩年的時光裡，和你們同甘共苦、互相扶持鼓勵，能認識你們真的很開心。還有其他在身邊鼓勵我的朋友們，雖然無法在此一一提及，但我心裡真的非常感激有你們在我身邊。

最後要感謝的是我的家人，默默地支持與鼓勵，並不時地關心我，是我在心力交瘁時還能保持鬥志的原動力。日後，我會更加努力地繼續前進，不辜負他們的期望。

Table of Content

List of Figures

List of Tables

Chapter 1. Introduction

With the growth of Internet, it changes not only the human's life but also the learning approaches. The technologies of e-learning are globally accepted for making learners study anytime and anywhere. Most e-Learning contents are placed on web servers, and learners can study with those contents by a web browser. In early years, these e-Learning contents are static and lack interactive features due to the pure HTML format, so they are uninteresting and can not show the real world scenario in a realistic way. However, some learnings need interactions with learners and scenario simulations.

In recent years, the web technology (such as Flash, JavaScript, AJAX) has become more and more mature and stable. Web has been able to present all kinds of multimedia information. As mentioned above, some learnings need interactions with learners and scenario simulations. For game-based learning, it has the characteristic of high interactions. Among several kinds of games, such as *Action Game*, *Role Playing Game (RPG)*, *Strategy Game* and so on, RPG is suitable for scenario simulations because it emphasizes the "real-world" side of science. Besides, Role-Playing mainly has three advantages which are "Motivating Students", "Augmenting Traditional Curricula", and "Learning Real-World Skills" [1].

Moreover, a platform is necessary to display the learning content. According to the statistics [2] from Adobe, Flash Player is the most pervasive software platform in the world, and supports many kinds of operating systems and mobile devices. In addition, it can represent rich 2D animations on WWW and handle various interactions by writing ActionScript code. Therefore, Flash file format (SWF) is very popular format in interactive learning content.

Due to the reasons above, we focus on RPG-like Flash interactive learning content in this thesis. But creating this kind of learning content is time-consuming and costly especially for nonprogrammer. Authors have to write ActionScript codes to handle various events. Even though the author has constructed RPG-like Flash content, he must modify the codes in order to apply to another similar scenario because the content is hard code. Therefore, how to facilitate the creation of RPG-like Flash content and reuse the learning content are important issues.

In general, RPG is composed of story (narrative), characters and scenes. All characters are arranged in the scene and the story describes how characters act to complete the playing of the game. These considerations must be taken into account. Therefore, in this thesis, we propose **Object Oriented Interactive Content Model (OOICM)** to represent the high level knowledge for RPG-like learning content. Authors can use the interface that OOICM provides to construct an RPG-like learning content easily without writing low level codes. OOICM is composed of three components which are **Story Control Flow (SCF)**, **Activity**, and **Scene Object (SO)**. SCF is a sequence of subgoals, Activity is the action that SOs can perform, and SO is the object in a game scene. In the system layer of OOICM, we apply Petri Net to model SCF and apply frame knowledge representation to model Activity and SO. Petri Net is a powerful language for process flow modeling, concurrency handling, and validation. These properties of Petri Net are suitable to represent the concept of the SCF. SOs have their own attributes, inherited attributes, and event procedure call. Frame knowledge representation is suitable to represent the properties of SOs. Final in order to communicate with SOs' frames, frame knowledge representation is also to represent activities.

Because we apply Petri Net and frame, the system environment must contain a Petri Net engine and a frame engine. We implement the two engines in ActionScript and a generator based on OOICM to assist authors in constructing an RPG-like learning content.

The remainder of the article is organized as follows. In Chapter 2, we introduce some related works about the story representation and authoring tools for creating the interactive content. Then, the proposed model OOICM and the system layer of OOICM are described in Chapter 3 and Chapter 4 respectively. Chapter 5 introduces the prototype system and an algorithm OOICM2AS. The implementation for OOICM and experiments are discussed in Chapter 6. Finally, Chapter 7 gives the conclusion and future work.

Chapter 2. Related Work

2.1. Story Representation

Story (Narrative) is a very important element in RPG. Story is a control flow that describes a sequence of events that the player has to experience in order to complete the game. There are several researches about the story representation and they propose their story structure. A well-designed story structure can help us verify the correctness of the story flow. In this section, we will describe some story representations.

In [3], they have studied many papers to analyze the narrative techniques. They think that storytelling applications can be classified as Rule base, State Transition based, Goal based, Permutation, Template based, Script based, Semantic Inference based, Emergent Narrative based, and Narrative Function based. We discuss several story structures as follows.

(1) Plan

In Mimesis [4][5], the planning representation is applied to represent the story. As shown in Figure 1, Gray rectangles represent character actions and are labeled with an integer reference number, the actions' names and a specification of the actions' arguments. Arrows indicate causal links connecting two steps when an effect of one step establishes a condition in the game world needed by one of the preconditions of a subsequent step. Each causal link is labeled with the relevant world state condition. The white box in the upper left indicates the game's current state description, and the box in the upper right indicates the current planning problem's goal description. The expressive power is proved in [6].

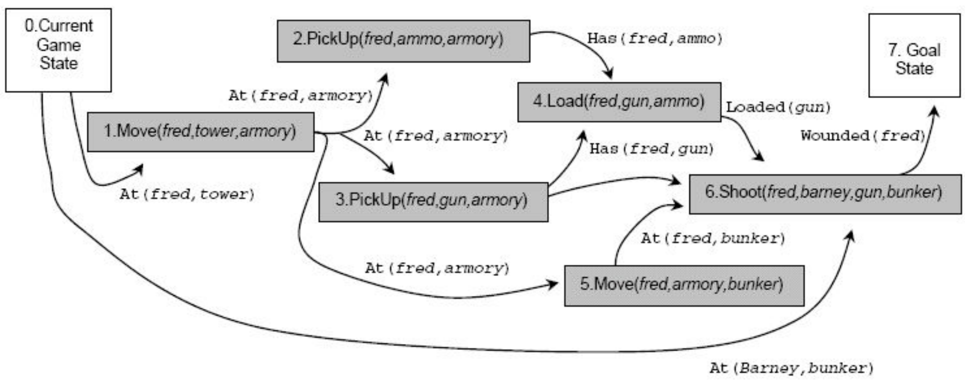


Figure 1: A Mimesis storyworld plan

(2) Partial-Ordering Graph

In Interactive Drama Architecture (IDA) [7][8], Brian Magerko used partial-ordering graph to represent story. As shown in Figure 2, each node in the graph is a *plot point*. A plot point has preconditions, actions, and a time constraint. The preconditions describe what should be true in the world in order for the plot point's actions to be executed. The actions are the plot events that are performed after all preconditions are fulfilled. The time constraint describes a time span during which every precondition must be true. This structure is similar to the planning language in Mimesis described earlier. The key difference is that this representation has no explicit concept of causality.

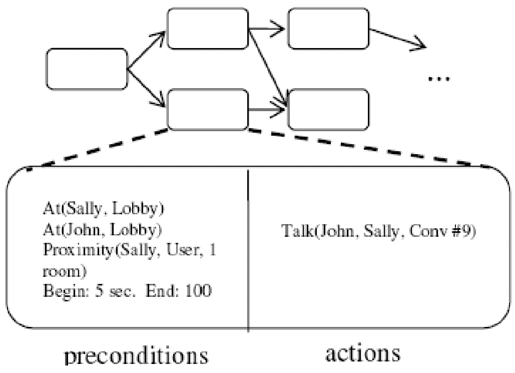


Figure 2: An example of partial-ordering of plot

(3) Petri Net

In [9], Petri Net is used to model an atomic action called transaction. There are three relationships of ordering and two logics relationships between transactions. The language that Petri Net generates can used to characterize the topology of the virtual space in a game [10]. In [11], in order to characterize narrative structures, they employ "narrative nets" (N-nets), which are based on colored Petri nets. In [12], Clark Verbrugge presented a representation framework called Narrative Flow Graphs (NFG), derived from 1-safe Petri Net. NFG can be used to verify desirable properties, or as the basis for a narrative development system.

