# Web2Animation - Automatic Generation of 3D Animation from the Web Text

Hyunju Shim, Bogyeong Kang and Kyungsoo Kwag
*Digital Content Technologies Group*
*Samsung Advanced Institute of Technology, Giheung-Gu, Gyeonggi-Do, South Korea*
*Email: hyunju67.shim, bogyeong.kang, kyungsoo.kwag@samsung.com*

***Abstract*—In this paper, we introduce an animation production system, which generates a story from the web text in a specific domain and presents it in 3D animation. With our approach, 3D animation is generated in real-time from a web recipe in natural language. As a user starts the animation, only domain related texts are extracted using their associated tags and their semantics are analyzed by a semantic parser. The results of this semantic analysis are mapped to the computer animation commands through the action generation algorithms. To infer the hidden context within the text, we designed and leveraged the recipe ontology. A set of XML schemas has been designed to store different representations for web recipes; from the natural language representation to the animation representation.**

***Keywords*-web intelligence; 3D animation; XML, semantic engineering;**

## I. Introduction

Recently, with the advances in the web technologies, users' needs for searching, creating, sharing, and consuming web contents have increased. User generated contents(UGC) have entered in the mainstream of the web media reflecting the trend that users desire is not only in consuming the multimedia contents but also in producing them. As the producer of web contents has been shifted from experts to general users, more users post their own stories and videos on the web and also share their opinions about news and products with strangers via social networking services [5].

While the desire to create and share web contents among users has enlarged, a limited number of users is producing visual contents due to the difficulties in creating and editing them. As a result, most contents on the today's web are presented in textual forms, and visual contents are used as supplementary means that enhance what is presented in text. To meet the users' enlarged desire in creating their own contents, the next generation web must empower users to easily produce their contents in various forms, especially in visual ones. In fact, merging 3D technologies into the web has emerged as one of the features of the next generation web, and W3C developed the X3D technology [9] which enables web applications to incorporate with the advanced 3D technologies. Our approach to satisfy users' needs is to enable them to create web contents in a traditional textual form while the output is presented in 3D animation.

This paper introduces a novel approach for real-time production of 3D animation from the web text. We developed a system that produces 3D animation by analyzing the semantics of the web text. On accessing a web page of a specific domain, recipe, users click the animation production button to watch the 3D animation of the web page. Then text in the web page is converted into XML representations through our system. The XML contents are then automatically integrated with 3D graphical objects and character animations by leveraging the recipe ontology and applying the action generation algorithms.

## II. Related Works

This section introduces related works about automatic generation of animation from text. BEAT [1] is a behavior animation toolkit that automatically generates nonverbal behaviors for the input text. Beat automatically synchronizes body and face behaviors with synthesized speech. The BEAT system is mainly composed of three modules of language tagging, behavior generation and behavior scheduling. The language tagging module annotates the input text with the linguistic information for nonverbal behavior assignment and scheduling. Through overall process, Beat uses the XML tree structure to annotate language level tag and behavior generation knowledge. Unlike BEAT which generates gestures and facial expressions for the input text, our work analyzes the semantics of the web text and presents them in 3D animation.

WordsEye [2] is a system that takes the input text and creates a 3D scene visualizing the contents in the input text. The WordsEye parser converts the input text into a dependency structure that represents the relation between words in the sentence. The dependency structure is then interpreted into a semantic representation which shows the descriptions of the entities to be visualized for the scene. Once the semantic relations are obtained, the depiction rules are applied to convert the semantic representations into a set of 3D object and place them into a 3D scene. While WrodsEye constructs a static 3D scene from the input text, we construct a dynamic 3D scene, which is animation.

T2D [4] introduced an approach for transforming the monological text into a dialog with character animations. In the T2D system, the parser analyzes the input text in terms of a specific theory of coherence relations known as Rhetorical Structure Theory (RST). Once analyzed, the text is mapped into a dialog structure, called DialogueNet

