Proceedings of the SPIE Conference on Sensor Fusion and Networked Robotics, VIII, Philadelphia, PA, October 1995.

Integration of sensor feedback and teleoperation into an open-architecture standard

John. L. Michaloski
National Institute of Standards and Technology

Paul G. Backes
Jet Propulsion Laboratory

Ronald Lumia
University of New Mexico

ABSTRACT

The Unified Telerobotics Architecture Project (UTAP) proposes a standard control architecture with well-defined interfaces as a means of promoting software reusability and component-based controller technology. Sensor-integration is a primary consideration within a telerobotic standard since teleoperation (or shared) control depends on integrating sensor feedback with motion control. A major hurdle to realizing a standard sensor-integration model involves the provision for ranges of capability, in effect, scaling the interfaces. Another problem is the need within UTAP applications to allow hybrid control, in which the system must accommodate position control in some axes as well as integrate force-sensing with motion control in other axes. This paper will examine the role of sensors as they apply to the UTAP standard telerobotic control architecture.

Keywords: standard, architecture, interface, control, robotics, sensor integration

1 INTRODUCTION

The Unified Telerobotics Architecture Project (UTAP) is an on-going effort to standardize telerobotic technology. UTAP has published a Working Draft document[1] for a standard open-architecture with well-defined interfaces aimed primarily for telerobotic maintenance applications. The UTAP project goal is to describe a reference architecture that can accommodate different types of sensing and actuated devices with differing degrees of sophistication; handle different part materials and part geometries; manage new tasks in the workplace; and provide a facility to upgrade/change computer equipment, sensors, and control mechanisms as technology advances. The UTAP reference model architecture serves as a guide as to how to structure the components in a system. UTAP describes a general model suitable for all modes of control: teleoperation, shared control, supervised autonomy and autonomous control. Depending on the application, a similar, but not necessarily duplicate instance of the reference architecture may be developed.

The goal of the UTAP reference model is to provide a systematic method for building an application system and for modeling the relationships among elemental components in the system. It is intended that commercial controller components and subcomponents will become feasible due to the specification of an architecture and standardization of system components. This will allow systems to be built from standard hardware and software modules which, rather than being custom developed, can be reused from other applications or purchased. The architecture therefore provides a framework for design and implementation of telerobotics systems for different

telerobotics applications using standardized hardware and software modules. The customization in developing a system will be in the selection of which modules to use rather than in the development of all the modules. This will allow both minimal, i.e., inexpensive, and complex, i.e., expensive systems to be built using the same architecture.

A major supposition to achieving the benefits of the UTAP reference model architecture is the necessity for open component technology. Openness via access and modification to component internals provides benefits and savings through flexibility and extensibility. Yet, openness alone does not address the issues of interoperability and portability since interfaces running under one vendor's system will generally not run under another vendor's system. For the UTAP to allow "plug-and-play," standard interfaces, which preserved the openness, were defined for UTAP component technology. For the UTAP interfaces to be successful, it was imperative for the defined component interfaces to be flexible and support a wide range of applications. An underlying requirement for flexibility was the need to explicitly address the use and integration of sensor technology. Within telerobotics the fusion of sensing and control is mandatory since telerobotics can have multiple sources for tool motion including hand controller, trajectory generator, and closed-loop, sensor-based control such as force control and proximity control. Further, within telerobotic applications the use of sensing may be necessary to overcome the impracticality of statically modeling all the objects within the work environment.

Sensor modeling involved three key areas within the UTAP interface definitions: 1) sensor data modeling, 2) sensor data routing, and 3) sensor-integration. To allow for a wide range of sensors, the description of sensor data is in a generic format such as a scalar, 1D, or 2D, since, for example, a 2D range map could be generated from a tactile or a range sensor. Further, UTAP uses a basic profile to describe performance parameters such as sampling rates, update frequency, or scaling factors of the sensor device. To enable sensor data routing, the UTAP interfaces support the ability to instruct a control module to connect and receive third party sensor information, such as, "Read input from sensor B every 10 milliseconds, and then use this reading to calculate an merge-offset to modify the nominal goal." To enable sensor integration, UTAP adopts a broadly-defined motion model with a scaling component to provide for dynamic integration of sensor knowledge. The importance of sensor integration cannot be understated in the UTAP applications of tooling and machining operations. Without sensor integration, hybrid and shared motion control could not be accomplished. For UTAP machining applications, sensor integration can provide values for position offsets to allow for thermal compensation, or provide a mechanism for sensor-driven feedrate override (to overcome chatter or reduce tooling forces and tool wear).

