Constructing a Role Playing Interactive Learning Content Model

Dung-Chiuan Wu[1], Huan-Yu Lin[2], Shian-Shyong Tseng*[3], Jui-Feng Weng[4], Jun-Ming Su[5]
Department of Computer Science National Chiao Tung University, ROC
Department of Information Science and Applications, Asia University, ROC[3]
donchen.cs94g@nctu.edu.tw[1], *huan.cis89@nctu.edu.tw*[2], *sstseng@cis.nctu.edu.tw*[3],
roy@cis.nctu.edu.tw[4], *jmsu@csie.nctu.edu.tw*[5]

Abstract

In the early years, the E-learning contents are always written by static HTML pages, which can not provide a highly interactive learning activity to fulfil the need of some simulation based learning, e.g., Standard Operation Procedure Training. As we known, the learning objective of simulation based learning can be achieved via Role-Playing Flash interactive learning contents. Since this kind of contents is usually manually created, the construction is time-consuming and costly. Thus, we propose an Object Oriented Interactive Content Model (OOICM) representing the high level game knowledge for authors with the interface of OOICM to assist authors in constructing a Role-Playing learning content easily without writing low level codes. OOICM is composed of three components: Story Control Flow (SCF), Activity, and Scene Object (SO). System layer of OOICM, where we apply Petri Net based rule set to model SCF and use frame knowledge representation to model Activities and SOs, is also defined as the middleware to separate OOICM from low-level implementation. Based on OOICM, we implement a generator with frame engine to help authors to construct the learning content. Finally, we compare the generator with other authoring tools and the experiment result shows the generator has better reusability.

1. Introduction

With the growth of Internet, the technologies of e-learning are globally accepted for making learners study anytime and anywhere. In early years, these e-Learning contents are static and lack interactive features due to the pure HTML format, so some learning objectives, involving scenario simulations and requiring the interactions with learners, are difficult to be fulfilled. For example, Standard Operation Procedure training, which needs a scenario simulation learning activity to improve the impression, can not be properly represented in the traditional web pages. Since role playing is an instructional alternative suitable for scenario simulations, it mainly has three advantages which are "Motivating Students", "Augmenting Traditional Curricula", and "Learning Real-World Skills" [1]. According to the statistics [2] from Adobe, Flash Player is the most pervasive software platform in the world, which can represent rich 2D animations on WWW and handle various interactions by writing ActionScript code, so Flash is an appropriate media to represent this kind of interactive learning contents. However, creating a Flash content for interactive learning is time-consuming and costly especially for nonprogrammer, and these contents are difficult to reuse and share, because authors have to write ActionScript codes to handle various events. Therefore, how to facilitate the creation of Role-Playing Flash content and reuse the learning content are important issues.

Role-Playing content is composed of story (narrative), characters and scenes. All characters are arranged in the scene and the story describes the goals flow for characters to act. According to this structure, we propose Object Oriented Interactive Content Model (OOICM) to represent the high level knowledge of Role-Playing learning content and make authors construct a Role-Playing learning content easily without writing low level codes. OOICM is composed of Story Control Flow (SCF), which is a sequence of sub-goals, Activity, which is the action for SOs to perform, and Scene Object (SO), which is the object in a scene. We also define the system layer to separate the high level knowledge from low level implementation. In system layer, we apply Petri Net based rule set, which can support process flow modeling, concurrency handling, and validation, to model SCF and apply frame knowledge representation, which can represent attributes, event procedure, inheritance relation, and provides a systematic inference mechanism, to model Activity and SO.

Because we apply rule set and frame, the system environment must contain a frame engine. We implement it in ActionScript and develop a generator based on OOICM to assist authors in constructing the Role-Playing learning content.

2. Related Work

2.1. Authoring Tools

Many tools can be used to create the interactive content.

Adobe Flash [3] is the original and popular authoring tool to create the flash format content. Since Flash is not originally designed to construct the Role-Playing content, authors have to add ActionScript code manually to simulate the behaviors of the role playing games. However, 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 [4], a powerful tool for creating role-playing games, provides a user-friendly editor interface for creating a role-playing game without writing low level codes, where events are attached to the grids in the game map and the story control flow is event-driven but not represented explicitly. What authors see is the game scene and a lot of related events. The scenario of the game can not be understood by authors. It results in that the complex role playing content is difficult to construct, reuse, and maintain.