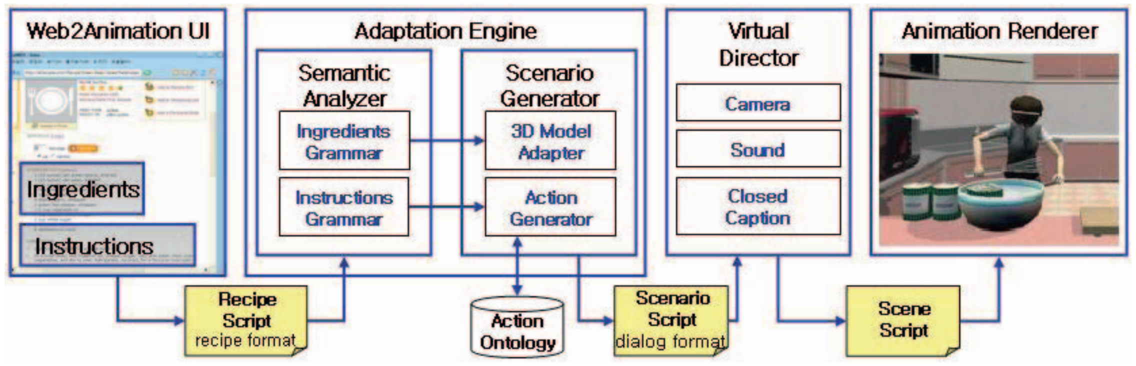


Figure 1. Web2Animation Internal Pipeline

structure. A DialogueNet structure corresponds to a text of the form *Speaker1 said P1*, *Speaker2 said P2*. The resulting DialogueNet structure is finally mapped into their 3D animation language, MPML3D to be rendered on a screen as 3D animation. While T2D mainly focuses on transforming monological text into a dialog, our approach focuses more on generating animation actions for the text based on its semantics.

e-Hon [7] is a system that transforms the web contents into a storybook with dialogues and animation. It is especially designed to assist children to understand web contents by animating them. The e-Hon system utilizes semantic tags that is associated with the text on the web. To transform the web contents into dialogues, the system generates a list of subjects, objects, predicates, and modifiers from the text and connects them in a colloquial style. In generating animation, a part-of-speech for each word in text is used to map them to characters and actions. Like e-Horn, our work automatically produces animation by generating dialogs for the input text. Ours, however, analyzes the semantics of the web text and generates a sequence of animation actions for 3D characters.

## III. Web2Animation System and Languages

In handling the unlimited vocabularies on the web text, we narrowed our focus into the domain of recipe. Our Web2Animation system constructs 3D animation based on the ingredients and instructions of the web recipes through the internal pipeline shown in Figure 1. The rest of this section introduces each component of this pipeline.

### A. User Interface

With our approach, converting the web text into animation is achieved through three steps of extracting domain-related texts, analyzing text semantics, and generating animation actions. The left most figure of Figure 1 shows the Web2Animation UI which simply embeds a web browser and provides a button for animation. As the user clicks the animation button, the Web2Animation UI extracts the information about the ingredients and instructions by looking

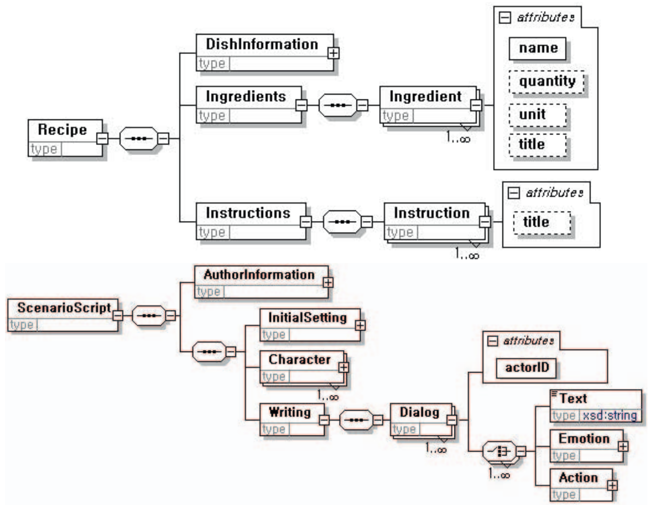
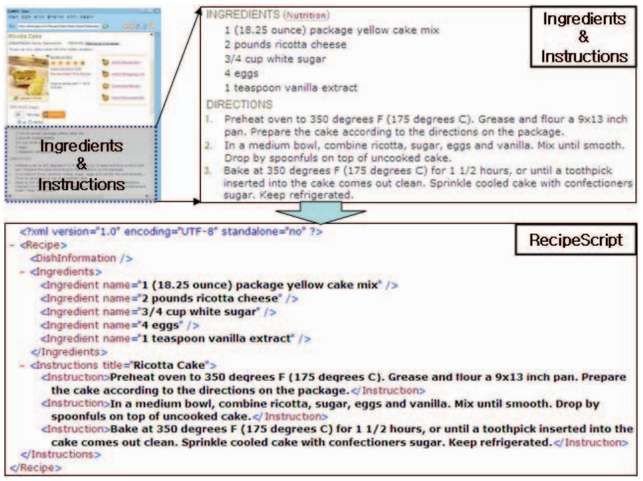


