The workshop on systems and architecture for computer assisted interventions (SACAI) aims to provide a forum for discussing open interfaces, interoperability, and standards for the development of computer assisted intervention systems.

**FINAL PROGRAM**

<table>
<thead>
<tr>
<th>Time</th>
<th>Session</th>
</tr>
</thead>
</table>
| 09:00 – 09:30 | Introduction and overview  
Peter Kazanzides, Johns Hopkins University, USA |
| 09:30 – 10:30 | Session 1:  
9:30-9:50 Burgert O., Treichel T., Gessat M, “The Role of DICOM for Modular Perioperative Surgical Assist Systems Design”, ICCAS, University of Leipzig, Germany  
10:10-10:30 Bohn S., Gessat M., Franke S., Voruganti A., Burgert O., “An integrated OR system based on open standards”, ICCAS, University of Leipzig, Germany |
| 10:30 – 11:00 | Break                                                                 |
| 11:00 – 12:30 | Industry Panel:  
11:00 - 11:15 Andinet Enquobahrie, Kitware, Inc.  
11:15 - 11:30 Simon DiMaio, Intuitive Surgical, Inc.  
11:30 - 11:45 Pratik Patel, BrainLAB AG  
11:45 - 12:00 Ahmad Kolahi, ClarionTechnology, Inc.  
12:00 - 12:15 Leslie Holton, Medtronic Navigation  
12:15 - 12:30 Luis F. Gutiérrez, Philips Healthcare |
<table>
<thead>
<tr>
<th>Time</th>
<th>Session 2:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>14:30-14:50 Chinzei K., Tokuda J., “Extension to openIGTLink: XML as message, error handling, logging, and one-to-multi relaying”, National Institute of Advanced Industrial Science and Technology (AIST), Japan</td>
</tr>
<tr>
<td></td>
<td>14:50-15:10 Joshi A., Scheinost D., Okuda H., Murphy I., Staib L., Papademetris X, “Unified framework for development, deployment and testing of image analysis algorithms”, Yale University, USA</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time</th>
<th>Session 3:</th>
</tr>
</thead>
<tbody>
<tr>
<td>15:30 – 16:00</td>
<td>Break</td>
</tr>
<tr>
<td>16:00 – 17:20</td>
<td>16:00-16:20 Birngruber E., Donner R., Langs G., “MatVTK - 3D Visualization for Matlab”, Medical University of Vienna, Austria</td>
</tr>
<tr>
<td></td>
<td>16:20-16:40 Pace D., Gobbi D., Wedlake C., Gumprecht J., Boisvert J., Tokuda J., Hata N., Peters T., “An open-source real-time ultrasound reconstruction system for four-dimensional imaging of moving organs”, Robarts Research Inst., Univ. of Western Ontario, Atamai, Natl. Research Council, Canada, and Brigham &amp; Women's Hospital, USA</td>
</tr>
<tr>
<td></td>
<td>16:40-17:00 Lassó A., Tokuda J., Vikal S., Tempany C.M., Hata N., Fichtinger G, “Implementation of a generic computer assisted intervention plug-in module for 3D Slicer with multiple device support”, Queen's University, Canada, and Brigham &amp; Women's Hospital, USA</td>
</tr>
<tr>
<td></td>
<td>17:00-17:20 Yamada A., Nishibori K., Hayashi Y., Chinzei K., Fujimoto H., “3D Slicer Based Surgical Robot Console System”, National Institute of Advanced Industrial Science and Technology (AIST), Japan</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time</th>
<th>Closing Session: Discussion and Wrap-up</th>
</tr>
</thead>
<tbody>
<tr>
<td>17:20 – 17:45</td>
<td></td>
</tr>
</tbody>
</table>
Introduction to DICOM

Motivation for using DICOM as standard for Surgical Assist Systems

Extending DICOM

DICOM Supplements relevant for modular perioperative Surgical Assist Systems

DICOM Working Group 24 “Surgery”

DICOM Supplements prepared by WG-24

Future enhancements of DICOM

Conclusion
1 Introduction to DICOM

Digital Imaging and Communications in Medicine (DICOM) is known as the major imaging and image communication standard in radiology. In 1985, the first version of the standard was published as ACR-NEMA Standards Publication No. 300-1985. Prior to this, most devices stored images in a proprietary format and transferred files of these proprietary formats over a network or on removable media in order to perform image communication – a situation which is comparable the situation in today’s pre- and intra-operative systems design. In 1993, the “Version 3.0” was released, to reflect the major changes the name of the standard was changed to Digital Imaging and Communications in Medicine (DICOM).

The DICOM Standard specifies a network protocol based on TCP/IP, so called “Service Classes” which allow for higher level communication like image storage of query, and Information Objects which are uniquely identified. The information objects cover not only images but also patients, studies, reports, and other data groupings. The DICOM standard is freely available as multi-part document at [http://dicom.nema.org](http://dicom.nema.org).

The major application field of DICOM was radiology, but the standard’s scope has been broadened within the last years to cover more use cases and application fields, like radiation therapy or pathology. It now aims on achieving compatibility and to improve workflow efficiency between imaging systems and other information systems in healthcare environments worldwide. A more detailed introduction and brief overview about the working principles of the DICOM Standard can be found in [1].

2 Motivation for using DICOM as standard for Surgical Assist Systems

Most surgical planning systems and many systems used intraoperatively are making use of radiology images. Those images are usually stored in a PACS (Picture Archiving and Communication System) in the DICOM format. In many applications it is desirable to store additional derived patient data, e.g. results of a surgical planning step, in the same location to facilitate integration of all data sources. In addition, if the PACS infrastructure allows for storage of those results, one can make use of the long term storage mechanisms already implemented in most PACS solutions – this solves the problem of maintaining different proprietary data formats for a long time. More use cases and rationales are given in a White Paper prepared by DICOM WG-24 “Surgery” [2].

Nonetheless, DICOM will not be the only standard used in the operating room (OR) of the future. It has no real time features; therefore it is not suitable for control tasks like control of a surgical instrument or robot. The scope of DICOM does not cover anesthesia with all its devices and it also does not support streaming data like live video transmission. Other standards will therefore complement the complete OR infrastructure of the future.
3 Extending DICOM

One common misunderstanding about DICOM is, that one can “implement version 3.0 which is the current one”. DICOM version 3 is constantly updated by supplements and modified by change proposals. Therefore, the annual version of DICOM (currently DICOM 2008) including the approved supplements finalized in 2009 is “the current version” of DICOM. There is no such thing like a DICOM 3.1 or 4.0. Extensions and modifications of DICOM are built in a sense that existing DICOM objects will remain valid and existing applications and devices will need no or just minor modifications to include the new features.

All systems claiming DICOM conformance must state in a so called “Conformance Statement”, which Service Classes and Information Objects they support. Therefore, it is not necessary to “implement DICOM” as a whole, one can focus on the things needed for a specific application. If new features are introduced by a supplement, each vendor can freely decide, whether those features are included in a product or not.

“Correction Proposals” (CP) aim on correcting minor errors in the existing text or reflect minor changes. If major additions, like new Information Objects or Service Classes, are introduced, a so called “Supplement” (Sup.) is prepared by one or more DICOM Working Groups (WG). To create a Supplement, the WG has to apply for a “Work Item” which describes the use cases and scope of the work. A comprehensive list of all Supplements and CPs is given by David A. Clunie at http://www.dclunie.com/dicom-status/status.html. All DICOM extensions and modifications must pass WG-06 (Base Standard) to ensure consistency of the overall DCIOM standard. Each supplement usually undergoes a “First Read”, optionally a “Second Read”, a preparation for the “Public Comment”-phase and a preparation for “Final Text”.

A DICOM Working Group consists of interested individuals representing vendors, manufacturers and end users of the respective domain. Everyone interested can participate in preparation of DICOM documents by joining a WG. From our point of view, the most relevant Working Groups for the field of Surgery are WG-02 and 12, both dealing with intraoperatively used imaging devices, 17 for storage of endoscope and microscope movies, 17 for image processing and modeling, 22 for dental and maxillofacial applications and 24 covering “surgery” as a whole. Two WGs play a special role: WG-10 considers issues and opportunities related to the strategic evolution of DICOM, WG-06 maintains the overall consistency of the DICOM standard. A complete list of WGs is given in Table 1.

<table>
<thead>
<tr>
<th>WG-01: Cardiac and Vascular Information</th>
<th>WG-15: Digital Mammography and CAD</th>
</tr>
</thead>
<tbody>
<tr>
<td>WG-02: Projection Radiography and Angiography</td>
<td>WG-16: Magnetic Resonance</td>
</tr>
<tr>
<td>WG-03: Nuclear Medicine</td>
<td>WG-17: 3D</td>
</tr>
<tr>
<td>WG-04: Compression</td>
<td>WG-18: Clinical Trials and Education</td>
</tr>
<tr>
<td>WG-05: Exchange Media</td>
<td>WG-19: Dermatologic Standards</td>
</tr>
<tr>
<td>WG-06: Base Standard</td>
<td>WG-20: Integration of Imaging and Information Systems</td>
</tr>
<tr>
<td>WG-07: Radiotherapy</td>
<td>WG-21: Computed Tomography</td>
</tr>
<tr>
<td>WG-08: Structured Reporting</td>
<td>WG-22: Dentistry</td>
</tr>
</tbody>
</table>

Latest version available at the Insight Journal link http://hdl.handle.net/1926/1338
Distributed under Creative Commons Attribution License
4 DICOM Supplements relevant for modular perioperative Surgical Assist Systems

There are several DICOM Supplements which have been introduced within the last years which are having relevance for the field of Computer Assisted Surgery. The following table gives a short overview, supplements relevant just for one clinical speciality are omitted:

<table>
<thead>
<tr>
<th>Supplement</th>
<th>Year</th>
<th>Relevance</th>
</tr>
</thead>
<tbody>
<tr>
<td>47: Visible Light Video</td>
<td>2003</td>
<td>Storage of video files</td>
</tr>
<tr>
<td>67: Configuration Management</td>
<td>2003</td>
<td>Self-configuration of devices</td>
</tr>
<tr>
<td>73: Spatial Registration</td>
<td>2003</td>
<td>Introduction of fiducials and registration using affine transformations</td>
</tr>
<tr>
<td>85: Web Access to DICOM Objects (WADO)</td>
<td>2003</td>
<td>In case you don’t get native PACS access…</td>
</tr>
<tr>
<td>111: Segmentation</td>
<td>2006</td>
<td>Store segmentation results as labelled images</td>
</tr>
<tr>
<td>112: Deformable Spatial Registration</td>
<td>2006</td>
<td>Adds deformation fields to registration</td>
</tr>
<tr>
<td>116: 3D X-Ray</td>
<td>2007</td>
<td>Intraoperative imaging</td>
</tr>
<tr>
<td>43: Storage of 3D Ultrasound Images</td>
<td>2008</td>
<td>Intraoperative image acquisition, in device or patient coordinate system.</td>
</tr>
<tr>
<td>123: Structured Display</td>
<td>2008</td>
<td>Screen layout</td>
</tr>
<tr>
<td>132: Surface Segmentation</td>
<td>2008</td>
<td>Introduces surface meshes to DICOM end extends the formerly image based Segmentation SOP class.</td>
</tr>
<tr>
<td>96: Unified Worklist and Procedure Step</td>
<td>Frozen</td>
<td>Worklist and Workflow Management</td>
</tr>
<tr>
<td>118: Application Hosting</td>
<td>Comment</td>
<td>Plug-In mechanism for PACS software</td>
</tr>
<tr>
<td>131: Bone Mounted Implants</td>
<td>Comment</td>
<td>Introduces 2D and 3D Implant Templates for Surgical Planning</td>
</tr>
<tr>
<td>134: Implantation Plan</td>
<td>Comment</td>
<td>Storage of Surgical Plans based on Supp 131</td>
</tr>
<tr>
<td>138: Crestal Implant</td>
<td>Cancelled</td>
<td>Use cases covered by Supp. 131</td>
</tr>
</tbody>
</table>

Table 2 Supplements relevant for Surgical Assist Systems. “Year” indicates the year the Supplement is introduced. “Comment” means, it’s in “Public Comment”-Phase, “Frozen” indicates a “Frozen Draft” which shall be tested by reference implementations before it is introduced in the standard.

---

1 At time of publication of this article. Actual status: “work”.

Latest version available at the Insight Journal link [http://hdl.handle.net/1926/1338](http://hdl.handle.net/1926/1338)
Distributed under Creative Commons Attribution License
5 DICOM Working Group 24 “Surgery”

In 2005, the DICOM Working Group 24 (WG-24) was installed with the goal to extend the DICOM standard towards applications in surgery. In contrast to other WGs, which are usually focused on one particular modality or application, WG-24 covers a broad range of applications and modalities. Therefore, it is organized in Clinical Project Groups which are responsible for the proper definition of clinical use cases and for ensuring the clinical relevance and applicability of the Supplements prepared by WG24. In addition, there Project Groups which are working on specific Supplements and Work Items. Those technical Project Groups are meeting based on their own schedule and report to the whole WG in biannual meetings. If a group or individual is interested in participating in the work of WG-24, they can contact the WG-24 chair or the respective PG-chair. DICOM WG-24 has currently over 100 members from industry, medical practice and academia.

6 DICOM Supplements prepared by WG-24

WG-24 has been strongly involved in the development of DICOM Supplement 132 “Surface Segmentation Storage SOP Class”. It introduces the Surface Segmentation IOD (Information Object Definition) which describes segmented surfaces in patients. The main contribution of this supplement is a module for the representation of surface meshes, i.e. piecewise bilinear approximations of surfaces. There are several use cases for surface meshes in CAS systems which go beyond the representation of segmentation results, as we discussed in [3]. Thus, the surface module which is specified within Supplement 132 was specified in way that facilitates re-use of this module in other IODs developed for other use cases.

The surface module contains mechanisms to describe a set of points \( P \) in 3D space, a set of vectors which represent the surface normal in each point, and one or more surfaces in the form of polygonal meshes with the points \( P \) as vertices. The surfaces are constructed from surface mesh primitives, such as triangles or planar polygons (facets).

While supplement 132, which was developed under the leadership of Working Group 17 (3D) is not aiming at a surgical use case per se, Supplements 131 “Implant Templates” and 134 “Implantation Plan SR Document” focus on a workflow which is definitely related to surgery: preoperative implantation planning.

Originally, implantation planning was performed using 2D drawings of implants printed on translucent slides which were manually overlaid to X-ray images. Such techniques were widely used, e.g. for the preoperative selection of the best-fitting implant for partial or total hip joint replacement. Along with adoption of digital acquisition and display of X-ray images, planning software which uses digital templates instead of the printed drawings emerged. Once detached from the plastic foils, implantation planning systems began to use 3D images and 3D templates where 2D information was insufficient, e.g. in dental surgery.

The availability of digital templates depicting the implant is a prerequisite for an implantation planning software to generate plans which make use of an implant. With no standardized format for digital
templates available, the extension of planning software or the communication of plans between different systems is difficult and error-prone. Supplement 131 specifies the generic implant templates IOD which is intended for the description of digital templates. The IOD allows the description of 2D and 3D templates, and includes mechanisms to describe how multi-component implants can be assembled and how anatomical landmarks can be used to find the optimal position of the implants.

Supplement 134 specifies a data structure for the storage and exchange of planning results. The implantation planning SR document is an instance which references all data which was used for or created during the procedure of planning an implantation for one patient. Figure 1 shows the structure of this document. The aim of the implantation plan defined by this Supplement is to provide vendor-independent information about the most important planning steps and results. It is not meant to replace the file handling and storage of a specific planning software since there might be several software and vendor specific information which shall not be stored in a public file format. Supplement 134 is based on the DICOM Structured Report (SR) document which allows to parse a tree-like structure of elements.

**Figure 1** Structure of the Implantation Planning SR Document
7 Future enhancements of DICOM

WG 24 will continue to pursue the standardization of data exchange in computer assisted surgery. Thereby, one task will be to identify the borders of a meaningful extension of DICOM. There will be use cases which technically can not be solved with DICOM (e.g., DICOM is not suitable when it comes to real-time communication). In addition, there might be others which could technically be solved with DICOM but appear not to be in the scope of DICOM since they are covered other standards such as HL7.

The strength of DICOM is its affinity to PACS, which is the infrastructure for the exchange and long term archival of images and image-related information. The authors foresee that DICOM will very likely be used to specify data exchange in use-cases where the primary requirements are: preoperative planning of interventions, intraoperative handling of patient models, and postoperative storage of information for surgery reports.

For the near future, WG 24 is planning to address the following subjects:

- **Patient Specific Implants**: Based on the Information Objects and Services introduced by Supplement 131 and 134, implantation procedures were patient specific implants are planned, crafted and implanted shall be supported. An IOD is required, which combines some of the attributes from the Generic Implant Template IOD from Supp. 131 with the Patient Module, which is used in DICOM to refer an instance to one patient.

- **Scanned Surfaces**: Supplement 132 covers segmented surfaces, i.e. surface meshes which are created as the boundaries of regions in three-dimensional images. Yet, there are other ways to generate surface-data of patients, like surface scanners which use laser or structured light to acquire surface information. WG-24, jointly with other interested WGs like WG-22 “Dentistry”, is preparing a work item description for the use case of patient surfaces which are acquired by such devices.

- **Parameters on Surfaces**: An extension of the surface module is planned which allows attaching parameter vectors other than a normal vector to the points of a surface. Use cases for such a data structure are the representation of simulation results and the distribution of measurements which were acquired at different locations on a surface.

- **Coordinate Systems**: There are several coordinate systems used by different modalities, but no unified and commonly accepted one which shall be used to integrate the different information about the patient. Especially intraoperative images and patient position can not be treated in one coordinate system. The aim of this work is to either propose a mechanism to link the different coordinate systems, e.g. by registrations, or to establish a new world coordinate system which can be used during all stages of therapy as a reference.
8 Conclusion

DICOM has the potential to play a major role as interoperability standard for surgical planning and intraoperative data exchange. It can be enhanced to meet the needs of many Surgical Assist Systems with moderate effort, e.g. it took roughly two years to introduce Surface Meshes. The benefits of a common standard in this field will be manifold: cost reduction, increased patient safety due to clearly defined interfaces, re-use of existing applications or integrated patient data presentation and handling. We strongly encourage the Computer-Assisted-Therapy-Community to participate in the standardization process and to implement the resulting Standard.

Acknowledgements

The authors wish to thank the chair Heinz U. Lemke all members of DICOM Working Group 24, especially those who contributed to the working group's white paper.

This work is sponsored by funds of the European Re-gional Development Fund (ERDF) and the state of Saxony within the framework of measures supporting the technology sector. ICCAS is funded by the German Federal Ministry of Education and Research (BMBF) and the Saxon Ministry of Science and Fine Arts (SMWK) within the scope of the "Unternehmen Region" initiative with grant numbers 03 ZIK 031 and 03 ZIK 032.

Reference


A role-based component architecture for computer assisted interventions: illustration for electromagnetic tracking and robotized motion rejection in flexible endoscopy.

Release 2.00

Jean-Baptiste Fasquel¹, Guillaume Chabre¹, Philippe Zanne², Stéphane Nicolau¹, Vincent Agnus¹, Luc Soler¹, Michel de Mathelin² and Jacques Marescaux¹

July 3, 2009

¹Computer Science Research and Development Department, IRCAD, France
²University of Strasbourg, France

Abstract

This paper presents an original role-based software architecture facilitating the flexible composition, configuration and collaboration of separated components in the field of computer assisted interventions. Roles, which can be seen as methods dynamically attached to objects, are embedded in components, to limit build level dependencies and improve flexibility. An appropriate component definition and composition language is proposed to declare softwares, without any specific initialization or glue code, this remaining a challenging issue in component oriented programming. The potential of this architecture is illustrated for a software coupling electromagnetic tracking with a robotized system dedicated to the physiological motion rejection in flexible endoscopy. This software consists in several independent components with are combined at runtime thanks to a concise XML-based declaration.

Contents

1 Introduction 1
2 Architecture 2
3 Illustration 6
4 Discussion 7

1 Introduction

This paper faces the design of a framework for the rapid prototyping of softwares in computer assisted interventions, this being currently an active area of interest [3], with existing open source IGT architectures...
such as CISST\cite{7}, Slicer IGT\cite{6} or IGSTK\cite{4}.