This paper further examines the role of sensors within the UTAP reference architecture and UTAP interface definitions. The next section reviews fundamentals of the UTAP architecture and application programming paradigm. The third section focuses on interfaces and the framework in which the interfaces were defined. The fourth section discusses UTAP provisions for sensor data modeling, sensor data gathering, and sensor-integration. The final section will review UTAP validation efforts and discuss the results.

2 UTAP ARCHITECTURE

The UTAP architecture development process incorporated telerobotics research and development results from universities and national laboratories, previous studies, and current off-the-shelf robotics capabilities. Most of the required capabilities have been demonstrated in point-solution systems, but without a common architectural approach. The unified architecture specifies the hardware and software modules so that telerobotics systems can be built from standard commercial components. The architecture described in this report is a summary of the architecture described earlier.[2] Figure 1 illustrates the application architecture in terms of its primary components.

The architecture includes both implementation and execution features although implementation and execution would be done at different times by different people. Central to the architecture is the application program. This

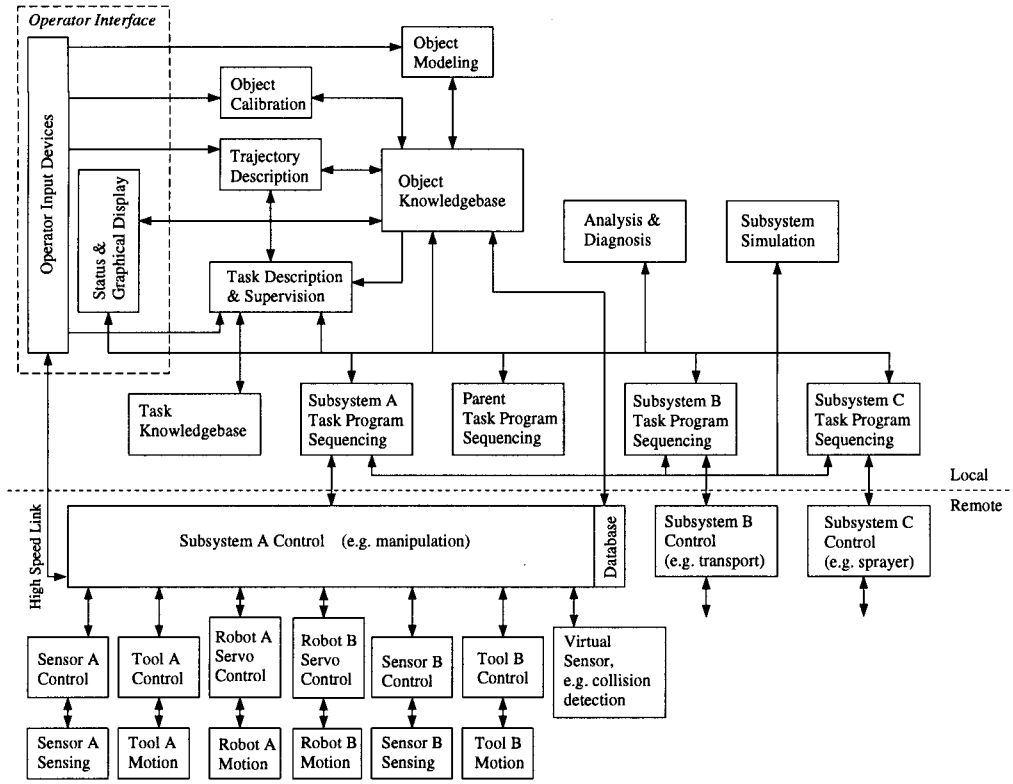

Figure 1: Telerobot architecture for aircraft maintenance and remanufacturing

is the program which is run by the operator to execute the telerobotic task. The application program is separated into subsystem task programs and a parent task program. A subsystem is characterized by having a separate task program. There may be separate task programs running on separate controllers for different robots or mechanisms, or separate task programs running on the same controller hardware. Coordinated control between separate task programs is achieved by direct communication between the subsystem task programs and/or through communication with a parent task program which communicates with the subsystem task programs to coordinate their control.

Different components of the system might run synchronously, asynchronously or upon request. For example, in real-time control, the closed-loop control components might run asynchronously at different rates, reading available data, and producing data to be read. Slowly changing information can be computed at a slower rate than it is used. Alternatively, these components could be synchronized and called in a given order. The task planning components will not be called at the high rates that the task control components are run. Therefore, they could more readily be implemented as agents, responding only when their services are needed.