3. Object Oriented Interactive Content Model (OOICM)

As mentioned above, it seems necessary to design a Role-Playing interactive learning content model for non-professional game developers, such as teachers, to facilitate constructing and reusing interactive learning contents. Therefore, we propose Object Oriented Interactive Content Model (OOICM) from the authors' viewpoint to let authors construct complete Role-Playing interactive learning content easily. In OOICM, object-oriented methodology is used to encapsulate the internal control codes of interaction handling and define the component types to ensure the correctness in component reusing and composing.

As shown in Figure 1, OOICM is composed of three components: Story Control Flow (SCF), Activity, and Scene Object (SO). Story Control Flow denotes a flow of sub-goals, Activity denotes a specific action that SOs can perform, and some activities may achieve sub-goals in the SCF. Scene Object denotes the objects that constitute the scene of learning content.

In other words, with OOICM, learners need to play a character, defined as an SO, to perform some actions, defined as Activities, with other SOs to achieve the sub-goal and finish the story, defined as SCF.

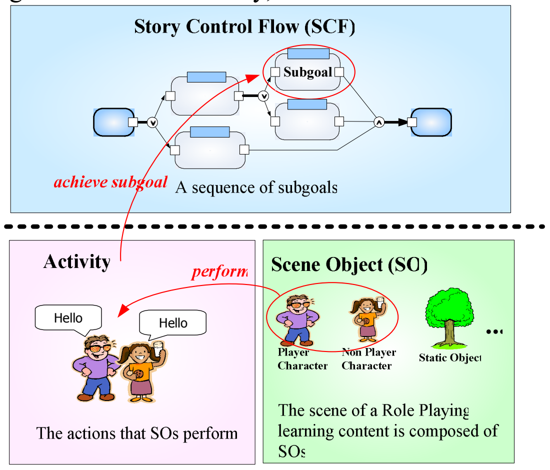


Figure 1: OOICM overview

3.1. Story Control Flow (SCF)

The SCF denotes a flow of sub-goals in a Role-Playing learning content. In other words, SCF is the story, which is a control flow to describe a flow of events that the learner has to experience, in the Role-Playing learning content. In [5, 6], the planning is applied to represent the story. In Interactive Drama Architecture (IDA) [7, 8], partial-ordering graph is used to represent story. 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 be used to characterize the topology of the virtual space in a game [10]. In [11], in order to characterize narrative structures, they propose "narrative nets" (N-nets), which are based on colored Petri nets. In [12], a representation framework called Narrative Flow Graphs (NFG), derived from 1-safe Petri Net, is proposed. NFG can be used to verify desirable properties, or as the basis for a narrative development system. These researches mainly attempt to verify the properties of the story flow such as the balance between the user control and the story coherence, but their story representation can not be easily understood by authors. Besides, a complete interactive learning content must contain Actions and Scene Objects, which can not be represented by above methodologies. Therefore, we define the SCF to model the story as follows.

Definition 1
The Story Control Flow is a 3-tuple **SCF** = (N, C, O), where

1. **N =** $\{n_1, n_2, \ldots, n_m\}$, which means the plots in a story, 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 the unique atomic nodes that specify the start and the end of a game respectively.
- The **Connective node** is an atomic node just for connecting different connectors.
- The **Regular node** is an atomic node that denotes the sub-goal in a game. It has an attached activity declaration.
- 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, which can be reused as a single node in 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.

2. **C** = $\{c_1, c_2, \ldots, c_n\}$, which means the flow directions of plots, 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**". If $|TN| = |HN| = 1$, the *t* would be "Linear", and if $|TN| = 1$, $|HN| > 1$, the branching sub-goal of *HN* can be "Concurrence" or "Selection" which means learners can achieve the all sub-goal or can select only one sub-goal. If $|TN| > 1$ and $|HN| = 1$, the sub-goal of *TN* may be "And" or "Or" means that all or one of the previous sub-goals should be achieved.

3. **O** = $\{o_1, o_2, \ldots, o_q\}$, which means the cast, 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.

To our knowledge, SCF is enough to represent most scenarios. The following is an example to illustrate SCF.

