Software architecture and middleware for artificial cognitive systems

Johan Wiklund, Klas Nordberg, Michael Felsberg
Computer Vision Laboratory, Linköping University, Sweden.

The DIPLECS project

The DIPLECS (Dynamic Interactive Perception-action LEarning in Cognitive Systems) project aims to design an Artificial Cognitive System capable of learning and adapting to respond in the everyday situations humans take for granted. The primary demonstration of its capability will be providing assistance and advice to the driver of a car. The system will learn by watching humans, how they act and react while driving, building models of their behaviour and predicting what a driver would do when presented with a specific driving scenario. The end goal of which is to provide a flexible cognitive system architecture demonstrated within the domain of a driver assistance system, thus potentially increasing future road safety.

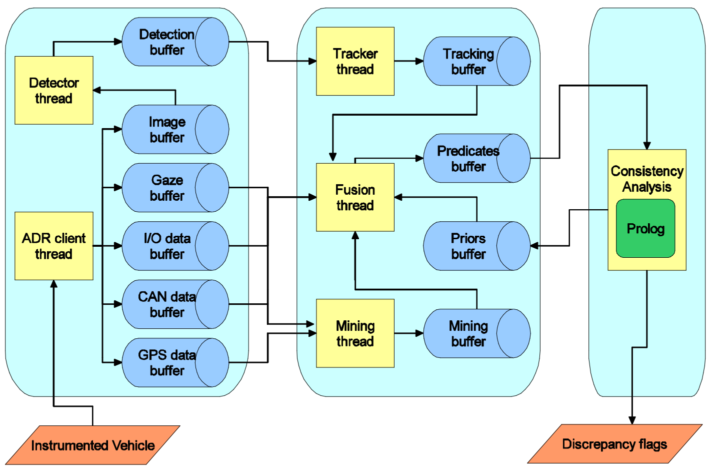
For more information about the DIPLECS project, see http://www.diplecs.eu/

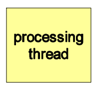
Middleware for system integration

The system integration task need tools to glue together the different parts of the DIPLECS system. After a review of available middleware, ICE was selected as the preferred solution for the DIPLECS system integration. ICE (Internet Communication Engine) is a mature middleware which provide tools for inter-process communication and synchronization.

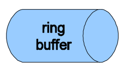
A couple of companies that are using Ice:
- Skype uses Ice as part of its communications infrastructure.
- Baosteel uses Ice as a part of the communications infrastructure for its real-time process control system.

See the homepage of ZeroC for more information: http://www.zeroc.com/





ICE provides a simple thread abstraction that makes it possible to write portable source code regardless of the native threading platform. This shields the application from the native underlying thread APIs and guarantees uniform semantics regardless of the deployment platform. The implemented threads in the DIPLECS system uses the ICE thread abstraction to simplify the integration.



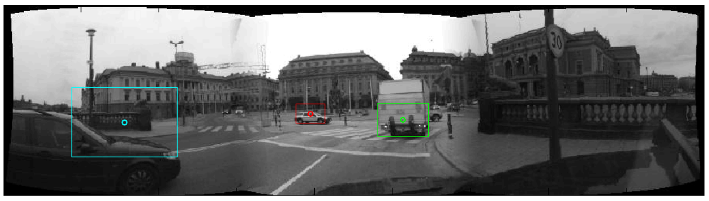
Communication and synchronization between the threads is implemented by using monitor-protected ring buffers. These buffers is implemented as a template class, using functionalities in the ICE middle-ware. The template class contains the access methods for synchronized reading and writing of buffer data. The actual data content is defined when a buffer is instantiated. The template buffer class is used for all buffer instances in the integrated system.

System parts

Available functionalities (instrumented vehicle, ADR data recorder/player) provided by Autoliv is based on .NET code, and the integration was done using the .NET mappings in the ICE middle-ware. The ADR client connects to the ADR data stream and stores the ADR data in ring buffers. The data consists of five different channels, which may have different data rates. Each channel is routed to a separate ring buffer. Every data item is tagged with a timestamp which follows the data through the following processing stages.

The detector, tracker, fusion and mining module implementations are each located in a separate thread. Each thread reads input data from one (or several) data buffers, and put the resulting data in a output buffer. The threads also have public methods to set different parameters.

The consistency analysis is based on the Prolog logic programming language. Python mappings in the ICE middleware have been used to build the connections between the fusion thread (which is C++ based) and the consistency analysis module. The Python language is used to synchronize and send/receive data between the Prolog based consistency analysis module and the C++ based data-buffers.



Interactive system monitoring

The ICE python language binding is also used to create an interactive command interface to the different modules. Using dynamic code generation, Slice (Specification Language for Ice) files are "loaded" at run time and dynamically translated into Python code, which is immediately compiled and available for use by the application. Dynamic code generation is convenient for a number of reasons, it avoids the intermediate compilation step required by static code generation. It also reduces complexity, which is especially helpful during testing, or when writing short or transient programs.

These features makes it an ideal tool during development/debugging. It is straightforward to view data buffer content, call available methods on the implemented modules, such as start/stop the different threads, read data from the ring buffers, setting thread parameters, etc. It is also very useful for visualization of the different processing stages, for example displaying the input image frames, with overlayed tracking results.

Conclusion

Our experience using ICE is very positive, it has really simplified the coding effort of the system integration. Specifically, we have made heavy use of the following features of ICE:
- Machine independence: It is easy to start system parts on different computers.
- Language independence: the system implementation includes .NET, C++, Python, Matlab and Prolog code.
- Operating system independence: Some system parts are only available on Windows, some parts are only available on Linux.
- Threading support: the implementation is multi-threaded.
- Synchronization support: synchronization between the threads is implemented using monitor-protected ring buffers.

Acknowledgement

The DIPLECS project has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement no 215078.