Karmafy for Video Games

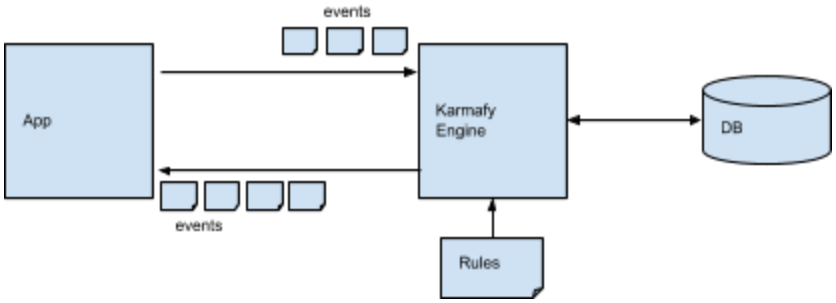
Architecture and Integration

Introduction

The Karmafy platform translates in-app user action to rewards, keeps track of the user's history and inventory of points, achievements, etc., and provides feedback to the user when rewards have been earned or converted into social impact.

At the core of the system are the rules engine and the database. The rules engine processes incoming "Karmafy Events" from the app, modifies the database accordingly, and provides messages to the user. The rules applied for each app are highly configurable.

This document will explain the most important data stored in the database, the data flow between the game and the Karmafy server, and the design of rules. Technically inclined readers will find this useful when integrating Karmafy with their app, and UX specialists will learn how expressive the rule system is.



Furthermore, we introduce the Karmafy SDKs available for certain mobile platforms. Using an SDK will greatly facilitate integration.

Karmafy Events

A Karmafy Event represents the fact that something has happened, or a request for something to happen. In general, events are created by the app, submitted to the Karmafy Server, and processed by the server. Processing one event may generate further events that are fed back into the event queue and processed. Finally, the app receives one or more events representing a history of how the original event was processed.

For example, consider a game where the objective is to kill monsters. The game has the following rules (summarised here, full listing in appendix):

- “monsterKilled” event earns you 10 KP for a dragon, 5 KP for a hamster. If your dragon kill count reaches 5, you get the “dragonSlayer” achievement.
- Reaching 100 KP earns the “reached100” achievement.
- A message is displayed in a popup on earning an achievement.

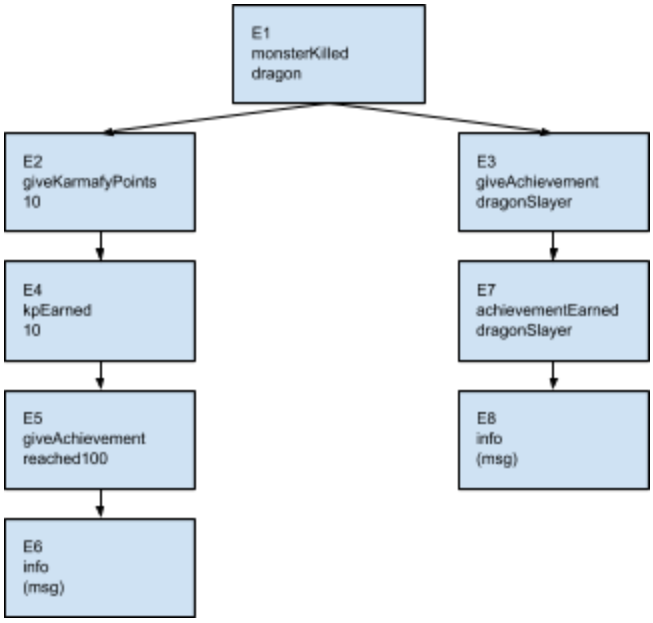
When a dragon has been defeated, the game submits this event to the server:

```
{
    kind: “monsterKilled”,
    game: “04a2cb7557fc48d2ba75336a9a84ed28”,
    user: “7dbed6af708c442fa5a14a0acc274407”,
    monsterType: “dragon”
}
```

When the event reaches the server, it is matched against the rules above to find a handlerfor the event kind “monsterKilled” within the game “04a2cb7557fc48d2ba75336a9a84ed28”.

- Killing a dragon gives 10 KP by rule 1. The server creates the event (let’s call it E2):
  ```
  {
      kind: “giveKarmaPoints”,
      game: GID,
      user: UID,
      kps: 10
  }.
  ```
- Assuming this brings the dragon kill to five, the server also generates E3:
  ```
  {
      kind: “giveAchievement”,
      achtpId: “dragonSlayer”,
      game: GID,
      user: UID
  }.
  ```

In total, these events are generated and processed:



Events are processed in depth-first order, so in the example, all events relating to the rewarding of 10 KP and the following achievement are processed prior to rewarding the “Dragon Slayer” achievement. Processing stops when no matching rules are found.