Figure 2. Schemas for RecipeScript and ScenarioScript

at the HTML tags in the loaded web page. Internally, the Web2Animation system uses XML as its primary data structure for storing recipes. Once extracted, the ingredients and instructions are stored into the XML format called RecipeScript, whose schema is shown on the top of Figure 2. Main role of RecipeScript is to store the recipe-related information from a web page in an XML format. The ingredients are stored as contents of *Ingredient* elements and the instructions are stored as contents of the *Instruction* elements. Figure 3 depicts each step of the conversion from a web recipe page to RecipeScript document.

### B. Adaption Engine

In order for a recipe in RecipeScript to be animatable, its contents must be mapped to actions and speeches of virtual characters. For this mapping, we designed an XML schema called ScenarioScript as shown on the bottom of Figure 2. The main goal of ScenarioScript is to store a story for 3D animation - it allows users to define a story using a set of

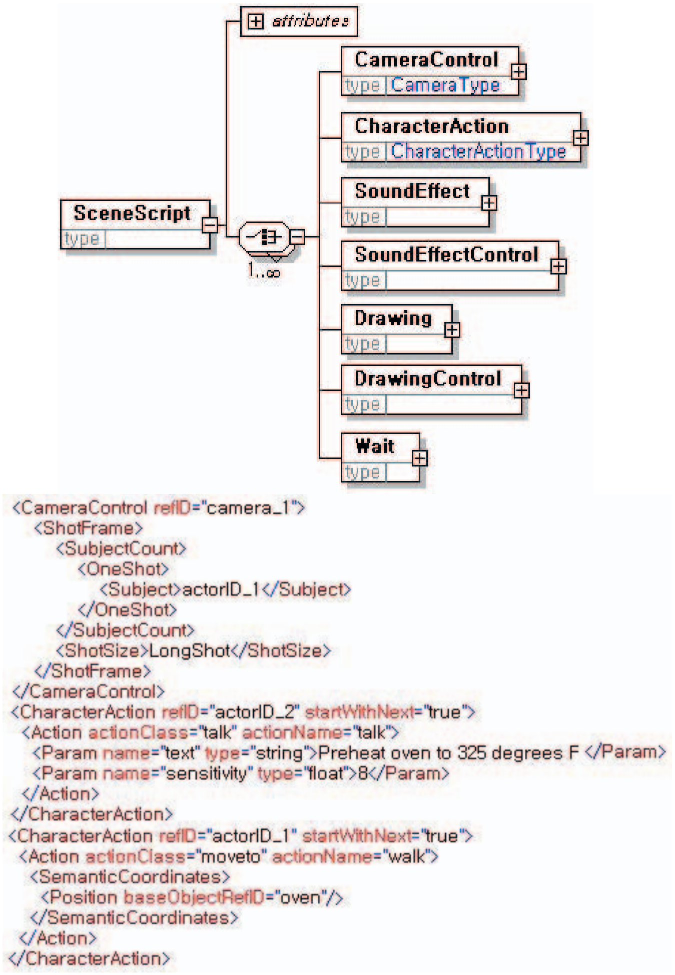


INGREDIENTS (Nutrition)
1 (18.25 ounce) package yellow cake mix
2 pounds ricotta cheese
3/4 cup white sugar
4 eggs
1 teaspoon vanilla extract