There may be multiple sources for tool motion, including hand controller, trajectory generator, and closed loop sensor based control such as force control and proximity control. Motion commands from each of these sources can be generated by specific software components associated with the motion source. These motion commands then have to be merged. This merging is done by the motion fusion component. There are many ways that motion can be merged, or fused, with motion commands of various types, e.g., disturbance forces, incremental motion, velocities and absolute positions. The motion source components therefore have to generate motion commands which are consistent with the motion fusion component input types.

The generic UTAP architecture actually has separate hardware and software conceptual architectures since for different implementations, software of a specific functionality may reside on different computational hardware.[3] For example, a Servo control module could reside on a special servo control board or on the same CPU board as Task Level Control module. The software module has clearly defined functionality, but where it is located is application dependent. For an UTAP application system, functional capability can be satisfied with aggregated software modules. For example, functionality for CAD and simulation can be composed as several modules in the UTAP architecture including: Task Description and Supervision, Object Knowledgebase, Object Modeling, Task Knowledgebase, and Subsystem Simulation.

A UTAP system is built as a connected set of modules consisting of hardware components and a software architecture derived from software components. The Parent Task Program Sequencing module controls a set of subsystems which are each managed by a separate Task Program Sequencing module. The bridge between the engineering and process description is done by the Task Program Sequencing module which is responsible for parsing the part program and interpreting its contents. From this parsed information, commands are generated for the Task Level Control including the trajectory generation functionality. The Task Level Control is responsible for coordinating control of the robot(s), sensor(s), and tool(s). The Robot, Sensor and Tool modules are responsible for control, sensing, and communication to individual devices.

3 INTERFACE DEFINITION

UTAP specifies an interface with a set of messages and uses the C/C++ language to define the messages. A function-call style Application Programming Interface (API) is also available to simplify the programming interface. UTAP defines messages using #defines to enumerate a unique message id, and then attaches a data structure to the unique message id. UTAP defines the information models (i.e., global data declarations) and UTAP messages within C/C++ header files. To handle variable length data, UTAP adopts the following strategy: first, declare the degrees of freedom as a mode parameter, and second, reference variable array data indirectly through a heap data storage mechanism that follows the message. Overall, the message structure can be represented using the following notation:

$$MESSAGE = HEADER + CONTENT + HEAP \tag{1}$$

where the *HEADER* contains protocol or "how-to" specific information, the *CONTENT* defines "what-is" or the message information, and the *HEAP* contains the variable-length data contents. UTAP interfaces divide communication into *mode* and *action* messages. The mode messages provide for event sequencing (e.g., start, halt, abort, etc.), set-up, algorithm selection (e.g., `PID`, `FEEDFORWARD`, etc.), and provide for loading control parameters. The action messages either write a command or initiate a sensor reading. Action messages treat communication as clocked data flow. The UTAP interfaces apply the following concepts.

Programmable I/O The UTAP control interface strategy adopts the command, status and mode concepts of programmable I/O (`PIO`) chips. `PIO` chips have operational modes and parameters that must be initialized before the chip is functional. Further `PIO` chips allow for combinations of selection modes. Selection vectors are of extreme relevance to teleoperated robotics - for example, the application of force control in one axis, while using position control in the other axes.

Generic State and Mode Generic messages were defined that are applicable to all modules in the UTAP architecture. Mode and state change commands are covered by the generic messages. Such state change commands include: start, halt, hold, resume, suspend, etc.

Servo Control Module interfaces that support sensor/effector directives mimic a servo control model with communication from a superior to a subordinate module treated as clocked data flow. Of note, the clocked data flow may only last one cycle. Interface messages of one clocked cycle are conceptually query/response interfaces. The clocked data flow can be either control commands or status readings. For control commands,

response to the command is not an answer, but a servoing action and status report. For status readings, response is either a status report or a sensor interpretation.

RS274D Control systems require synchronization of devices within the system. For example, one would like for a tool change to complete before initiating the next tooling motion. For the UTAP interfaces, synchronization is achieved through the use of the `BEGIN_BLOCK`, and `END_BLOCK` generic messages. This construct is similar to the block concept in RS274D.[4] Messages that arrive between the `BEGIN_BLOCK` and `END_BLOCK` messages are treated as a unit. It is assumed that the receiving module understands as well as documents how operations are synchronized.

Data Routing Data flow of every parameter of the system state at every clock cycle is senseless. Some data-flow spigot is necessary. The UTAP interfaces support a `ROUTE` concept that was intended to provide a contextual-based mechanism for sampling and posting of state information. A module receives get-info queries and then posts the desired state information. Depending on the type of get, the state information could be posted once or periodically updated.

