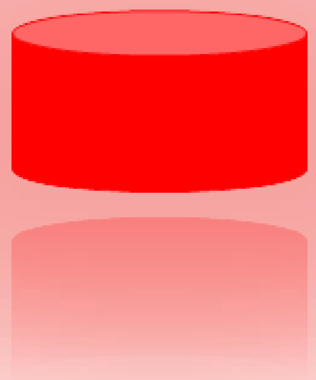
RedSib: a Smart-M3 Semantic Information Broker implementation



F. Morandi, L. Roffia, A. D'Elia, F. Vergari, T. Salmon Cinotti

Alma Mater Studiorum , Bologna University

{fmorandi, lroffia, adelia, fvergari, tsalmon}@arces.unibo.it

FRUCT12 Oulu, 8 November 2012

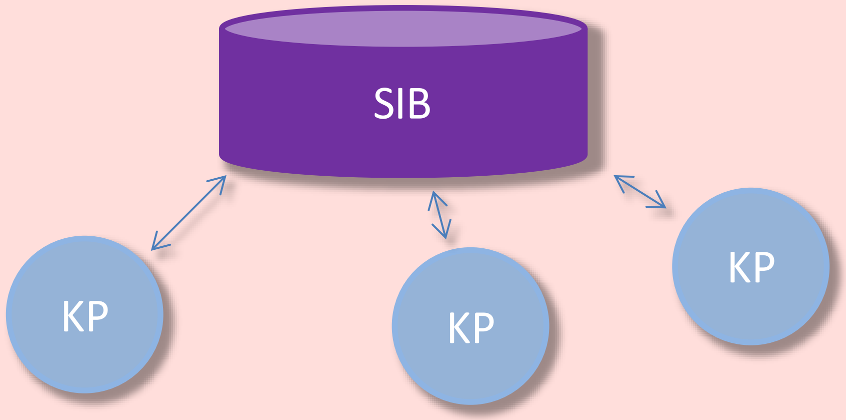
Motivation for an open interoperability platform

- A platform supporting environment related co-operative services is needed
- Heterogeneous devices and data need to be handled
- Required non-functional qualities are scalability, reliability, security, short reaction time and easy access

Introduction to Smart M3

SMART-M3 is based on a blackboard architectural model, supports space-based computing and consists of two main components: semantic information broker (SIB) and knowledge processor (KP).

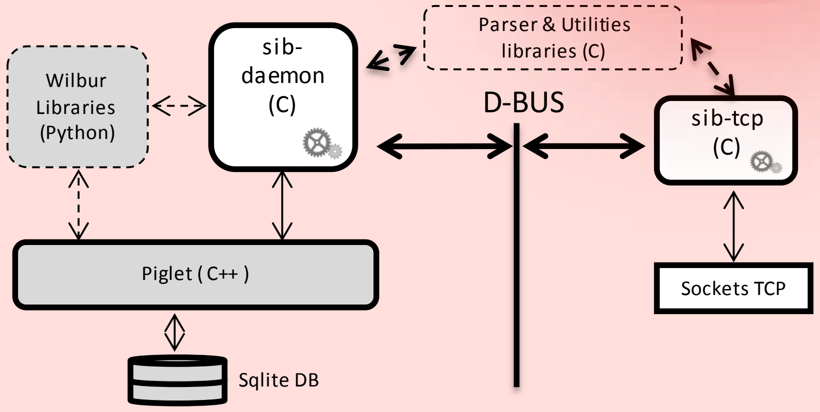
- The information in the SIB is stored as a RDF graph
- KPs communicate with the SIB using an XML based protocol called SSAP (*Smart Space Access Protocol*)
- A subscribe-notify mechanism for context reactivity is provided
- Several "Knowledge Processor Interface" (KPI) (developed in popular programming languages including C, C#, Java, PHP, JavaScript and Python) can abstract from the low-level protocol details.