DIRECTIONS
1. Preheat oven to 350 degrees F (175 degrees C). Grease and flour a 9x13 inch pan. Prepare the cake according to the directions on the package.
2. In a medium bowl, combine ricotta, sugar, eggs and vanilla. Mix until smooth. Drop by spoonfuls on top of uncooked cake.
3. Bake at 350 degrees F (175 degrees C) for 1 1/2 hours, or until a toothpick inserted into the cake comes out clean. Sprinkle cooled cake with confectioners sugar. Keep refrigerated.

```
  <?xml version="1.0" encoding="UTF-8" standalone="no" ?>
- <Recipe>
    <DishInformation />
  - <Ingredients>
      <Ingredient name="1 (18.25 ounce) package yellow cake mix" />
      <Ingredient name="2 pounds ricotta cheese" />
      <Ingredient name="3/4 cup white sugar" />
      <Ingredient name="4 eggs" />
      <Ingredient name="1 teaspoon vanilla extract" />
    </Ingredients>
  - <Instructions title="Ricotta Cake">
      <Instruction>Preheat oven to 350 degrees F (175 degrees C). Grease and flour a 9x13 inch pan. Prepare
        the cake according to the directions on the package.</Instruction>
      <Instruction>In a medium bowl, combine ricotta, sugar, eggs and vanilla. Mix until smooth. Drop by
        spoonfuls on top of uncooked cake.</Instruction>
      <Instruction>Bake at 350 degrees F (175 degrees C) for 1 1/2 hours, or until a toothpick inserted into the
        cake comes out clean. Sprinkle cooled cake with confectioners sugar. Keep refrigerated.</Instruction>
    </Instructions>
  </Recipe>
```

Figure 3. Conversion from Web Texts to RecipeScript



```
<CameraControl refID="camera_1">
  <ShotFrame>
    <SubjectCount>
      <OneShot>
        <Subject>actorID_1</Subject>
      </OneShot>
    </SubjectCount>
    <ShotSize>LongShot</ShotSize>
  </ShotFrame>
</CameraControl>
<CharacterAction refID="actorID_2" startWithNext="true">
  <Action actionClass="talk" actionName="talk">
   <Param name="text" type="string">Preheat oven to 325 degrees F </Param>
   <Param name="sensitivity" type="float">8</Param>
  </Action>
</CharacterAction>
<CharacterAction refID="actorID_1" startWithNext="true">
  <Action actionClass="moveto" actionName="walk">
   <SemanticCoordinates>
     <Position baseObjectRefID="oven"/>
   </SemanticCoordinates>
  </Action>
</CharacterAction>
```

Figure 4. Schema for SceneScript and a Part of SceneScript Example

dialogs for 3D characters.

In formatting ScenarioScript, the traditional screenplay format is applied as a guideline [3]. In traditional screenplay, a dialog is normally composed of an actor's name with speeches, actions, and emotions. Using ScenarioScript, a recipe is described as a story with a sequence of dialogs. Inside the Web2Animation system, AdaptionEngine operates as an XML transducer that takes RecipeScript as an input and produces an output XML, ScenarioScript. Mainly, it converts recipe instructions into a story with a set of dialogs and assigns each dialog to a virtual character. The *Ingredient* elements of RecipeScript are mapped with 3D graphical objects and placed under the *InitialSetting* element. The *Instruction* elements are converted into the *Text*, *Emotion*, and *Action* elements, and assigned to a virtual character in the *Dialog* elements. Detailed transformation processes of AdaptionEngine are explained in Section IV.

### C. Virtual Director

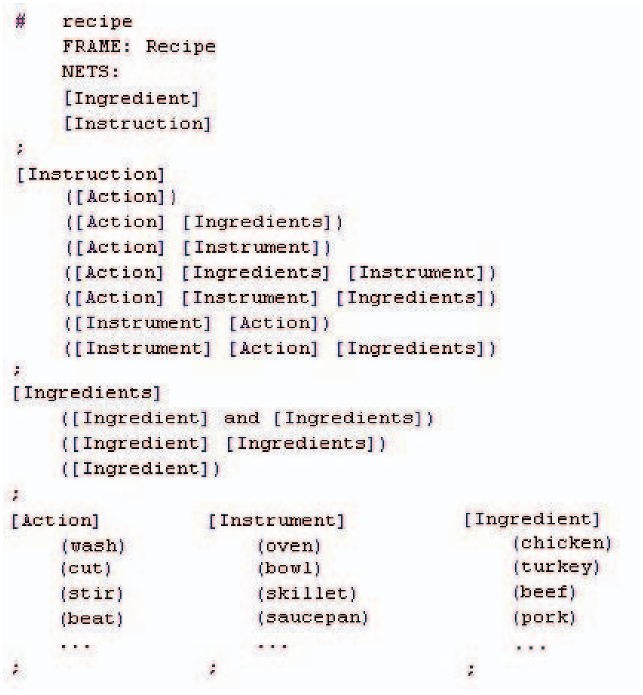
In order to generate 3D animation for dialogs, a virtual character must act according to the dialogs while a virtual camera is appropriately positioned. It is VirtualDirector that adds camera, sound, and closed-captions to the dialogs in ScenarioScript. The output of VirtualDirector is an animatable script called SceneScript whose schema is shown at the top of Figure 4. The SceneScript schema defines an XML format for 3D animation including character action, camera, sound, and drawing - a drawing is a 2D plane that may attach a text and be positioned on a screen to be used as closed-caption. At the bottom of Figure 4, a part of the SceneScript example is shown. In this example, a virtual camera aims a virtual character named *actorID_1* as a long shot. Then a virtual character says "*Preheat oven to 325 degrees F*" and walks to oven. SceneScript is rendered on a computer screen as 3D animation by the animation player that our team developed. Details about VirtualDirector and the resulted animation are introduced in Section V.