The researches discussed above mainly attempt to verify the properties of the story flow such as the balance between the user control and the story coherence. But they did not take the construction of a story into consideration. These story structures are not intuitively understood by authors. Therefore, a game author must understand the story structure before designing a game. Besides, there are other considerations such as scenes, actions for an object, when designing a whole game.

2.2. Authoring Tools

Many tools can be used to create the interactive content. **Adobe Flash** [13] is the original and popular authoring tool to create the flash format content. Flash is not designed to construct the RPG-like content, and therefore authors have to add ActionScript code manually to simulate the behaviors of the role playing games. It is not easy for a programmer to write ActionScript code to handle varied events in a game, not to mention a nonprogrammer.

RPG-Maker [14] is powerful tool for creating role playing games. It provides the user-friendly editor interface. Authors create a role playing game without writing low level codes. Events are attached to the objects or characters in the scene. The story control flow is event-driven and is not represented explicitly. What authors see is the game scene and a lot of events. The scenario that the game presents can not be understood by authors immediately. Therefore, it is hard to construct, reuse, and maintain a more complex role playing game.

Chapter 3. Object Oriented Interactive Content Model (OOICM)

As mentioned before, it is time-consuming and costly to create the RPG-like learning content. Because authors have to write low level codes to handle various events, we want to propose a model to assist authors in constructing the learning content without low level programming. To design this model, some difficulties must be solved. First, how do we represent the story explicitly for authors? Second, the objects in a scene have their own attributes, inherited attributes, and event procedure calls. Writing codes for every object instances is costly. Final, how do we transform the model into low level codes?

The idea for this model is described as follows. We provide authors with the high level game knowledge from authors' viewpoint so that author can construct an RPG-like learning content easily without understanding how the system actually work actually. Based upon this idea, we propose **Object Oriented Interactive Content Model (OOICM)** to represent the high level knowledge for RPG-like learning content. Authors can use the interface of OOICM to construct an RPG-like learning content easily without writing low level codes. Furthermore, the story control flow is represented explicitly in order to show the scenario of the learning content clearly. Object oriented methodology is also used for reusability and eases coding effort.

As shown in Figure 3, OOICM is composed of three components which are **Story Control Flow (SCF)**, **Activity**, and **Scene Object (SO)**. **Story Control Flow** a sequence of subgoals. **Activity** denotes the action that SOs can perform. Some activities may achieve subgoals in the SCF. **Scene Object** denotes the objects that constitute the scene of learning content. In order to complete the playing of the game, the player has to perform some activities to achieve the subgoals in the SCF. The detail of the three components will be described in this Chapter.

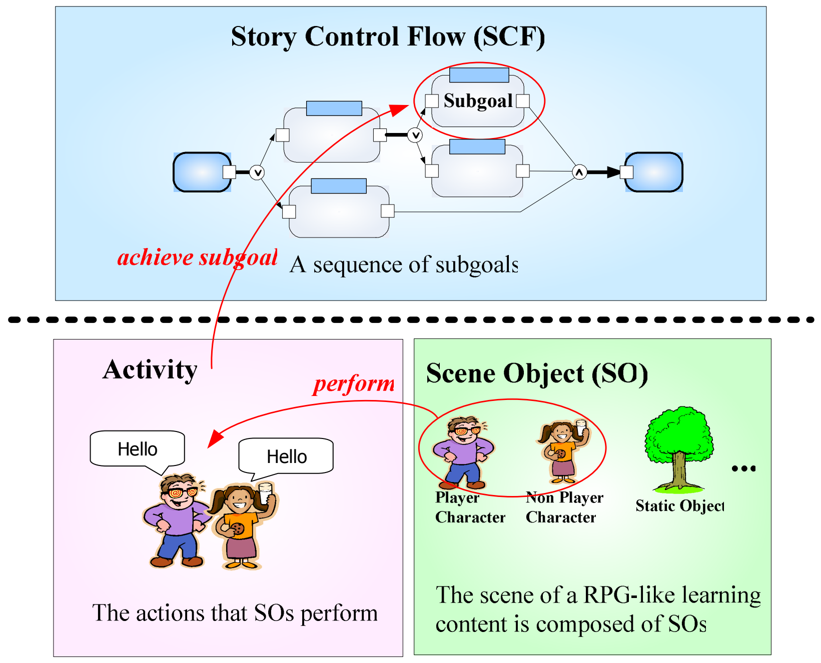


Figure 3: OOICM overview

3.1. Story Control Flow (SCF)

The SCF denotes a sequence of subgoals in an RPG-like learning content. The definition of the SCF is described as following.

Definition 1: The **Story Control Flow** is a 3-tuple

SCF = (N, C, O), where

1. **N** $= \{n_1, n_2, ..., n_m\}$ is a finite set of SCF nodes. n_i includes four types which are **Start**, **End**, **Connective**, **Regular**, and **Super**.
   - The **Start node** and **End node** are atomic nodes that specify the start and the end of a game respectively. Start node and End node in an SCF must be unique.
   - The **Connective node** is an atomic node just for connecting different connectors.
   - The **Regular node** is an atomic node that denotes the subgoal in a game. It has an attached activity declaration. In this section, we skip the detail of the activity. It will be discussed in 3.2.
   - The **Super node** is a composite node that can be taken apart into nodes and connectors. An SCF can be encapsulated into a Super node and the Super node can be reused to simplify another SCF. When an SCF is transformed into a Super node, the Start node and End node in the SCF will both become a Connective node.

   The detail is described in Table 1.
2. **C** $= \{c_1, c_2, ..., c_n\}$ is a finite set of SCF connectors. A connector $c_i = (TN, HN, t)$ is considered to be directed from *TN* to *HN*.
   - $TN \subset N$. *TN* is called the *PreNodes* of c_i. The side that connects to *TN* is called *tail* of c_i.
   - $HN \subset N$. *HN* is called the *PostNodes* of c_i. The side that connects to *HN* is called *head* of c_i.

- *t* denotes the type of the connection. There are five types which are "**Linear**", "**Concurrence**", "**Selection**", "**And**",and "**Or**". The values of $|TN|$ and $|HN|$ are dependent on *t*. As described in Table 2, $|TN| = |HN| = 1$ for a Linear connector, but $|TN| = 1$, $|HN| > 1$ for a Concurrence connector.

3. $\mathbf{O} = \{o_1, o_2, \ldots, o_q\}$ is a finite set of Scene Object declarations. o_i = (*type*, *variable*), where *type* is the type of o_i, *variable* is the identification for o_i.

Table 1: The SCF node

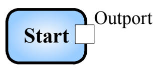
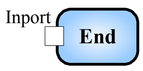
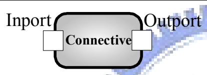
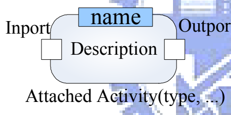
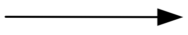
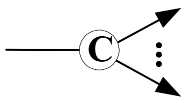
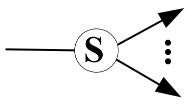
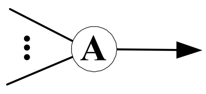
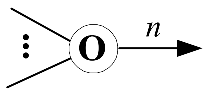
SCF Node Type	**SCF Node Notation**	**Description**
Start		The start of a game. Only one outport.
End		The end of a game. Only one inport
Virtual		A middle node for connecting different connectors
Normal	 name Inport Description Outport Attached Activity(type, ...)	The subgoal of a game

Table 2: The SCF connector

SCF Connector Type	**SCF Connector Graph**	**Description**
Linear		One to one. If the PreNode of this connector is achieved, player can progress in PostNode of this connector.
Concurrence		One to many. If the PreNode of this connector is achieved, player can progress in any PostNode of this connector.
Selection		One to many. If the PreNode of this connector is achieved, player can progress in only one PostNode of this connector.
And		Many to one. If all PreNodes of this connector are achieved, the player can progress in PostNode of this connector.
Or		Many to one. If any *n* PreNodes of this connector are achieved, the player can progress in PostNode of this connector. The default value of *n* is 1.

From author's viewpoint, N means the plots in a story, C means the flow directions of plots, and O mean the cast.

