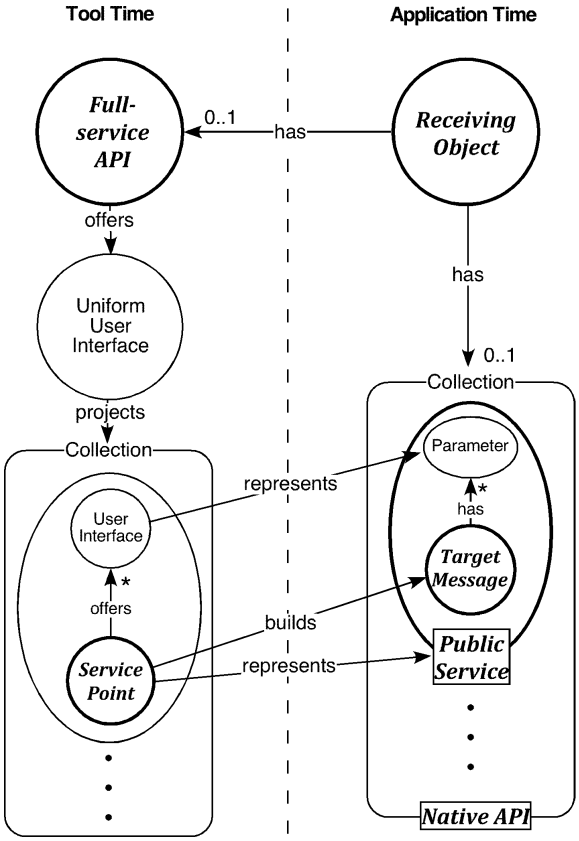
The Full-Service API

Why: The purpose of the ***full-service API*** of a ***receiving object*** is to enable a person building an application to use the ***public services*** of that object, with minimal knowledge of the object's ***native API***. It is a linking language within construction tools jointly used by programmers and non-programmers.

What: There are two execution domains that must be distinguished in the following figure. Terms in this text in ***bold italics*** appear in the figure.

- **Tool time:** A user is interacting with an application-building tool.
- **Application time:** A user is interacting with the application being built.



If the tool satisfies the conditions of the humane dozen[1], in particular the *Symmetrical* property, each user event (e.g., mouse click), depending on its location in the tool+application UI, determines which execution domain immediately follows. This execution domain is in effect only for the duration of the processing of that user event. In such a tool the receiving object is accessible in both execution domains.

[1] http://melconway.com/Home/pdf/humanedozen.pdf

Twitter: @conways_law

Date of pdf: 6/14/2018
conway.mel@gmail.com

A full-service API is a public property of every conforming object, which must provide its full-service API on demand. (The default return value is empty. However, every full-service API, even an empty one, offers its uniform user interface as described below.) In a fully supporting application-building tool, a full-service API gives you everything you need in order to use the public services of its receiving object: to understand and test these services, and to decide whether or not to use one of them *without ever leaving the user interface of the full-service API*.

All tool-time user interfaces conform to **the *Self-revealing Principle*: The user is never required to construct grammatical expressions according to some syntax; rather, the user interface presents choices for selection, which it explains to the extent that the user requests.**

A full-service API offers a uniform user interface that projects a (possibly empty) collection of ***service points***. A service point represents one public service of the receiving object and provides all needed information to the tool's user so that the application being built can activate the fully parameterized service corresponding to this service point at application time. This service activation occurs in application time when a ***target message*** is sent to the receiving object according to the receiving object's native API. If the user of the tool needs additional information about the service point, or if the target message needs to be further parameterized, this is done *within a public-service-specific user interface provided by the service point*.

How: Within a supporting application-building tool the way for a user to specify how to get a receiving object to perform one of its public services is for the tool to ask the receiving object for its full-service API, which then displays in its uniform user interface the collection of its service points. The user can choose a service point from this collection. Help information for this service point is made available and, if necessary, further specification of the service point is provided within a user interface provided by the service point; this user interface also conforms to the Self-revealing Principle. When this choice of service point is confirmed by the user[2], the tool builds into the application a fully specified target message for the receiving object's native API that can be sent to the receiving object in application time.[3]

---

[2] There must be provision for undoing this confirmation in later tool sessions.

[3] Tool dependencies can be minimized by building all tool-time UIs and processes in the Web. This suggests a standard Web-based tool for both defining and building the tool-time side of all full-service APIs.

Twitter: @conways_law

Date of pdf: 6/14/2018
conway.mel@gmail.com