The main originality of the proposed C++ cross-platform (Windows, Linux, Mac Os X) architecture relies in the coupling of a component approach with the notion of role-based programming \cite{8} in this application. Roles define the composition by dynamically associating different tasks to the same conceptual entity although they are defined in separated code elements, similarly to dynamic inheritance. Such a mechanism is not supported by traditional object oriented languages and in particular by the C++ language traditionally used in our application domain for runtime performances reasons. Role orientation facilitates component collaboration as component instances share a common data support (the base object). Component orientation aims at removing, thanks to role abstraction, build level dependencies \cite{9}.

On top of this approach, we propose a concise component definition and composition language \cite{9} facilitating the definition of applications independently from the implementation language, without any specific glue code and therefore with a high degree of flexibility, this being challenging \cite{9}.

Compared to our previous work \cite{5}, this paper presents two main improvements. In terms of architecture, we simplified both class distributions (e.g. merging of both ExtPtService and ExtService) and the XML-based description formalism. Moreover, specific initialization or glue code is not required anymore, allowing to entirely define applications with a pure XML-based declaration. In terms of functionality, compared to \cite{5}, the application field is extended to robotized system management combined with an Aurora-based electromagnetic tracking \cite{1}.

2 Architecture

As traditionally considered in component oriented programming, the proposed framework is based on a strong separation between data (base object, directly inheriting from the ::layer::Object in fig. 1, being data container restricted to getters and setters) and functionalities (services, being implementations of the ::layer::IService interface in fig. 1).

![Simplified UML class diagram of the architecture](image)

Figure 1: Simplified UML class diagram of the architecture (the layer package being the core of the architecture). The io package defines a functionality type (IReader), while aurora package (only partially described for clarity) represents an implementation package (a specific base object (Tube) and a specific service (Reader)). The anubis content is not detailed for clarity.

Services (roles) can be dynamically associated to a base object, using, at code level, a generic invocation
such as: `::layer::add(myObject,"::io::IReader","::aurora::Reader")`, where `myObject` is an instance of `::aurora::Tube`, and will play the `::io::IReader` role type. Note that the notion of service type facilitates both code factorization and object classification (useful to perform some tasks according to the type, independently from effective low-level implementations). Service state (attributes’ value) is defined using an XML like structure passed as parameter (`IService::setConfiguration(cfg:XML)` and `IService::configure()` methods in fig. 1).

```xml
<plugin id="aurora">
  <library name="libAurora"/>
  <point id="::aurora::Reader" schema="AuroraReader.xsd">
    <implements>::io::IReader</implements>
    <implements>::aurora::Tube</implements>
  </point>
</plugin>
```

Figure 2: Example of component description file content: `point` declares, with a uniform formalism, that `::aurora::Reader` inherits from `::io::IReader` and can play a reading role for objects of type `::aurora::Tube`.

Figure 3: The complexity of the component distribution (top) is hidden by the XML-based graph provided by their descriptions (middle). Applications (bottom) can be seen as new types (`newType`, see figure 4 for full declaration), possibly declared in separated components (e.g. `myApp`), consisting in particular compositions and configurations of base objects and services. Note that `myApp` can be restricted to a description file only (i.e. no dynamic library). All together forms a uniform object-oriented like graph integrating both definitions and new declarations. Thanks to the `implements` XML statement, `Reader` seems to inherit from `Tube`, although it concerns role declaration only.

At initialization (`IService::start()` method in fig. 1), a service can dynamically affect shared attributes (optional attributes [5]) to its base object (natively restricted to hard-coded intrinsic attributes [8]).

At runtime, depending on the functionality, a service can perform computations using its own attributes, intrinsic attributes of its base object, or even a combination of all. If the service execution depends on attributes of another service, it is preferable (inter-service independency) that this other service exports the required attributes to its base object, as shared attributes. Therefore the union of the base object’s intrinsic and shared attributes represent the data support for service execution (i.e. `IService::update()` method in fig. 1). Service execution ordering (chaining) can be managed in a generic manner using a list of identifiers.
(explicit chaining), as each service can be retrieved from a unique identifier (e.g. invocation looking like ::layer::get(myObject,serviceUID)->update()). Chaining can also be implicit (implicit chaining) using the event-based collaboration mechanism previously described [5] (communication entity in fig. 1).

Thanks to the abstraction provided by the layer (e.g. add, get methods and IService API), we can avoid build level inter-service dependencies by embedding services in components [5]. A component is defined by a XML description file (e.g. see figure 2), (optionally) coupled with a dynamic library and resources (e.g. icons,...), being bundled into a single entity (e.g. a directory of a filesystem). The dynamic library embeds the implementation of a (set of) service(s) defined in the XML description file. When requesting an attachment (e.g. ::layer::add(myObject,"::io::IReader","::aurora::Reader")), the component management system (wrapped in the layer package) analyzes the description files of the component distribution, loads the appropriate dynamic library (e.g. libAurora) to instantiate the appropriate service and finally to store the resulting association in the ::layer::ObjectRegistry (singleton storing all instantiated base objects and attached services). The formalism used to describe components (component definition language) leads to a graph (see figure 3) similar to an inheritance hierarchy (where each class is declared with a XML element named point). Such a graph-based representation appears more concise and structured than previously [5], and hides the complexity and the specificity of the component distribution (as illustrated by figure 3).

```
<extension id="newType" implements="::aurora::Tube" >
  <object uid="root" type="::aurora::Tube">
    <service uid="reader" type="::io::IReader" implementation="::aurora::Reader">
      <location>...</location>
    </service>
    <service uid="render" type="::render::IRender" implementation="::aurora::Render">
      <window id="900"/>
    </service>
    <service uid="tracker" type="::tracker::ITracker" implementation="::aurora::Tracker" />
    <service type="::gui::IAspect" implementation="defaultAspect">
      <windows>
        <window id="900"/>
      </windows>
    </service>
    <start type="::gui::IAspect" />
    <start uid="tracker" />
    <start uid="reader" />
    <update uid="reader" />
    <stop uid="tracker" />
    <stop type="::gui::IAspect" />
  </object>
</extension>
```

Figure 4: XML-based software declaration: Aurora-based electromagnetic tracking (tracker), including 3D rendering (render) in a specific view (corresponds to the left side of the snap shot given in fig. 7). The ::aurora::Tube base object is a set of points, their position being modified by the tracker, involving 3D rendering refresh (implicit chaining).

On top of this, new types (i.e. base objects with specific service attachments to be performed at instanciation time) can be defined using a dedicated XML declaration (composition language), as illustrated in figure 4. The root XML element extension enables to integrate this declaration into the graph resulting from component definition, leading to a uniformization of both definitions and compositions. Note that optional
XML elements can specify services to start (start statement) and execute (update statement) when ending instanciation, and those to stop (stop statement) when destroying the object (taking the declaration order into account). Compared to our previous work [5], such a declaration is used to define a complete software (evaluated for the VR-Render freeware [2]), using a generic launcher (the root object type, e.g. newType, being a parameter), independent from any component (dependency limited to the layer entity) and without glue code. Note that service existence and attachment compliance are checked through a graph analysis. Services’ state (children of service-named XML elements) can be checked using XSD schemas provided as component resource files and specified XML descriptions (e.g. schema attribute value of point XML element in figure 2).

```
<extension id="illustration" implements="::layer::Object">
  <object uid="root" type="::layer::Object">
    <object uid="aurora" type="::aurora::Tube">
      <service uid="reader1" type="::io::IReader" implementation="::aurora::Reader">
        <location>xxx</location>
      </service>
      <service uid="render" type="::render::IRender" implementation="::aurora::Render">
        <window id="900"/>
      </service>
      <service uid="tracker" type="::tracker::ITracker" implementation="::aurora::Tracker"/>
    </object>
    <object uid="anubis" type="::anubis::Robot">
      <service uid="reader2" type="::io::IReader" implementation="::anubis::Reader"/>
      <service uid="motionFilter" type="::layer::IControler" implementation="::anubis::Motion">
        <uid>s1</uid>
        <uid>s2</uid>
        <uid>s3</uid>
      </service>
      <service uid="s1" type="::tracker::ITracker" implementation="::anubis::Tracker"/>
      <service uid="s2" type="::anubis::ICommand" implementation="::anubis::Command"/>
      <service uid="s3" type="::fwRender::IRender" implementation="::anubis::Image">
        <windowContainerId="902"/>
      </service>
      <service type="::gui::IAspect" implementation="defaultAspect">
        <windows>
          <window id="900"/>
          <window id="901"/>
          <window id="902"/>
        </windows>
      </service>
      <start type="::gui::IAspect"/>
      <stop type="::gui::IAspect"/>
    </object>
  </object>
</extension>
```

Figure 5: XML-based declaration of the application managing the robotized flexible endoscope (anubis) coupled with the tracking system (aurora).
3 Illustration

The XML declaration of the application considered in this paper is given by figure 5. Figure 6 (resp. 7) concerns the experimental setup (resp. the software). Both anubis and aurora objects are declared as independent elements composing the application root object. This root object has only one IAspect service in charge of configuring the overall layout of the GUI. The core of the previous newType declaration (figure 4) has been entirely reused without any code modification or specific glue code.

![Figure 6: Experimental setup related to the system managed by the application (snapshot in fig. 7).](image)

The aurora object focuses on the monitoring of flexible endoscope motion using an electromagnetic tracking system: a set of sensor coils are regularly placed on a catheter (::aurora::Tube in figure 5) which is introduced in a channel of the flexible endoscope. Associated services perform tracking as well as 3D rendering (both tracker and render services in figure 5) in a predefined view area of the application (right panel in fig. 7, corresponding to the window identifier 900 in the declaration). For this functionality, the native event-based mechanism is used to synchronize both tracking and rendering (similarly to the IGSTK approach [4] with notions of SpatialTransform and SpatialObject).

