MSG VWML Integration (draft version)

Table of Contents

Overview

This is the preliminary version of integration document which describes base command based interface between virtual and real worlds.

Configuration

Configuraton integration module should be implemented as ‘fringe’ which allows to load configuration of virtual world in run-time.

Quartal

Quartal’s configuration in VWML is defined in ew.wvml file:

```
QuartalsConf ias (
        /* initial quartals (closed) per player */
        QuartalsPerPlayer ias 1;
        /* selected per quartal in random way */
        SherifBribeRanges ias (100 200 300 100 200 400 500 100 200 100 100 50 10 20 40);
        /* available resource types and their initial value */
        ResourceTypes ias ((vodka 10000) (gold 20000) (food 300000));
        /* available quartal's resources */
        /*
                index:
                        0 => type
                        1 => quantum (payment per period)
                        2 => period
                        3 => number of payment's periods
                        4 => quartal's open time
                        5 => minimal resource's quantum when sherif agrees to open information about quartal

         */
        Resources ias ((vodka 100 500 12 3000 100) (gold 10 500 10 4000 70) (food 50 500 9 5000 50));
);
```

The fringe must implement method ‘loadQuartalsConf’ which creates configuration entity (EWEntity) in following format (will be discussed):

((QuartalPerPlayer N) (SherifBribeRanges (<list of numbers>)) (ResourceTypes ((<pair_resource_name_value>)…())) (<random quartal conf>))

The format and description of <random quartal conf> see definition of entity Resources.