Most scenarios are not beyond the scope of the above definitions. Therefore, the expressive power of the SCF is enough. We give three examples to explain the SCF.

Example 1. A sample conversation scenario

Figure 4 shows a sample SCF graph composed of three nodes and two connectors. The three nodes are Start node, End node, and Regular node *node1*. The two connectors are *c1* and *c2* which are both linear type. The story describes that the player has to talk to Mary to complete the game. The subgoal is described in *node1*.

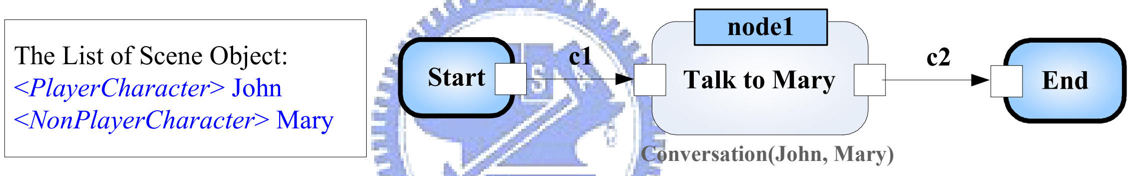


Figure 4: A simple conversation SCF.

Example 2. The procedure to leave school.

Figure 5 shows a part of the procedure to leave school. It is composed of seven SCF nodes and four SCF connectors. The story describes that the player has to talk to Mary and will get hints. After that, the player knows that he/she has to copy five theses and give two theses to the library, three to the laboratory. In the same time, the player also has to return the key of the laboratory to the manager. When finishing all missions, the player completes the game.

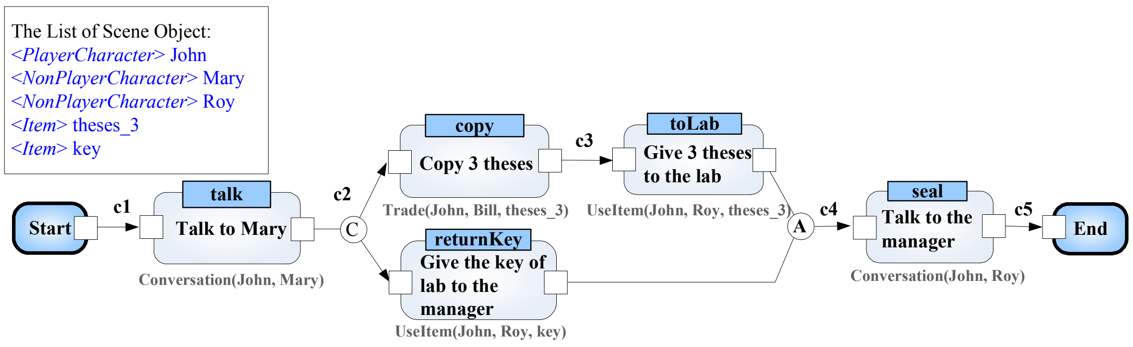


Figure 5: A complex SCF

Example 3. Use Super node to simplify a complex SCF.

As shown in Figure 6, the SCF in Example 2 can be encapsulated into an SCF *Super node*. The Start node and End node in the original SCF will become Connective nodes in Super node. The Connective node is a node used to join two different connectors.

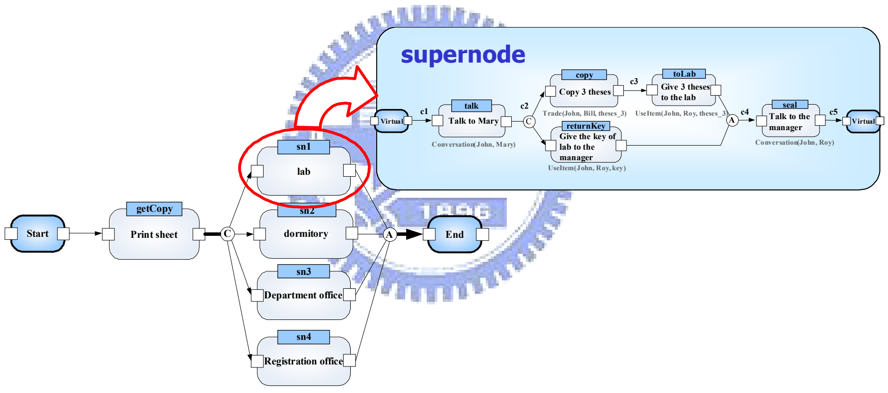


Figure 6: An SCF Super node

3.2. Activity

Activity is the action that the player can perform. Some activities may achieve some subgoals in the SCF, but some may not. In order to complete the game, the player has to perform some activities to achieve the subgoals in the SCF. For example, John has to have a conversation (Activity) with Mary so that the subgoal can be achieved. The basic principle is that if the precondition of an activity is satisfied, this activity will be executed, and after that, causes some post action to be executed. As shown in Table 3, authors must assign values to three basic attributes for every activity. The

In this thesis, we propose five kinds of activity templates which are "*Trade*", "*Use Item*", "*Take Ite*m", "*Conversation*", and "*Time*". For the learning purpose, these five activities are often used. If a new kind of activity is required, a new template can be added to extend the activity templates. These five templates have the default preconditions and are detailedly described in Table 4.

Table 3: The attribute of an activity that authors have to assign values to.

Attribute	Description
Participants	Specify the participants for the activity. The format is (<t_1> p_1, <t_2> p_2, …, <t_i> p_i, ….). Scene object p_i must conform with type t_i.
Effect	Specify the effect after the activity is finished. Some APIs will be provided for authors.
LifeCycle	Specify when the activity can be performed. There are six options. ①"Default": If this activity is attached to some SCF node, it can be performed when the node is enabled. If this activity is not attached to any SCF node, it can be performed anytime. ②"Always": This activity can be performed anytime. ③"AE_*NodeName*": After the given SCF node is enabled, this activity can be performed. ④"AF_*NodeName*": After the given SCF node is finished, this activity can be performed. ⑤"BE_*NodeName*": Before the given SCF node is enabled, this activity can be performed.

	⑥"BF_*NodeName*": Before the given SCF node is finished, this activity can be performed.
Subgoal	Specify the node in the SCF. The activity will be attached to a given node (subgoal) in SCF. After the activity is finished, the subgoal will be achieved.

Table 4: Activity templates

Activity Template	**Attribute**		**Description**
Conversation	Participants Type	(<*PlayerCharacter*> *pc*, <*NonPlayerCharacter*> *npc*)	The player has a conversation with others.
	Default Precondition	*pc* collides with *npc* and the mouse click *pc*.	
	Dialog	{(speaker$_1$, content$_1$), (speaker$_2$, content$_2$), …}	
Take Item	Participants Type	(<*PlayerCharacter*> *pc*, <*Item*> *item*)	The player takes some item to his inventory.
	Default Precondition	*pc* collides with *item*, and *item* is selected by the mouse.	
Use Item	Participants Type	(<*PlayerCharacter*> *pc*, <*Item*> *item*, <*SceneObject*> *target*)	The player uses some item in his inventory (to some target).
	Default Precondition	<u>Case 1: *target* is null</u> The player clicks the useButton for *item* in *pc*'s ventory. <u>Case 2: *target* is not null</u> When *pc* collides with *target*, the player clicks the useButton for *item* in *pc*'s ventory.	
Trade	Participants Type	(<*PlayerCharacter*> *pc*, <*NonPlayerCharacter*> *npc*, <*Item*> *goods*)	The player buys some item from a trader and then the item is put in his inventory.
	Default Precondition	*pc* collides with *npc* and the player clicks the buyButton for *goods*.	
Time	Participants Type	(<*SceneObject*> *so*)	Some event will be triggered after a given interval of time.
	Default Precondition	A given time interval elapses.	