The anubis object is mainly dedicated to motion cancelation in flexible endoscopy, in the case of the robotized system described in [10]. This system is represented by a specific base object (of type ::anubis::robot), which mainly consists in data related to both motors and the image (video) being visualized by the head of the endoscope. A tracking service performs the visual tracking (s1 in fig. 5), and a command service (s2 in fig. 5) controls endoscope motors. GUI aspects have been implemented in separated services, preserving the control loop (s1 and s2) from the GUI specificity, regarding both the video rendering and the interactive definition of the target to be tracked (s3 in fig. 5). In addition to motion filtering, the user can control the orientation of the head of the flexible endoscope (ctrl in fig. 5). Due to the video frame rate, a complete control loop cycle must be shorter than 40 ms. For this reason, a control service (motionFilter in figure 5), triggered by video acquisition (reader2 in figure 5), manages both control loop and video rendering refresh (explicit chaining). As rendering is optional, the s3 service is executed only if enough time remains. Despite the proposed abstraction (well known to reduce performances due to indirections), it has been observed that runtime performances were compliant with requirements. In our sense, this is facilitated by the role orientation, as services collaborate through a direct access to their common base object content, preserving performances. Indirections mostly concern system initialization (e.g. service attachment and configuration) and observation based collaborations (which can be, as for the considered control loop, replaced by explicit chaining, depending on required reactivity).
4 Discussion

In our sense, the strength of the role orientation in the presented framework is that component composition is natively supported (common data support), as well as behavioral collaboration using the integrated observer design pattern (used in the illustration for electromagnetic tracking). Component abstraction and (build level) independencies facilitate application prototyping with a concise XML declaration without specific glue code or initialization code. Due to direct access to the base object, it seems that such an architecture, despite its abstraction, can be advantageously used for critical applications such as the one presented here. Besides, due to the well structured organization of base objects and services, including communications and, at service level, the state pattern, any application can be easily monitored at runtime. Such monitoring capability appears essential for debugging and could be advantageously used for managing safety, this being critical in softwares dedicated to computer assisted interventions.

References


An integrated OR system based on open standards

Release 1.10

Stefan Bohn, Michael Gessat, Stefan Franke, Arun Voruganti and Oliver Burgert

August 14, 2009

Innovation Center Computer Assisted Surgery (ICCAS), University of Leipzig, Germany

Abstract

An open standards based modular OR integration system was designed, which is modelled as component-based service-oriented architecture. Medical devices, IT hardware, and software are integrated as independent components and interconnected through a TCP/IP based Ethernet network. The integrated system facilitates service discovery, time synchronization, systems monitoring as well as component control and network administration. Each of the components exposes its semantic information about offered services as service description to peer components. The service description object is based on a generic model and contains all information about attributes, functions, streams as well as information on the used protocols that are required to access the services. An open standards based programming library for the rapid development of server/client components including example modules was developed. A prototype setup of the entire integrated OR system was realized within a demonstrator OR lab.

Contents

1 Introduction................................................................................................................................................... 1
2 Architecture....................................................................................................................................................... 2
3 Implementation................................................................................................................................................ 4
4 Results............................................................................................................................................................ 7
5 Conclusion...................................................................................................................................................... 8

1 Introduction

Today’s operating rooms (OR) accommodate a wide variety of medical device hardware, modern image acquisition and processing technologies, IT systems and systems for computer assisted surgery (CAS). Unlike in other industry domains, there exists no communication platform within the OR that enables interoperation among heterogeneous medical devices or IT components. The lack of a common OR
infrastructure and the large amount of information sources within the OR leads to unergonomic and uneconomic conditions as well as an impaired surgical workflow. Besides focusing on the surgical intervention the surgeon is required to mentally integrate all necessary information coming from various spatial distributed IT components. Furthermore, due to the absence of a common OR infrastructure there is lack of a consistent data flow between preoperative, intraoperative and postoperative phases. Data, which already exist in clinical information systems, e.g. hospital information systems or radiological information systems, need to be entered manually into different OR systems. Portable media are used to transfer data from preoperative planning to intraoperative applications as well as interoperatively acquired data to postoperative documentation. These limited state of the art approaches are highly susceptible to type errors or mixing of patient identification information and thus can cause undesirable data inconsistencies in clinical information systems and PACS archives.

All affected participants across the medical device industry, clinical end-users, and researchers recognized the benefits that would come from plug-and-play device interoperability within the OR as well as centralized user access to an integrated system. Several commercial vendors provide such integrated operating rooms (e.g. BrainLab, Storz, Stryker), which mainly integrate their own components using proprietary interfaces. Although a wide variety of communication protocols and standardized physical interfaces for systems intercommunications are available, none of them are established or commonly accepted in the medical device domain until now. There exists no generic communication framework that provides communication protocols for syntactic and semantic interoperability among medical devices or CAS systems.

The aim of our work is the design and implementation of an integrated OR infrastructure as research platform that is based on open standards and protocols. This should show the feasibility of vendor-independent system intercommunication within the medical device domain and especially in the operating room. Clinical users ask for an efficient and user friendly system that can be applied in different contexts depending on the surgical intervention. Therefore, we focus on the generic design of an autonomous and fault-tolerant infrastructure, which can assert safety, reliability, robustness and reconfigurability.

2 Architecture

The development of the OR integration concept started with an analysis of the required system components [1], surgical workflow analysis’s [2] and selected clinical use cases [3]. The design of our modular OR integration infrastructure follows the Therapy Imaging and Model Management System (TIMMS) meta-architecture, which was published by Lemke and Vannier in 2006 [4]. It introduces several engines, repositories, and communication channels, suitable for all kinds of computer assistance in surgery and therapy.

From this we derived a concrete modular integration architecture that consists of a system of distributed modules (hardware and software) as well as several system core components. The architecture of the integrated OR system is shown in figure 1. In order to achieve vendor-independent data exchange, the modules interact using a set of standard protocols for session management, data exchange, remote control, time synchronization as well as systems monitoring. Since the OR is dominated by an ad hoc context, the overall system needs to cope with the absence of any a-priori knowledge about possible peers and particular communication mechanisms.
Figure 1 The architecture of the integrated system with TIMMS modules, network infrastructure, core components, and end users.

2.1 TIMMS Modules

Modules that act on the patient during the intervention such as sensors, imaging modalities, or CAS systems consist of hardware and software of various kinds. Each of them strongly differs in the requirements regarding data exchange with peer modules. Online data sources require real-time communication of streaming data while modelling modules or image acquisition devices produce large data objects at irregular intervals. These need to be transported from one module to another resulting in heavy network load for a short period. At the same time status messages are exchanged in regular or irregular intervals, which also need to be transmitted. These applications bring along different requirements on all levels of the Open Systems Interconnection (OSI) reference model. To satisfy all requirements at the same time, the TIMMS backbone network infrastructure needs to be implemented using different technologies on all levels of the OSI reference model.

To integrate these various technologies into a homogeneous framework over the complete system operation cycle, each module is abstracted based on a generic device model [5]. The device model describes the functional capabilities of each module, its data, behaviour, and the used communication protocols in a generic manner. The device model is exported as service description object to peer modules during the initial negotiation phase of the intercommunication. Thus, using technologies of auto-configuration, TIMMS modules can seamlessly join the network, discover peer modules, connect and retrieve the service description, as well as learn and make use of the offered services to perform their particular tasks within the integrated system.
2.2 TIMMS Core Components

The integrated OR system contains several core components that provide the overall integrated functionality to the end-user such as clinicians as well as the technical administrator. Furthermore, the core components perform several managing functions such as maintaining the patient context, data exchange with clinical information systems, time synchronization, and logging.

The TIMMS Component Controller (TCC) is the central managing component that administrates and supervises all interconnected modules within the integration system. The TCC tracks the system health status of the entire networked modules, monitors the data flow on the network and ensures rights and access control for security and safety. Additionally, it provides a time server in order to keep modules that perform time critical tasks synchronized. The TCC models and controls the integrated system and allows different user groups to view and access the OR functionality from different perspectives at different workstations. On the one hand, the clinical user at the surgical cockpit gets a condensed view, visualization, and smart control of the OR functionality, where technical details are abstracted to an appropriate level. The administrator user on the other hand has an in-depth view and access to all parameters of the integrated TIMMS system and is able to adopt specific configuration issues on demand.

The integrated OR system is required to exchange data with different clinical information systems and repositories to enable a consistent data flow between preoperative, intraoperative, and postoperative phases. Therefore, a COM-Server maintains access to the IT world outside the OR, e.g. to retrieve patient information from the hospital information system, worklists from PACS archives as well as to store OR reports and acquired data back for surgical documentation and reporting.

The context module and session repository maintain all information about the current intervention, patient information and associated files, e.g. planning data or device profiles. The session repository stores linking information about all data that is generated during the intervention by TIMMS modules, e.g. screenshots, recordings of biosignals etc. At the end of the intervention the surgeon obtains an overview of all acquired information to choose those items that should be used for surgical documentation.

The intervention logger provides a central logging service to record all relevant information from TIMMS modules and core components such as the appearance of alarms, user inputs, systems failures, and status data. This enables retrospective analysis of the intervention as well as can be used within the OR documentation.

Later versions of the integration system will encompass further core components such as a workflow management system for knowledge support and intelligent systems control. A patient model component might integrate the diversity of anatomical and functional patient data into one unique representation for the surgeon.

3 Implementation

The use and application of established communication standards is one of the main requirements for the successful implementation and acceptance of an integrated system. Intercommunications within the proposed integrated OR system are based on a set of standard protocols on top of a TCP/IP based Ethernet.
network. A C++ library was developed as general communication interface for TIMMS modules. The TIMMS Communication Library API (TiCoLi API) is a hybrid library that combines several existing open-source libraries with dedicated functionalities into one common object-oriented framework.

3.1 TiCoLi API Protocol Stack and Services

Figure 2 shows the TiCoLi API protocol stack. TiCoLi API provides a C++ application-level interface to the functionalities of the underlying protocols and specifies several services (e.g. Service Discovery, Message Exchange, Time Synchronization etc.).

![Figure 2: The Protocol Stack of the TIMMS Communication Library API (TiCoLi API) with services and corresponding protocols.](image)

The design of the TiCoLi API was challenging since the programming interface should be easy-to-use and hide complex implementation issues of the underlying protocols, e.g. streaming or messaging. Therefore, different manager classes (e.g. for devices, methods, attributes and streams) were implemented that operate beyond the TiCoLi API. The manager classes transparently maintain socket connections, threads for non-blocking API calls and device description exchange. The TiCoLi API uses the OpenIGTLink Library which is provided by NA-MIC [7], the open-source RTP implementation jRTPLIB, POSIX Threads, the ZeroConf implementation BonjourSDK, the TinyXML library and the SNMP implementation of Agent++.

3.2 Service Discovery

Each of the TIMMS modules is required to automatically detect services with certain characteristics within the network. Automatic configuration and plug-and-play service discovery of TIMMS modules are realized using ZeroConf [6]. Within ZeroConf the DNS-Service Discovery (DNS-SD) protocol is applied that specifies how DNS service messages are used to describe services offered by the modules. Modules that enter or leave the network send short status messages to all components on the network. Thus, modules can be added or removed at runtime without affecting the overall network integrity. The TiCoLi API offers a set application level service primitives to register and discover services within the integrated system.
3.3 Service Description

The service description of a server module is a document that contains information about the services (messaging, streaming, attribute access, method calls) the module currently offers via TiCoLi. The TiCoLi API provides a programming interface where each of the particular variables, methods and streams are registered during server initialization. The service description object is then transparently assembled at runtime. Each server module holds one default message socket open (as registered with ZeroConf) at which peer devices can connect and retrieve the device description.

