NEWS FROM SWIM IN SPACE

Frank Morlang

German Aerospace Center DLR, Lilienthalplatz 7, 38108 Braunschweig, Germany, frank.morlang@dlr.de

ABSTRACT

All the future air traffic participants are requested to act as system wide information management (SWIM) communicating sub-systems by the future Single European Sky Air Traffic Management Research (SESAR) SWIM "Intranet for ATM" concept. Against the background of the global character of future commercial space transportation (CST) operations and the associated SWIM harmonization need referring the U.S. Next Generation Air Transportation System (NextGen) and SESAR, a first solution based on the already harmonized data format standards Aeronautical Information Exchange Model (AIXM) and Flight Information Exchange Model (FIXM) had been realised in the Tool Command Language (Tcl). The new version's improved performance by the usage of a C code runtime embedding module, a data representation proposal for vehicle specific hazard area information as well as the way for flexible extensions and enhancements in the future by a fundament for mixed language programming additions are presented.

1. INTRODUCTION

The SWIM ideology can be abstracted as the enabler of data delivery at the right time to the right people in terms of quality information that is commonly understood, achieved with the help of open standards based on service oriented architecture (SOA). Facing the integration challenge of space traffic in the current Air Traffic Management (ATM) needs SWIM compliancy of future commercial space transportation (CST) vehicles having "landing like an aircraft at an airport" characteristics.

2. THE SOLUTION ENHANCEMENT

2.1. Performance Improvement

A greater than 30% performance (Fig. 1 and Fig. 2) improvement was successfully realised by replacing the CalculateHeading and CalculateHazardZone Tcl procedures by C code, benefiting from the "Compiled Runtime In Tcl" (CriTcl) package [1][2]. It is a self-contained package to build C code into an extension on the fly, making it possible to wrap C code into cached chunks which compile into a Tcl extension. This allows the usage of embedded C code like callable Tcl procedures.