3.3. Scene Object (SO)

Scene Object denotes the objects that constitute the game scene, for example, a person, a dog, a tree, and so on. In order to describe the features and concepts of various SOs, we propose a Scene Object Ontology to classify SOs, analyze the inherited attributes and relations among SOs. The proposed ontology as shown in Figure 7, SOs are classified into **Dynamic Object** which is an animate object such as an animal and **Static Object** which is an inanimate object such as a building. There are two relations in Scene Object Ontology which are "*a kind of*" and "*a part of*". "*A kind of*" relation denotes the inheritance from the parent. For example, if SO_1 is a kind of SO_2, it denotes that some attributes of SO_1 inherit SO_2. "*A part of*" relation denotes scene object can be classified into several classes. For example, if SO_1 is a part of SO_2, it denotes that SO_1 is the subset of SO_2. The details of all scene objects are shown in Table 5.

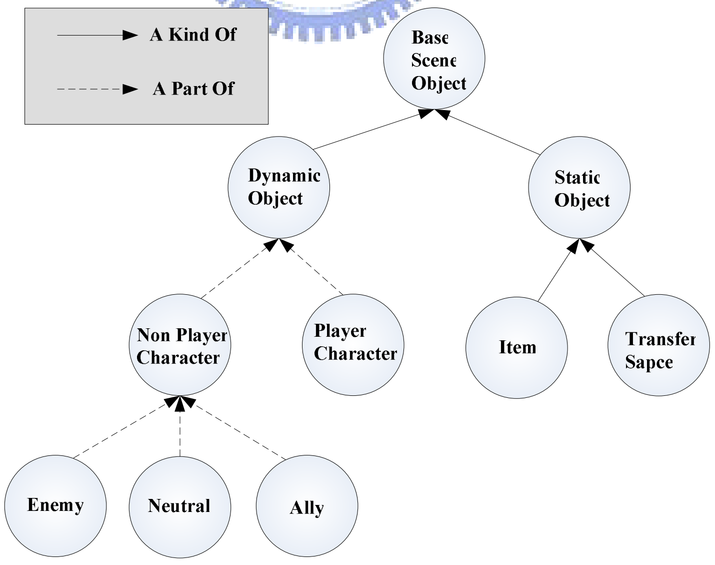


Figure 7: Scene Object Ontology

Table 5: The descriptions of concepts in Scene Object Ontology

Scene Object Type	**Description**
Base Scene Object	The base object in a scene of a game.
Dynamic Object	The animate object. For example, a dog.
Non Player Character	The AI object.
Player Character	The object that can be controlled by the player, i.e. the protagonist.
Enemy	The object that will attack Player Character.
Neutral	The object between Enemy and Ally.
Ally	The object that will help Player Character.
Static Object	The inanimate object. For example, a building.
Item	The object that can be taken and used by the Player Character. For example, a pen.
Transfer Space	When touching the Transfer Space, the player will be transferred to another scene. For example, a door.

3.4. Application of OOICM

In this section, we will explain how SCF, Activity, and SO communicate each other. As shown in Figure 8, there are four interfaces among the three components. ① An SCF has activities that attached to the SCF nodes. The types of activities must all conform to the types that SCF nodes specify. ② An SCF has a list of SOs. The types of SOs in the scene must all conform to the types that the SCF specifies. ③ An activity specifies the types of participant SOs. The types of SOs must all conform to the types that the activity specifies. ④ The states of SOs can be changed by an activity.

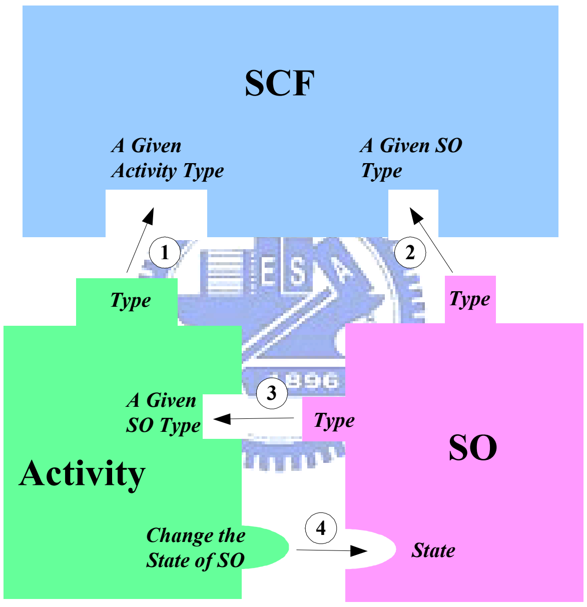


Figure 8: A diagram for communications among SCF, Activity, and SO.

The Construction and reuse of an RPG-like learning content are described as follows.

(1) The Process of Constructing an RPG-like Learning Content

Step 1. Construct an SCF by combining the SCF nodes and SCF connectors.

Step 2. Create SOs and configure their attributes.

Step 3. Create activities and configure their attributes.

Step 4(optional). Attach the activities in Step 3 to SCF nodes.

Example 4. Construct a conversation learning content.

(Step 1) Construct an SCF as same as Example 1. (**Step 2**) Add a *Player Character* named John and a *Non Player Character* named Mary to the scene. Then set the properties of John and Mary such as location, width, height. (**Step 3**) Add a *Conversation* activity named CS. Then set John and Mary as the participants of CS. (**Step 4)** Finally, attach CS to node1.

(2) Reusing an Existing SCF

Import an existing SCF into a new learning content and name the SCF scf.

Step 1. For each SO so_i in **O** of scf, create a new SO whose type is the same as so_i's.

Step 2. For each node n_i in **N** of scf, check the attached activity a_i of n_i and then create a new activity na_i whose type is the same as a_i's.

Step 3. For each na_i in step 2, check the types of participants (i.e. SOs) of a_i and then add the same type participants into na_i.

Step 4. Attach na_i to n_i.

Example 5. Construct a conversation learning content from an existing SCF.

We name the SCF in Example 1 scf and reuse scf to construct a new learning content. **(Step 1**) there are *Player Character* and *Non Player Character* type SOs in the cast of scf. Therefore, create *Player Character* named Bill and *Non Player Character* named Kelly. (**Step 2**) node1 has an attached *Conversation* activity which participants are *Player Character* and *Non Player Character* type. Therefore, create a *Conversation* activity named cs. (**Step 3**) Set Bill and Kelly as the participants of cs. (**Step 4)** Attach cs to node1.

Chapter 4. System Layer of OOICM

The system layer of OOICM specifies how OOICM actually works in our prototype system. We apply Petri Net to model SCF and apply frame knowledge representation to model Activity and SO. Petri Net is a powerful language for process flow modeling, concurrency handling, and validation. These properties of Petri Net are suitable to represent the concept of the SCF. SOs have their own attributes, inherited attributes, and event procedure call. Frame knowledge representation is suitable to represent the properties and functionality of SOs. In order to communicate with SO's frames, frame knowledge representation is also to represent activities. Then, Petri Net and frames will be transformed into ActionScript code. Although what authors see are SCF, activities, and SOs, our system actually handles Petri Net and frames

Adding the system layer for OOICM has the advantage described as follows. Transforming the OOICM into Petri Net and frames and then transforming Petri Net and frames into ActionScript codes is easier than transforming the OOICM into ActionScript codes straight. Besides, OOICM can be extended without modifying or adding ActionScript codes, because the system actually handles Petri Net and frames which originally have the fixed execution mechanism.

Petri Net and frames will be described detailedly in this Chapter respectively.

4.1. Petri Net for SCF (SCFPN)

Petri Net is a powerful language for process flow modeling, concurrency handling, and validation. These properties of Petri Net are suitable to represent the concept of the SCF. Therefore, we apply the Petri Nets to model the SCF. The definition of Petri Net for the SCF is described as follows.

Definition 2: The **Petri Net for Story Control Flow** is a 5-tuple.

SCFPN = (P, T, F, W, M$_0$), where

1. **P** $= \{p_1, p_2, ..., p_m\}$ is a finite set of places. P includes five types of places.
   - P_P : The progress of a story.
   - P_S : The start of a story.
   - P_E : The end of a story.
   - P_L : Check whether the activity is completed.
2. **T** $= \{t_1, t_2, ..., t_n\}$ is a finite set of transitions which disjoint form P ($P \cap T = 0$)
3. **F** $\subseteq (P \times T) \cup (T \times P)$ is a set of arcs (flow relation).
4. **W** : F→{1,2,3,…} is a weight function.
5. **M$_0$** $: P \rightarrow \{0,1,2,3,...\}$ is the initial marking and $M_0(P_S) = 1$.

