Ishtar: a flexible and lightweight software for remote data access

Antoine Beyeler[†] Stéphane Magnenat[*] Alexandre Habersaat[†]

École Polytechnique Fédérale de Lausanne (EPFL)
[†]Laboratory of Intelligent Systems, Station 11
[*]Laboratoire de Systèmes Robotiques, Station 9
CH-1015 Lausanne, Switzerland
antoine.beyeler@epfl.ch

Abstract

In this paper, we present *Ishtar*, a lightweight and versatile collection of software for remote data access and monitoring. The monitoring architecture is crucial during the development and experimentation of autonomous systems like Micro Air Vehicles. Ishtar comprises a flexible communication layer that allows enumeration, inspection and modification of data in the remote system. The protocol is designed to be robust to the data loss and corruption that typically arises with small autonomous system, while remaining efficient in its bandwidth use.

In addition to the communication layer, Ishtar offers a flexible graphical software that allows to monitor the remote system, graph and log its data and display them using a completely customisable cockpit. Emphasis is put on the flexibility to allow Ishtar to be used with arbitrary platforms and experimental paradigms. The software is designed to be cross-platform (compatible with Windows, Mac OS and Linux) and cross-architecture (it is compatible with both microcontroller- and embedded-PC-based remote systems). Finally, Ishtar is open source and can therefore be extended and customised freely by the user community.

1 Introduction

The monitoring architecture is crucial during the development and experimentation of autonomous systems. This is particularly true in the context of **m**icro **a**ir **v**ehicles (MAV), where the required capabilities of the monitoring architecture usually includes inspection of internal variables, modification of flight parameters, and logging of sensor data. It is also used to remotely control of the aircraft in case of emergency.

In the context of MAVs, weight and power consumption requirements lead designers to select communication hardware that only tightly fits the range and bandwidth specifications. This often translates into communication links that are not always perfect in terms of data integrity and sometimes exhibit data loss and/or corruption. These limitations call for a communication layer that is both immune to data corruption and able to transmit payload with a minimum of overhead. This layer should also be compatible with a wide range of communication hardware to allow flexible adaptation of the flying platform to new experiments and missions.

The requirements for flexibility are not limited to communication hardware. In order to streamline development and experimentation, a monitoring system should handle a wide range of flying platforms and experimental scenarios. This calls for a generic architecture that allows inspection of arbitrary data content, that is compatible with a wide range of underlying hardware, and that provides a customisable **g**raphical **u**ser **i**nterface (GUI).

In this paper, we present *Ishtar*, a lightweight and versatile collection of software for remote data access that addresses the requirements of communication robustness and flexible operations. Ishtar includes a communication layer that implements a generic protocol able to handle arbitrary data structures while retaining efficient bandwidth use and robustness against data loss and corruption. This layer exposes an **a**pplication **p**rogramming **i**nterface (API) both for the server (i.e. the remote system) and for the client (i.e. the monitoring station).

On top of the client API, Ishtar offers a flexible GUI. This software allows to inspect and plot remote data and build custom cockpits tailored to arbitrary platform and experimental setup by linking reusable widgets to specific variables in the remote system.

The next section describes the communication architecture implemented in Ishtar; Section 3 describes the GUI layered on top of the communication architecture; and Section 4 presents an application of Ishtar for the monitoring of a lightweight MAV.

1.1 Related work

We can classify remote data access software in three categories. The first is ad hoc serialization/deserialization mechanisms developed for a specific platform or experimental paradigm, with no attempt at a generic design. While such solution may efficiently solve some problems, they usually do not offer the same level of flexibility as Ishtar.

The second encompasses a wide range of solutions that are usually referred to as *remote method invocation*. In this this type of architecture, the remote target is considered as an object with related methods, in a way that is similar to object oriented programming languages. This provides a natural way of interacting with the remote target, but is not always optimal from a bandwidth and latency viewpoint. Indeed, the client initiates all the transfers, and it must thus poll the target for reading any data; and the request-answer cycle adds latency compared to a model where the target streams the data, which is typically desirable for the monitoring of MAV. Most of these systems are based on CORBA [8, 1, 7], but some use HTTP [6], and others have developed their own communication layers [2, 5].

A third category takes a data centric approach. It resembles the previous category, except that communication is seen as explicit data access rather than method calls. This allows additional possibilities such as asynchronous data transfer [4], or streaming of data from the target, as Ishtar is capable.

Many MAV-oriented monitoring GUI have been developed to complement autopilot systems, for example by *Procerus Technologies*[1], *Schiebel*[2] or *CDL-Systems*[3]. However, they are all tightly coupled to the corresponding autopilot hardware and limit the possibilities for customisation due to their closed-source nature.