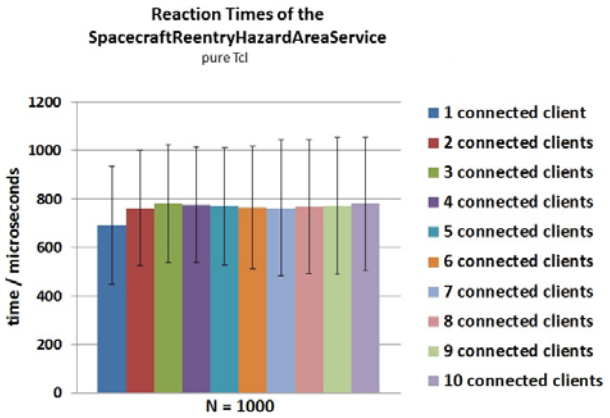

Figure 1. Pure Tcl performance

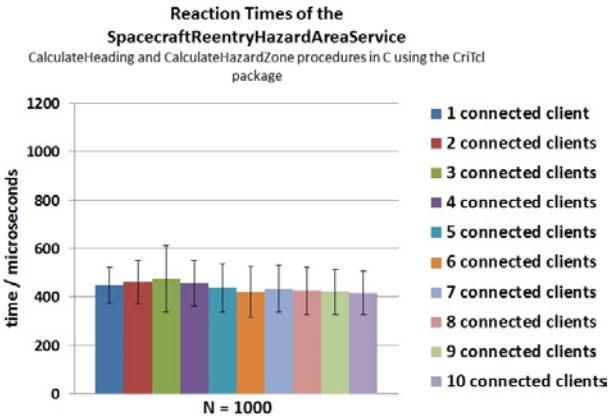

Figure 2. Performance with CriTcl

The principle is exemplarily shown in Fig. 3 to Fig. 5 by means of the CalculateHeading procedure. An equivalent in C only needs integration in the Tcl source code and can be easily switched to this alternative by commenting out the invocation of the original Tcl procedure. While the C function compiling needs a few seconds when running the first time, starting without code modifications does not need that time, just loading of the corresponding shared library, which was generated and cached during the compilation process, is requested. An also cached MD5 checksum of the source code is used to identify if recompiling is needed, which results in an almost instantaneous start behaviour of subsequent runs.

```
# ------------------------------------------------------------------------------
# CalculateHeading --
#     Calculates heading based on two points lat and lon.
#
# Arguments:
#     lat and lon of the two points
#
# Returns:
#     calculated heading
proc CalculateHeading {lat1in lon1in lat2in lon2in} {
    variable Pi
    set LocalOperator1 [expr {cos(($lat2in*$Pi)/180) * sin((($lon2in*$Pi)/180) - (($lon1in*$Pi)/180))}]
    set LocalOperator2 [expr {cos(($lat1in*$Pi)/180) * sin(($lat2in*$Pi)/180) - sin(($lat1in*$Pi)/180) \
            * cos(($lat2in*$Pi)/180) * cos((($lon2in*$Pi)/180) - (($lon1in*$Pi)/180))}]
    set LocalHeading [expr {atan2($LocalOperator1, $LocalOperator2) * (180 / $Pi)}]
    if {$LocalHeading < 0} {
        set LocalHeadingBuffer $LocalHeading
        set LocalHeading [expr { 360.0 + $LocalHeadingBuffer }]
    }
    return $LocalHeading
}
```

Figure 3. CalculateHeading procedure in Tcl

```
critcl::cproc c_calcheading {double lat1in double lon1in double lat2in double lon2in} double {
    /* this is C code */
    double localoperator1;
    double localoperator2;
    double localheading;
    localoperator1 = cos((lat2in * pi) / 180) * sin(((lon2in * pi) / 180) -  ((lon1in * pi)/180));
    localoperator2 = cos((lat1in*pi)/180) * sin((lat2in*pi)/180) - sin((lat1in*pi)/180) * \
          cos((lat2in*pi)/180) * cos(((lon2in*pi)/180) - ((lon1in*pi)/180));
    localheading = atan2(localoperator1, localoperator2) * (180 / pi);
    if (localheading < 0)
        localheading += 360.0;
    return localheading;
}
```

Figure 4. CalculateHeading procedure in C using CriTcl

```
# set CurrentHeading [CalculateHeading $HistoryLat $HistoryLon $CurrentLat $CurrentLon]
set CurrentHeading [c_calcheading $HistoryLat $HistoryLon $CurrentLat $CurrentLon]
```

Figure 5. Simple switching from Tcl to C in the Tcl main file

Although the Simplified Wrapper and Interface Generator (SWIG), driven by its high level of automation, was taken into consideration as a tool to glue Tcl and C [3], CriTcl was chosen, because it is a loadable package for Tcl itself, thus fitting better in the Tcl extension structure and making the whole solution easier to deploy.

2.2. Data Representation

The paper proposes vehicle specific hazard area information to be stored and managed as a representation of data with the extensible markup language (XML) according to Fig. 6 and Fig. 7. Although this representation has a data-centric character, which is usually associated with XML-enabled databases [4], a native XML database (NXD) called BaseX[1], using XML documents as the fundamental unit of storage, was chosen because of the following reasons:

- Planned service output enrichment of the SpacecraftReentryHazardAreaServer might turn the data representation to a document-centric character, usually associated with NXD [4].
- It provides handling of multi user write and simultaneous read operations based on a client/server architecture with future usage potential towards database splits, where a server dedicated to a subset of clients will only store data related to these clients.
- It offers a wide range of interfaces.

```
<?xml version="1.0" encoding="UTF-8"?>
- <hazardareas>
  - <spacecraft status="test">
      <id>00001</id>
      <name>test</name>
    - <hazardarea>
        <footprintrefalt unit="FL" relative="agl">0</footprintrefalt>
      - <altitude unit="FL" relative="msl" value="0">
          <arealength unit="km">0</arealength>
          <areawidth unit="km">0</areawidth>
        </altitude>
      - <altitude unit="FL" relative="msl" value="2000">
          <arealength unit="km">200</arealength>
          <areawidth unit="km">25.0</areawidth>
        </altitude>
      - <altitude unit="FL" relative="msl" value="2050">
          <arealength unit="km">205</arealength>
          <areawidth unit="km">25.6</areawidth>
        </altitude>
      - <altitude unit="FL" relative="msl" value="2100">
          <arealength unit="km">210</arealength>
          <areawidth unit="km">26.2</areawidth>
        </altitude>
      - <altitude unit="FL" relative="msl" value="2160">
          <arealength unit="km">216</arealength>
          <areawidth unit="km">27.0</areawidth>
        </altitude>
      - <altitude unit="FL" relative="msl" value="2210">
          <arealength unit="km">221</arealength>
          <areawidth unit="km">27.6</areawidth>
        </altitude>
      - <altitude unit="FL" relative="msl" value="2270">
          <arealength unit="km">227</arealength>
          <areawidth unit="km">28.4</areawidth>
        </altitude>
      - <altitude unit="FL" relative="msl" value="2310">
          <arealength unit="km">231</arealength>
          <areawidth unit="km">28.9</areawidth>
        </altitude>
      </hazardarea>
    </spacecraft>
</hazardareas>
```

Figure 6. Hazard area description example in xml

```
<?xml version="1.0" encoding="UTF-8"?>
<hazardareas>
    <spacecraft status="test">
        <id>00001</id>
        <name>test</name>
        <hazardarea>
        </hazardarea>
    </spacecraft>
    <spacecraft status="test">
        <id>00002</id>
        <name>test2</name>
        <hazardarea>
        </hazardarea>
    </spacecraft>
</hazardareas>
```

Figure 7. Hazard areas' representation of two spacecrafts

A first database connection test setup was successfully realised, benefiting from the "BaseXClient-Tcl" package of Danilo Chang[2], which uses the BaseX server protocol to communicate with the database server.

[1] www.basex.org

[2] https://github.com/ray2501/BaseXClient-Tcl

2.3. Flexibility For The Future

Scalability

Although the prototype's single system performance is sufficient for most small and medium sized cases (Fig. 8), considerations were taken how to make the solution scale better. A first version of a SWIM asset connector proxy was developed that works on top of the prototype solution servers (Fig. 9).