## IV. Adaption Engine

In the previous section, we show that AdaptationEngine takes ingredients and instructions in RecipeScript and outputs them as a story with a set of dialogs in ScenarioScript. This section explains how the internal process of the AdaptationEngine, whose basic idea is depicted in the Figure 1, works.

### A. Semantic Analyzer

To obtain the semantics within the contents of the ingredient and instruction elements in RecipeScript, we developed SemanticAnalyzer that utilizes Phoenix parser [8]. Phoenix is a rule-based parser based on manually constructed semantic grammars. It parses input string into a sequence of semantic frames - a frame is a named set of slots, where each slot represents related pieces of information. Each slot is associated with context-free grammars that specify word



```
#   recipe
    FRAME: Recipe
    NETS:
    [Ingredient]
    [Instruction]
;
[Instruction]
    ([Action])
    ([Action] [Ingredients])
    ([Action] [Instrument])
    ([Action] [Ingredients] [Instrument])
    ([Action] [Instrument] [Ingredients])
    ([Instrument] [Action])
    ([Instrument] [Action] [Ingredients])
;
[Ingredients]
    ([Ingredient] and [Ingredients])
    ([Ingredient] [Ingredients])
    ([Ingredient])
;
[Action]          [Instrument]          [Ingredient]
    (wash)            (oven)                (chicken)
    (cut)             (bowl)                (turkey)
    (stir)            (skillet)             (beef)
    (beat)            (saucepan)            (pork)
    ...               ...                   ...
;                 ;                     ;
```

Figure 5. Phoenix Grammars for Web2Animation

string patterns. When parsing, grammars in slots are matched against the input string.

Figure 5 shows the Phoenix grammars for a recipe. In this grammar, the main frame, called *Recipe*, is consisted with two slots named *[Ingredient]* and *[Instruction]*. The *[Ingredient]* slot matches the input string against a set of terminals which are the names of ingredients. Grammars in the *[Instruction]* slot are matched against the contents of the *Instruction* elements in RecipeScript. The *[Instruction]* slot is designed based on following semantics of recipe instructions:

- Recipe instructions are imperative sentences that start with a verb in a base form. The *[Instruction]* slot handles input strings that start with verbs. The string patterns of verbs are enumerated in the *[Action]* slot.
- Verbs in recipe instructions are either intransitive or transitive. In case of an intransitive verb, it is described without an object. This corresponds to the first pattern of the *[Instruction]* slot.
- In case of a transitive verb, it is accompanied with an object which usually corresponds to either an ingredient or a cooking instrument. This is handled by the rest of patterns in the *[Instruction]* slot where actions are accompanied with *[Instrument]* or *[Ingredient]*
- The grammar for the *[Ingredients]* slot is defined recursively to handle multiple ingredients that appear consecutively.
- Occasionally, cooking instruments appear before an action verb. This case is handled by putting the *[Instrument]* slot before the *[Action]* slot. This corresponds to the sixth and seventh patterns of the *[Instruction]* slot.
- The *[Action]*, *[Instrument]*, *[Ingredient]* slots enumerate the string patterns for the cooking actions, ingredients, and instruments, respectively.