The *Paparazzi* project[4] has similar goals and features, but has the potential for more customisation as it is open source. However, this solution is also tightly integrated to its autopilot system and does not offer the flexibility of a generic system like Ishtar.

2 Communication layer

Ishtar include a lightweight communication layer for dynamically accessing remote data. Its architecture is asymmetric client-server: the servers expose the data the clients access (Figure 1, e and b).

At the high-level (Figure 1, c and Figure 2), the protocol handles the enumeration of available data, as well as the reading and writing. Ishtar also allows a client to request a specified subset of data to be sent at regular intervals. This feature – the snapshot – allows the

[1] http://www.procerusuav.com/productsGroundControl.php
[2] http://www.schiebel.net/
[3] http://www.cdlsystems.com/index.php/vcs4586
[4] http://paparazzi.enac.fr/wiki/index.php

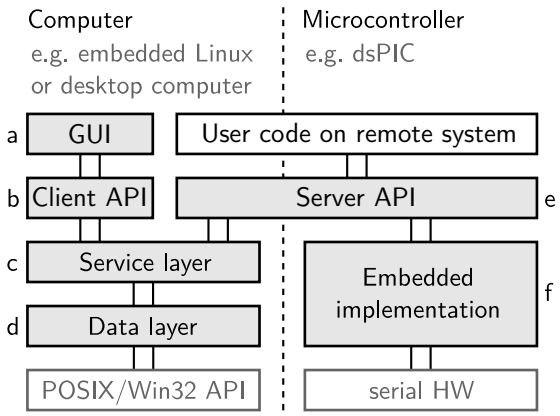

Figure 1: The software architecture of Ishtar. The cells with gray background represent elements that are part of Ishtar. Servers contain the data and clients connect to them.

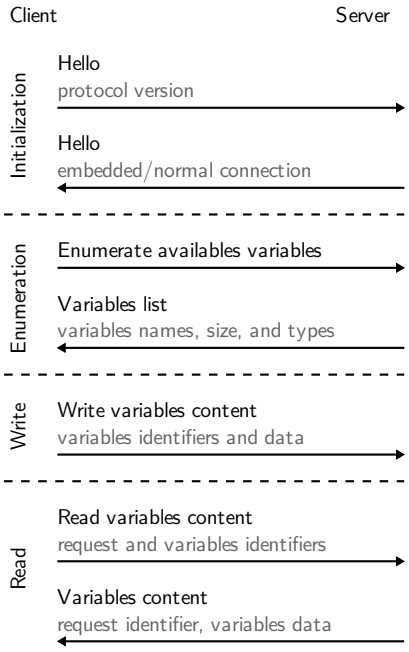

Figure 2: The high-level protocol of Ishtar. The arrows indicate the direction of transmission. Over the arrows, the name of the message is in black and its content in grey. This figure shows the four phases of the protocol. Any phase can safely arise at any time, and their messages can be interleaved. For each phases, this protocol is safe with respect to packets loss. In the enumeration phases, requests to the server can be resent until an answer is received. In the write data phase, if a packet is dropped, the data to write are lost but the protocol does not deadlock. In the read data phase, requests have a unique identifier, and corresponding answers includes it. This allows the client to interpret the structure of incoming data without explicit description in the message. This saves bandwidth while maintaining robustness to packets loss.

server to stream data to the client, which is useful in the context of telemetry.

To minimise bandwidth use, Ishtar remembers its recent data queries by associating each of them with a unique identifier. When data are received, the query identifier is transmitted alongside, which allows Ishtar to interpret their structure. As Ishtar remembers all queries that it has not received data for, this mechanism is robust to packet loss.

The data are organised as a set of typed vectors of fixed size, called *variables*. This structure is general enough for most applications while being sufficiently simple for low-footprint embedded implementations. Servers indicate the amount, names, and types of vectors upon connection of a new client.

At the low-level (Figure 1, d), Ishtar provides robustness against data corruption by optionally adding checksums to each packet. This allows recovery of operations even in cases where arbitrary data loss or corruption can happen. To reduce the overhead when using a safe transport layer, such as TCP sockets, this feature is negotiated upon connections and used only if necessary. At the lowest level, Ishtar uses Dashel[5], a low-level abstraction library that allows cross-platform connection over arbitrary transport link (TCP, serial, etc.).

The standard implementation of Ishtar is in C++ (Figure 1, a to e). This implementation is both efficient and readable, and makes use of C++ standard template library containers. In particular, it is very easy to expose a C++ variable through an Ishtar server, using the server API (Figure 1, e). Likewise, for its client API (Figure 1, b), Ishtar provides a C++ class that integrates seamlessly with third party user code through inheritance. In addition, a compact C implementation of the server, limited to a single connection, allows the use of Ishtar on embedded platforms, such as microcontroller-based MAVs (Figure 1, f).