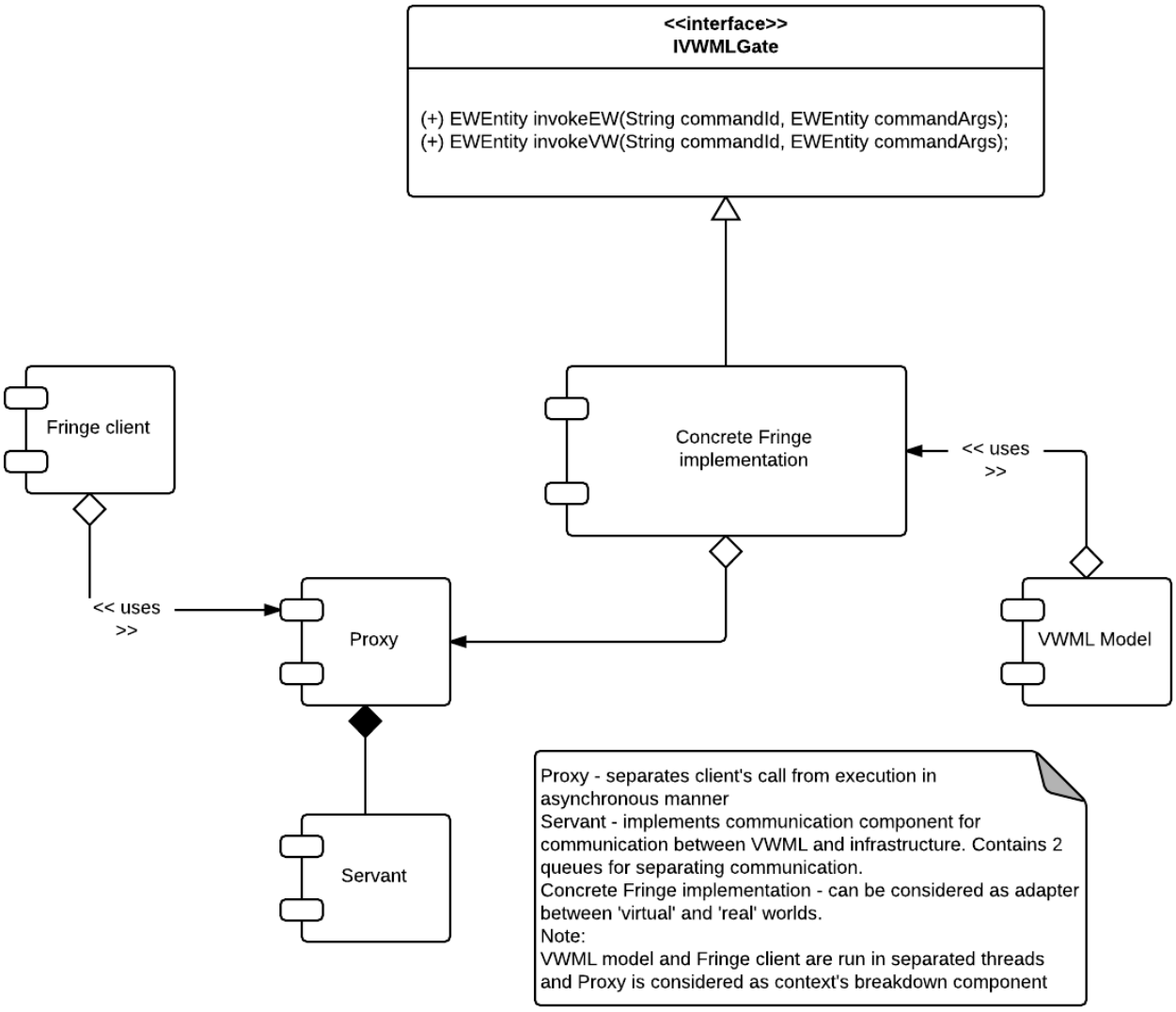
Battle graph configuration

The battle graph is defined as list of adjacent (to be defined). As example see graph/battleField.graph

Unit configuration

The fringe must implement method ‘loadUnitsConf’ which creates configuration entity (EWEntity). The format will be discussed when document which covers unit’s properties will be ready. At this time unit’s property are passed during creation one and described in ‘hire’ units command

General fringe design



Communication protocol

Players, battle, attack and units management

Adding player to game (aka login)

The login process is implemented by infrastructure part of project and in case if it was successful the command is sent to VWML model, no 'fail' login command is sent.

The command has following format

(pm 0 playermanagment playeradd <player id> nil)

Example:

Adding new player with id 0 => (pm 0 playermanagment playeradd 0 nil)

The response is sent back to 'real' world must be specified. The proposed format is

((playermanagment playeradd ok) (Player <player id>) ()) where <player id> is Id of player which is passed during 'playeradd' command

Remove player from game and stop its activity

This command is by infrastructure part when player decides to leave game and stop its world's activity. The command has following format

(pm 0 playermanagment playerstop <player id> ())

Example:

Removing player with id 0 and closing its session (pm 0 playermanagment playerstop 0 ())

The response is sent back to 'real' world must be specified. The proposed format is

((playermanagment playerstop ok)(Player <player id>)()) where <player id> is Id of player which is passed during 'playeradd' command.

Stop the whole world (all players are stopped)

This command is sent by infrastructure part when administrator decides to stop server. Pay attention that all users activities are stopped.

Send command for each logged in player

(pm 0 playermanagment playerstop <player id> ())

When all players have stopped, following commands must be sent:

(gbanker 0 any exit none ()) => stopps 'global banker' which is responsible for resource transfer among players

(manager 0 nil exit 0 ()) => stops game's manager

No specific response is defined, the infrastructure part detects that world stopped by exiting game logic loop.

Player hires units

This command is sent when player wishes to hire units for either attack or defence

(player <player id> hiring starthiring 0 ((<hiring properties>) (<unit's defence and attack properties>)))

(<hiring properties>): ((resource and cost) (hiringtime <ms>) (id <unit_id>))

(<resource and cost>): (cost <resource_type> <resource_cost>)

<ms>: hiring time in ms

<unit_id>: (player_id <unit_id>)

<resource_type>: coincides with quartal's resource configuration

(<unit's defence and attack properties>): (<kind of unit>) (attack (<attack_props>)) (defence (<defence_prop)))

<kind of unit>: skeleton | zombac | etc

(<attack_props>): combination of attacking properties (must be specified in unit's training document)

(<defence_props>): combination of defence properties (must be specified in unit's training document)

Example:

(player 0 hiring starthiring 0 (((cost vodka 200) (hiringtime 7000) (id (0 2))) ((kind skeleton) (attack (yes yes yes)) (defence (no no yes)))))

Here attack and defence properties defined by combination of yes | no and described in document of battle offered by Michael Groozman

The response is sent back to 'real' world must be specified.