When matched strings are found from the element contents in RecipeScript by SemanticAnalyzer, this information is handed over to ScenarioGenerator for the further process.

## *B. Scenario Generator*

The main role of ScenarioGenerator is to generate a scenario from the matched strings. For the matched ingredients, ScenarioGenerator adds the information about the corresponding 3D objects to the *InitialSetting* element of ScenarioScript so that they can be loaded in the final 3D animation scene. For the matched instructions, *Action Generator* creates a set of dialogs under the *Writing* element leveraging *Action Ontology* and *ActionGenerator* which are introduced in the following sections.

*1) Action Ontology:* Before developing action algorithms, we first designed an action ontology for cooking and attributed the ontological individuals as shown in Figure 6. Note that only a part of individuals are shown in Figure 6 while every string patterns listed in the *[Action]* slot is defined as individuals in the actual ontology.

At the top of the ontology is a root class, *action*. The *action* is either a positional, interactive, or stand-alone. These meta actions are again classified by a set of animation actions. The meta actions and the animation actions have *hasAnimation* relation. Under the animation actions, similar individual actions, which are shown as leaf nodes in Figure 6, are grouped together. In this way, similar actions can be animated using a single animation action. For example, *fry*, *boil*, and *beat* are animated using the same character animation named *stir* in which a character moves its arms around. The animation actions and individuals are related by *hasIndividual* relation.

Each individual is associated with a set of properties. These properties are used to infer the hidden context of the input when it has insufficient information for animation. Consider an instruction "*Chop beef*". A complete sentence for this instruction would be: "*Grip a knife and chop beef on a cutting-board*". In order to generate a feasible animation for the analogy-requiring instructions, we associated each individual with the required appliance, container, and instrument. In this way, when an action that requires a property appears in an instruction without its property, its ontological properties are inferred to acquire the necessary information. For the example instruction above, we can generate animation that a virtual character grips a knife and chops beef on a cutting-board.

*2) Action Generator:* Having designed the action ontology, we developed ActionGenerator that leverages the ontology and generates a sequence of animation actions

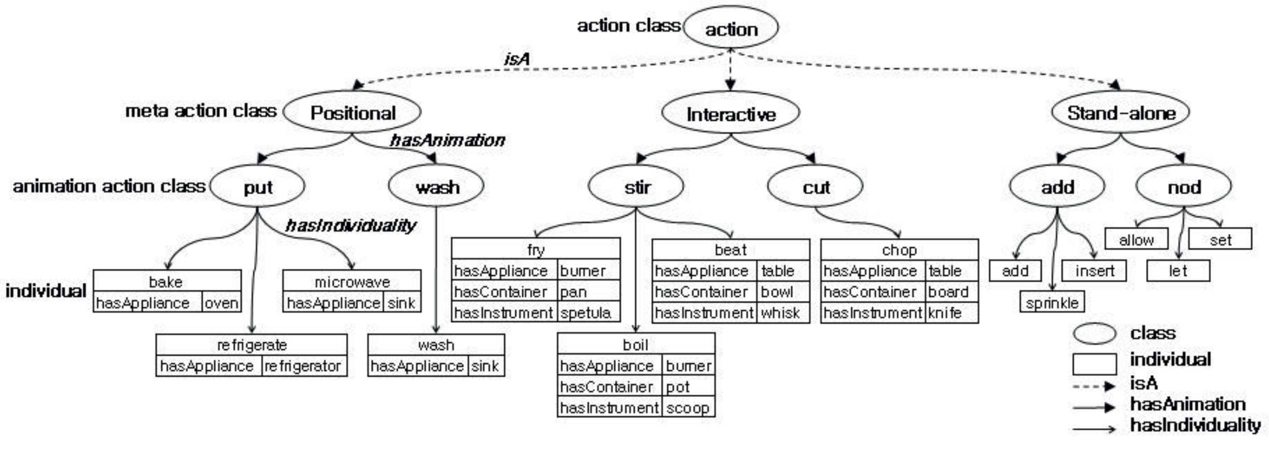

action class
action
isA
meta action class
Positional
Interactive
Stand-alone
hasAnimation
animation action class
put
wash
stir
cut
add
nod
hasIndividuality
individual
bake | hasAppliance | oven
microwave | hasAppliance | sink
refrigerate | hasAppliance | refrigerator
wash | hasAppliance | sink
fry | hasAppliance | burner | hasContainer | pan | hasInstrument | spetula
boil | hasAppliance | burner | hasContainer | pot | hasInstrument | scoop
beat | hasAppliance | table | hasContainer | bowl | hasInstrument | whisk
chop | hasAppliance | table | hasContainer | board | hasInstrument | knife
add
insert
sprinkle
allow
set
let
class
individual
isA
hasAnimation
hasIndividuality