3.4 Message Exchange, Remote Procedure Calls and Attribute Access

TiCoLi API specifies an acknowledged message exchange service, which is based on OpenIGTLink [7]. The message exchange service can be used to exchange the service description objects as well as to exchange application-level messages between client and server. TiCoLi holds references to all registered objects in the service description objects (see 3.3) to transparently facilitate reading or writing of attribute values, execution of remote procedure calls, as well as to initiate and control data streams. The TiCoLi API specifies a range of common data types for attributes (e.g. boolean, numeric, or character strings), status messages as well as service types and stream types.

3.5 Data streaming

The TiCoLi API implements an unicast and multicast streaming service based on the Real-Time Transport Protocol (RTP). RTP is an application-layer protocol, which facilitates packet-based data exchange via the User Datagram Protocol (UDP) transport layer. RTP consists of two components, the Data Transfer Protocol and the RTP Control Protocol (RTCP), which is used for exchange of reception quality feedback and for synchronization. TIMMS modules can connect to streams offered by servers by sending a connection request via a previously opened messaging session (see 3.4). Quality of service (QoS) information, such as stream and video characteristics, are exchanged as stream-description object via the messaging session prior to the stream connection. The TiCoLi handles the synchronization of the streams internally. The streaming service continuously sends frames to subscribed client modules according to the frame-rate negotiated during QoS exchange.

3.6 Time Synchronization

Time synchronization within the integrated system is facilitated using the Network Time Protocol (NTP). NTP is designed to compensate for network latencies, which are a common problem in packet-switched domains such as IP based networks. NTP is based on a hierarchical client-server architecture with highly reliable clocks such as atomic clocks on the highest level. The implementation in the proposed integrated system follows the IHE Profile “Consistent Time”, where the TCC implements a grouped time server with which each of the modules can be synchronized.

3.7 Systems Monitoring und Diagnosis

The life critical domain within the OR demands safe and reliable operation of the integrated OR components. Therefore, the TiCoLi implements a technical supervision framework that facilitates systems monitoring and diagnosis of the integrated system. Management agents are used to acquire performance and diagnostic information at network backbone level, computer hardware level, and in software
applications. The transfer of diagnostic information between management agents and the supervisory module of the TCC is based on the “Simple Network Management Protocol” (SNMP). Monitoring at code implementation level is facilitated using the Open Group Standard “Application Response Measurement” (ARM). The combination of ARM and SNMP enables software performance measurements as well as alive-state supervision using software watchdogs [8].

4 Results

The presented modular OR system integrates medical hardware and software components into a common research platform. Integration takes place at three different levels: 1st data, 2nd functions, and 3rd applications. A prototype setup was established in a demonstrator OR laboratory (Figure 3). The TIMMS modules operate on standard PC hardware and are interconnected using a standard 100/1000Base-T Ethernet network.

Several software applications have been implemented as modules based on the TiCoLi API, such as a streaming server for continuous object tracking, an image-guided surgery application using navigation and patient modelling (Figure 4) [9], streaming of biosignals, a server for display and video routing, as well as PACS access to DICOM objects, e.g. imaging data or reconstructed surfaces from surgical planning software.

The user interface for the clinical user provides centralized access and control using ceiling-mounted booms with touch screen displays (Figure 3). At the central user interface, software applications and device functions are integrated based on remote display software using the Remote Framebuffer Protocol (RFB) as well as video routing technologies.

The user interface for the technical supervisor at the TCC software component provides an overview and access to all connected TIMMS modules (Figure 5). It displays the particular device models as well as provides access to configurable data elements.

Figure 3: Integrated OR prototype within ICCAS demonstrator lab.

Figure 4: Image-guided-surgery application for brain tumor surgery [9].
The supervising and monitoring module provides information to detect system anomalies such as network bottlenecks, cache and hard disc space exceeds, or CPU consuming software processes. The visualization comprises simple numerical values of performance measurements as well as graphical trend views for time-dependent values (e.g. network load).

The integrated OR system itself is independent of any particular surgical discipline. Due to its modular architecture the system can be adapted to specific clinical requirements and applications on demand.

5 Conclusion

A generic open standards based infrastructure that supports the integration of components for computer assisted surgery has been developed. The presented programming interface TiCoLi API enables rapid development of CAS applications for the proposed integration architecture. Methods of auto-configuration, plug-and-play service discovery, message exchange and device control, streaming as well as systems supervision and time synchronization are accessible through a set of high-level C++ API calls. Since the OR is characterized by rapid changing technologies, the proposed systems integration architecture aims for the development of a stable long-term integration solution that can be used in different surgical disciplines. Standard communication protocols as used within the TiCoLi API and technology independent service descriptions offer the potential to solve this issue while providing sufficient flexibility due to the independency of special, programming languages, operating systems or network technologies. Future developments aim on the investigation of semantic interoperability as well as automated workflow assisted interventions.

Acknowledgments

The Innovation Center Computer Assisted Surgery (ICCAS) at the Faculty of Medicine at the University of Leipzig is funded by the German Federal Ministry for Education and Research (BMBF) and the Saxon Ministry of Science and Fine Arts (SMWK) within the scope of the initiative Unternehmen Region with the grant numbers 03 ZIK 031 and 03 ZIK 032.
References


Latest version available at the Insight Journal link http://hdl.handle.net/10380/3081
Distributed under Creative Commons Attribution License
Abstract

The Surgical Assistant Workstation (SAW) modular software framework provides integrated support for robotic devices, imaging sensors, and a visualization pipeline for rapid prototyping of telesurgical research systems. Subsystems of a telesurgical system, such as the master and slave robots and the visualization engine, form a distributed environment that requires an efficient inter-process communication (IPC) mechanism. This paper describes the extension of the component-based framework provided by the cisst libraries, on which the SAW depends, from a multi-threaded (local) to a multi-process (networked) environment. The current implementation uses the Internet Communication Engine (ICE) middleware package, but the design does not depend on ICE and can accommodate other middleware choices. A telesurgical robot system based on the da Vinci hardware platform is used as a test case, with a research application that requires coordination between the left and right robot arms (a bimanual knot tying task). Performance measurements on this platform indicate minimal overhead due to the networking.

Latest version available at the Insight Journal [http://hdl.handle.net/10380/3079]
Distributed under Creative Commons Attribution License

Contents

1 Introduction 2
2 Overview of Telesurgical Robot System 3
3 Network Implementation of Inter-Task Communication 4
  3.1 Design Concept: Proxy Pattern .......................... 4
  3.2 Proxies in the cisstMultiTask library .................... 5
  3.3 Code-level Changes ..................................... 5
4 Performance Analysis for Telesurgical Robot System 6
1 Introduction

The Surgical Assistant Workstation (SAW) modular software framework provides integrated support for robotic devices, imaging sensors, and a visualization pipeline for rapid prototyping of telesurgical research systems [7]. It is based on the cisst libraries for computer-integrated surgical systems development [2]. The cisst libraries can be classified in three categories: 1) foundation libraries, 2) component-based framework, and 3) component implementations (see Figure 1). The SAW is an architectural framework that sits on top of cisst; as such, it draws heavily from the component-based framework and from the collection of implemented components, such as robots, collaborative robots (e.g., telesurgical robots), and a 3D user interface toolkit.

One major limitation of the framework reported last year was that it was only usable within a single process; that is, it was designed for safe and efficient data exchange in a multi-threaded environment within a single process. Clearly, this is a serious limitation for systems, such as telesurgical robots, that often require multiple computers. As a result, it was not possible to consistently employ the same inter-task communication mechanisms in such systems. Instead, researchers were forced to implement their own inter-process communication (IPC), often resorting to a custom-made socket-based protocol. Although use of a standard protocol, such as OpenIGTLink, would be better, it would still require researchers to employ different communication mechanisms, depending on whether or not the tasks were in the same process.
Clearly, a toolkit that is designed to facilitate the development of computer assisted intervention systems must be distributed by nature. For code reuse, however, it is desirable to have an implementation that is independent of the topology. In this paper, we report on the development of a network interface for the cisst component framework that extends the current inter-task communication mechanism to the multi-process and multi-computer scenarios. The network interface currently uses the Internet Communications Engine (ICE) to provide the low-level functionality, but still preserves the existing inter-task programming model; in fact, other than some additional configuration, researchers will not see a difference in how they create their multi-task applications. It is important to note that the design does not depend on ICE—it is possible to replace ICE by any other package that provides the required features (even standard sockets would do, with a bit more programming effort). There are a plethora of middleware packages that could be considered as alternatives to ICE. Some examples include the Spread Toolkit, Data Distribution Service (DDS), and Common Object Request Broker Architecture (CORBA).

We chose ICE as our network middleware packages for two reasons. The first is that the basic concept of the ICE architecture is the Proxy Pattern [3, 6], which is also the main design concept of the new cisst-MultiTask library (see Section 3.1). The second is that ICE provides the SLICE—Specification Language for ICE—which is conceptually similar to Common Object Request Broker Architecture (CORBA) Interface Description Language (IDL). SLICE provides flexible and extensible ways to define or modify data structures and interfaces that play a critical role to share information between components [1].

This paper uses a telesurgical robot system, with virtual fixtures, as an illustrative example. This system is described in the following section.

2 Overview of Telesurgical Robot System

The basic telesurgical robot system consists of two master/slave manipulator pairs, a stereo viewing console (all from Intuitive Surgical, Inc.), custom-designed motor controller/amplifier boards and control software based on the SAW application framework. Due to physical hardware limitations, one control workstation cannot host all the required controller/amplifier boards for the two master/slave manipulators pairs. Therefore, each master/slave pair is connected to one control workstation. For this teleoperation configuration, no networked communication is needed (see Figure 3a).

In other configurations where two masters are coordinated, or master/slave reside on different computers, networked communication between multiple computers is required, thus motivating the network extension to the inter-task communication model.