Example 1. The standard graduation procedure training

Figure 2 shows a partial graduation procedure, which is a standard operation procedure for a graduate to get a graduation certificate. In the procedure, the player has to talk to Mary to receive hints, and then the player knows that he/she has to make three copies of papers and give them to the laboratory. At the same time, the player also has to return the key of the research laboratory to the lab manager. In SCF, we use regular nodes to represent all the sub-goals, such as "Talk to Mary", "Make three copies of papers", use "Concurrence", marked as "C", to represent that the two sub-goals can be achieved concurrently, and use "And", marked as "A", to represent that the two sub-goals should be achieved both. When the learner reaches the End node, the activity is completed. This SCF can be encapsulated into single Super node, which can be reused as a part of story in the larger scenario of standard graduation procedure training.

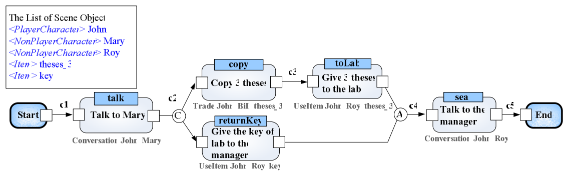


Figure 2: The SCF of the standard graduation procedure training

3.2. Activity

Activity is the action that the player can perform. Some activities may achieve some sub-goals in the SCF. The basic principle is that if the precondition of an activity is satisfied, this activity will be executed, and causes some post action to be executed. As shown in Table 1, authors must assign values to four basic attributes for every activity.

Table 1: The attributes of an activity

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.
Sub-goal	Specify the node in the SCF. The activity will be attached to a given node (sub-goal) in SCF. After the activity is finished, the sub-goal will be achieved.

In this paper, we predefine five kinds of activity templates which are "Conversation", "Take Item", "Use Item", "Trade", and "Time". "Conversation" denotes that the player has a conversation with another Non Player Character. "Take Item" denotes that the player takes some item to his inventory. "Use Item" denotes that the player uses some item in his inventory (to some target). “Trade” denotes that the player buys some item from a trader and then the item is put in the player’s inventory. “Time” denotes that some event will be triggered after a given interval of time. 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 described in Table 2.

Table 2: Activity templates

Activity Template	Attribute	
Conversation	Participants Type	(*<PlayerCharacter> pc*, *<NonPlayerCharacter> npc*)
	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*)
	Default Precondition	*pc* collides with *item*, and *item* is selected by the mouse.
Use Item	Participants Type	(*<PlayerCharacter> pc*, *<Item> item*, *<SceneObject> target*)
	Default Precondition	Case 1: *target* is null The player clicks the useButton for *item* in *pc*’s ventory. Case 2: *target* is not null 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*)
	Default Precondition	*pc* collides with *npc* and the player clicks the buyButton for *goods*.
Time	Participants Type	(*<SceneObject> so*)
	Default Precondition	A given time interval elapses.

3.3. Scene Object (SO)

Scene Object denotes the objects that constitute the scene, e.g., a person, a dog, a tree, and so on. In order to describe the features and concepts of various SOs, we propose Scene Object Ontology to classify SOs with the inherited attributes and relations. As shown in Figure 3, 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. The relation among SOs in Scene Object Ontology is “*A kind of*”, which denotes the inheritance from the parent. For example, “Non Player Character” is a kind of “Dynamic Object”, so the attributes of “Dynamic Object” are inherited by “Non Player Character”. The details of all scene objects are shown in Table 3.

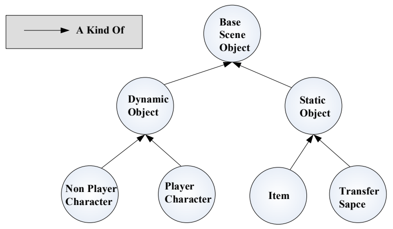


Figure 3: Scene Object Ontology

Table 3: The descriptions of concepts in Scene Object Ontology

Scene Object Type	Description
Base Scene Object	The base object in a scene.
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.
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.

4. System Layer of OOICM

In order to separate the high level knowledge from low level implementation, the system layer of OOICM, where we apply Petri Net based rule set to model SCF, and the frame knowledge representation to model Activity and SO, is defined. The system layer of OOICM can make transformation from OOICM into ActionScript easier with the powerful knowledge representations, frames and Petri Net, as middleware, and OOICM can be extended without modifying ActionScript code structures, because the system actually handles rule set and frames with fixed execution mechanism.

4.4. Petri Net for SCF (SCFPN)

The Petri Nets are used to model the SCF, and can be transformed into rules, which are easy to be executed, when system is running. The definition of Petri Net for the SCF is described as follows, and the transformation method is shown in Table 4, Table 5.