Keyword Context One simplification was to use a keyword convention to categorize control, data, parameter and mode message traffic. The generally self-explanatory keywords are grouped by type:

MESSAGING – `BLOCK`, `MACRO`, `PLAN`, `EVENT`, `SELECTION`

SEQUENCING CONTROL – generics (i.e., `STARTUP`, `SHUTDOWN`, `ENABLE`, `DISABLE`, etc.)

MODALITY – `USE`, `START`, `STOP`, `COMPUTE` (e.g., `USE_PID`, `COMPUTE_THRESHOLD_VALUES`)

PARAMETRIC – `LOAD`, `INCREMENT`, `SELECT` (e.g., `LOAD_VELOCITY_LIMIT`)

DATA COMMAND – `SET`, `ADJUST`, `GET` (e.g., `SET_POSITION`, `GET_READING`)

STATUS – `POST` (e.g., `POST_2D_SENSOR_READING`)

Figure 2 shows the module specification in terms of categorizing by keywords and by flow of information. The flow of traffic is divided into: 1) control sequence, 2) modes, 3) algorithm selections, 4) parameter settings, 5) real-time data, 6) information requests, and 7) information responses. The information flow is equivalent for superior as well as subordinate connections, except that there can be multiple instances of subordinate information flow.

4 SENSOR MODELING

4.1 Data Modeling

UTAP defines standard data representations for object knowledge within the sensor and world model. The standard UTAP data representation provides communicating modules a like-representation of knowledge. Sensor data representation is generically categorized by dimensionality as scalar, vector, or two dimensional array. Object knowledge within UTAP includes representation for the devices, parts, modules, and general system state information. Object knowledge is defined with attributes, and access to information is through query/response connection. UTAP further defines attribute modifiers including maximum, minimum, average, actual, desired, last, and timed historical reading. For example, one can get and post the desired and the actual position. The attribute in this case is position, and the modifiers are desired and actual.

UTAP also defines messages for the setup and profile of a general sensor. Many of these are convenience items that make a programmer's life easier. For instance, instead of mandating SI units, UTAP offers the ability to define the sensor reading units and data representation. Another UTAP sensor provision was to include common sensor facilities such as the ability to apply transforms, filters, and/or scaling offsets to modify the sensor readings. Sensor performance messages were added to provide the capability to program the sampling

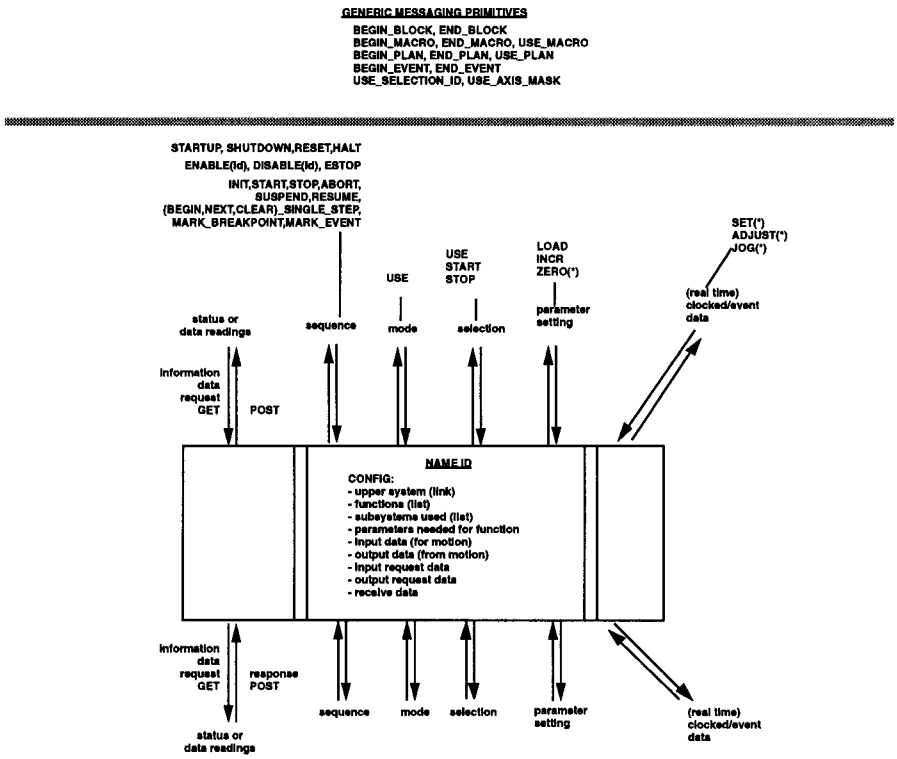

Figure 2: UTAP Model for Module Specification with Keyword Interface Breakdown

and frequency of a sensor directly across the interface, not just in a statically defined configuration file. It is important to note that not every UTAP message must be accepted by a sensor, instead a sensor publishes which messages it can accept. Overall, the following list highlights some of the messages defined for sensor setup and profiling: `SENSOR_USE_MEASUREMENT_UNITS`, `SENSOR_LOAD_SAMPLING_SPEED`, `SENSOR_LOAD_FREQUENCY`, `SENSOR_LOAD_TRANSFORM`, `SENSOR_LOAD_FILTER`, `START_TRANSFORM`, `STOP_TRANSFORM`, `START_FILTER`, `STOP_FILTER`.