Since every SCF node and SCF connector can be transformed into Petri Net, as shown in Table 6 and Table 7 respectively, every SCF can be also transformed into Petri Net.

Table 6: The corresponding Petri Net for SCF nodes.

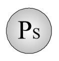
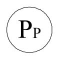
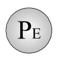
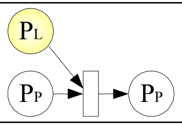
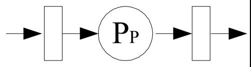
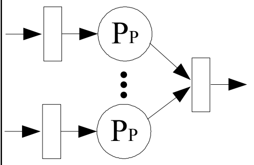
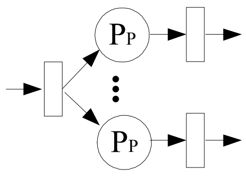
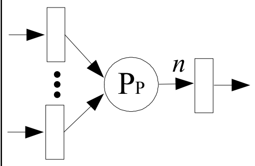
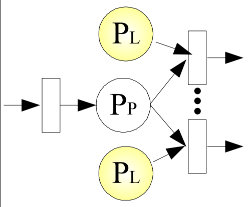
SCF Node Type	Petri Net Notation	SCF Node Type	Petri Net Notation
Start		Virtual	
End		Normal	

Table 7: The corresponding Petri Net for SCF connectors.

SCF Connector Type	Petri Net Notation	SCF Connector Type	Petri Net Notation
Linear		And	
Concurrence		Or	
Selection			

We also propose a transformation algorithm **SCF2PN** to transform the SCF into Petri Net. The algorithm is shown as follows.

Algorithm : SCF2PN

Input

scf denote the SCF that will be transformed into Petri Net.

Output

The transformed Petri Net.

Step 1. Create a new empty Petri Net pn.

Step 2. For every node n_i in scf, **ND2PN**(n_i, pn).

Step 3. For every connector c_i in scf, **CRT2PN**(c_i, pn).

Step 4. Output pn.

Procedure ND2PN(Node *n*, PetriNet *pn*)

Convert an SCF node to the corresponding Petri Net npn, and attach npn to *pn*.

Parameter

n is the SCF node that will be transformed into Petri Net.

pn is the attached Petri Net.

Step 1. Check the type of *n*.

Step 2. According

Table 6, add corresponding Petri Net block to *pn*.

Step 3. Assign the inport and outport places of *n*.

Procedure CTR2PN(Connector *c*, PetriNet *pn*)

Convert an SCF connector to the corresponding Petri Net cpn, and attach cpn to *pn*.

Parameter

c is the SCF connector that will be transformed into Petri Net.

pn is the attached Petri Net.

Step 1. Check the type of *c*.

Step 2. According Table 7, add the corresponding Petri Net block pnb to *pn*.

Step 3. Assign the tail and head transitions of *c*.

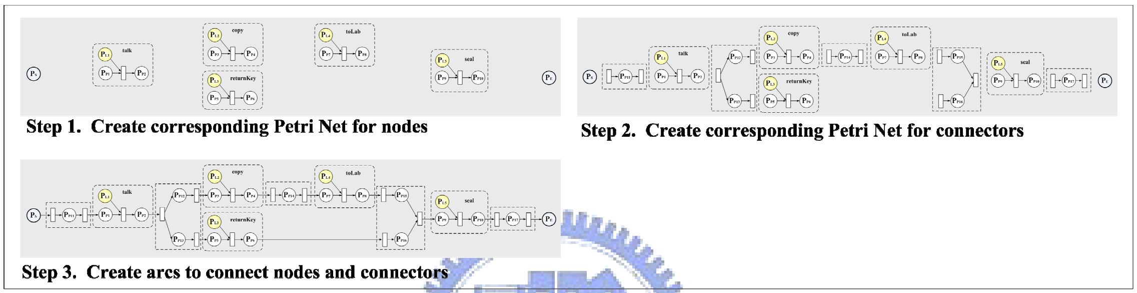
Step 4. Connect *c* with its PreNodes and PostNodes.

Step 4.1. For every PreNode n_i of *c*, create an arc to connect outport place of n_i and tail transitions of *c*.

Step 4.2. For every PostNode n_j of *c*, create an arc to connect head transition of *c* and inport place of n_j.

Example 6. Transform SCF in Example 1 into SCFPN.

Figure 9 shows the steps to transform the SCF of example into the corresponding Petri Net. According to the algorithm **SCF2PN,** ① Create a new empty Petri Net pn. ② Transform every node n_i into the corresponding Petri Net. ③ Transform every connector c_i into the corresponding Petri Net and create arcs to connect PreNode and PostNode of c_i.


Step 1. Create corresponding Petri Net for nodes
Step 2. Create corresponding Petri Net for connectors
Step 3. Create arcs to connect nodes and connectors
talk
copy
returnKey
toLab
seal
Figure 9: The steps to transform SCF in example 1 into Petri Net.

Example 7. Transform SCF in Example 2 into SCFPN.

The steps are the same as Example 3. The corresponding Petri Net of Example 2 is shown in Figure 9.

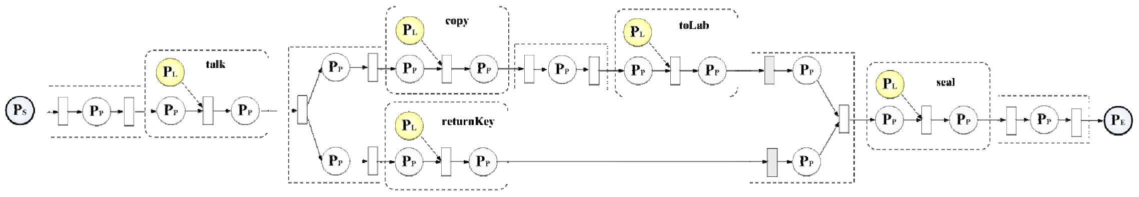


Figure 10: The corresponding Petri Net of example 2.

4.2. Activity Frame

In order to communicate with Scene Object frames, the frame knowledge representation is applied to model the functionality of activities. The frame of an activity is shown in Table 8. *PreCondition* slot and *InActivity* slot of the five templates all have default values, and the other slot values are specified by the content author as shown in Table 9.

Table 8: The descriptions of the slots in an activity frame.

Activity		
Slot Name	**Type**	**Description**
Actor	Scene Object	Scene objects that participant in this activity.
PreCondition	Rule	The condition that triggers this activity to start.
LifeCycle	String	Specify when the activity can be performed.
InActivity	Procedure	The actions that will be executed when the activity is proceeding.
Result	String	The result of this activity. So far, it is useful for only *Conversaton* activity.
PostAction	Rule	The actions that will be executed after the activity is finished.
GoalTest	Rule	The subgoal that this activity will achieve in SCF.

Table 9: Five kinds of templates of activity

Activity Template	**Default Slot Value**	
Conversation	PreCondition	If Actor.Collision(SceneObject *target*) == true AND , *target*.onRelease == true, then trigger InActivity.
	InActivity	Call Conversation.run()
Take Item	PreCondition	If Actor.Collision(Item target) == true and target.MouseDown == true, then trigger InActivity.
	InActivity	Call TakeItem.run()
Use Item	PreCondition	If Actor.Collision(SceneObject target) == true and Actor.Inventory.Items[i].clickUse == true, then

		trigger InActivity.
	InActivity	Call UseItem.run()
Trade	PreCondition	If Actor.Collision(SceneObject target) == true and target.goods[i].clickBuy == true, then trigger InActivity.
	InActivity	Call Trade.run()
Time	PreCondition	If Time.Elapsed == timeInterval, then trigger InActivity.
	InActivity	Call Time.run()

Example 8. A *Conversation* Frame Instance

The *Conversation* frame instance is shown in Figure 11. If John collides with Mary, a Conversation procedure will be called to show the dialogs of John and Mary. When the conversation ends, a result value will be added to the *Result* slot. After that, *PostAction* and *Goal* will be triggered. *PostAction* checks the *Result* slot value, if the value is "result1", then the appearance of Mary will be changed. *Goal* also checks the *Result* slot value, if the value is "result1", then it means subgoal1 is achieved.