A specific example is provided by the Bimanual Knot Placement project, where we use virtual fixtures to provide guidance in performing the suture knot placement task using both master/slave teleoperated manipulators in the same workspace (see Figure 3b). This is an extension of earlier work that used a simpler experimental setup (Cartesian robots under cooperative control) to demonstrate the concept [5]. In the current system, the surgeon operates the master manipulators, and the slave manipulators follow the masters under certain motion optimizations, which are implemented with virtual constraints such that the correct sliding friction and knot motion trajectory are maintained to place the knot. This project requires coordination of both master manipulators and a stereo vision system to track the position of the knot and both slave tool tips, which must be fed back to the two master manipulators. In this project configuration, each master/slave manipulator pair is controlled by a workstation and both connect to the stereo vision system on another workstation via the network extension.

In the Swapped Slave configuration, the right master controls the left slave and vice versa. In this scenario,
a master must use network communication to transmit Cartesian frame information to the opposite slave, which is controlled by a different workstation (see Figure 3c).

In all these configurations, the control task implementation for each master and slave manipulator is independent of the topology. The only differences are during system configuration, where the specific required and provided interfaces are connected. The three configurations shown in Figure 3 have all been implemented and tested, showing the flexibility of the network extension.

3 Network Implementation of Inter-Task Communication

One of the key features of the cisstMultiTask library is to allow data exchange between tasks in a thread-safe and efficient way. The initial implementation focused on multithreading within a single process. Nevertheless, the key design concepts—component-based philosophy and self-describing interfaces—allow painless integration via the Proxy Pattern [3, 6] so that cisstMultiTask can support IPC.

The low-level data communication between proxies is built upon ICE which provides a simple, flexible, light-weight but versatile network middleware. This low-level data communication layer has been abstracted so that the new cisstMultiTask library is loosely coupled with ICE, which means the overall design does not depend on ICE. It can be easily replaced with any other package, even native sockets, if they can support the abstraction layer appropriately.

3.1 Design Concept: Proxy Pattern

Using the Proxy Pattern, the cisstMultiTask library is able to handle proxy objects as it does the original objects. In Figure 4, object A is locally connected to object B. With the introduction of our proxy-based model, object A is locally connected to the object B proxy but is practically linked to object B across a
3.2 Proxies in the `cisstMultiTask` library

For efficient data exchange in a thread-safe way, the `cisstMultiTask` library uses several components such as `tasks`, `interfaces`, `commands`, and `events`. The implementation of IPC requires these components to be split across a network, which means several proxy classes should be defined as well. In order to increase code reusability and to manage proxy classes in a consistent way, we defined three base classes—`mtsProxyCommonBase`, `mtsProxyBaseClient`, and `mtsProxyBaseServer`—and then created the other derived classes. Table 1 shows the complete list of proxy classes defined in the `cisstMultiTask` library.

Note that these proxy classes are entirely hidden from the application layer and are created and managed internally. This design allows programmers to use any class or object derived from the `cisstMultiTask` base types over the network, thereby enabling network extension with minimal code changes (see Section 3.3).

3.3 Code-level Changes

Use of the Proxy Pattern allows `cisstMultiTask` library users to extend their single process application to a multi-process system that works across a network in a very simple and easy way. If there are two tasks—a server task and a client task—that run in the same process (see Figure 5), the core part of the application configuration code would be the following:

```cpp
taskManager->AddTask(clientTask);
taskManager->AddTask(serverTask);
taskManager->Connect("clientTask", "requiredInterfaceName",
"serverTask", "providedInterfaceName");
```
If, however, these two tasks are distributed over a network, the configuration code would be as follows:

**Server Process** – Simply add the server task and make sure the task manager publishes it:

```cpp
    taskManager->SetTaskManagerType(mtsTaskManager::TASK_MANAGER_CLIENT);
    taskManager->AddTask(serverTask);
```

**Client Process** – Add the client task and connect to the server task via the global task manager:

```cpp
    taskManager->SetTaskManagerType(mtsTaskManager::TASK_MANAGER_CLIENT);
    taskManager->AddTask(clientTask);
    taskManager->Connect("clientTask", "requiredInterfaceName",
                          "serverTask", "providedInterfaceName");
```

To summarize, there is very little code overhead for application programmers because all proxy objects such as task proxies, interface proxies, command proxies, and event proxies are dynamically created and run internally. The only methods to be implemented are serialization and deserialization if the commands and events use data type that are not supported by the cisst package (serialization methods for vectors, matrices, frames, strings and common robot data types are already provided). For new data types, the implementation of serialization should be trivial because the library provides examples and auxiliary methods.

## 4 Performance Analysis for Telesurgical Robot System

Using the cisstMultiTask network extension, the conversion of a single process application to a multi-process application requires additional processing such as serialization, networking overhead (physical data transmission over a network), dynamic object creation, and deserialization. This naturally leads to the lowered performance of the overall system. Thus, it is important to perform experiments to understand how significantly these factors affect the overall performance of the system.

The core idea for testing is not to rely on synchronization between the client and server to measure the network overhead. Instead, we use a data object which goes from server to client and back to the server (we also have a benchmark which uses client to server and back to client). As our datatype carries a timestamp,
we can compare the time it took with and without the middleware. By subtracting, we get an estimate of the network overhead, including serialization, dynamic creation and de-serialization.

For the communication from server to client and back to server (SCS), we first use a Read command. Since the Read command uses a circular buffer for thread safety, the client receives an object which is no older than one period plus the transmission time. The client then uses a Write command to send back to the server. This command is queued and executed at the beginning of the next period. The total time elapsed should be lower than 2 periods (assuming both client and server use the same period and are sufficiently synchronized).

For the communication from client to server and back to client (CSC), we first use a Write command. As in the previous example, the command is queued. When the command is executed, it sends the data back using an event. The event is queued on the client side. Again, the total time elapsed should be lower than 2 periods.

The goal of this experiment is not to compute the total loop time but to estimate the overhead introduced by the middleware and the required serialization, dynamic creation and de-serialization. For our experiments, we relied on the cisst data type prmPositionCartesianGet which contains a rotation matrix, position vector (all doubles), a timestamp, a valid flag and a couple more fields for a total of 120 bytes.

Table 2 shows our experiment results using two machines (Ubuntu 8.0.4 with RTAI installed, kernel 2.6.24-16-rtai i686 GNU/Linux, Pentium 4 3.2GHz, 1G RAM).

For both SCS and CSC tests, Linux with RTAI performed better than Linux without RTAI, as expected. The most notable thing here is that the average elapsed time is small (less than 1 msec), even in the worst case (SCS testing, 0.9229 msec under Linux with RTAI disabled). This implies that the overhead due to the introduction of networking is minimal.

The total overhead can be separated into three parts: preparation and recovery overhead (serialization/dynamic creation/deserialization), transmission overhead (networking that includes middleware layer processing), and execution overhead (pointer casting/dereferencing). The preparation and recovery overhead were measured using an independent test program that repeats a testing loop 10000 times that serializes a variable, dynamically creates it, and deserializes it. The results—0.004239 msec and 0.0043557 msec for Linux with

Table 1: List of Proxy Objects in the cisstMultiTask Library

<table>
<thead>
<tr>
<th>Class Type</th>
<th>Original Class</th>
<th>Proxy Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base</td>
<td>-</td>
<td>mtsProxyBaseCommon</td>
</tr>
<tr>
<td></td>
<td></td>
<td>mtsProxyBaseServer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>mtsProxyBaseClient</td>
</tr>
<tr>
<td>Derived</td>
<td>mtsTaskManager</td>
<td>mtsTaskManagerProxy</td>
</tr>
<tr>
<td></td>
<td></td>
<td>mtsTaskManagerProxyServer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>mtsTaskManagerProxyClient</td>
</tr>
<tr>
<td>Derived</td>
<td>mtsDevice</td>
<td>mtsDeviceProxy</td>
</tr>
<tr>
<td>Derived</td>
<td>mtsDeviceInterface</td>
<td>mtsDeviceInterfaceProxy</td>
</tr>
<tr>
<td></td>
<td></td>
<td>mtsDeviceInterfaceProxyServer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>mtsDeviceInterfaceProxyClient</td>
</tr>
<tr>
<td>Derived</td>
<td>mtsCommandVoid</td>
<td>mtsCommandVoidProxy</td>
</tr>
<tr>
<td></td>
<td>mtsCommandWrite</td>
<td>mtsCommandWriteProxy</td>
</tr>
<tr>
<td></td>
<td>mtsCommandRead</td>
<td>mtsCommandReadProxy</td>
</tr>
<tr>
<td></td>
<td>mtsCommandQualifiedRead</td>
<td>mtsCommandQualifiedReadProxy</td>
</tr>
<tr>
<td>Derived</td>
<td>mtsMulticastCommandWrite</td>
<td>mtsMulticastCommandWriteProxy</td>
</tr>
</tbody>
</table>

Latest version available at the Insight Journal [http://hdl.handle.net/10380/3079](http://hdl.handle.net/10380/3079)
Distributed under Creative Commons Attribution License
Table 2: The overhead measurement results

and without RTAI, respectively—show that this overhead is negligible. Similarly, the execution overhead is also very small. Thus, the transmission overhead accounts for the largest part of the total overhead, which is still relatively small. This low overhead for ICE has already been extensively tested and examined [4]. To summarize, the experiment results show that users can extend a single-process application to a multi-process system with just a small sacrifice in performance.

5 Conclusions

We created a component-based framework for the development of computer-assisted intervention (CAI) systems, focusing initially on achieving high-performance multi-tasking by implementing all tasks as threads within a single process. The component model consists of command objects in provided interfaces that are bound to corresponding objects in required interfaces. Conceptually, the task with the provided interface can be considered the server for the task with the required interface (the client), although a task can have both types of interfaces (e.g., acting as both a client and server in a hierarchical structure). This component model has proven useful for the creation of several CAI systems, but the limitation to a single process has been problematic. A particularly compelling example is a telesurgical robot system which, due to hardware constraints, consists of several robotic arms connected to different computer workstations. Thus, we extended our existing component model to a network configuration by creating proxy objects for the key elements of our component-based framework – specifically, for the task manager, tasks (and devices), interfaces, and commands. We used the Internet Communication Engine (ICE) to provide these proxies, though the design allows other middleware packages to be used. We tested the software by implementing it on the telesurgical robot system described above. Performance results indicate minimal overhead (less than 1 msec) due to the network implementation.

The SAW framework and underlying cisst libraries are being released under an open source license. Currently, most relevant portions of the cisst libraries (i.e., the foundation libraries and component framework illustrated in Fig. 1) are available at trac.lcsr.jhu.edu/cisst. Current work is focused on increasing the set of
implemented components that enable creation of different CAI systems within the SAW framework.

