PILS: A General Plugin and Interface Loading System

PILS: A General Plugin and Interface Loading System

A component of the Open Clustering Framework Reference Implementation

Alan Robertson

IBM Linux Technology Center

alanr@unix.sh

Agenda

- What is the OCF Reference Implemention?
- Why plugins?
- Goals and features of PILS
- Why not other plugin software?
- Sample PILS usage
- Future Enhancements

Purpose: To give an overview of PILS for developers and system architects.

Terminology

- **OCF: Open Cluster Framework – a set of standard clustering API being developed**
- **interface: a unique set of imported and exported functions**
- **implementation: a set of functions (in a plugin) which provide a particular interface**

Note: A single plugin may implement more than one interface

What is the OCF Reference Implemenation?

- ♦ The OCF reference implementation is a general framework for implementing cluster management systems based on the OCF APIs.
- ♦ It is very general and open-ended.
- ♦ It is oriented to making every major function replaceable and configurable at run time.
- ♦ The goal is to fork components, not the framework.

Why Plugins?

- Plugins allow great flexibility and help in creating a powerful system.
- Plugins allow easy updates and new capabilities to be added to running systems.
- Plugins encourage simpler system architecture – vital for OSS projects.
- Plugins are ideal for an open–ended system with open community participation.

Goals of PILS

- Be portable to other Operating Systems (OSes)
- Be immediately usable (as a shared library) by any project
- Encourage reuse of plugins
- Support many kinds of plugins simultaneously
- Provide information on which plugins of a given type are available
- Allow a given shared object to provide several interfaces