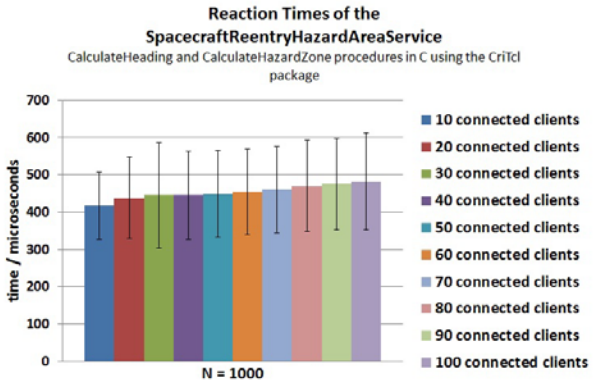

Figure 8. Performance with CriTcl for larger numbers of connected clients

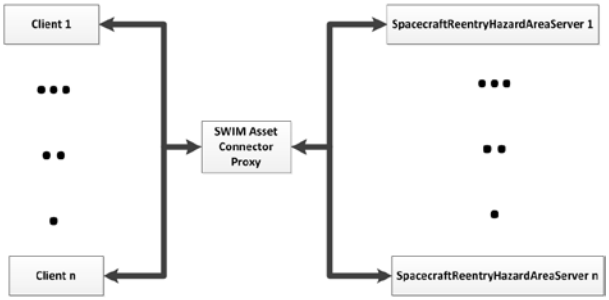

Figure 9. Proxy application that works on top of servers

A single interface is exposed and points clients to the appropriate servers by sending the requests to target servers and passing the results back to the callers. The realisation benefits from a remote communication facility application programming interface (API) for Tcl called "comm", part of the standard collection of utility modules for Tcl called "Tcllib". This API provides an inter-interpreter remote execution facility with control over the remote execution path. If needed, the SWIM asset connector proxy is able to remotely start a new server instance as well as to automatically actualise a list of channel connections. Asynchronous result generation is implemented by the technique of having a remotely invoked command indicate that it will not deliver an immediate, synchronous result. Thus the proxy can continue processing further requests with no blocking and no nesting of event loops. The code realisation consists of the two procedures shown in Fig. 10. The forward procedure takes the next target server as the first argument and the other arguments are processed as commands to be passed to that target server, setting up the asynchronous return command to be used in the forwardReturner procedure, where the result is just passed back to the caller of the forward procedure.

```
proc forward {target args} {
   comm::comm send -command [list forwardReturner [comm::comm return_async]] \
         $target {*}$args
}

proc forwardReturner {todo args} {
   $todo return -code [dict get $args -code] \
         [dict get $args -result]
}
```

Figure 10. Realisation of asynchronous operations invocation

Mixed language programming

The way for flexible extensions and enhancements in the future is paved by the fact that the prototype solution bases on a solid, mature and evolving cross-platform fundament for mixed language programming additions because of the following reasons:

- Tcl's cross-platform high-level API permits written code working on a wide range of Unix / Linux platforms, Macintosh and Windows.
- According to its roots in 1988 (version 1.0) and the fact that new features are still being added under active development (latest release: version 8.6.6 as of July 27th, 2016), Tcl/Tk benefits from both, being mature and evolving.
- Interfacing with other programming languages is possible by taking advantage of the fact that Tcl interfaces natively with the C language on the one hand and has associated bridging extension modules on the other hand, e.g. loading a Java interpreter into an existing Tcl process for using functionality implemented in Java [5].

3. OUTLOOK

Currently, the SpacecraftReentryHazardAreaServer solution consumes the aircraft state data within the Flight Object FIXM message and publishes the hazard area as temporary flight restriction (TFR) airspace output using AIXM version 5.1 for consumption and display by other interested parties. Enrichment of this output is planned in terms of delivering air traffic information about flights that will interfere with the hazard area and might need special attention. Realisation is foreseen to benefit from using the FlightXML 2.0 API for fetching relevant information from Flight Aware's hyperfeed [6].

4. SUMMARY

The paper presents the improved performance by the usage of a C code runtime embedding module of the SpacecraftReentryHazardAreaServer's new version. A data representation proposal of vehicle specific hazard area information based on a native XML database is introduced and justified. Flexibility for the future is addressed by a SWIM asset connector proxy realisation working on top of servers for facing possible future scalability needs. An outlook is given to enrich the service output with hazard area surrounding air traffic information.

9. REFERENCES

1. Landers, S. & Wippler, J-C. (2002). CriTcl - Beyond Stubs and Compilers. In Proc. 9th. Annual Tcl/Tk Conference, Tcl Community Association, Vancouver, Canada.

2. Kupries, A. (2016). C Runtime In Tcl. In Proc.23rd. Annual Tcl/Tk Conference, Tcl Community Association, Texas, USA.

3. Beazley, David M. (1998). Tcl and SWIG as a C/C++ Development Tool, http://www.swig.org/doc.html

4. Pavlovic-Lazetic, G. (2007). Native XML databases vs. relational databases in dealing with XML documents. Kragujevac Journal of Mathematics. 30(2007), 181-199.

5. B. Johnson, T. Pointdexter and D. Bodoh. 2011. JTcl and Swank: What's new with Tcl and Tk on the JVM. Proc. of 18th Annual Tcl/Tk Conference. (Oct. 2011), http://www.tclcommunityassociation.org/wub/proceedings/Proceedings-2011.html

6. Conn, Z. (2016). Hyperfeed: FlightAware's parallel flight tracking engine. In Proc.23rd. Annual Tcl/Tk Conference, Tcl Community Association, Texas, USA.