Using Code Generation to Build a Platform for Developing & Testing Dialogue Games

Tommy Yuan, Suresh Manandhar, & Simon Wells
Universities of {York | Edinburgh Napier}
CMNA 14 @ JURIX 2014, Krakow
Introduction

ProtOCL -

a prototype tool & workflow for describing & implementing dialogue games

(1) Describe game using industry standard tools

(2) Implement using code generation

(n) Build on generated code using API

Execution Platform -

API, Code, & Tools for using ProtOCL games

Exemplar of the process of integrating ProtOCL with a wider system
Motivation

Not always a big intersection between academic & industrial/commercial tools

But, increasing intersection of academia & business

projects (particularly larger EU), spin-outs

Legitimate to investigate applied issues

NB. Also an increasing focus on argumentation in relation to HCl & UX
Specification Methods

- Natural Language
- Formal/logical Notation
- Domain Specific Language (DSL)
- Diagrammatic
  - + various hybrids
Move Types

**Assertions:** The content of an assertion is a statement P, Q, etc. or the truth-functional compounds of statements: “Not P”, “If P then Q”, “P and Q”.

**Questions:** The question of the statement P is “Is it the case that P?”

**Challenges:** The challenge of the statement P is “Why P?”

**Withdrawals:** The withdrawal of the statement P is “no commitment P”.

**Resolution demands:** The resolution demand of the statement P is “resolve whether P”.

Dialogue Rules

**RFORM:** Participants may make one of the permitted types of move in turn.

**REPSTAT:** Mutual commitment can only be asserted when a question or challenge is responded.

**QUEST:** The question P can be answered only by P, “Not P” or “no commitment P”.

**CHALL:** “Why P?” has to be responded to by either a withdrawal of P, a statement that challenger accept, or a resolution demands of the previous commitments of the challenger which immediately imply P.

**RESOLVE:** A resolution demand can be made only in situations that the other party of the dialogue has committed in an immediate inconsistent conjunction of statements, or he withdraws or challenges an immediate consequent of previous commitments.

**RESOLUTION:** A resolution demand has to be responded by either the withdrawal of the offending conjuncts or confirmation of the disputed consequent.

**LEGALCHALL:** “Why P?” cannot be used unless P has been explicitly stated by the dialogue partner.

Commitment Rules

**Initial commitment, CR₀:** The initial commitment of each participant is null.

**Withdrawals, CR_W:** After the withdrawal of P, the statement P is not included in the move makers store.

**Statements, CR_S:** After a statement P, unless the preceding event was a challenge, P is included in the move makers store.

**Defence, CR_Ys:** After a statement P, if the preceding event was Why Q?, P and If P then Q are included in the move makers store.

**Challenges, CR_Y:** A challenge of P results in P being removed from the store of the move maker if it is there.

Termination Rules

1. The game will be ended when a participant accepts another participants view.
There have been a variety of approaches to the diagrammatic description of dialogue protocols. For example, in the Toulmin Dialogue Game (TDG) \[33\] a character of DGDL descriptions.

There is also a pattern of commitments in both players' commitment stores when it is played. This game is for purely illustrative purposes and is indicative of the features and descriptive commitment rules in multi-agent systems \[35\]. UML sequence diagrams also provide a way to diagrammatically depict dialogue protocols. For example, Agent UML \[36\] extends the unified modelling language (UML) to model intelligent software agents and related agent-based systems. FIPA adopted this approach (AUML) to specify agent communication protocols such as the Subscribe Interaction Protocol, which enables an agent to subscribe to messages from another agent with which it has entered into a dialogue. Finite State Machines (FSMs) have long been used to define network protocols and have been widely used to model, analyse and simulate protocols. In this example game a turn structure, two named players, and a commitment store for each player are defined. A single turn is assumed to be a move that is either a statement or a withdrawal, and a dialogue is ended when a participant accepts another participant's view.