The original Piglet SIB

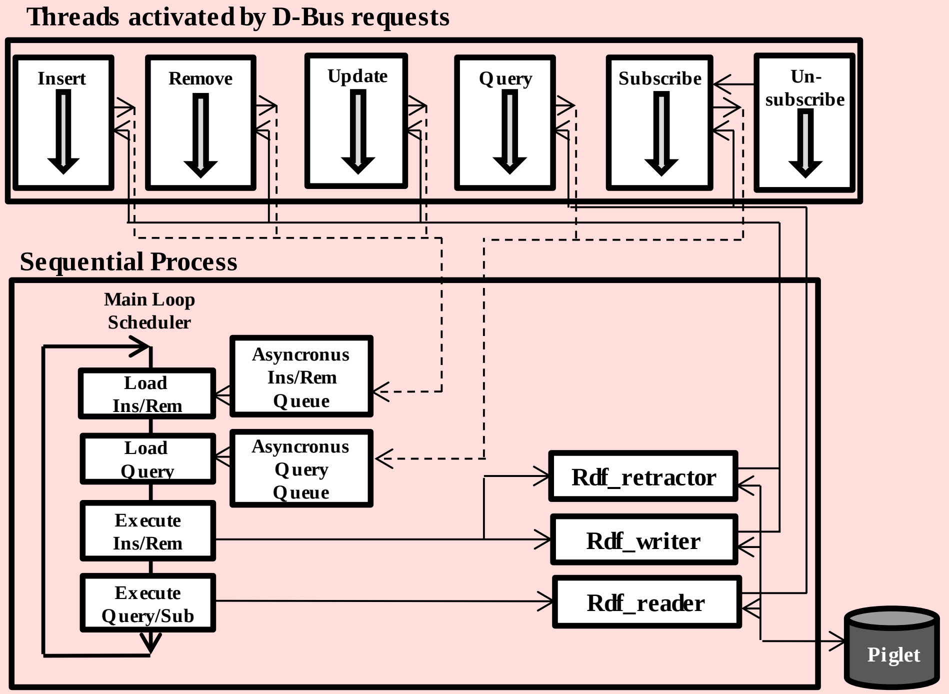
SIB Architecture:

- The *Piglet SIB* is composed of two daemons and a set of libraries (e.g. Piglet RDF store, Wilbur query engine).
- The daemons are seen from the operating system (i.e. a *Linux* based system) as independent processes that exchange content through the D-Bus



SIB-Daemon Implementation:

- The *sib-daemon* is implemented as a multithreading application where the scheduler is the main loop thread. Every time a request operation (i.e. remove, insert, update, query, subscribe, unsubscribe) comes from the D-Bus, a new thread is allocated.
- Subscriptions implemented as queries threads that constantly reload themselves into the query queue until the corresponding unsubscription occurs.



Piglet SIB SWOT Analysis

Strengths

- **C core implementation:**

Native C implementation of daemons and support libraries optimizes:
> Optimize performances
> Memory usage

- **Separate processes:**

D-Bus (originally designed for X Windows) acts as an inter-process communication (IPC) mechanism.
> Load independently connectivity modules and the sib core.

Weaknesses

- **Diverging DB size and performance:**

Piglet RDF store is an experimental software with some issues left:
> Continuous increase of the DB size even upon removes
> Persistent SQL Lite implementation only .

- **Subscription performance impact and stability:**

> Subscriptions are managed by the SIB as slow recursive queries on the triple store.
> The SIB does not detect lost connections and it continues to perform useless queries.

Opportunities

- **RDF++ materialization:**

Starting from the asserted triples and applying the rules defining the RDF++ semantics it is possible to infer new information.
> Inferred triples are automatically inserted into the store, augmenting the knowledge.

- **Synchronization at triple level:**

The need to control concurrent access on shared RDF sub-graphs is an important feature in a multi agent scenario.
In the Piglet SIB implementation a prototype of access control was implemented.

Threats

- **Absence of SPARQL support:**

Wilbur queries are not maintained anymore and the only implementation is a Python based library that works strictly connected with the Piglet RDF store.
> Furthermore, the W3C recommended query method for RDF since 2008 is SPARQL

- **Lack of support for RDF/XML triple encoding:**

RDF/XML is one of the most common syntaxes for serializing RDF knowledge and OWL ontologies.
> According to the Semantic Web stack, the ontology layer is based on RDF and on XML .