Conversation	
Slot Name	**Value**
Actor	John
PreCondition	if Actor.Collision.Name == Mary then trigger InActivity
InActivity	Conversation procedure
Result	result1
PostAction	if Result == result1, set Mary.appearence = "smile.gif"
GoalTest	if Result == Result1, set subgoal1 = true

If added
Trigger
PostAction and
GoalTest

Figure 11: An instance of *Conversation* frame.

4.3. Scene Object Frame

We apply the frame knowledge representation to describe the attributes of the scene object. The attributes of the scene object can be classified into three categories which are **resource**, **profile**, and **behavior**. Resource denotes the external files of the scene object. Profile denotes the personal data of the scene object. Behavior denotes the event-driven behavior. The definition is described as follows.

Definition 3: The **Scene Object Frame** is a 4-tuple.

SOF = (FN, Rel, S), where

1. FN is the name of a frame.
2. Rel = {rel_1, rel_2, …,rel_h} and rel_k = <relation, FN> which is the relation with other frame specified by frame name FN. There are three types of relation –"*a kind of*", "*a part of*".
3. S = {s_1, s_2, ..., s_n}is a finite set of slots, and s_i = <SN_i, V_i, P_i>, where
   - SN_i is the name of the i-th slot
   - V_i is the value of the i-th slot.
   - P_i is a attached procedure that can be triggered by "if added" events.

Example 9. A frame instance for *Non Player Character*

Figure 12 shows an example of *Non Player Character* frame. The scene object Mary will be located at the coordination (10,100), and its appearance is the image from "C:\Mary.png". The Collision slot is null because no other SOs collide with it.

	Non Player Character	
	Slot	**Value**
	Name	Mary
profile	Location	(10,100)
	Size	Width = 20, Height = 30
resource→	Appearance	"C:\Mary.png"
behavior→	Collision	null

Figure 12: The frame representation of a SO.

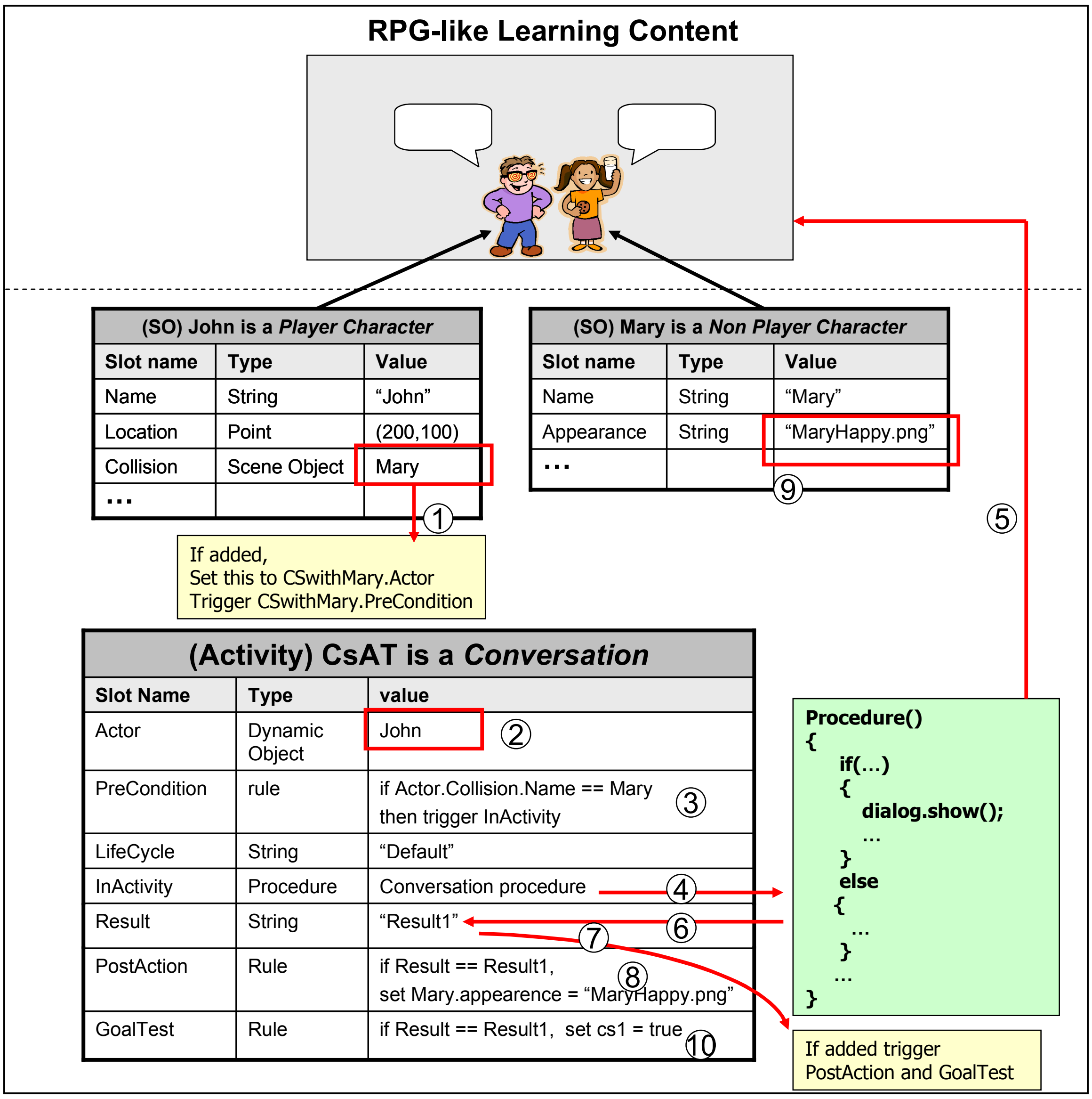
4.4. The OOICM Running Process

In this section, we will discuss the running process of frames and Petri Net. We give an example of a simple conversation scenario to explain that.

Example 10. The learning content of a simple conversation scenario.

Figure 4 is the SCF of this game. The initial mark of SCFPN is State 1 shown in Figure 14. P_S has one token. As shown in Figure 13, the scene of the learning content contains two scene objects which are *Player Character* type and *Non Player Character* type respectively. For the sake of convenience, we simplify the two frames in our example. The man called John and the girl called Mary will have a conversation activity. The frame representation of the conversation activity called *CsAT* is shown in the down side of Figure 13.

After game starts, the Petri Net runs and becomes State 2 in Figure 14. All scene objects are set according to their frames. The player can press up, down, left, and right key to move John. When the player presses left key, John's location is changed and user event interrupt happens. Therefore, the Location slot value of John frame is updated. If John collides with Mary, *CsAT* will be triggered as shown in Figure 13. The inference process is ① the Collision slot value of John frame is added and then the attached procedure is triggered. ② Actor slot value of *CsAT* is added. ③ Check whether PreCondition is satisfied. ④ PreCondition is true, so run procedure of InActivity. ⑤ The procedure of InActivity causes John and Mary to converse. ⑥. After the conversation, the procedure of InActivity add result to Result slot. ⑦ Result slot is added, so attached procedure is triggered. ⑧ The PostAction procedure is run to set Appearance slot of Mary frame. ⑨ Appearance slot of Mary frame is updated. ⑩ The GoalTest procedure is run to add token in Petri Net, and therefore Petri Net for SCF becomes State 3 in Figure 14. Final, the Petri Net runs again and becomes State 4 in Figure 14. Because P_E in Petri Net has one token, the story ends.