<table>
<thead>
<tr>
<th>Pre-Conditions - Commitment Store Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>( C \in CS_n )</td>
</tr>
<tr>
<td>( C \notin CS_n )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Post-Conditions - Alterations to Commitment Stores</th>
</tr>
</thead>
<tbody>
<tr>
<td>( CS_{n+1} = CS_n \cup {C} ) Commitment ( C ) is added to commitment store ( CS )</td>
</tr>
<tr>
<td>( CS_{n+1} = CS_n \setminus {C} ) Commitment ( C ) is removed from commitment store ( CS )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Move Specifications (utilising pre- &amp; post-conditions)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Statement</strong>(( S_x ))  Pre: ( \emptyset )</td>
</tr>
<tr>
<td>Post: ( CP_{n+1} = CP_n \cup {S_x} \wedge CO_{n+1} = CO_n \cup {S_x} )</td>
</tr>
<tr>
<td><strong>Withdrawal</strong>(( S_x )) Pre: ( \emptyset )</td>
</tr>
<tr>
<td>Post: ( CP_{n+1} = CP_n \setminus {S_x} )</td>
</tr>
</tbody>
</table>
Simple{
    {turns,magnitude:single,ordering:strict}
    {players,min:2,max:2}
    {player,id:Player1}
    {player,id:Player2}
    {store,id:CStore,owner:Player1}
    {store,id:CStore,owner:Player2}
    {Assert,{p},"I assert that",
        {store(add, {p}, CStore, Speaker),store(add, {p}, CStore, Listener)}
    }
}
**claim** (C)
Description: P asserts that C
Preconditions: P has control of the dialogue
Postconditions: O has control of the dialogue
Completion Conditions: C is popped from the claim stack

**why** (C)
Description: O seeks data supporting C
Preconditions: O has control of the dialogue
Postconditions: P has control of the dialogue
Completion Conditions: C is not top of claim stack
1. Describe a generic dialogue game UML object model

2. Describe specific rules for updating that model in OCL

Use standard UML tools to produce the OCL description

Compile against object model

Auto-generates a dialogue game framework with Java API
Overview: ProtOCL Lite

OCL Annotation

UML Class Diagram

annotates

code generation

Game Engine API

use

Argumentation Software Tools
Overview: ProtOCL

- DGDL Description
- Argument.DTD
- AIF2 ???

OCL Annotation

- UML Class Diagram

Game Engine

- API

Game Transcript

- Argumentation Software Tools

This diagram illustrates the process of ProtOCL, showing how various components interact to generate code and transcripts.

- DGDL Description generates OCL Annotation
- Argument.DTD generates OCL Annotation
- AIF2 ??? generates OCL Annotation

OCL Annotation annotates UML Class Diagram

Code generation leads to:
- Game Engine
- API

API uses Argumentation Software Tools

Game Transcript exports from Game Engine.
Object Model

Fig. 2. A generic model for dialogue games expressed using the UML class diagram notation. This model captures the general core elements of dialogue games and provides the basis for an API for generated code.
Move Types

Assertions: The content of an assertion is a statement $P$, $Q$, etc. or the truth-functional compounds of statements: “Not $P$”, “If $P$ then $Q$”, “$P$ and $Q$”.

Questions: The question of the statement $P$ is “Is it the case that $P$?”

Challenges: The challenge of the statement $P$ is “Why $P$?”

Withdrawals: The withdrawal of the statement $P$ is “no commitment $P$”.

Resolution demands: The resolution demand of the statement $P$ is “resolve whether $P$”.

--Player makes a legal move
context Player::makeMove():Move
   --Permitted move types:
   post: Set{’Assertion’, ’Question’, ’Challenge’, ’Resolve’, ’Withdrawal’} ->includes(result.getType())
Execution Platform

- Rules+Agents+Knowledge = Platform
- ProtOCL generated rules
- Java Agents (extend abstract agent classes from the platform) - should be an agent framework (e.g. JADE)
- XML Knowledge-Bases (KBManager Graphical Tool)
Dialogue History

01: agent2> Is it the case that CP is acceptable?
02: agent1> Yes, I think CP is acceptable.
03: agent2> I think CP is not acceptable.
04: agent1> Is it the case that mistakes rarely happen during judicial process?
05: agent2> Yes, I think mistakes rarely happen during judicial process.
06: agent1> I think 'it is wrong to take a human life' implies 'CP is not acceptable'.
07: agent2> Is it the case that 'execution of murderers is fair for the people being murdered' implies 'murderers should receive capital punishment'.
08: agent2> Yes, I think 'execution of murderers is fair for the people being murdered' implies 'murderers should receive capital punishment'.
09: agent2> Is it the case that scientific techniques will increase the success of justice?
10: agent1> No, I think it is not the case scientific techniques will increase the success of justice.
11: agent2> I think murderers should receive capital punishment.
12: agent1> I don't think murderers should receive capital punishment.

Welcome to the game, please choose 'New game' item from the top menu bar!
Benefits

Flexible, Expressive, & Comprehensive:

- Dialogue Game API
- Object Model

Common/Popular Rules

Increased testability of game rules

Reduced likelihood of implementation errors (code gen)
Conclusions/Discussion

- Approaches to specification - many too distant from user(dev) experience
- Identified existing, well supported tools within industry/commercial software dev
- Developed preliminary workflow for bringing together those software tools with concepts from argumentation domain.