Figure 6. Cooking Action Ontology

for the recipe instructions. As a result, *Instruction* elements in RecipeScript are reconstructed as *Dialog* elements with *Action*s, *Emotion*s, and *Text*s in ScenarioScript.

When ActionGenerator receives the matched strings from the *[Instruction]* slot from SemanticAnalyzer, it runs its action generation algorithms. Internally, ActionGenerator has a set of action algorithms to handle different cases of matched strings. Figure 7 shows the flowchart of the action algorithm for when an action and ingredient(s) are matched - this falls under the second pattern of the *[Instruction]* slot in Figure 5. Note that only a part of the algorithm is shown in Figure 7 to give a general idea how the ActionGenerator works.

In this flowchart, as no instrument is matched from the input string, it first checks if the matched action requires any instrument by making a query to the ontology. This is achieved by the *action = ontology(Action)* command querying ActionOntology with the matched string of *[Action]* and retrieving the corresponding individual along with its properties and the *hasAnimation* parent. If the retrieved individual has the *hasInstrument* property, the execution follows the *yes* path and generates an action sequence of: a virtual character grips the ingredient(s) and puts them into the ontological container - note that, in our cooking ontology, the *hasInstrument* property is always accompanied by the *hasContainer* property. Then the virtual character grips the ontological instrument and walks back to the ontological container to perform the animation action that is specified by the *hasAnimation* parent. Finally, the virtual character puts the instrument to the container. When the retrieved individual has no *hasInstrument* property, the execution follows the *no* path and the *hasContainer* property is checked for the further process.

## V. Virtual Director and Execution

The output of AdaptationEngine is a scenario with a set of dialogs in ScenarioScript, which is converted from the recipe

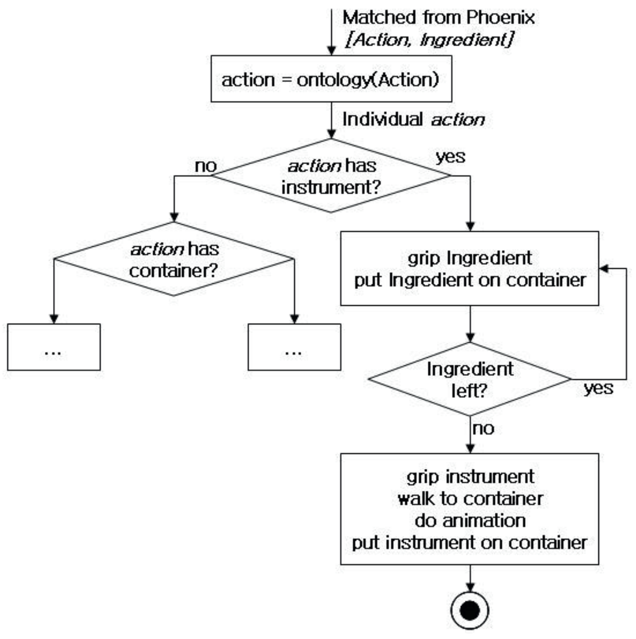


Figure 7. Flowchart for Action Generating Algorithm

in RecipeScript. In order to produce 3D animation from this scenario, actions in dialogs must be mapped to character animations and a virtual camera must be properly positioned. In our previous work, we developed VirtualDirector that automatically produces 3D animation in SceneScript format from the user stories in ScenarioScript format [6] by performing followings:

- Converting texts in ScenarioScript into the characters' speeches
- Converting emotions in ScenarioScript into the characters' facial expressions
- Converting actions in ScenarioScript into the characters' animation actions