Acknowledgments

The cisst libraries have been created by many individuals; primary contributors include Ankur Kapoor, Ofri Sadowsky, Balazs Vagvolgyi, and Daniel Li. We acknowledge the contributions of Simon DiMaio, Chris Hasser and colleagues at Intuitive Surgical, Inc. towards the SAW framework. This work is supported in part by National Science Foundation EEC 9731748, EEC 0646678, and MRI 0722943.

References


[3] E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994. 1, 1, 3


Latest version available at the Insight Journal [http://hdl.handle.net/10380/3079]
Distributed under Creative Commons Attribution License
Extension to OpenIGTLink; Smart Socket Connection, XML as Message, Logging, and One-to-multi Relaying

Release 0.00

Kiyoyuki Chinzei¹ and Junichi Tokuda²

August 15, 2009

¹National Institute of Advanced Industrial Science and Technology (AIST), Tsukuba Japan
²Brigham and Women’s Hospital and Harvard Medical School, Boston, USA

Abstract

We extended OpenIGTLink with following useful features; 1) AgentSocket class to capsule socket initiation for smarter socket connection with try-catch implementation, 2) XmlMessage class to handle an XML text message. Using this extension, we will implement a data-logging DB that can relay the data to multiple clients.

This will impact the following 1) a pair of agents takes care of establishing, maintaining, terminating the communication channel, as well as attempting re-connection in case of network trouble, 2) modern error handling, 3) arbitrary format of data transfer by formatting an XML, including MRML data transmission via OpenIGTLink, 4) data logging by XML format. The work is yet in an early stage of implementation and a simple client example will be demonstrated.

Contents

1 Introduction 2
2 Intelligent Surgical Instruments Project 2
3 Missing Features from Current OpenIGTLink 3
4 New features 4
5 Sample of AgentSocket Usage 5
6 Log DB; Application of the Extensions 5
1 Introduction

OpenIGTLink is a messaging protocol for communicating a position, a transformation, an image, a command or system status value, and its implementation to send and receive between two processes. For details, refer to [1, 2]. OpenIGTLink is implemented as a C++ library on Linux, Windows XP or newer, Mac OS 10.4 or newer. It uses cmake as a make tool, and the rest of tools and libraries necessary to build and use OpenIGTLink are the standard ones found in these operating systems.

Since its birth in early 2008, OpenIGTLink has been accepted by certain number of research groups and commercial companies. Its popularity as a research and development tool is rooted from its simple yet extensible nature. Although it is simple, it has modern programming features such as the namespace, the multi-thread-safe coding, the use of socket, and multi-platform compatibility.

However, we experienced an extensive amount of custom coding for our new system development, which was indeed not particularly large scale. We believe our self-coding parts are useful for others.

This paper describes several extensions we made and plan to make to the current OpenIGTLink library. This paper is organized as the following: Section 2 briefly introduces our ongoing project and its requirement from the architecture. Section 3 describes some limitations in the current OpenIGTLink and Section 4 states our proposal to extend it. Section 5 is a code fragment using these new features, and Section 6 is an application of these extensions.

2 Intelligent Surgical Instruments Project

The authors are working with other companies and research groups to develop an integrated endoscopic surgical robot system. Its research topics includes 1) sensors to locate tumors, 2) logging of system behavior and playback, 3) monitoring system that detects 'unusual' situations from these sensor information and issues warning, 4) training system using the logging and playback functions.

Its communication related schema is in Fig. 1. It has the following characteristics:

1) Processes communicate by OpenIGTLink except some time or performance critical connections, e.g., between the master-slave robots, or when noise tolerance is required, or 3D or video images.

2) All the OpenIGTLink communications are recorded as a log in Log DB. Log DB collects any communication by OpenIGTLink.

3) Monitoring systems are collection of different algorithms. They need to have access to the log data, and each algorithm requires different combination of incoming log data.

4) Number of PCs for sensors and tracking devices are not fixed.
Figure 1: The basic communication components in the Intelligent Surgical Instrument system. Blue rectangles are computers with OpenIGTLink library. There is a master database machine for "Log DB" (the red rectangle). Monitoring Systems are collection of algorithms to detect 'unusual' situation from various combination of incoming log data.

In this configuration, a tracker probe is associated to its subjects to track. If a tracking probe is associated to a sensor, the data will be delivered to 1) the sensor so that the Sensor PC binds the sensor value with its position, 2) the Log DB to store the data (Fig. 2), and 3) Monitoring System may or may not use the position information. If another tracking probe is to represent robot transformation, the value will be delivered to Robot Console and the Log DB. In short, the tracking probes' information will be delivered differently according to their associated subjects to track.

3 Missing Features from Current OpenIGTLink

1) When there are many OpenIGTLink servers and clients distributed in the system, managing the server-client pairs will be painful.
The basic connection mechanism of the OpenIGTLink library is making a pair of `ServerSocket` and `ClientSocket` classes. `ServerSocket` takes a port number to cast the data and `ClientSocket` takes a hostname and a port to specify the server. Additionally, `ServerSocket` process must start before `ClientSocket` process starts, otherwise `ClientSocket` will fail. If there is only one pair of server-client, it is manageable. However, if the system has many servers / clients sparsely located in many computers, maintaining the correct pairs of the hostname and port information and the order to start the program will be a painful work.

2) If server-client socket is broken or fails to establish communication, need to redo the initiation process.
3) No built-in time-out mechanism.

If the application program does not handle such situations properly, the program will 'freeze' or abnormally terminate.

4) OpenIGTLink library has `StatusMessage` class that can communicate any string data.

Any string data means there is no common semantics in the string. Application programs need to define it.

4 New features

1) `AgentSocket` class to capsulate Socket classes for better communication with try-catch implementation

We designed `AgentSocket` class to hide the details of initiation and re-connection procedures. `AgentSocket` takes care of the following tasks; 1) connect clients and servers, 2) when server is not responding, try to redo it specified times with specified interval, 3) when requested, periodically examine keep-alive test of each other, 4) when time-out condition met, try to re-initiate the socket, 5) these configurations are read from a configuration file. We plan to implement an exception handler to capture these situations. Inside `AgentSocket` is a socket connection to a daemon process running on the localhost. The daemon process takes care of the actual connection to the server. We assume that the socket connection on the localhost is robust enough to rely on.

2) `XmlMessage` class to handle an XML text message

`XmlMessage` class is a subclass of `StatusMessage` class. Except defining a new header identifier "XMLSTATUS", the behavior is same as `StatusMessage`. We used libxml2 as the XML parser, however, we did not include it as a part of OpenIGTLink library - it can be any parser preferred.

---

1 Writing conventions in this paper: Every OpenIGTLink class is defined like `igtl::ServerSocket`. In this paper, we omit namespace portion (`igtl::`).
5 Sample of AgentSocket Usage

Below is the code snippet to establish connection. Instead of directly connecting to a server by specifying the hostname and the port, AgentSocket class takes care of initiating, maintaining and terminating the socket connection.

Code 1: Sample of connection process

```cpp
try {
    igtl::ConfigFactory cfg(configfilepath);
    igtl::Socket::Pointer socket = igtl::AgentSocket::New();
    socket->Initialize(cfg);
    int r = socket->ConnectToServer(cfg.timeoutConnect);
    ...
}
```

The remaining part is not necessary to modify from the current codes by this change.

6 Log DB; Application of the Extensions

We are developing an application of these extensions to realize the Log DB in Fig. 1. As described above, a tracking probe position data may be sent to 1) the associated sensor, 2) the Log DB, 3) Monitoring System may or may not use it. We are implementing as the following; 1) Tracker PC explicitly configures to connect to Sensor PC, 2) AgentSocket automatically opens a connection to the Log DB, 3) the Log DB knows about which Monitoring Systems need this information and connects to these (Fig. 3). As such socket connections increase, there may be an issue of latency. We will evaluate the latency and other performance loss.

Acknowledgement

This work has been funded by NEDO P08006 "Intelligent Surgical Instruments Project", Japan. J.T. was supported by NIH 5U54EB005149 and 5U41RR019703.

Reference


Unified framework for development, deployment and testing of image analysis algorithms

Release 0.00

Alark Joshi\textsuperscript{1}, Dustin Scheinost\textsuperscript{1}, Hirohito Okuda\textsuperscript{4}, Isabella Murphy\textsuperscript{1}, Lawrence H. Staib\textsuperscript{1,2,3} and Xenophon Papademetris\textsuperscript{1,2}

August 15, 2009

\textsuperscript{1}Department of Diagnostic Radiology, Yale University
\textsuperscript{2}Department of Biomedical Engineering, Yale University
\textsuperscript{3}Department of Electrical Engineering, Yale University
\textsuperscript{4}GE Healthcare, Japan

Abstract

Developing both graphical and command-line user interfaces for image analysis algorithms requires considerable effort. Generally developers provide limited to very rudimentary user interface controls. These image analysis algorithms can only meet their potential if they can be used easily and frequently by their intended users. Deployment of a large suite of such algorithms on multiple platforms requires that the software be stable and appropriately tested.

We present a novel framework that allows for rapid development of image analysis algorithms along with graphical user interface controls. Additionally, our framework allows for simplified nightly testing of the algorithms to ensure stability and cross platform interoperability. It allows for development of complex algorithms by creating a custom pipeline where the output of an algorithm can serve as an input for another algorithm. All of the functionality is encapsulated into the object requiring no separate source code for user interfaces, testing or deployment. This makes our framework ideal for developing novel, stable and easy-to-use algorithms for computer assisted interventions (CAI). The framework has both been deployed at Yale and released for public use.

Latest version available at the Insight Journal [http://hdl.handle.net/10380/3078]
Distributed under Creative Commons Attribution License

Contents

1 Introduction 2
2 Related Work 3
1 Introduction

Image analysis algorithms are typically developed to address a particular problem within a specific domain (functional MRI, cardiac, image-guided intervention planning and monitoring, etc.). Many of these algorithms are rapidly prototyped and developed without considerations for a interface (GUI), robust testing, and integration into a large software package. Sometime these features are added later, but require considerable amount of effort on the part of the developer of the original algorithm. This makes it increasingly difficult for deployment and widespread adoption of the newly developed algorithms especially for CAI algorithms where robust testing and easy-to-use interfaces are critical.