The proposed format for 'start hire operation' is:

((hiring starthiring inProgress) (Player <player_id) (unit <unit_id>) (<requestId>))

<request_id>: current request id. Using this id player can cancel hiring operation

The proposed format for 'negative' response (case if user doesn't have enough resources):

((hiring starthiring failed noResources) (Player <player_id>) (unit <unit_id>))

The proposed format for 'positive' response is:

((hiring starthiring done) (Player <player_id) (unit <unit_id>) (<requestId>))

Player cancels hiring process for specific unit

This command is sent when player wishes to cancel hire operation.

(player <player_id> hiring cancelhiring <request_id> ())

<player_id>: player's id

<request_id>: request id which player received as immediate response on 'starthiring' operation.

See ((hiring starthiring inProgress) (Player <player_id) (unit <unit_id>) (<requestId>))

The response is sent back to 'real' world must be specified.

The proposed format for 'positive' 'cancelhiring' is:

((hiring cancelhiring done) (player <player_id>) (unit <unit_id>) (<request_id>))

The proposed format for 'negative' 'cancelhiring' is:

((hiring cancelhiring failed invalidRequestId) (player <player_id>) (unit <unit_id>) (<request_id>))

Player wishes to build defence

In order to build defence player has to change player's mode to defence. In order to do it following command is sent:

(player <player_id> battle setmode builddefence ())

The response is sent back to 'real' world must be specified.

The proposed format is:

((battle setmode ok builddefence) (player <player_id>))

Player wishes to recruit units (defenders)

When player in 'defence' mode it can recruit units. In order to do it player sends following command:

(player 0 battle recruitunit <unit_id> ())

Pay attention that unit identified by <unit_id> must be hired before (see 'starthiring' command)

The response is sent back to 'real' world must be specified.

In case of 'negative' response proposed format is:

((battle recruitunit failed notHiredDefence) (player <player_id>) (unit <unit_id>))

In case of 'positive' response proposed format is:

((battle recruitunit ok defence) (player <player_id>) (unit <unit_id>))

Player finishes defence building (battlefield's settings are confirmed)

When player decides that defence has already built it has to send following command, otherwise 'battlefield' will not be considered as ready for attack (isn't visible to other players)

(player <player_id> battle apply 0 ())

The response is sent back to 'real' world must be specified.

The proposed response is:

((battle apply ok ()) (player <player_id>))

In case of 'failed':

((battle apply failed ()) (player <player_id>))

Player resets defence (battlefield's settings are reset, defence mode)

When player decides to reset defence's configuration (units' disposition) it has to send following command

(player <player_id> battle reset 0 ())

All units are returned to player's resource pool.

The response is sent back to 'real' world must be specified.

The proposed response is:

((battle reset ok ()) (player <player_id>))

Player dismisses recruited unit from battlefield (defence mode)

When player decides to remove unit from battlefield it has to send following command (opposite to recruitunit command):

(player <player_id> battle dismissunit <unit_id> ())

The response is sent back to 'real' world must be specified.

The proposed response is:

((battle dismissunit ok defence) (player <player_id>) (unit <unit_id>))

Player recruits units for attack, attacking group (attack mode)

In case if player wants to attack another player (selected from list – produced by match-making) it has to send following command:

(player <player_id> attack recruiteunit <unit_id> ())

The response is sent back to 'real' world must be specified.