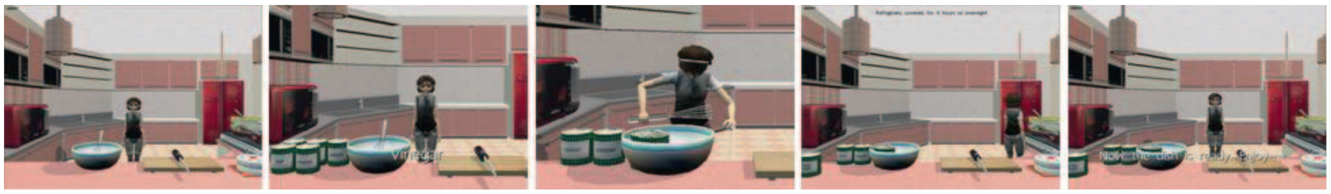


Figure 8. Snapshots for Web2Animation Excution

- Adding a virtual camera according to the characters' positions and actions
- Adding closed captions for the characters' speeches

Figure 8 shows the snapshots of the output animation that is generated from the recipe for *Green Been Salad*. The first snapshot is the opening of the animation showing the virtual cook. The cooking animation begins with the introduction of ingredients as shown in the second snapshot. While the names of ingredients are introduced by computer-synthesized voice, 3D cans labeled with their ingredient appear on the cooking table. The third and forth snapshots show the character animations; in the third one, the character grips a whisk and mixes ingredients in the bowl and, in the fourth one, the character walks to the refrigerator as the recipe instruction says to refrigerate. The last snapshot shows the character saying the ending comment as the cooking has finished. In our animation, the characters' speeches are synthesized with computer-generated voices.

## VI. Conclusion

In this paper, we introduced an approach for animating text on the web. As the vocabularies on the web are almost unlimited, we narrowed our focus into the domain of recipe so that we could design a feasible semantic analyzer and character animations.

In brief, there are two reasons why our approach is worth of exploring. Firstly, we introduced the Web2Animation system that provides a complete pipeline for generating 3D animation from web text in natural language. Our Web2Animation system analyzes the semantics of recipes on the web and generates animations for them. We designed a set of XML schemas to store different representations of recipes as they are processed and transformed from natural language to computer animation. We also designed an ontology for cooking actions to classify them and to infer the hidden context of instructions. Finally, we developed a set of action generation algorithms that utilizes the semantic analyzer and ontology for generating a series of animation actions out of the recipe instructions.

The second contribution of our research is on demonstrating a new approach to generating and enjoying media on the web - described in natural language and enjoyed in a visual form of 3D animation. This approach brings several benefits to users; while traditional visual contents are created by a limited number of people using specialized tools, with our approach, users who can write texts can also create and enjoy visual contents by themselves. We believe that this approach to easy generation of visual contents on the web would be one of the main feature for the next generation web.

## Acknowledgment

The authors would like to thank Youngmok You and Seungeun Song for their great assistance in developing our system.

## References

[1] J. Cassell. Beat: The behavior expression animation toolkit. In E. Fiume, editor, *SIGGRAPH 2001, Computer Graphics Proceedings*, pages 477–486. ACM Press / ACM SIGGRAPH, 2001.

[2] B. Coyne and R. Sproat. Wordseye: An automatic text-to-scene conversion system. In E. Fiume, editor, *SIGGRAPH 2001, Computer Graphics Proceedings*, pages 487–496. ACM Press / ACM SIGGRAPH, 2001.

[3] J. H. Haag and H. R. Cole. *The Complete Guide to Standard Script Formats: The Screenplay*. CMC Publishing, Hollywood, 1980.

[4] P. Piwek, H. Hernault, H. Prendinger, and M. Ishizuka. T2d: Generating dialogues between virtual agents automatically from text. In *Proceedings of International Conference on Intelligent Virtual Agents*, pages 161–174. Springer, Heidelberg, 2007.

[5] B. Quain. *Pro-sumer Power!* International Network Training Institute, 2000.

[6] H. Shim and B. G. Kang. Cameo - camera, audio and motion with emotion orchestration for immersive cinematography. In *Proceedings of international conference on Advances in Computer Entertainment Technology*, pages 115–118, 2008.

[7] K. Sumi and K. Tanaka. Transforming e-contents into a storybook world with animations and dialogues using semantic tags. In *Online Proceedings of WWW-05 Workshop on the Semantic Computing Initiative*, 2005.

[8] W. Ward. The phoenix system: Understanding spontaneous speech. In *IEEE ICASSP*, 1991.

[9] extensible 3d (x3d). http://www.web3d.org/x3d.