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

SCFPN = (P, T, F, W, M_0), where

1. **P** = $\{p_1, p_2, \ldots, p_m\}$ 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** = $\{t_1, t_2, \ldots, t_n\}$ 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** : $F \rightarrow \{1,2,3,\ldots\}$ is a weight function.
5. $\mathbf{M_0}$: $P \rightarrow \{0,1,2,3,\ldots\}$ is the initial marking and $M_0(P_S) = 1$.

Table 4: The corresponding Petri Net for SCF nodes.

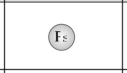
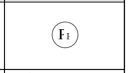
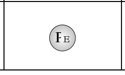
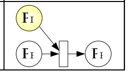
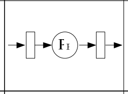
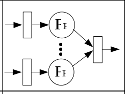
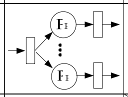
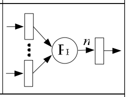
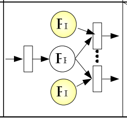
SCF Node Type	Petri Net Notation	SCF Node Type	Petri Net Notation
Start		Virtual	
End		Normal	

Table 5: The corresponding Petri Net for SCF connectors.

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

Example4. Transform SCF in Example 1 into SCFPN.

Figure 4 shows the steps to transform the SCF of Example 1 into the corresponding Petri Net. In Step 1, a new empty Petri Net pn is created. In Step 2, every node n_i is transformed into the corresponding Petri Net. In Step 3, every connector c_i is transformed into the corresponding Petri Net and arcs are created to connect PreNode and PostNode of c_i.

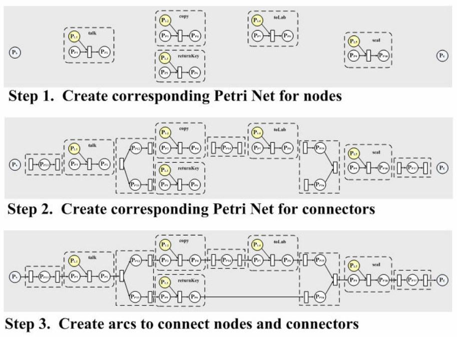

Step 1. Create corresponding Petri Net for nodes
Step 2. Create corresponding Petri Net for connectors
Step 3. Create arcs to connect nodes and connectors

Figure 4: Transform SCF in example 1 into Petri Net

4.5. Activity Frame

The frame of an activity is shown in Table 6. *PreCondition* slot and *InActivity* slot have default values, and the other slot values are specified by the author, as shown in Table 7.

Table 6: The descriptions of the slots in an activity frame.

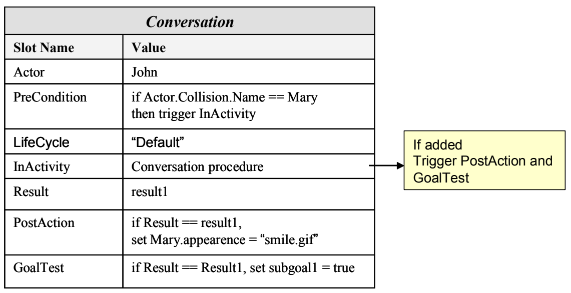
Activity		
Slot Name	**Type**	**Description**
Actor	Scene Object	Scene objects that participant in this activity.
PreCondition	Rule	The condition that triggers this activity to start.
LifeCycle	String	Specify when the activity can be performed.
InActivity	Procedure	The actions that will be executed when 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 sub-goal that this activity will achieve in SCF.

Table 7: 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 5. A *Conversation* Frame Instance

The *Conversation* frame instance is shown in Figure 5. 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 sub-goal1 is achieved.



Conversation	
Slot Name	**Value**
Actor	John
PreCondition	if Actor.Collision.Name == Mary then trigger InActivity
LifeCycle	"Default"
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 5: An instance of Conversation frame

4.6. 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 6. A frame instance for *Non Player Character*

Figure 6 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**
profile	Name	Mary
	Location	(10,100)
	Size	Width = 20, Height = 30
resource →	Appearance	"C:\Mary.png"
behavior →	Collision	null