3 Monitoring interface

Built on its client API, Ishtar includes a monitoring GUI. This software has been designed to be adapted to a wide range of platforms and experimental scenarios. Its interface[6] is organised in several components, as represented in Figure 3. First, the variables exposed by the connected server is hierarchically displayed in a list that allows to read and set the values (Figure 3, a). Each of these variables can be plotted in real time in one or more graph windows (Figure 3, b), either against time or against another variable (e.g. to plot trajectories). If the connected robot is equipped with a positioning system, it can be displayed on a map (Figure 3, c) that uses imagery automatically downloaded from Google Map[7]. The map also displays navigation waypoints when they are supported by the connected platform, and allows to modify their location in real time using drag-and-drop. Finally, the GUI can display one or more custom cockpits (Figure 3, d) tailored to the platform and experimental setup. These cockpits are built from a collection of user interface elements called widgets. A configuration file specifies the layout of the cockpit and the connections between specific variables in the remote system and the various gauges and indicators of the widgets. If required, widgets specific to a new platform can easily be implemented and added to the collection.

Finally, Ishtar being open source and based on a rich and open GUI toolkit[8], everyone can extend and adapt it to their particular needs[9].

4 Application

We present in this section the application of Ishtar to a MAV participating to the EMAV '08 flight competition. It is based on the Swift II R/C model from MS Composit[10]. The total weight is about 350 g and the wingspan has been reduced from 82 cm to 75 cm to fit the requirements of the competition. The aircraft is equipped with a dsPIC-based[11] autopilot, called *Aeropic 6*[12], developed at the Laboratory of Intelligent Systems, EPFL [3]. The Aeropic board uses for stabilisation and navigation a GPS, two pressure sensors (airspeed and altitude), three accelerometers and three gyroscopes. A downward-looking ultra-sonic sensor has also been added to the aircraft for low-altitude terrain following, as well as a wireless camera (2.4 Ghz) for vision-based navigation and target detection. The aircraft communicates with the monitoring station with a XBee link (2.4 Ghz).

On the monitoring station (see Figure 3), a cockpit has been customised using both generic and custom widgets to monitor the different sensors and the decisions of the navigation algorithm of the aircraft. The primary flight display provides information about the estimated angles of the MAV, the altitude and the airspeed, and the autopilot target for these parameters. The autopilot panel has been created entirely using the available buttons and spin boxes, each mapped to the corresponding variable on the MAV. Several internal variables of the navigation algorithm are also displayed for debugging purposes. The battery level is shown on a multi-colored bar and sounds are emitted to warn the operator when the voltage reaches a critical value. Three red/green lights have been mapped to critical variables (corresponding to battery level, GPS fix and radio-receiver