The initial UTAP world model concentrated on such airplane parts as wings and fuselages. These parts can be described as a combination of geometry, topology, and shape. UTAP defines an information model that is a subset of the ISO STEP Part for Geometrical Shape and Material Information Models.[5] For UTAP, the EXPRESS Geometry and Topology Part Models were directly translated into C++ language for the UTAP information model. UTAP defines part features to identify the focus of attention for the tooling operation which were described as simple shapes that are filled by motion patterns, as represented with the following equation:

$$feature = pattern \cdot shape \tag{2}$$

For the UTAP application domain, features are a composition of a motion pattern applied to faces of different geometrical shape - e.g., orbital motion against a flat surface. Some patterns are merely shorthand notation for a larger set of motions. For example, a raster motion sweep can be composed as a set of linear motions.

4.2 Data Routing

Across each sensor category, a UTAP **GET_VALUE** message is universal. The verb **GET** is used to initiate a response - either a status report or a sensor reading. **GET** has a corresponding response **POST**. The **GET** is part of the query/response interface connection. Each **GET** message contains routing, attribute and modifier information. For example, the generic message **US_GET_VALUE** requires routing, an attribute and a modifier.

```
#define US_GET_VALUE 55
struct us_get_value_msg_t {
        int msgid;
        ROUTE r;
        Attribute_t items;
        Modifier_t modifiers;
};
```

The verb **POST** is used to convey the notion of an output reading. This output reading could be either a status report, or a sensor reading. Further, the output reading could be posted to the superior or to the Object Knowledge module. Within the **GET** a field is set aside to designate the destination of the subsequent **POST** - either to the superior, Object Knowledge module or both. The **POST** messages use this information for return routing. UTAP posting messages were often customized according to expected sensors readings. For example, although one can construct a force/torque query message from generic building blocks, it is redundant since this sensor is quite common. (For example one can use the generic message **GET_DATA_LIST** with *attribute* = *_force|_torque*.) Wherever possible, sensor readings that were anticipated to be common were given a distinguishing message name. The following example, which includes data routing covered next, outlines a UTAP interface to a force torque sensor:

```
DEVICE dev;
ROUTE route;
Attribute_t attr;
Modifier_t modifier;
us_ft_sensor_post_reading_t *reading;
int errorstat;                                          // error
double fx,fy,fz;                                        // force
double tx,ty,tz;                                        // torque

    // setup parameter attribute and modifier info
    attr = Attribute_t.force | Attribute_t.torque;
    modifier=Modifier_t.actual;

    // setup routing info
    route.type = ROUTE.STATUS;
    route.times = 1;

    dev=use_selection(get_selection("TLC:A:FORCE_TORQUE_SENSOR"));
    dev.load_dof(3);
    dev.load_sampling_speed(100*Hz);                      // in Hertz
    dev.load_frequency(.10);                              // update every 100 milliseconds
    dev.load_filter(sensor_load_filter_msg_t.HI_PASS,1000);
    dev.start_filter();                                   // start filter
    dev.start();                                          // start sensor

    while(1)
    { errorstat=dev.sensor_get_attributes_reading(route,attr,modifier,&reading);
      fx=reading[0]; fy= reading[1]; fz=reading[2];
      tx=reading[3]; ty= reading[4]; tz=reading[5];
      // Now, use to compute ....
    }
```

As mentioned earlier, one doesn't want every conceivable piece of system state information flowing through the system at every clock cycle. One would prefer that under certain circumstances, relevant state information is posted in the timely manner desired. For example, under normal operation, it would be desirable to post the current position as status every 10 milliseconds. For gain tuning, one may require position readouts every millisecond. Under maintenance operation, it might be desirable to post the current position and encoder readings so that a problem can be tracked down. UTAP interfaces were designed to be flexible and allow a range of state information to be posted.