(SO) John is a *Player Character*		
Slot name	Type	Value
Name	String	"John"
Location	Point	(200,100)
Collision	Scene Object	Mary
…		

(SO) Mary is a *Non Player Character*		
Slot name	Type	Value
Name	String	"Mary"
Appearance	String	"MaryHappy.png"
…		

If added,
Set this to CSwithMary.Actor
Trigger CSwithMary.PreCondition

(Activity) CsAT is a *Conversation*		
Slot Name	Type	value
Actor	Dynamic Object	John
PreCondition	rule	if Actor.Collision.Name == Mary then trigger InActivity
LifeCycle	String	"Default"
InActivity	Procedure	Conversation procedure
Result	String	"Result1"
PostAction	Rule	if Result == Result1, set Mary.appearence = "MaryHappy.png"
GoalTest	Rule	if Result == Result1, set cs1 = true

```
Procedure()
{
    if(…)
    {
        dialog.show();
        …
    }
    else
    {
        …
    }
    …
}
```

If added trigger
PostAction and GoalTest

Figure 13: The OOICM running process.

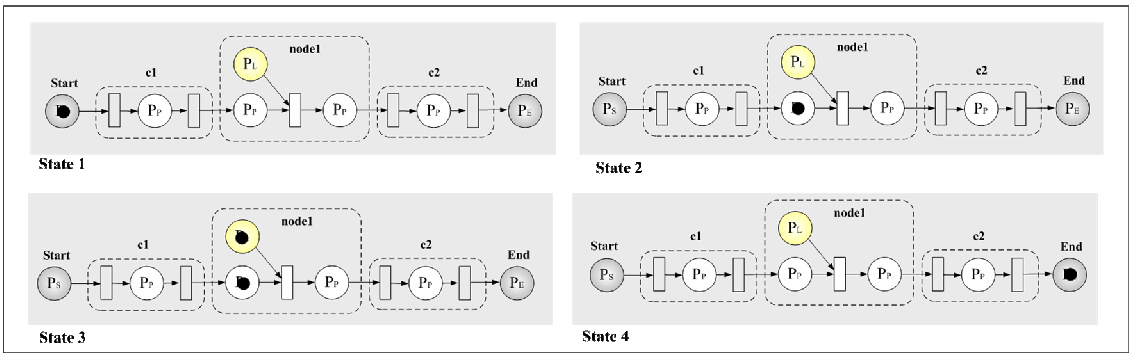


Figure 14: The running process of Petri Net

Chapter 5. ActionScript Transformation Process

In this Chapter, we will introduce our prototype system. Figure 15 shows the prototype system. The SCF will be transformed into Petri Net by mean of **SCF2PN** that has been discussed in 4.1. Activity and SO will be transformed into frames by filling in the slots of frames according to the attributes of Activity and SO. Because we apply Petri Net to model SCF and apply frame knowledge representation to model Activity and SO in the system layer of OOICM, the system environment must contain a Petri Net engine and a frame engine. Therefore, we implement the Petri Net engine and frame engine in ActionScript. We also propose an algorithm OOICM2AS to transform Petri Net and frames into ActionScript codes.

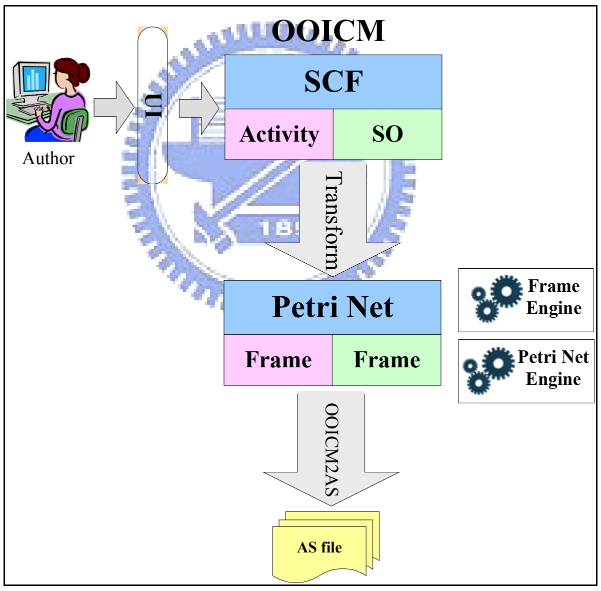


Figure 15: A prototype system based on OOICM

(1) Petri Net Engine

Besides the basic capability of Petri Net engine, there are two additional requirements for this engine in order to handle the SCFPN.

a. The type P_L place is a place which the activity frame can add tokens or remove tokens to. Therefore, the Petri Net engine must be able to receive the commands form frames to add tokens or remove tokens of places.

b. The end of a game depends on whether the type P_E place has tokens. Therefore, the Petri Net engine must be able to dispatch an ending message when the type P_E place has tokens.

(2) Frame Engine

A frame engine is necessary in order to handle the inference of activity frames and SO frames. We implement each activity frame and each Scene Object frames as several ActionScript classes. The communications among frames and Petri Net mainly rely on the ActionScript API *dispatchEvent()*.

(3) OOICM2AS

OOICM must be transformed into ActionScript (AS) code so that the compiler can compile the code into SWF file. We have implemented SCFPN AS class, Activity AS class, SO AS class. Besides, we also give a middle AS file called "template.as" to help the transformation. Figure 16 shows the outline of "template.as".

```
import mx.controls.*;
import mx.utils.Delegate;
class FlashGame
{   …
    //<SceneObject declare>
    …
    //<Activity declare>
    …
    //<Petri Net instance>
   …
    //<SceneObject instance>
    …
    //<SceneObject setting>
    …
    //<Activity instance and setting>
    …
}
//<Activity PostAction setting>
...
```

Figure 16: The outline of "template.as"

The **OOICM2AS** algorithm is described as follows.

```
Algorithm: OOICM2AS
Input
    SCFPN, Activity Frames, SOFs, and an AS file "template.as".
Output
    An ActionScript File.
Step 1. Create a new file "source.as".
Step 2. Read the next line from "template.as", and write the line in "soruce.as"
Step 3. Check the line in Step 2.
    Case 1: the line is "//<SceneObject declare>"
        For each SOF, get the value of slot Name, and write the declaration in
        "source.as".
    Case 2: the line is "//<Activity declare>"
        For each Activity Frame, get the value of slot Name, and write the declaration in
        "source.as".
    Case 3: the line is "//<Petri Net instance>"
        For each place, transition, and arc, write their declaration and attributes setting in
        "source.as".
    Case 5: the line is "//<SceneObject instance>"
        For each Activity Frame, get the value of slot Name, and write the declaration in
        "source.as".
    Case 6: the line is "//<SceneObject setting>"
        For each SOF, get the attributes from its frame and write the attribute setting in
        "source.as".
    Case 7: the line is "//<Activity instance and setting>"
        For each Activity Frame, get the attributes from its frame and write the attribute
        setting in "source.as".
    Case 8: the line is "//<Activity PostAction setting>
        For each Activity Frame, get the value of slot PostAction and write a PostAction
        function in "source.as".
Step 4. if the line is final line in "template.as", then go to Step 5, else go to Step 2.
Step 4. Output "source.as"
```

Chapter 6. Experiment

For evaluating the OOICM model, we implement a generator based on OOICM. The generator transform XML file into Flash file. Authors construct the RPG-like learning content by editing XML. In addition, several scenarios are given to evaluate the expressive power of OOICM and we also compare performance of the generator with Adobe Flash and RPG-Maker.

6.1. System Implementation

Our generator has four inputs and one output. “Story.xml”, “Activity.xml”, and “Scene.xml” are the specifications of *SCF, Activity*, and *Scene Object* respectively. *Transform process* will parse these xml files and then apply **OOICM2AS** algorithm to transform them into ActionScript file called “FlashGame.as”. “Source.xml” describes what asset is imported, such as images and sounds. It is the input for *Swfmill* [15] that is an xml2swf and swf2xml processor with import functionalities. *Swfmill* will import the assets described in “Source.xml” into a blank flash file called “source.swf”. After that, “FlashGame.as” and “source.swf” are inputted to *MTASC* [16]. *MTASC* is an ActionScript 2 Open Source free compiler. It can compile large number of .as class files in a very short time and generate directly the corresponding SWF bytecode without relying on Adobe Flash or other tools. “FlashGame.as” may call function from Library, so MTASC has to import class from Library. Final, MTASC will generate a Flash file called “FlashGame.swf”.