The proposed response is:

In case of 'negative' response proposed format is:

((attack recruitunit failed notHiredAttack) (player <player_id>) (unit <unit_id>))

In case of 'positive' response proposed format is:

((attack recruitunit ok ()) (player <player_id>) (unit <unit_id>))

Player dismisses recruited unit from attacking group (attack mode)

When player decides to remove unit from battlefield it has to send following command (opposite to recruitunit command):

(player <player_id> attack dismissunit <unit_id> ())

The response is sent back to 'real' world must be specified.

The proposed response is:

((attack dismissunit ok ()) (player <player_id>) (unit <unit_id>))

Player resets attacking group (battlefield's settings are reset, attack mode)

When player decides to reset attacking group it has to send following command

(player <player_id> attack reset 0 ())

All units are returned to player's resource pool.

The response is sent back to 'real' world must be specified.

The proposed response is:

((attack reset ok ()) (player <player_id>))

Player attacks another player

When player wishes to attack another player (the opponent is selected with help of match-making algorithm) it has to send following command:

(player <player_id_attacker> attack prepareinvasion 0 (<player_id_attacked>))

The response is sent back to 'real' world must be specified.

The proposed responses are:

Response in case if attacker wasn't built attacking group:

((attack prepareinvasion failed noAttackingGroup) (player <player_attacker_id>) (player <player_attacked_id>))

Response in case of attacking himself

((attack prepareinvasion failed attackHimself) (player <player_id>))

Response in case if attacking player did not build defence

((attack prepareinvasion failed invalidAttackedDefence) (player <player_attacker_id>) (player <player_attacked_id>))

Response in case if attacker successfully started attack

((attack prepareinvasion ok ()) (player <player_attacker_id>) (player <player_attacked_id>))

Pay attention that attack phase has more than one intermediate commands which are sent among components - and one of them is 'setupattackers' that should be considered separately:

The command 'setupattackers' is sent by attacker, just after command 'attack' (last command in prepareinvasion's commands' chain) is sent, and contains list of attacking units (called invasion group/attacking group) which were recruited before. I guess that 'real' world should get notification from 'virtual' world about 'readiness' of invasion group on opponent's battlefield – so this process should be animated somehow.

Proposed response:

((attack setupattackers ok) <list of attacking units> (player <attacked player id>))

Capitulation of attacking player

If during attack, attacker, sees that attack is going to be failed it may initiate process of capitulation in order to save remained units:

(player <player_id> battle {surrender | fullback} 0 ())

Pay attention on that this command is sent to attacked player (who owns battlefield). In this case all remained units are returned to players' resource pools, but attacker lost battle. The process of rewards will be described

in separated document. For now battle's cost (resource) is passed to attacking player and settles on it bank's account.

The command is sent to attacked player due to game scenario where attacker may capitulate only.

The response is sent back to 'real' world must be specified.

The proposed response is:

((battle surrender ok) (player <player_attacker_id>) (player <player_attacked_id>))

Since this command include more than one intermediate commands following response commands must be processed:

Units are returned to player's pool (will be specified later)

Units are released (killed) (will be specified later)

Update players' bank accounts (will be specified later)

Battle status

The proposed response is:

((battle attack ok finished) (player <winner_player_id>) (player <lost_player_id>))

Unit was killed during the battle

Since battle is implemented on 'real' world, 'virtual' world has to know everything what happened during the battle. In case if unit was killed - the 'real' model must notify 'virtual' world by sending command:

(player <player_id> battle unitkilled <unit_id> ())

No specific response on this command. The battle status response can be sent in case if battle's status was changed (as example when last attacker or defender was killed):

The proposed response is:

((battle attack ok finished) (player <winner_player_id>) (player <lost_player_id>))

Unit management commands (on resource manager)

All operations on units' pool are implicit and sent by other 'virtual' components during a flow. For example unit hired, returned to pool, taken from pool or released at all (aka killed) – but in any case 'real' world must be notified in order to be synchronized with 'virtual' world.

Hiring agency sends command to resource manager about hired unit

The response is sent back to ‘real’ world must be specified.

The proposed response is:

((resourcemanager hireunit done) (player <player_id>) (unit <unit_id>))

Unit taken from resource manager (aka recruited)

The response is sent back to ‘real’ world must be specified.