The **ROUTE** data structure defined below was intended to provide a contextual-based mechanism for posting sensor readings and state information. A module would receive a get-info query and then post the desired state information. Depending on the type of get, the state information could be posted once or periodically updated. The same mechanism can be used to read state information data from the Object Knowledge module.

```
struct ROUTE {
        enum {  _STATUS         = 1,        // post response to questioner
                _WRITE_TO_OK    = 2,        // posting response values to object knowledge module
                _READ_FROM_OK   = 4,        // read from obj know base
                _ITH_OFFSET     = 8,        // use data as ith offset
                _MERGE          = 16,       // merge cmd dx,dy,dz,rx,ry,rz
             } type;                        // Bitmask for response dest
        int times;                          // -1= continuous, 0=stop, 1=1,...
        TIME update_period;                 // frequency of update
};
```

4.3 Sensor Integration

The integration of sensing with motion control can take on many forms including axes-feedback, force/torque feedback, proximity sensing, operator-input, among others. This feedback can be used as an offset against the nominal trajectory path or to modify the rate-control (e.g., feedrate override). Sensor feedback can also be used for servo compensation. This section will study sensor-integration within the Task Control Module which handles trajectory generation.

Since a multitude of control options can occur, it was decided that the UTAP interfaces must support simultaneous multiple control modes. The keywords **START** and **STOP** were defined the notion for initiating/terminating simultaneous or multiple selections. Thus, one can **START** one or multiple algorithms when necessary. Then, the command **STOP** is used to discontinue the algorithm. For example, a standoff motion could be used to describe a distance that an axes must maintain from a given surface. Thus a **START_STANDOFF_MOTION** message remains in effect until terminated with a **STOP_STANDOFF_MOTION** message. The other axes may use position control. The following control modes were provided for combinations of sensor-integration at the Task Level Control (or trajectory) level:

manual motion – when an axis is under operator control

automatic motion – when an axis is under autonomous control

guarded motion – when an axis is about to contact a surface

compliant motion – when an axis is continuing contact with a surface

traverse motion – when an axis is moving in free-space

fine motion – when an axis motion is defined with tolerances

standoff motion – when an axis motion maintains a proximity distance

force position motion – when an axis does force reflection

move until motion – when an axis comes in contact

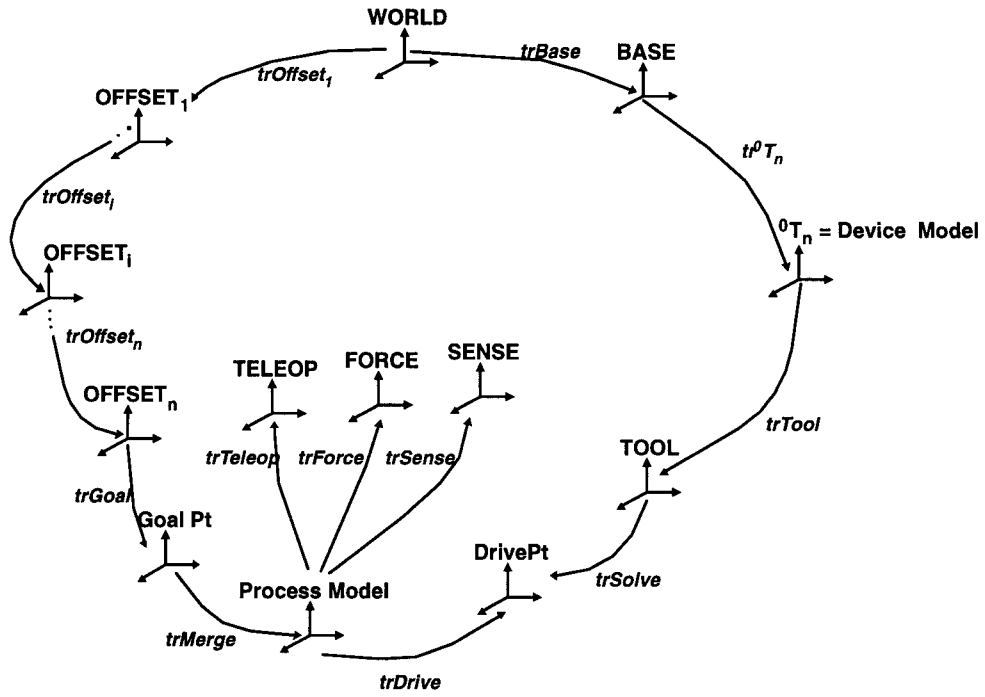

Figure 3: Diagram of generalized kinematics ring

In these cases of sensor-integration, a vendor-supplied trajectory generator module is directly responsible for integrating sensor-feedback into the motion control. On the other hand, users may wish to modify the motion in their own manner. For user or third party modifications, UTAP used the common technique of kinematic rings[6] which is equivalent to position/force equations[7] to describe the nominal motion position with allowances for dynamic offset transforms. Within the kinematic ring, some offset transformations could then be derived from sensor-readings as a part of the process model. The UTAP kinematic ring is shown in figure 3. The figure shows the various coordinate frames and the transformations between them (defined in the nomenclature defined below.) The coordinate frame nomenclature in the diagram corresponds to the following definitions:

WORLD – common frame for specifying position in workspace

BASE – base coordinate frame

Device Model – coordinate frame mapping from device base to device wrist dependent on device kinematics

TOOL – coordinate frame mapping corresponding to tool offset

Drive Pt – coordinate frame mapping as scaled by the trajectory motion

Process Model – coordinate frame in which sensor knowledge is merged

Goal Point – nominal goal described as position or segment

Offset – series of coordinate frame mappings from object base to part (e.g. table, program offset, part edge)

The transformations are defined as follows:

trOffset $= {}^{WORLD}_{OFFSET}T$ = nominal goal segment on part

trOffset$_i$ $= {}^{OFFSET_i}_{OFFSET_{i+1}}T$ = series of workspace offsets

trGoal $= {}^{OFFSET_i}_{GOAL}T$ = nominal goal segment on part

trMerge $= {}^{GOAL}_{MERGE}T$ = process model for sensor fusion

trDrive $= {}^{MERGE}_{DRIVE}T$ = trajectory generator motion

trSolve $= {}^{DRIVE}_{SOLVE}T$ = kinematic solution

trTool $= {}^{WRIST}_{SOLVE}T$ = device tooling offset

tr^0**T**$_n$ $= {}^{BASE}_{WRIST}T$ = kinematic chain from base to wrist

trBase $= {}^{WORLD}_{BASE}T$

Kinematic ring equations are used to describe the goal point and motion of the device and the relationship between them. The general UTAP kinematic ring equation is:

$$trBASE \cdot {}^{n}_{0}T \cdot trTOOL \cdot = trOFFSET_1 \cdot \ldots \cdot trOFFSET_n \cdot trMERGE \cdot trDrive \tag{3}$$

For UTAP to standardize and allow for scaling with this model, a selection mask was defined for potential fields in the kinematic ring. The selection mask provides one with the ability to define levels of positioning control - from simple position updates – to allowing sensor fusion and specifying transform models for the base, tool, object base. This scaling of specificity allows a broader range of modules to use the same generic interface without unnecessary burden on simpler control modules.

```
#define US_TLC_USE_KINEMATIC_RING_POSITIONING_MODE 626
struct us_tlc_use_kinematic_ring_msg_t {
        int id;
        Measurement_units_type units;
        enum {    BASE           = 0x0000001,
                  TOOL           = 0x0000002,
                  SENSOR_FUSION  = 0x0000004,
                  // RHS
                  MERGE          = 0x0000010,
                  OBJECT         = 0x0000020,
                  OBJECTBASE     = 0x0001000,
                  OBJECTOFFSET2  = 0x0002000,
                  OBJECTOFFSET3  = 0x0003000,
                  OBJECTOFFSET4  = 0x0004000,
             } ring_mask;
};
```

For the integration of sensor-readings, UTAP provides a routing parameter within the **GET** command that can instruct a subordinate module to read sensor or operator feedback from another module at a designated location. This routing can be tied to one of the kinematic ring offset locations. Thus, a user or third-party process running concurrently can provide dynamic adjustments to the kinematic ring.

One major consideration for UTAP flexibility was the ability to do on-line configuration and assignment of communications. For example, the addition of a new sensor to a system implies the need to connect the sensor to

some other module. UTAP provides for the use of the **GET** message with the **USE_SELECTION** message to enable sensors to feed readings into a designated sensor fusion slot. Applying this to the trajectory generation, one specifies the routing information within a **GET** message as a **MERGE** or **ITH** offset destination. Thus, the module uses a subordinate reading to modify its nominal goal with a merge offset. The following code demonstrates this concept. The code embeds a **SELECTION** and **GET** message within a **MACRO** message that will instruct a subordinate to use one of its subordinates readings as a merge offset value.

```
sensor=us_use_selection(us_get_selection("ROBOT.A:SENSOR.B"));
dev=us_use_selection(us_get_selection("ROBOT.A"));
dev.us_begin_macro("Sensor  Fusion");
dev.us_get_value(ROUTE._ITH_OFFSET,
                         sensor,                    // from where
                         -1,                        // forever
                         .002,                      // every 2 milliseconds
                         1);                        // in ith offset location 1
dev.us_end_macro("Sensor Fusion");
dev.us_use_macro("Sensor Fusion");
```