6.2. Experiment Designs

We use the generator to generate the learning content and compare with Adobe Flash and RPG-Maker. The experiment gives a scenario of the standard operation procedure. The player plays the role of a graduate. The scenario describes the player has to complete several procedures in order to get the graduation certificate. Figure 17 shows the SCF, and Figure 18 shows the screenshot of the game.

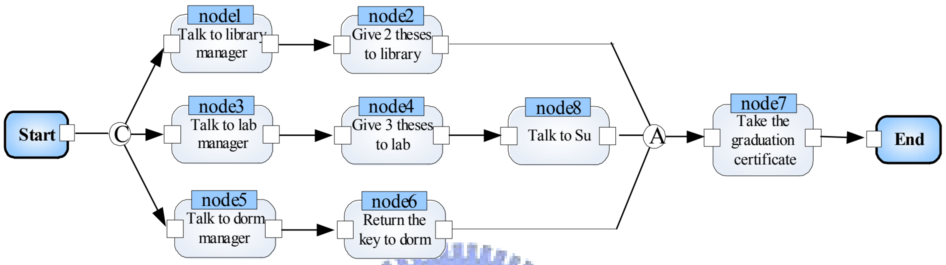
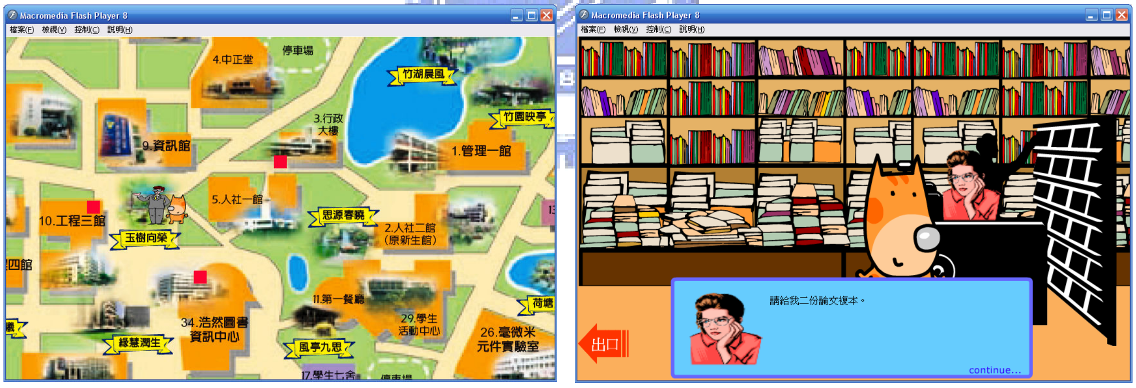


Figure 17: The SCF of our experiment



請給我二份論文複本。

Figure 18: The screenshot of the learning content

6.3. Experiment Results

We count the steps of constructing this learning content for our generator and RPG-Maker. Figure 19 shows the comparison of the number of steps. When constructing a new learning content in our experiment design, the generator spends about 350 steps and RPG-Maker spends about 250 steps. Because we have to construct SCF, the cost for constructing a new learning content is more than RPG-Maker. However, the generator spends fewer steps than RPG-Maker for reusing a learning content. In the experiment, we reuse the SCF, the author just reconfigure the activities and SOs. In addition, the AS code, generating by generator, contains 686 lines. If we take the other library into account, authors have to write more than 686 lines of code in Adobe Flash.

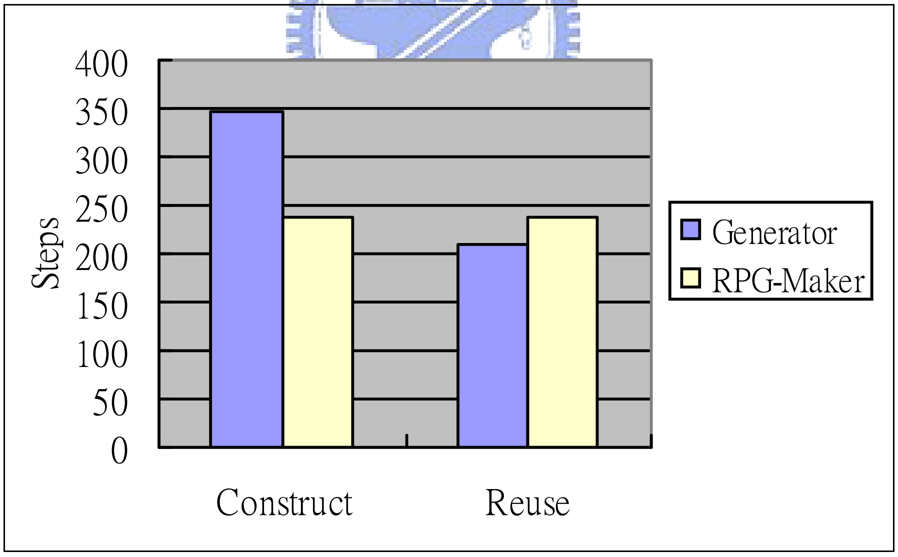


Figure 19: The comparison of the number of steps

Chapter 7. Conclusion

In this thesis, we apply knowledge-based approach to propose OOICM, which is composed of SCF, Activity, and SO. We apply Petri Net to model SCF and apply Frame knowledge representation to model Activity and SO. An ontology is proposed to describe the relations of all kinds of SOs. Moreover, we also implement a generator to evaluate OOICM. This generator can help authors construct an RPG-like flash learning content without low level programming.

It is tedious to edit the XML files for the input of the generator. Therefore, we will develop an authoring tool to help authors edit XML file by a user-friendly UI in near future. In addition, MMORPG (Massive Multiplayer Online Role Playing Game) has become very popular in recent years, because the player can play the game with other real people rather than AI agents. We are trying to support MMORPG in OOICM. Moreover, the battle, which is a complex activity actually, is very attractive to most players in the computer game. We will also add the battle activity into OOICM in the future.

References

[1] Role-Playing Exercises, Created by Rebecca Teed, SERC, Carleton College, http://serc.carleton.edu/introgeo/roleplaying/index.html

[2] Flash Player Penetration,http://www.adobe.com/products/player_census/flashplayer/

[3] Arturo Nakasone, Mitsuru Ishizuka, "Storytelling Ontology Model using RST", *Proceedings of the IEEE/WIC/ACM International Conference of Intelligent Agent Technology 2006*

[4] Mark Riedl, C. J. Saretto, R. Michael Young, "Managing Interaction between Users and Agents in a Multi-agent Storytelling Environment", *Proceedings of the second international joint conference on Autonomous agents and multiagent systems, 2003*

[5] RM Young, MO Riedl, M Branly, A Jhala, RJ Martin, C. J. Saretto, "An Architecture for Integrating Plan-based Behavior Generation with Interactive Game Environments", *Journal of Game Development, 2004*

[6] Mark O. Riedl and R. Michael Young, "From Linear Story Generation to Branching Story Graphs", *IEEE Computer Graphics and Applications Special Issue on Interactive Narrative, 2006*

[7] B Magerko, JE Laird, M Assanie, A Kerfoot, D Stokes, "AI Characters and Directors for Interactive Computer Games", *16th Innovative Applications of Artificial Intelligence Conference, 2004*

[8] Brian Magerko, "Story Representation and Interactive Drama", *1st Artificial Intelligence and Interactive Digital Entertainment Conference, 2005*

[9] S Natkin, L Vega, "A Petri Net Model for Computer Games Analysis", *International Journal of Intelligent Games & Simulation, 2004*

[10] S Natkin, L Vega, S Grünvogel, "A new Methodology for Spatiotemporal Game Design", *Proceedings of CGAIDE, 2004*

[11] Martin Purvis, "Narrative Structures for Multi-Agent Interaction", *Proceedings of*

the IEEE/WIC/ACM International Conference on Intelligent Agent Technology, 2004

[12] Clark Verbrugge, “A Structure for Modern Computer Narratives”, *CG’2002: International Conference on Computers and Games, 2002 - Springer*

[13] Adobe Flash, http://www.adobe.com/products/flash/

[14] RPG Maker, http://www.enterbrain.co.jp/tkool/RPG_XP/eng/

[15] Swfmill, http://osflash.org/swfmill

[16] MTASC, http://www.mtasc.org/