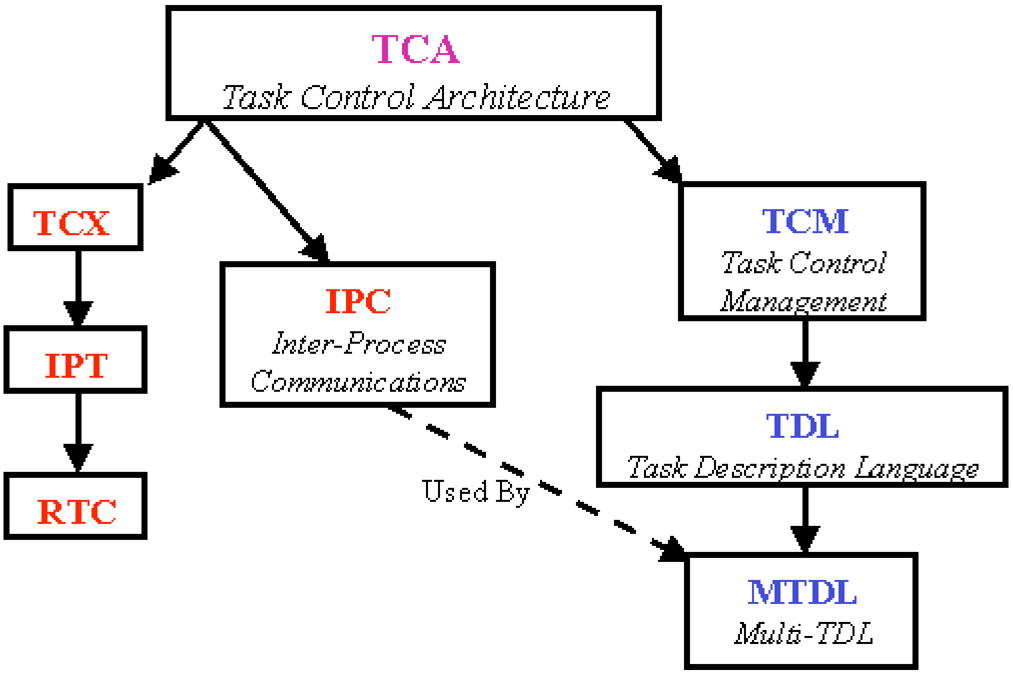
A Brief History of Our Research in Task-Level Control Architectures

Reid Simmons, March 2001


TCA
Task Control Architecture
TCX
IPT
RTC
IPC
Inter-Process Communications
TCM
Task Control Management
TDL
Task Description Language
MTDL
Multi-TDL
Used By

Work on the Task Control Architecture (TCA) began in 1988, to support the Ambler six-legged walker (funded by NASA). TCA includes capabilities for both inter-process communications and task-level control. The inter-process communications features anonymous socket-based communication using TCP/IP that supports both publish/subscribe and client/server modes of message passing. TCA also supports automatic marshalling and unmarshalling of data based on a flexible format definition language. The language supports efficient transmission of all C primitive data types, as well as structures, pointers, and fixed and variable length arrays. All communications in TCA are routed through a central server, which can log all message traffic.

The task-level control portion of TCA includes capabilities for hierarchical task decomposition, task sequencing and synchronization, resource management, execution monitoring, and exception handling. TCA is based on a hierarchical representation of task execution called *task trees*. Task trees represent both the parent-child relationships between tasks, as well as temporal constraints between subtasks. TCA uses a library API to create task trees dynamically – TCA message handlers can send special "goal" and "command" messages that inform the central server to add new task tree nodes under the current node, and to add temporal constraints between nodes. The central server then dispatches tasks (i.e., sending out messages) when the temporal constraints are satisfied. User code can also terminate nodes, raise exceptions, and add new nodes dynamically. This provides a very flexible method for controlling the execution of tasks.

In 1990, Chris Fedor (who helped implement TCA) developed TCX, which is basically the communications infrastructure of TCA. TCX was used in a number of projects, most prominently Dante and the Beesoft mobile robot software (developed by Sebastian Thrun and colleagues at University of Bonn). TCX features peer-to-peer communications and a queue-based paradigm for messages. Jay Gowdy developed the IPT communications system for the UGV (Unmanned Ground Vehicle) program. IPT was implemented in C++ and had many of the same features of TCX. In the late 1990's, Jorgen Pedersen developed the RTC (Real-Time Communications) package for robotic projects at the National Robotics Engineering Consortium (NREC). RTC featured very efficient message passing for real-time applications (including capabilities for shared memory communications). While it uses TCA-type format strings to describe data structures for automatic marshalling and unmarshalling, it does not allow passing of data structures with pointers or variable-length arrays.