The proposed response is:

((resourcemanager recruitunit done inAction) (player <player_id>) (unit <unit_id>))

Unit returned to resource manager (aka dismissed or returned from battle, etc)

The response is sent back to ‘real’ world must be specified.

The proposed response is:

((resourcemanager returnunit done free) (player <player_id>) (unit <unit_id>))

Unit released from resource manager (aka killed)

The response is sent back to ‘real’ world must be specified.

The proposed response is:

((resourcemanager releaseunit done released) (player <player_id>) (unit <unit_id>))

Economic and deal with sheriff and quartals

Number of qurtals are configured per player. When player's world is created the configured number of quartals are created and player is linked to quartals in automatic manner.

Linking quartal to player's economic gives ability to make deal with sheriff

(player <player_id> deal linkquartal 0 (<quartal_id>))

<quartal_id>: (<player_id> <simple_quartal_id>)

<player_id>: owner of quartal

<simple_quartal_id>: number or string which must be uniq in player's scope

The response is sent back to 'real' world must be specified.

The proposed response is:

((deal linkquartal done) (player <player_id>) (quartal <quartal_id>))

Unlinking from quartal

(player <player_id> deal unlinkquartal 0 (<quartal_id>))

<quartal_id>: (<player_id> <simple_quartal_id>)

<player_id>: owner of quartal

<simple_quartal_id>: number or string which must be uniq in player's scope

The response is sent back to 'real' world must be specified.

The proposed response is:

((deal unlinkquartal done) (player <player_id>) (quartal <quartal_id>))

Open quartal's information

This command is sent to player's linked quartal:

(player <player_id> deal setintention 0 (<quartal_id> open (<cost>)))

<cost> : quartal's payable resource. The sheriff asks for bribe in order to open information about quartal

The response is sent back to 'real' world must be specified.

The proposed response are:

Sheriff doesn't wants more resources (wishes more than proposed)

((deal setintention failed ((tooSmall open) (<proposed_price>) (<desired_price>))) (player <player_id>) (quartal <quartal_id>))

Player doesn't have resources to pay for quartal's information

((deal setintention failed (open notEnoughResources)) (player <player_id>) (quartal <quartal_id>))

Sheriff opens information about quartal

((deal setintention done (open (<quartal_info> <period_ms>))) (player <player_id>) (quartal <quartal_id>))

<quartal_info>: opened information about quartal (visible for player). Format will be specified during integration phase

<period_ms>: quartal's information is being opened during this period.

<period_ms> and <quartal_info> are configurable entities and loaded during configuration reading phase

After the 'positive' response sheriff opens information about quratal. The information is being opened during configurable time, which in turn is part of quartal's configuration.

Player gives bribe to sheriff

Quartal's information must be opened before

(player <player_id> deal setintention 0 (<quartal_id> pay (<cost>)))

<cost> : quartal's payable resource. The sheriff asks for bribe in order to open information about quartal

The response is sent back to 'real' world must be specified.

The proposed response are:

Sheriff thinks that bribe to small

((deal setintention failed ((pay tooSmall) (<proposed_bribe>) (<desired_bribe>))) (player <player_id>) (quartal <quartal_id>))

Player doesn't have resources to pay for bribe

((deal setintention failed (notEnoughResources pay)) (player <player_id>) (quartal <quartal_id>))

Sheriff takes bribe and starts contribution process

((deal setintention inProgress (pay (<quartal_info>))) (player <player_id>) (quartal <quartal_id>))

After the 'positive' response sheriff starts immediately contribution process. Contributing period and number resource's quantums are configurable also. During the contribution process player's bank account is being updated.

Sheriff closes information about quartal (time expired)

The response is sent back to ‘real’ world must be specified.

The proposed response are:

((deal setintention done (closed open)) (player <player_id>) (quartal <quartal_id>))

Sheriff finishes to pay contribution

The response is sent back to ‘real’ world must be specified.