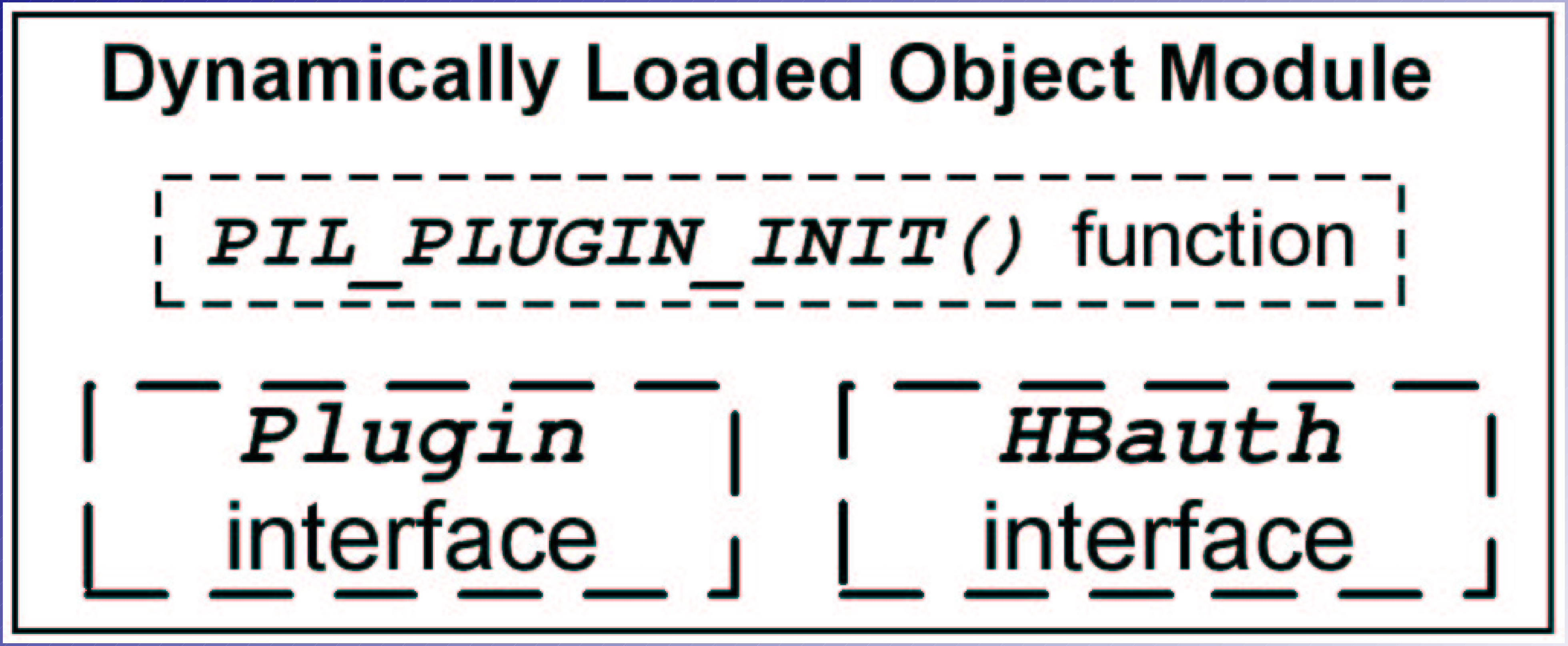
Features of PILS

- Distinguishes plugins (.so files) from interfaces (sets of functions).
- Each interface exports a set of functions, and imports a set of functions.
- *In addition*, each plugin imports a standard set of functions, and exports a standard set of functions.
- Plugin loading is by interface type/name.
- Plugin unloading by reference count.
- Built on top of libtool for maximum portability

Why invent a new system?

- Usable by any application as a library
- Provide imports to plugins for reusability and portability
- Named (not #defined) plugin types
- Highly portable system

Components of a Plugin



Sample Plugin Usage

Goal: Load "md5" authentication ("HBauth") plugin

```
PILPluginUniv* PluginSys = NULL;
GhashTable*    AuthFuncs = NULL;

PILGenericIfMgmtRqst Requests[] =
{"Hbauth" &AuthFuncs, NULL, NULL, NULL},
{NULL,    NULL,       NULL, NULL, NULL}};

/* Create Plugin Universe and load plugin
 * manager.
 */
PluginSystem = NewPIPluginUniv("/usr/lib/foo");

PILLoadPlugin(PluginSys,"InterfaceMgr",
,  "generic", &Requests);
```

Sample Plugin Usage (continued)

```
struct hb_auth_ops* Auth;
char                result[64];

/* Load and use md5 plugin */
PILLoadPlugin(PluginSys,"hbauth","md5",NULL);

Auth = g_hash_table_lookup(AuthFuncs, "md5");

Auth->auth(&authinfo, "SignMe", result
,       sizeof(result));

/* Unload plugin */
PILIncrIFRefCount(PluginSys,"HBauth", "md5",-1);
Auth = NULL;
```

Sample Plugin

```
#define PIL_PLUGINTYPE Hbauth
#define PIL_PLUGIN      md5
#define PIL_PLUGIN_S    "md5"
static int md5_auth_calc(...);
static int md5_auth_needskey(void);
static struct HBAuthOps md5ops =
{md5_auth_calc, md5_auth_needskey};

/* Called before unloading */
static void md5closepi(PILPlugin* pi) { }

/* Called down to shut down the interface */
static PIL_rc md5closei(PILInterface* i, void*pp)
{ return PIL_OK; }

/* Standard boilerplate stuff */
PIL_PLUGIN_BOILERPLATE("1.0", Debug, md5closepi);
```

Sample Plugin (continued)

```
static const PILPluginImports* PiImports;
static PILPlugin*                OurPI;
static PILInterface*             OurIntf;
static void *           IntImports, intprivate;

/* Plugin Initialization function */
PIL_rc
PIL_PLUGIN_INIT(PILPlugin* us
   ,  const PILPluginImports* imp) {
   PiImports = imp;
   OurPI = us;

   /* Register us as a plugin */
   imp->register_plugin(us,&OurPIExports);

   /* Register our md5 authentication interface */
   imp->register_interface(us,"Hbauth","md5",&md5ops
   , md5closei, &OurIntf, &IntImports,   &intprivate);
}
```

Ideas for the Future

- Interface aliases
- PATH–like plugin searching
- Security awareness and checking
- Cryptographically signed plugins
- Interface version management
- Independent project (if sufficient interest)

References

- http://linux–ha.org/download/
- http://linux–ha.org/
- http://opencf.org/

Alan Robertson *alanr@unix.sh*
http://linux-ha.org/

Legal Statements