Open source software RefPerSys

(logo donated by Gaëtan Tampon)

The free software project RefPerSys (REFlexive PERsistent SYStem) is an open source C++ software project under GPLv3 (or CeCILL) license. So ***without warranty***.

Its mostly C++ code is on github.com/RefPerSys/RefPerSys and sometimes downloadable from http://refpersys.org/refpersys-snapshot.tar.bz2 (that tarball also contains some data in JSON and the repository `.git/` …)

In october 2025 the following features are more or less implemented

- *orthogonal persistence of data*. The entire data heap is loaded from disk files (JSON, C++ etc…) at startup and dumped on disk at normal termination (or periodically, e.g. every 15 minutes). Some data (explicitly temporary) are not dumped (e.g. when they could be recomputed). Mutable data is organized in objects (heavy in memory, with a fixed memory address) and in lighter movable immutable data. We call *value* something which may be persisted (tagged integer, object, scalar, composite value) and is represented by a 64 bits word (often a pointer, except for tagged integer). The C++ `NULLPTR` is not a value (but interpreted as absence of value).
- *precise garbage collection*. Objects are marked, mutable and heavy, but immutable data is moved (managed as in generational GC).
- *Immutable data* include *tagged integers* (63 bits of precision, the LSB being 1), scalar: boxed floats, boxed UTF8 strings, JSON, [lexical tokens perhaps], etc… and composite values: tuples of objects, ordered finite set of objects, abstract syntax nodes, instances (having immutable classs and immutable fields), closures, etc….
- *Mutable objects*. Each of them has a *globally unique identifier* -its objid-, knows its last mutation time, a `std::recursive_mutex` (C++) to serialize access, a unique class (which is reified as a RefPerSys object), a `std::vector` of components, a `std::map` of attributes, and an optional *payload* it belongs to some space (identified by an object, or null for temporary objects; the *space* defines how and where is the object persisted on disk) and a few optional C++ function pointers (used in closures, etc) The objid is a globally unique “random” 128 bits number represented textually as an alphanum string like `_1aGtWm38Vw701jDhZn` …). About a hundred of objects are *roots* (created empty before loading the heap). Constant objects are usable after heap load (which is using `dlsym=` and can be referred by C++ code using C++ raw pointers `rpskob_41OFI3r0S1t03qdB2E` …

When an object has some payload (of C++ superclass `RPS_PAYLOAD`) its payload is unique and belongs to the owning object. The payload may contain values and other data (and is known by the garbage collector and the dumper thru C++ virtual methods).

- The *class model* is ObjVlisp like. RefPerSys classes has a payload containing the superclass, the sequence of fields, the mutable dictionary of methods (mapping message selectors to closures; a message selector is an object).
- **Dynamic code generation** is important for us (and tied to partial evaluation, can happen at runtime or dumptime). Code generation at runtime (or dump time) is not (in october 2025) fully implemented. C++ generation (compiled to `dlopen`-able plugins) follows ideas from `https://arxiv.org/abs/1109.0779` and machine code is also generated using GNU lightning and/or libgccjit and/or libjit.
- **Runtime reflection and introspection** is possible using the libbacktrace and with *RefPerSys frames* described by some objects. Each RefPerSys frames know its calling frame and the values local to it (used by the garbage collector).
- RefPerSys (in contrast to clipsrules) is multi-threaded and uses an **agenda** mechanism. The agenda is reified in some singleton root object. It contains **tasklets**. *A tasklet should always run quickly* (milliseconds, some flag is changed by signal handlers). There is a small fixed number (eg a dozen) of worker pthreads related to that agenda. Each worker agenda pthread fetches a tasklet from the agenda and runs it (in milliseconds).
- Our inference rules should have a set of conditions and a sequence of actions. Some of them could be metarules inspecting the callstack of the working thread (not of other threads) and inspecting and changing the agenda. The insight is to generate code implementing the inference, using later machine learning techniques.

Logiciel Libre RefPerSys

Le projet logiciel libre RefPerSys est un **logiciel libre moteur d'inférences en intelligence artificielle symbolique**.

Le code en C++ est sous licence libre GPLv3+ (ou CeCILL) disponible en https://github.com/RefPerSys/RefPerSys/ (donc ***sans garantie***)

Les idées directrices de RefPerSys sont : les connaissances déclaratives, les métaconnaissances, la métaprogrammation, le raisonnement symbolique, les règles d'inférence (« systèmes experts »), la persistance orthogonale, la réflexivité, le parallélisme par threads POSIX, et l'apprentissage machine ….

Nous recherchons des contributions à ce logiciel libre. Sont actuellement (octobre 2025) réalisés:

- La *persistence orthogonale des données*. Le tas mémoire est chargé au démarrage depuis des fichiers sur disque (JSON, C++ etc….) et il est vidé à la fin (ou periodiquement, peut-être tous les quart d'heures). Certaines données (temporaires) ne sont pas écrites.
- Un ramasse-miettes (R-M copieur pour les valeurs immuables, marqueur pour les objets)
- Des valeurs immuables scalaires (légères) emboîtant: les flottants, les chaînes de caractères, la représentation textuelle JSON (cf `json.org`), des lexèmes.
- Des objets mutables et totalement ordonnés, représentées textuellement par un identifiant unique (tel que `_41OFI3r0S1t03qdB2E`)
- Des valeurs immuables composites (légères et déplaçable par le R-M) pour les tuples, les ensembles finis d'objets, des noeuds d'objets.
- Les objets sont lourds et fixes en mémoire, contenant un verrou d'accès individuel, une séquence de composants, une association entre attributs et leurs valeurs, une charge utile.
- Le modèle objet est celui d'ObjVLisp: héritage simple, classes et métaclasses réifiées en des objets, dictionnaire de méthode.
- Les valeurs composites comme les objets peuvent être chacuns temporaires ou **persistents**.
- Le tas mémoire des valeurs persistentes est chargé au démarrage (depuis des fichiers textuels); à la fin de l'exécution, ce tas -qui a été modifié par l'exécution- est écrit sur disque dans des fichiers (en JSON et C++). Les valeurs temporaires ne sont pas écrites dans des fichiers.
- La génération dynamique de code est esquissée: code C++ généré sur les idées de https://theses.fr/1990PA066799 et https://arxiv.org/abs/1109.0779 et les livres de Jacques Pitrat, code machine généré à l'aide de *GNU lightning* et *libgccjit*.

Le mécanisme d'inférences et la syntaxe des règles et métarègles sont discutées sur https://framalistes.org/sympa/arc/refpersys-forum (en anglais)

Pour en savoir plus, contacter: Basile STARYNKEVITCH, 8 rue de la Faïencerie, 92340 Bourg-la-Reine, France ou par courriel basile@starynkevitch.net

N'hésitez pas à rediffuser ce document verbatim.