In 1994, Reid Simmons developed the IPC (Inter-Process Communications) package for the NASA DS1 New Millennium mission. While, in the end, IPC was not actually used on DS1, it has since been used in numerous projects at CMU, NASA, DARPA, and elsewhere. As with TCA, IPC features efficient transmission of general C data types, anonymous publish/subscribe and client/server capabilities, and automatic marshalling and unmarshalling. IPC features both centrally routed messages (which can be logged and visualized using a graphical tool developed at CMU) and peer-to-peer communications. It has support for timers and much more flexibility for passing data messages, which trade off flexibility for efficiency. IPC runs under Linux, Windows, MacOS, SunOS, Solaris, IRIX, VxWorks, and several other operating systems. It also has a Lisp implementation, and a Java implementation is being contemplated. IPC is actively being supported and extended – a recent addition is a means of automatically generating format strings from XDR data structure definitions.

In 1997, under NASA funding, we began to develop TCM, a library of task-level control functions. TCM is a complete reimplementation, written in C++, of the TCA task-level control capabilities. Instead of having a central server that maintains the task tree representation and dispatches tasks (via message passing), TCM is a library that is linked into user code and maintains the task tree representation locally, dispatching tasks via callbacks. TCM also provides a much richer set of temporal constraints, including the ability to dispatch a task at a given time or at some period of time after another task completes. Currently, TCM is just single-threaded, but we have plans to implement true multi-tasking within the library.

In 1998, we began development of the Task Description Language (TDL), a superset of C++ that includes explicit syntax for task-level control capabilities. The objective was to facilitate writing task-level control programs by embedding such syntax in a language familiar to roboticists. One problem with TCA and TCM, which are both library-based, is that people typically need to be familiar with details of the API in order to use it, even for the simplest application. The design philosophy behind TDL was that simple things should be simple to encode, but that complex things should not be precluded. In particular, you can include as many, or as few, of the TDL capabilities in a program as needed. TDLC, a translator written in Java, transforms TDL code to pure C++, plus calls to a TDL and the TCM libraries. TCM and TDL have been used by a number of projects at CMU, NASA, and elsewhere. TCM and TDL were strongly influenced by other work in execution languages, including Erran Gat's ESL (an extension to Lisp), James Firby's RAPs, and Drew McDermott's RPL.

Most recently (starting in 2000), we have been extending TCM to work in a distributed fashion. In particular, the idea is to distribute the task tree representation so that one process can add nodes to the task tree of another process, monitor the execution of tasks on other processes, handle exceptions raised by other processes, etc. The distributed version of TCM uses IPC to communicate between processes. We are currently extending the syntax of TDL to utilize these distributed capabilities (MTDL). The intention is to use MTDL for two CMU projects in heterogeneous, multi-robot coordination.

Task Control Architecture

The Task Control Architecture (TCA) simplifies building task-level control systems for mobile robots. By "task-level", we mean the integration and coordination of perception, planning and real-time control to achieve a given set of goals (tasks). TCA provides a general control framework, and it is intended to be used to control a wide variety of robots. TCA provides a high-level, machine independent method for passing messages between distributed machines (including between Lisp and C processes). Although TCA has no built-in control functions for particular robots (such as path planning algorithms), it provides control functions, such as task decomposition, monitoring, and resource management, that are common to many mobile robot applications.

TCA can be thought of as a robot operating system --- providing a shell for building specific robot control systems. Like any good operating system, the architecture provides communication with other tasks and the outside world, facilities for constructing new behaviors from more primitive ones, and means to control and schedule tasks and to handle the allocation of resources. At the same time, it imposes relatively few constraints on the overall control flow and data flow in any particular system. This enables TCA to be used for a wide variety of robots, tasks, and environments. It also allows researchers to experiment easily with different instantiations of robot control schemes. To date, we know of about a dozen robot systems that have employed TCA, including both indoor and outdoor, autonomous and teleoperated robots. Within NASA, TCA has been used on the Ambler, Ratler and Nomad rovers, on the Tessalator tile inspection robot, in the VEVI teleoperator interface, and for autonomous spacecraft simulation. In addition, descendants of the TCA communication mechanisms (TCX and IPC) have been used in the Dante robot, and the Aercam project.