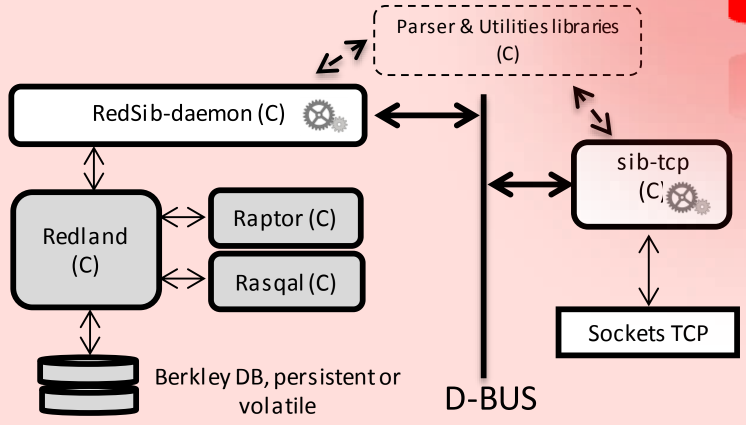
RedSib implementation:

Main Changes in overall Sib Architecture :

Redland replaced Piglet:

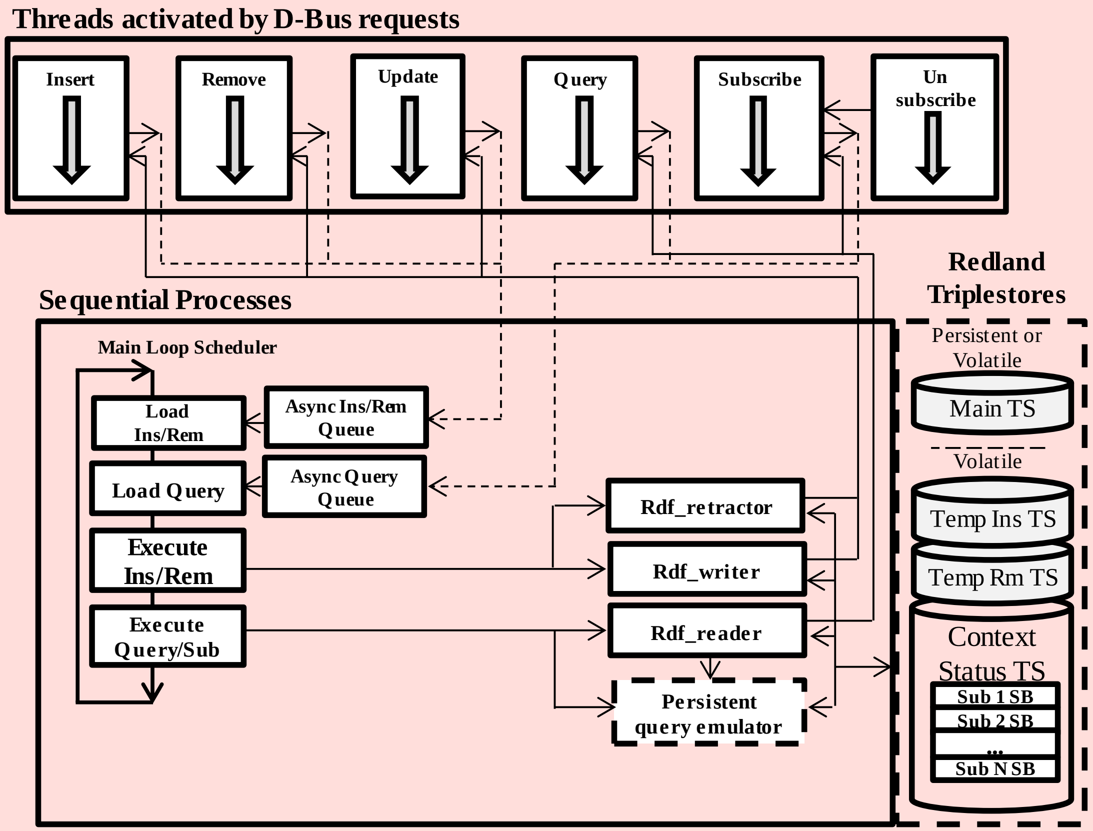
> Native support for SPARQL (1.0 , 1.1 partial) (implemented in Rasqal)