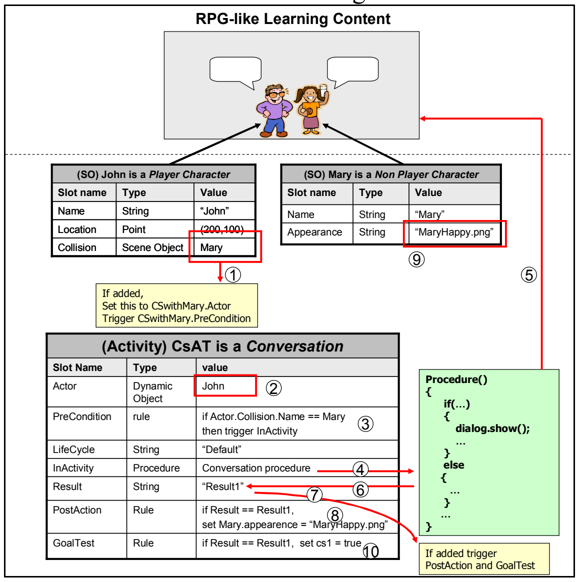
Figure 6: The frame representation of a SO

4.7. 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 7. The learning content of a simple conversation scenario.

Continuing from the Example 5, the initial mark of SCFPN is State 1 shown in Figure 8. P_S has one token. As shown in Figure 7, the scene of the learning content contains two scene objects which are *Player Character* type and *Non Player Character* type respectively. To simplify out description, we only show the partial 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 bottom side of Figure 7.



(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 7: The OOICM running process

After the learning activity starts, the Petri Net runs and becomes State 2 in Figure 8. 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, the user event interrupt happens, and if John collides with Mary, *CsAT* will be triggered as shown in Figure 7. In the inference process 1, the Collision slot value of John frame is added and then the attached procedure is triggered. In 2, Actor slot value of *CsAT* is added. In 3, the process should check whether PreCondition is satisfied. In 4, if PreCondition is true, the procedure of InActivity is run. In 5, the procedure of InActivity causes John and Mary to converse. In 6, after the conversation, the procedure of InActivity adds result to Result slot. In 7, Result slot is added, so attached procedure is triggered. In 8, the PostAction procedure is run to set Appearance slot of Mary frame. In 9, Appearance slot of Mary frame is updated. In 10, the GoalTest procedure is run to add a token in Petri Net, and therefore Petri Net for SCF becomes State 3 in Figure 8. Final, the Petri Net runs again and becomes State 4 in Figure 8. The learning activity will continue until P_E in Petri Net has one token and the story ends.

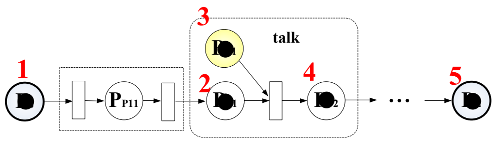


Figure 8: The running process of Petri Net

5. Implementation

In this Section, we will introduce our generator that can generate ActionScript codes and then compile the codes into Flash file. Figure 9 shows the prototype system of the generator. The SCF will be transformed into Petri Net based rule set. 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 rules 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 frame engine to make rules and frames work cooperatively. Therefore, we implement the Petri Net engine and frame engine in ActionScript, and propose an algorithm OOICM2AS to transform rule set and frames into ActionScript codes.

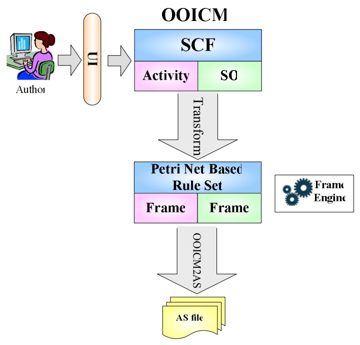


Figure 9: A prototype system based on OOICM

(1) 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()*.
(2) 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. OOICM2AS is a tedious file and string processing. It writes appropriate string into AS code file according to the setting of Petri Net and frames.
(3) Generator
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 "FlashLC.as". "Source.xml" describes what asset is imported, such as images and sounds. It is the input for *Swfmill* [13] 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, "FlashLC.as" and "source.swf" are inputted to *MTASC* [14]. *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. "FlashLC.as" may call function from Library, so MTASC has to import class from Library. Final, MTASC will generate a Flash file called "FlashLC.swf".

6. Experiment

For evaluating the OOICM model, we have implemented a generator based on OOICM. The generator transforms XML file into Flash file. Authors construct the Role-Playing learning content by editing XML. In addition, a scenario is given to evaluate the expressive power of OOICM and we also compare the generator with Adobe Flash and RPG-Maker.

6.1. Experiment Design

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 10 shows the SCF, and Figure 11 shows the screenshot of the game.