Inter-Process Communication

TCA allows you to construct a distributed system without having to build your own remote procedure call mechanism. At its core, TCA provides a flexible mechanism for passing coarse-grained messages between processes (which we call modules). The communication mechanisms automatically marshall and unmarshall data, invoke user-defined handlers when a message is received, and include both publish/subscribe and client/server type messages, and both blocking and non-blocking types of messages. TCA also provides orderly access to robot resources so that you don't have to build your own queuing mechanisms.

TCA is available for both C and Allegro Common Lisp implementations. It currently runs on the following architectures and operating systems: Sparc (running SunOS, Solaris and Mach), Intel x86 and 486 processors (running Linux, Mach, DOS, Windows 3.1, Windows NT, Windows 95), 680xx processors (running VxWorks), SGI, HP, NeXT. It is easily ported to any machine that supports Unix-style sockets.

Task Management

TCA provides a variety of control constructs that are commonly needed in mobile robot applications, and other autonomous systems.

Planning and Execution

The fundamental capability of a robot is to achieve its goals. TCA enables developers to easily specify hierarchical task-decomposition strategies, such as how to navigate to a particular location or how to collect a desired sample. This can include temporal constraints between sub-goals, leading to a variety of sequential or concurrent behaviors. TCA schedules the execution of planned behaviors, based on those temporal constraints.

Execution Monitoring and Error Recovery

TCA provides constructs that enable the robot system to monitor selected sensors and inform the system when the monitored conditions are triggered. To recover from errors in plans, TCA utilities enable robot systems to reason about plans, terminate or suspend portions of plans, add patches, and retry plans. TCA provides a hierarchical exception-handling mechanism for specifying context-dependent error procedures.

Human/Robot Interaction

No robot will be fully autonomous; interaction with humans is a necessity. TCA provides users with the ability to interact with the robot at any level of the task hierarchy. Users may also view the current task decomposition that the robot is executing, and modify it on the fly, if need be. We are currently developing tools to facilitate the task of designing and debugging complex concurrent, distributed systems.

Note that TCA may not be an appropriate framework for real-time control systems. We are, however, currently integrating TCA with ControlShell, in order to provide both task-level and real-time control within a single architectural framework.

The current version of TCA runs on the following machines types:

- SPARC running: SunOs 4.1.3, Solaris, Mach 2.6, VX works
- Motorola 680xx running VxWorks
- x86 running: DOS (with FTP Software's socket implementation), Windows 3.1, Windows NT, Windows 96 (using the Winsock interface), Linux, and Mach
- Dec 3100 (pmax) running: Mach 2.6
- Dec Alpha running: osf 1.3
- SGI Iris
- NeXT
- Soon to be released: Machintosh running: System 7.

The system should run on any machine that supports sockets and has a ANSI c compiler (gcc). It also runs on Allegro Common Lisp on Sun machines, running either SunOS or Solaris. For information on porting TCA to a new architecture, contact Reid Simmons

Send mail to reids@cs.cmu.edu to be added to the tca mailing list, tca-users@cs.cmu.edu.

Support Tools

We have developed a number of support tools and packages for building distributed, concurrent software systems. All of these tools and packages are available through the ftp source (see below).

- comview: A tool for graphically displaying the message traffic between TCA modules. Works by parsing a TCA log file.
- tview: A tool for graphically displaying the hierarchical task composition of a TCA system. Works by parsing a TCA log file.
- devUtils: A package for connecting and maintaining connections (e.g., TCA, X, tty-input, RS232).
- nanny/runConsole: Tools for automatically starting up, killing, restarting, and interacting with processes on multiple machines (currently runs on SunOS and Linux).

Source

- TCA ftp Directory

Documentation

- TCA Manual, version 8.5
- Comview Manual, version 1.0

Release Notes

- Version 7.4
- Version 7.5
- Version 7.6
- Version 7.7
- Version 7.8
- Version 7.9
- Version 8.0
- Version 8.1
- Version 8.2
- Version 8.3
- Version 8.4
- Version 8.5

Last Updated: May 7, 1997 reids+@cs.cmu.edu