The proposed response are:

((deal setintention done (finished pay)) (player <player_id>) (quartal <quartal_id>))

Player’s bank account commands

All player’s bank account commands are explicit commands which are being sent during various flows – when player hires unit, open quartal’s information or pays bribe and receives contribution from Sheriff

The response is sent back to ‘real’ world must be specified.

The proposed response are:

((banking withdraw done)(player <player_id>)(<resource>))

((banking recharge done)(player <player_id>)(<resource>))

<resource>: (<resource_type> <resource_quantity> <balance>)

Pay attention that <resource> is configurable and is part of quartal’s configuration

Updating player's account in runtime

This command allows to change player's account configuration in runtime and at this time is going to be used during game balance phase

(player <player_id> main updateaccount 0 <accountconf>)

<accountconf>: (<resource>..<resource>)

<resource>: (<type> <quantity>)

Example:

(player 0 main updateaccount 0 ((vodka 10000) (gold 20000) (food 300000)))

The proposed response is:

((main updateaccount ok) (player 0))

Update quartal's configuration in runtime

This command allows to player to change quartal's configuration in runtime and at this time is going to be used during game balance phase

(player <player_id> deal setintention 0 (<quartal_id> updateconf (<quartalconf> bribe)))

<quartalconf>: (<type> <quantum> <period> <number of payment periods> <quartal open time> <quartal open info price>)

The description of parts of quartal's configuration fully coincides with configuration described in configuration section

The proposed response is

((deal updateconf done) (player <player_id>) (quartal <quartal_id>))

Global player account configuration

When player is created its account should be configured. The following message should be sent in order to configure it in runtime

(pm 0 playermanagment gaccountconf 0 <account_conf>)

<account_conf>: <account configuration as defined in file msg\conf\accountconf.conf>

Response:

((playermanagment gaccountconf ok))

Global quartal's configuration

When quartal is initialized it takes initial configuration from global configuration entity. The following message should be sent in order to configure it in runtime

(pm 0 playermanagment gquartalconf 0 <quartal_conf>)

<quartal_conf>: <account configuration as defined in file msg\conf\quartalconf.conf>

Response:

((playermanagment gquartalconf ok))

Modify player's balance

The command should be sent to player in following format in order to modify player's account:

Withdraw resource (dec)

(player <player_id> banking withdraw 0 (player nil nil nil (cost <resource_type> <resource_value>)))

The response is defined in section

Recharge resource (inc)

(player <player_id> banking recharge 0 (player nil nil nil (cost gold 250)))

The response is defined in section

Examples

Player 0 withdraws and recharges resources

(player 0 banking withdraw 0 (player nil nil nil (cost vodka 150)))

(player 0 banking recharge 0 (player nil nil nil (cost gold 250)))

Dismissing unit

Unit can be dismissed (stopped and removed from resource manager) when it has been returned to resource manager. If unit was hired, firstly, it must be returned to resource manager by sending commands defined in sections, Player dismisses recruited unit from battlefield (defence mode), Player dismisses recruited unit from attacking group (attack mode) (depending on mode). Upon receiving response the following command must be sent in order to finalize dismissing procedure

(player <player_id> resourcemanager releaseunit <player_id> (<unit_id>))

The responses are:

When resource manager sent command to unit to stop it (can be ignored)

((resourcemanager unitstopped done) (player <player_id>) (unit <unit_id>))

When unit actually stopped – instance of unit removed from model

((resourcemanager releaseunit done) (player <player_id>) (unit <unit_id>))

Configuration files (examples)

All formats described in section Configuration

accountconf.conf

((vodka 10000) (gold 20000) (food 300000) (viski 50000))

quartalconf.conf

```
(
        (QuartalPerPlayer 1)
        (SherifBribeRanges (300 200 300 100 200 400 500 100 200 100 100 50 10 20 40))
        (ResourceTypes ((vodka 50000) (gold 20000) (food 300000)))
        (Resources ((vodka 100 500 12 3000 100) (gold 10 500 10 4000 70) (food 50 500 9 5000 50)))
)
```