> Native support for RDF/XML syntax



Main Changes in Sib Daemon

- Subscription related triple patterns are cached in RAM to avoid the need for disk queries when an insert or remove primitive is performed
- New subscription mechanism based on the *state buffer* concept, implemented adding a new method (*Persistent query emulation*)



RedSib implementation, Additional features:

- ***Garbage collector for subscriptions both in sib-tcp and RedSib-daemon:***
  - > Automatically unsubscribe with subscriptions “broken” sockets.
  - > Tests socket status transmitting a periodically “space” character (US-ASCII 32) on all subscription sockets.

- ***SSAP extensions:***
  - > SPARQL query request and response.
  - > RDF/XML insert, remove and update.

- ***RDF/XML insert, remove and update.***

  Below are reported the rules implemented in the RDF++ module. If a rule is satisfied, the corresponding inferred triples are added to the triple store, increasing the semantic knowledge.

  (1) $holds(p, s, o) \Rightarrow property(p)$

  (2) $holds(rdfs{:}subClassOf, x, y) \Longleftrightarrow class(x) \wedge class(y) \wedge [\forall z : type(z, x) \Rightarrow type(z, y)]$

  (3) $holds(rdfs{:}subPropertyOf, x, y) \Longleftrightarrow property(x) \wedge property(y) \wedge [\forall o, v : holds(x, o, v) \Rightarrow holds(y, o, v)]$

  (4) $holds(rdfs{:}domain, p, c) \Rightarrow property(p) \wedge class(c) \wedge [\forall x, y : holds(p, x, y) \Rightarrow type(x, c)]$

  (5) $holds(rdfs{:}range, p, c) \Rightarrow property(p) \wedge class(c) \wedge [\forall x, y : holds(p, x, y) \Rightarrow type(y, c)]$

  The following rules have not been included in the RDF++ reasoner because they would introduce to much overhead:

  (6) $holds(owl{:}sameAs, x, y) \Longleftrightarrow x = y$

  (7) $holds(p, x, z) \wedge holds(p, y, z) \wedge type(p, owl{:}InverseFunctionalProperty) \Rightarrow holds(owl{:}sameAs, x, y)$

- ***Synchronization at triple level.***

  Data access control at triple level is a feature implemented in the *Piglet SIB* and all the functionalities have been maintained on the *RedSib*. Furthermore, the *RedSib* also support this mechanism with the new RDF/XML triples encoding.

RedSib, Evaluation of the subscription algoritm:

To validate the proposed subscription algorithm some performance comparison between the RedSib and the Piglet SIB were carried out.

Chosen Device:

(HW) - Fit-PC2, i.e. an embedded PC based on a 1600 MHz, dual core Intel® Atom™ Processor with 1 GB RAM.
(SW) - Linux Mint , 32 bit distribution.