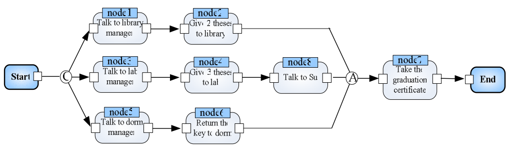


Figure 10: The SCF of our experiment

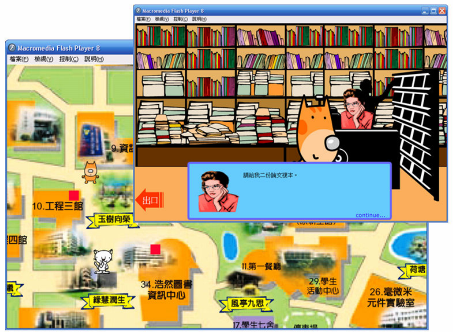


Figure 11: The screenshot of the learning content

6.2. Experiment Results

We count the steps of constructing this learning content for our generator and RPG-Maker. Figure 12 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.

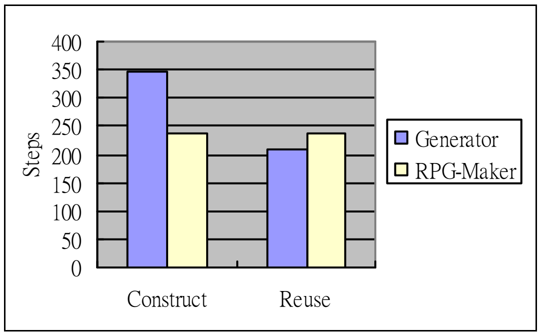


Figure 12: The comparison of the number of steps

7. Conclusion

In this paper, 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. Ontology is proposed to describe the relations of all kinds of SOs. Moreover, we also implement a Role-Playing interactive learning content generator and evaluate the effectiveness of Role-Playing interactive learning content authoring by means of OOICM. The result shows that this generator can help authors construct a Role-Playing flash learning content without low level programming and it can make reuse of content scenario easily.

8. Acknowledgement

This research was partially supported by National Science Council of Republic of China under the number of NSC95-2520-S009-007-MY3 and NSC95-2520-S009-008-MY3.

9. References

[1] Teed, R. *Role-Playing Exercises*. 2006 [cited 2007 Jul 20]; http://serc.carleton.edu/introgeo/roleplaying/.

[2] Adobe. *Flash Player Penetration* 2007 [cited 2007 Jul 20]; http://www.adobe.com/products/player_census/flashplayer/.

[3] Adobe. *Adobe Flash*. 2007 March 3 [cited 2007 Jul 20]; http://www.adobe.com/products/flash/.

[4] Enterbrain. *RPG Maker*. 2006 Jan 31 [cited 2007 Jul 20]; http://www.enterbrain.co.jp/tkool/RPG_XP/eng/.

[5] Riedl, M.O., C.J. Saretto, and R.M. Young, *Managing interaction between users and agents in a multi-agent storytelling environment*, in *Proceedings of the second international joint conference on Autonomous agents and multiagent systems* 2003.

[6] Young, R.M., et al., *An Architecture for Integrating Plan-based Behavior Generation with Interactive Game Environments*. Journal of Game Development, 2004. **1**(1).

[7] Magerko, B., et al., *AI Characters and Directors for Interactive Computer Games*, in *Proceedings of the 2004 Innovative Applications of Artificial Intelligence Conference*. 2004: San Jose, CA.

[8] Magerko, B., *Story Representation and Interactive Drama*, in *1st Artificial Intelligence and Interactive Digital Entertainment Conference*. 2005.

[9] Natkin, S. and L. Vega, *A Petri Net Model for Computer Games Analysis*. International Journal of Intelligent Games & Simulation, 2004. **3**(1).

[10] Vega, L., et al., *A new Methodology for Spatiotemporal Game Design*, in *Fifth Game-On International Conference on Computer Games: Artificial Intelligence, Design and Education*. 2004.

[11] Purvis, M., *Narrative Structures for Multi-Agent Interaction*, in *Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology*. 2004.

[12] Verbrugge, C., *A Structure for Modern Computer Narratives*, in *International Conference on Computers and Games*. 2002.

[13] OSFlash. *swfmill*. 2007 Mar 16 [cited 2007 Jul 20]; http://osflash.org/swfmill.

[14] Motion-Twin. *MTASC*. 2005 [cited 2007 Jul 20]; http://www.mtasc.org/.