The app that originally generated the “monsterKilled” event receives the entire sequence E2, E4, E5, E6, E3, E7, E8. The app may choose to incorporate data from the events into its own UI (such as displaying the new total of KP, or showing badges for achievements).

Events of kind “info” are special in that they contain a message intended for display to the user (e.g., “You just got an achievement for reaching 100 KP.”). The messages themselves may contain calls to action, which in turn may generate further events. For example, the “reached100” message may invite the user to choose a cause to support.

The flow above is just an example. The range of possible rules includes:

- Different KP credits for different kinds of monsters.
- Tracking repeated behaviour, such as killing a monster each of five consecutive days.
- Further cascading of events, such as awarding KPs for earning the tenth achievement.
- Event processing depending on user identity, any stored user data (see below), user belonging to different cohorts in A/B testing, etc.
- Events involving multiple entities, such as installing the fifth game from a particular publisher, or contributing more than 100 KP to two different causes.

Furthermore, we use the event system to script other aspects of the Karmafy user experience.

- Issuing the "karmafyMenu" event from the app displays the Karmafy main menu (or rather, the "karmafyMenu" event generates a suitable "info" event with the contents of the Karmafy menu, and so on).
- The exact flows for logging in, registering as a new user, choosing causes to support, etc., are expressed as rules and can be modified without changing the app.
- General information, "message of the day", etc., can be expressed as rules as well.

Rules for a particular app are expected to be written by a Karmafy Rules author in collaboration with the app developer. It is the responsibility of the app developer to handle the events returned from the server. SDKs exist for certain platforms that assist in this, but developers are free to implement their own solution using the HTTP API only.

User Data and Rules

The Karmafy server collects and stores information about users. This overview is not a comprehensive database schema, but it provides enough information to understand how rules can be written to be dependent on user status.

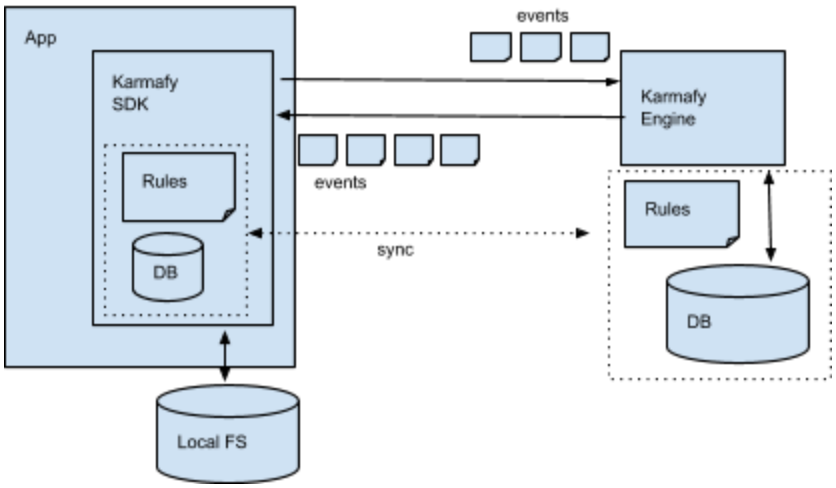
- Basic. User ID, username, image, total KP.
- Credentials. We allow several sets of credentials to be associated with a single user account. These include email/password pairs, in-app identity (as managed by app developer) and external identities from Facebook and others.
- The causes the user has chosen to support for each game.
- The number of KPs assigned to causes.
- Achievements earned for each game.
- Counts of certain kinds of events. Can be used to determine the number of times the user performed an action, or to determine whether the user performed an action for a particular number of consecutive days (weeks, months, years).
- User's friends. Only if submitted from app or if Karmafy is allowed Facebook access.
- Additional data volunteered by the app, including user's preferred language, etc.
- Assignment of user to experiment cohorts.

Using an SDK

Karmafy provides an SDK for Unity3d on iOS, Android, Windows, and OSX. We plan to develop additional SDKs for "native" iOS and Android.

The SDK offers several advantages over integrating directly with the HTTP-based API:

- Incorporating into a project takes literally minutes (see SDK integration manual).
- Encapsulates all HTTP communication.
- Manages user identity based on stored information on the device.
- Support for displaying messages contained in “info” events.
- Improved off-line behaviour through partial replication of database and rule set to device.