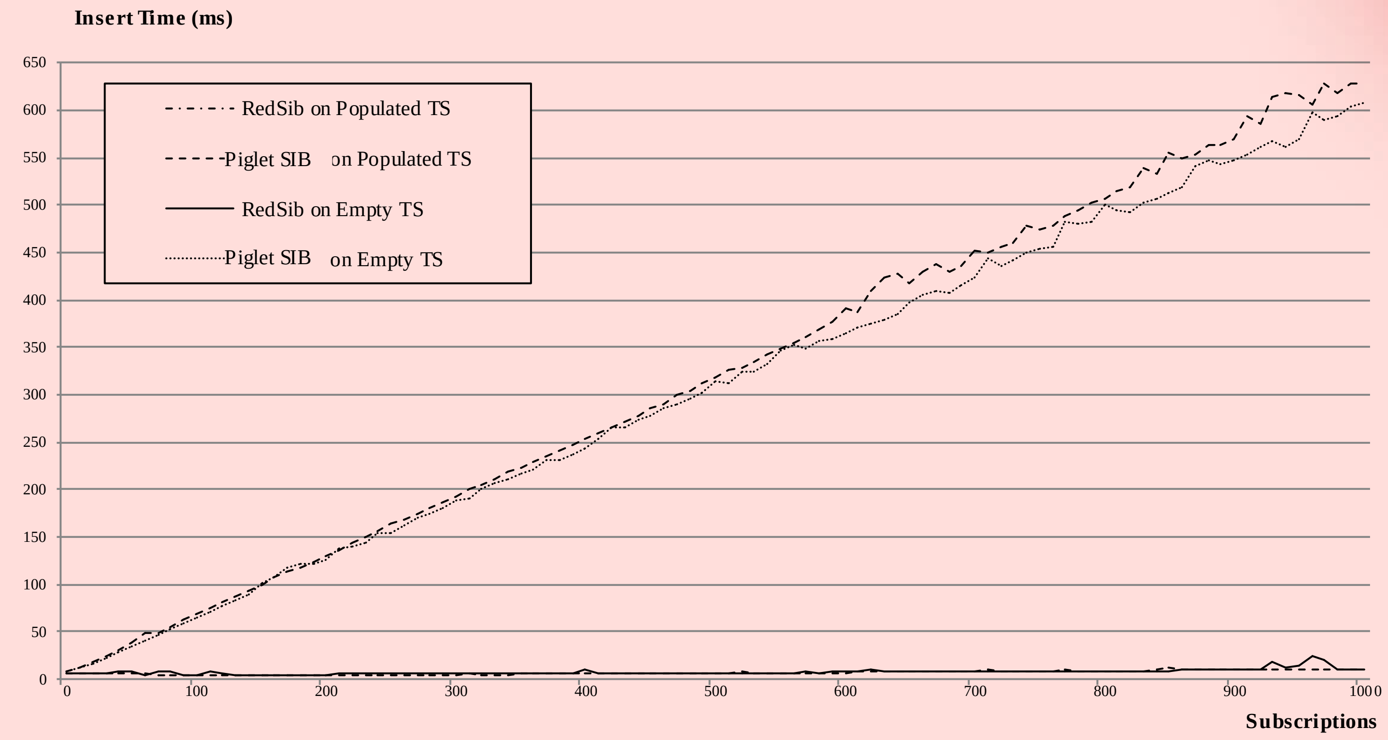
Test Description:

- The two implementations (RedSib and Piglet SIB) are compared with their RDF store entirely located in non-persistent memory (RAM).

- At any iteration (until a maximum of 1000 iterations) a subscription to a single random triple is added, and several (in our case 50) *inserts* and *removes* of single random triples which do not trigger any notification are repeated.
  Every time a new triple is inserted or removed both SIBs check, each one using its own implementation, if some notification messages have to be delivered.
  In this way the average insert (and remove) time as a function of pending subscriptions may be estimated.

- This test was repeated twice for each SIB, once with an initially empty store and once with nearly 15000 triples sitting in the RDF store.

Subscription handling algorithm evaluation



- With no active subscriptions both SIBs have similar performances (5.8 ms for the *RedSib*, 8.9 ms for the *Piglet SIB*)
- Insert time rises linearly in all trends (slightly noised by some unpredictable OS overhead)
- Significant increase rate differences may be motivated by two different implementations of the persistent query emulation algorithm.

Conclusions

The proposed implementation is already used by several research projects, e.g. ARTEMIS-CHIRON (Health Care), ARTEMIS Internet of Energy (electric mobility and smart grids)

Available new features:

- Overall maturity increase with respect to previous implementations
- SPARQL (and RDF/XML) support

Preliminary results show:

- Improved response time and number of active subscriptions supported
- improved performance profile and stability of the subscription checking mechanism

MAIN ISSUES:

- *Security* at service and triple level entirely missing from this implementation
- Too much of overhead *in SSAP Protocol* (based on preliminary tests, performances are strictly dependent on the length of the SSAP messages in TCP, while the SIB is mostly idle. Several SSAP parameters are currently *unused .*

CHALLENGES:

- Turning Smart-M3 into a mature implementation in order to become the **best** interoperability **solution** for Smart Environments
- Improve security and use Smart-M3 to support Distributed Smart Space scenarios (e.g. the V SIB considered within an EIT ICT Labs 2012 Task).