[5] Dashel: http://gna.org/projects/dashel

[6] Demo video available at: http://download.gna.org/ishtar/vid/demo.avi

[7] Google Map: http://map.google.com

[8] Qt: http://www.trolltech.com

[9] The project is hosted here: http://gna.org/projects/ishtar/

[10] Swift II: http://www.mscomposit.com/

[11] dsPic: http://www.microchip.com

[12] Aeropic: http://lis.epfl.ch/smavs

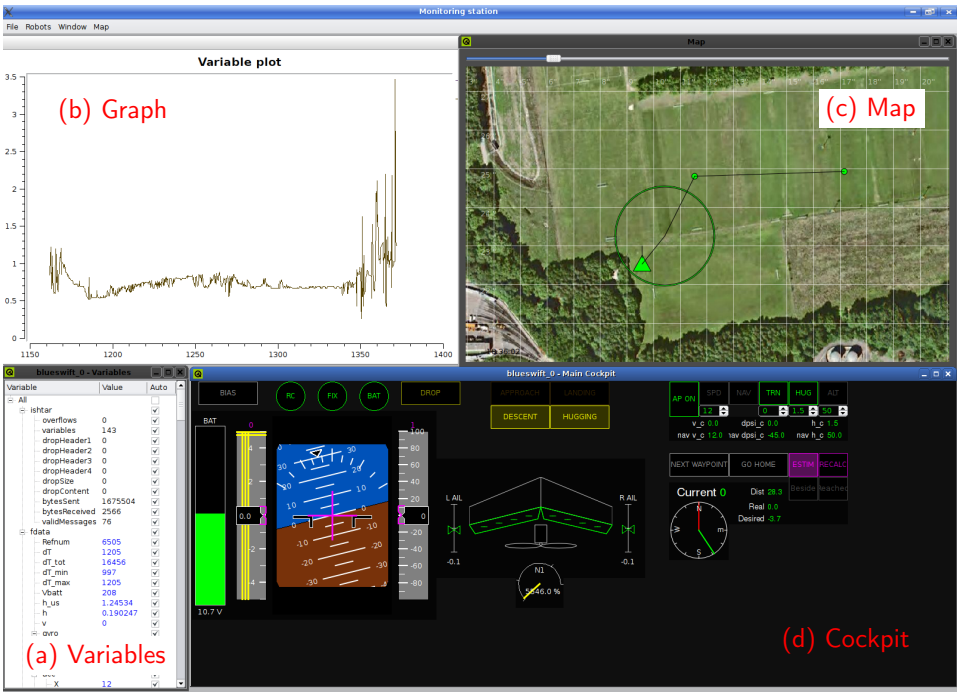

Figure 3: Screenshot of the monitoring GUI included in Ishtar. Its main components are (a) a list of the variables available on the remote system, (b) real-time plots of arbitrary variables, (c) a map with the MAV and navigation waypoints represented and (d) a completely customisable cockpit to represent and control the MAV.

signal strength) and are used during the *pre-take-off* checks. Among the widgets that have been specially developed for the MAV, the status display shows the state of the ailerons and the motor power of the platform.

The MAV exposes 143 variables through Ishtar. These variables are sensor values, commands, or parameters for the stabilisation and navigation algorithm. In typical operational scenarios, 89 variables are sent in snapshots, at a rate of 10 Hz. These packets are 264-byte long and contain 15 bytes of overhead (or less than 6%) for data integrity and protocol structure. For the uplink, values are sent only occasionally when a button is pressed. In these cases, the relative overhead is much more significant as the data payload is very small (15 bytes of overhead for one or two bytes of payload). However this happens rarely compared to the rate of snapshot messages.

During a typical 30-minute preparation flight for the EMAV '08 flight competition, less than 10 packets are typically dropped due to data loss or corruption, but the consistency of the communication is never disrupted and the operations are, in practice, not affected.

5 Conclusion

Ishtar is a generic remote data access solution that is suitable for a wide range of scenarios. We showed with a real example that its capabilities fit well the requirements for the monitoring of MAVs. It also streamlines the development process by making no assumption on the nature of the transport link nor on the internal data organisation of the remote system.

MAV monitoring is not the only domain that can benefit from such architecture; monitoring of any type of robotic system can be implemented using Ishtar. Furthermore, the minimal footprint on server allows Ishtar to be used for remote monitoring of arbitrary software system. For example, we use Ishtar in our laboratory to dynamically inspect and configure large simulations running on computer clusters.

Acknowledgements

We thank Francesco Mondada, Michael Bonani, and Dario Floreano for their support on the original version of Ishtar. The original version of Ishtar was supported by the Swarm-bots and the ECAgents projects, which are funded by the Future and Emerging Technologies program (IST-FET) of the European Community. The information provided is the sole responsibility of the authors and does not reflect the Community's opinion. The Community is not responsible for any use that might be made of data appearing in this publication.

References

[1] N. Ando, T. Suehiro, K. Kitagaki, T. Kotoku, and Woo-Keun Yoon. Rt-middleware: distributed component middleware for rt (robot technology).

In *International Conference on Intelligent Robots and Systems (IROS)*, pages 3933–3938. IEEE Press, 2005.

[2] B. Gerkey, R.T. Vaughan, and A. Howard. The playerstage project: Tools for multi-robot and distributed sensor systems. In *International Conference on Advanced Robotics (ICAR)*, pages 317–323. IEEE Press, 2003.

[3] S. Leven, J.-C. Zufferey, and D. Floreano. A Simple and Robust Fixed-Wing Platform for Outdoor Flying Robot Experiments. In *International Symposium on Flying Insects and Robots*, pages 69–70, 2007.

[4] Stéphane Magnenat, Valentin Longchamp, and Francesco Mondada. Aseba, an event-based middleware for distributed robot control. In *Workshops and Tutorials CD - IEEE/RSJ 2007 International Conference on Intelligent Robots and Systems*, 2007.

[5] A. Makarenko, A. Brooks, and T. Kaupp. Orca: Components for robotics. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 163–168. IEEE Press, 2006.

[6] Makoto Mizukawa, Hideo Matsuka, Toshihiko Koyama, Toshihiro Inukai, Akio Nodad, Hirohisa Tezuka, Yasuhiko Noguch, and Nobuyuki Otera. Orin: Open robot interface for the network. In *SICE*, pages 925–928. IEEE Press, 2002.

[7] H. Utz, S. Sablatnog, S. Enderle, and G. Kraetzschmar. Miro - middleware for mobile robot applications. *Robotics and Automation, IEEE Transactions on*, 18(4):493–497, Aug 2002.

[8] S. Vinoski. Corba: integrating diverse applications within distributedheterogeneous environments. *Communications Magazine, IEEE*, 35(2):46–55, Feb 1997.