Method Calls Rather than HTTP

For example, to request display of the Karmafy menu in a Unity3d app:

```
KarmafyEvent ke = new KarmafyEvent();
ke.dict = new Dictionary<string, object> () {  { "kind", "karmafyMenu" } };
KarmafyEngine.instance.sendEvent(ke, null);
```

User Identity Management

The SDK, having access to local storage on the device, keeps a record of the most recent user and ensures that this user identity is used for subsequent app activation. It also supports the concept of an “anonymous” user who can enjoy the Karmafy experience without identifying themselves.

When an app starts for the first time after Karmafy integration, an anonymous user will be created. The anonymous user receives KPs and achievements along with the regular Karmafy messaging encouraging selection of a cause, etc., and may eventually choose to create a Karmafy account, either registering with email/password or logging in with Facebook. If an account already exists with the submitted credentials, it will be merged

with the anonymous account, and the user's KPs and other settings will be accessible from all devices and all Karmafy-enabled apps.

Display of Messages

The developer may choose to interpret the "info" events delivered by the rules engine, but to get started it may be simpler to use the display feature built into the SDK. In general, when the engine has something to display, it will first notify the app that a popup sequence is about to start (the app is expected to suspend what it is doing). Then, the SDK will display one or more messages and finally notify the app that the popup sequence has ended.

The app developer remains in control of when exactly messages are shown by setting the "allowed priority" of messages. Thus, in a highly interactive game screen, messages can be disallowed altogether, and when the allowed priority is reset after the "level complete" screen, the SDK proceeds to show queued-up messages generated during the game.

Off-line Processing

As the Karmafy SDK launches, we synchronise the rule set for the app and relevant parts of the database. This allows a significant part of the rules to be processed locally on the user's device with no need for an Internet connection. Events that require a persistent record to be created on the server, such as earning KPs or changing support from one cause to the other, do need a connection, though.

Required and Recommended Events

See also the *Generic Events* spreadsheet.

`karmafyMenu`. To allow the user to access the Karmafy Menu, we recommend that your user interface include a button or similar that, when activated, sends the karmafyMenu event.

To track app usage and reward repeated visits, we suggest sending **`appActivated`** when the app launches or gains focus.

In order to track revenue generated by Karmafy-engaged users, we ask that you send an event when the user makes an in-app purchase or watches advertising content. See *Karmafy Features & User Experience* (K0005).

If your app integrates with Facebook or a similar Oauth2-based service, you may choose to submit the user's identity and access token from that service. This allows us to provide an identical Karmafy experience across several devices and to leverage the user's social graph as part of the experience.

If you choose to participate in our cross-promotion program, we would like you to tell us when an install has been positively attributed to our messaging. As there are many attribution systems on the market, this will require a bit of bespoke integration work from your side. Sample code for integration with Branch.io is available on request.

Most importantly, we invite you to collaborate with Karmafy to incorporate as many app-specific events as you find relevant. This allows you to reward all sorts of user behaviour with KPs and makes for a compelling and meaningful user experience.

Appendix: Example Rules

Example rules for game `04a2cb7557fc48d2ba75336a9a84ed28`		
	Incoming event	Resulting event(s)

1	`{`
	`  kind: "monsterKilled",`
	`  user: $user,`
	`  monsterType: $mtype`
	`}`
		`if ($mtype == "dragon")`
		`  $kp = 10`
		`else if ($mtype == "hamster")`
		`  $kp = 5`
		`{`
		`  kind: "giveKarmaPoints",`
		`  user: $user,`
		`  kps: $kp`
		`}`
		
		`If (5 dragons killed in total)`
		`{`
		`  kind: "giveAchievement",`
		`  achtpId: "dragonSlayer",`
		`  user: $user`
		`}`

2	`{`
	`  kind: "giveKarmaPoints",`
	`  user: $user,`
	`  kps: $kp`
	`}`
		`Credit user $kp`
		`{`
		`  kind: "kpEarned",`
		`  user: $user,`
		`  kp: $kp,`
		`  previousBalance: …,`
		`  newBalance: ...`
		`}`

3	`{`
	`  kind: "giveAchievement",`
	`  achtpId: $achievementType,`
	`  user: $user`
	`}`
		`Reward achievement`
		`{`
		`  kind: "achievementEarned",`
		`  achtpId: $achievementType,`
		`  user: $user,`
		`}`

4	{
	Kind: “kpEarned”,
	Kp: $kp,
	User: $user,
	previousBalance: …,
	newBalance: ...
	}
		If (new balance > 100)
		{
		kind: “giveAchievement”,
		achtpId: “reached100”,
		user: $user
		}

5	{
	Kind: “achievementEarned”,
	achtpId: $achievementType,
	User: $user
	}
		{
		kind: “info”,
		msg: { … }
		}

Document History

Version	Date	Description
0.9	2017-06-08	Initial draft