5 DISCUSSION

Validation of the UTAP standards effort is being performed on several test applications, including an Advanced Chamfering and Deburring System, an Advanced Grinding System, and a Next Generation Armament Loading system. This section will review one validation system – the Advanced Deburring and Chamfering System (ADACS) which is a machining workcell designed to put precision chamfers on parts made with hard metals.[8] Chamfering aircraft components made of hard metals – such as inconel and titanium – requires special tooling and exacting procedures to produce suitable chamfers with tolerances approaching 0.07mm. To achieve such tolerances, ADACS is a study in integration of computer, control, and sensing subsystems. ADACS has CAD modelling, graphical simulation, real-time motion control, adaptive force-based tooling, and operator-supervisory control.

Single-vendor, closed-controller technology did not span the breadth of the ADACS application requirements. Instead ADACS was built with subcomponent controller technology that was open and provided access to internals. Since standard interfaces for commercial components do not exist, middleware was developed to map each vendor-specific interface to the UTAP interface specification. Based on interim results, ADACS demonstrated that a numerically-controlled finishing systems could and should be built from open component technology. To satisfy the UTAP interface specification, in-house middleware was developed as a programming front-end to individual components. Overall, the UTAP reference model architecture was used in the design of the system and did not impede the ability of the system to perform real-time part finishing that included coordination of multi-vendor software components to chamfer airplane parts. When applying open-architecture concepts within a heterogeneous and distributed system environment, one finding is that it is crucial to have available a robust infrastructure and set of integration tools - including common data structures, component naming, network communication, etc. Ideally, a UTAP system infrastructure including configuration and integration specification could be developed and ultimately standardized.

In conclusion, this paper presented an overview of the role of sensors within the UTAP standard environment. While the UTAP reference architecture is reasonably stable, the UTAP interfaces are still preliminary and undergoing further review. Evaluation and acceptance of the sensor interface framework and its assumptions is critical to success. Industry repeatedly requests the ability to easily integrate sensor products with control products. The hope is that the UTAP reference architecture and interface framework can satisfy this need. Validation of the interfaces and the sensor framework through implementation and testing at several different facilities is ongoing. Infrastructure and low-level communication issues have been sketched by the interface framework, but not formalized. Despite the concerns, it is felt that the UTAP framework is a good starting point and with further

evaluation and feedback it should evolve into a viable standard.

ACKNOWLEDGMENTS

This standards effort was carried out under the sponsorship of the Air Force Material Command (AFMC) Robotics and Automation Center of Excellence (RACE).

6 REFERENCES

[1] R. Lumia, J.L. Michaloski, R.T. Russell, T. Wheatley, P.G. Backes, R. Steele, and S. Lee. *Unified Telerobotic Architecture Project (UTAP) Standard Interface Environment (SIE).* Gaithersburg, MD 20899, May 1985.

[2] NASA JPL. *"Final Report: A Generic Telerobotics Architecture for C-5 Industrial Processes".* Air Force Material Command (AFMC), Robotics and Automation Center of Excellence (RACE), San Antonio Air Logistics Center, Kelly AFB, TX 78241, 1993.

[3] Paul G. Backes, Wayne Zimmerman, and Michael B. Leahy, Jr. A telerobotics architecture for aircraft maintenance and remanufacturing. In *Proceedings World Automation Congress*, pages 147–154, Maui, Hawaii, August 14–17 1994. TSI Press, Albuquerque, New Mexico. Published in Intelligent Automation and Soft Computing: Trends in Research, Development, and Applications, Vol. 2.

[4] Engineering Industries Association, Washington, D.C. *EIA Standard - EIA-274-D, Interchangeable Variable, Block Data Format for Positioning, Contouring, and Contouring/Positioning Numerically Controlled Machines*, February 1979.

[5] International Organization for Standardization. *ISO 10303-42 Industrial Automation Systems and Integration Product Data Representation and Exchange - Part 42: Integrated Resources: Geometric and Topological Representation.*

[6] Paul G. Backes. Generalized compliant motion with sensor fusion. In *Proceedings 1991 ICAR: Fifth International Conference on Advanced Robotics, Robots in Unstructured Environments*, pages 1281–1286, Pisa, Italy, June 19–22 1991.

[7] Rick Guptill and Paul Stahura. Multiple robotic devices: Position specification and coordination. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 1655–1659, Raleigh, North Carolina, March 31–April 3 1987.

[8] R. Russell, J. Michaloski, and K. Stouffer. The adacs implementation of the utap architecture. In *6th International Conference on Manufacturing Engineering*, Melbourne, Australia, November 29 – December 1 1995.