BioImage Suite [12] is a comprehensive, multi-platform image analysis suite comprised of many different image analysis algorithms with a focus on epilepsy neurosurgery. In previous versions of BioImage Suite (up to version 2.6 released in November 2008), all algorithms were implemented in C++ and invoked from either command line scripts or GUI modules both written in the Tcl scripting language. However, the command line scripts and GUI modules were two separate implementations of essentially the same algorithm and would invariably diverge without extensive coordination. This required developers to create both command line scripts as well as complex GUIs. Testing became problematic as two new applications need to be tested for each new algorithm. Finally as new algorithms became more complex, basic components (e.g., image smoothing) were often reimplemented instead of using existing implementations of these components.

To address the issues discussed above, we developed a framework that unifies the algorithm that is being invoked from the command line as well as from the user interface. We have chosen a component-based software approach which has been widely used and researched in the field of software engineering [10]. In our framework, a component performs an operation (smoothing, surface extraction and so on) on the specified input data (images, surfaces, transformations). The main algorithm is developed in C++ while its functionality is encapsulated into an [Incr Tcl] object. [Incr] Tcl[9] is one of the most commonly used object-oriented extension for Tcl, which is not an inherently object oriented language. The encapsulation of C++ classes allows the user to instantiate an object in a tcl script and handle the input/output as well as the GUI via inherited methods.

With this framework, the developer can focus on the creation of the algorithmic component and not worry about software engineering aspects needed for CAI algorithms such as testing, integration, and creating customized workflows. The GUI is automatically generated by the algorithm object when the object is invoked – however, given that this is an object-oriented framework, the developer may customize this by overriding...
the appropriate methods. Testing is handled by the algorithm object by specifying the inputs, expected outputs, and the test flag. Additionally, in this new framework, it is possible to create data workflows where the output of an algorithm can then be used as the input of another algorithm. Thus, developers can reuse existing algorithms saving time and reducing programming complexities.

In addition, this framework, with its central definition of parameter handling code, enables BioImage Suite components to output descriptions for both the Slicer Execution Interface and the LONI Pipeline at no extra work to the developer – this is handled by the abstract parent class of the component hierarchy.

2 Related Work

Software development work in the field of medical image analysis has focused on describing the architecture for a specialized setting. Coronato et al. [1] developed an open-source architecture for immersive medical imaging that used 3D graphics and virtual reality libraries. Additionally, they also include ubiquitous computing principles for context aware interaction with mobile devices. Shen et al. [6] discuss their system which works with stereoscopic displays and uses projectors to provide an immersive experience in environments such as the CAVE.

In the field of medical image analysis, medium to large imaging software such as Slicer3D [7] have a component based approach to developing software that allows for the easy development of user interfaces for developers. Here each algorithm generates an XML file specifying instructions for creating a GUI, which is then read by the main application at run time to create a GUI. One of the limitations of this approach is that an external application creates the GUI and 3D Viewer components which increases the complexity of the external application and implies that the algorithm implementation cannot function as a stand-alone application. Medical Imaging Interaction Toolkit (MITK) [3] is a medical imaging toolkit for image analysis, which has some features similar to those in our framework. However, it is intended to be used as a toolkit and “is not intended as an application framework” [3] that can be used for development of novel image analysis by users.

The functionality included in all the abovementioned image analysis software systems and others is very similar. Our contribution is in the ability to provide researchers in image analysis an open-source platform-independent framework that allows them to focus on developing new algorithms. The software engineering aspects of interface design, testing protocols, and code reusability are automatically provided to the researcher, assisting deployment and widespread adoption of the newly developed CAI algorithms.

3 System Overview

Our unified framework allows for easy development, deployment, and overall packaging of image analysis algorithms. Using this framework, developers can easily create user interfaces and seamlessly test their algorithm on multiple platforms. Novel algorithms can be added and custom workflow pipelines can be constructed where each piece of the pipeline is an algorithm that takes an input and performs an operation. Figure 1 shows a flowchart for an image analysis algorithm. Each new algorithm takes a combination of images, surfaces, transformations, and input parameters and produces a combination of images, surfaces, and transformations as outputs.

BioImage Suite algorithms are packaged into a single set of [Incr Tcl] classes [9]. These classes are characterized by two key methods, Initialize and Execute. In the Initialize method, each algorithm
Figure 1: This diagram provides an overview of the new unified framework for image analysis algorithm development. Any image analysis algorithm has a combination of images, surfaces, and transformations (from registrations) that serve as input to the algorithm. The algorithm most probably has some input parameters (which can be specified on the command line or can become GUI components). In our framework, the input parameters can be one of boolean, real, integer, string, listofvalues (for drop down options when using a GUI) or a filename. The output too can be a combination of images, surfaces and transformations.

explicitly defines three sets: (i) inputs, which are objects such as images, surfaces, landmarks etc., (ii) parameters, which are single values such as integers, strings, filenames, and (iii) outputs, which are also objects. Figure 2 shows a detailed example of an Initialize method. The Execute method invokes and passes object to/from the underlying C++ source code which consists of classes derived from VTK parent classes[8]. Some of the C++ code also leverages functionality from ITK [4].

Based on the definition of the input and output sets, the base abstract classes have functionality (which need not be touched by more concrete implementations) to (i) parse command line arguments if the algorithm class is invoked as an application; (ii) automatically create a GUI using the CreateGUI method (this method can be overridden by some algorithms to generate a more customized interface); and (iii) perform testing by parsing a test file. These classes can then be used (i) to invoke the algorithm (using an Execute method), (ii) to become a components of other algorithms (e.g. the image smoothing algorithm is invoked by the edge detection algorithm), (iii) to create a standalone applications with an image viewer and a GUI, and (iv) to integrate individual components into a larger application. Figure 3 shows how the same code is invoked using (i) the command line, (ii) the GUI, (iii) a larger application, and (iv) for nightly testing.

Figure 2: Sample implementation for a image smoothing algorithm. A new algorithm usually requires the implementation of two methods. The first is Initialize (shown in detail in this figure) where the inputs, outputs and parameters are defined. The second is Execute (not shown) which simply takes the specified inputs and parameters and runs the actual algorithm to generate the desired output. Derived classes can have customized GUI by overriding the CreateGUI method.

Latest version available at the Insight Journal [http://hdl.handle.net/10380/3078] Distributed under Creative Commons Attribution License
3.1 Core Classes:

The new framework has at its core the following [Incr Tcl] classes:

1. `bis_option` encapsulates an option value (e.g. smoothness factor, etc.). An option can have a type of: listofvalues, boolean, real, integer, string or filename. Within this class there is functionality for creating an appropriate GUI for each option.

2. `bis_object` encapsulates the input and output objects of the algorithms. The core objects supported are: image, transform (both linear and non-linear), polygonal surface, landmark set and electrode grid.

3. `bis_basealgorithm` is the core algorithm class from which all algorithms are derived. It has all the functionality for manipulating options, inputs and outputs.

4. `bis_algorithm` is derived from `base_algorithm` and adds the functionality needed for taking an algorithm and making it into a component or an executable. More specialized classes are derived from `bis_algorithm` such as `bis_imagetoimagealgorithm` which serves as a base for algorithms which take a single image as an input and produce a single image as an output.

5. `bis_guicontainer` is a derived class of `bis_algorithm` and serves as a parent class for creating multi-algorithm containers (e.g. a tabbed-notebook style GUI where each tab is a separate algorithm).

4 Current Status

In this section, we discuss the current status of the new framework. We provide details about the invocation of the algorithms, discuss our nightly testing setup, show an example of a customized data workflow that is being used for processing SPECT data for epilepsy neurosurgery and discuss the interoperability that this framework facilitates.

4.1 Algorithm interfaces

An algorithm can be invoked in three ways: (i) command line, (ii) GUI, and (iii) managed graphical interface. The framework facilitates the invocation of the same code regardless of the manner in which the script is invoked.

Figure 3: This schematic shows the unified nature of our framework. The command line invocation of an algorithm as well as the GUI use the same algorithm source code and can be controlled using a simple `-dogui` flag. For integration into a larger user application, the same source code is used to give the user a toolkit of similar algorithms that can be used. Additionally, for nightly testing the same algorithm code is invoked with different input parameters that test the algorithm on various platforms.
4.2 Nightly Testing

Nightly testing is done with the help of the functionalities in CDash. The implementation of our testing framework allows for easy addition of test cases. For testing, we maintain a list of all the test cases which

Figure 4: This figure shows all the different ways in which a script can be invoked. Label (A), (A1) and (A2) shows the graphical user interface (A) with the parameters in the standard tab (A1) and the advanced tab (A2). Additionally, the user can click on the “Show Command” button highlighted with a red rectangle that shows how the script can be invoked on the command line (C). The script can also be invoked on the command line (B) and in the situation where incorrect input parameters are provided, a Unix-style help is shown that shows the format for the input and input parameters. Additionally, the script can be contained in a managed framework (D) where it becomes a menu item that invokes the same graphical user interface options as in (A).
have the following format:

algorithm name : input parameters and their values: input files : expected output file
bis_smoothimage: -blursigma 2.0: MNI_T1_1mm.nii.gz : MNI_T1_1mm_sm2.nii.gz

When the nightly testing process starts, it goes through and tests each algorithm. For each algorithm, it
looks up its name in the first column of the list, and if the name matches then it reads in the remaining
arguments and performs the test. As shown above, to test the image smoothing algorithm we specify the
name of the script, the input parameters and their values (blursigma=2.0 in this case), the input file name
and the expected output file name to compare the output with. The obtained output is compared with the
expected output and based on the comparison a “test passed” or “test failed” result is obtained. Therefore,
adding more test cases is as simple as adding another line to the list of nightly tests for that algorithm.

Figure 5 shows a composite figure of the nightly dashboard which shows the platforms that the scripts are
tested on. As of now, the nightly tests run on Linux (CentOS), Windows (Vista and XP), Power Mac and
Mac OS X. On the right is a screenshot of a list of some of the scripts as can be seen on the dashboard. This
allows us to readily know whenever a script fails on a particular platform.

Figure 5: In this figure, we can see the various platforms that the scripts are tested on. This happens on a nightly basis and allows
for multi-platform testing. The right image shows a sample list of scripts being tested and their status on a particular platform.

4.3 Customized workflow - Diff-SPECT processing for epilepsy

Using this framework, customized workflows can be created to enable the development of complex and
streamlined algorithms. In these customized workflows, the output of one algorithm can be used as the input
to another algorithm. Here we present an example of a customized workflow for Ictal-Interictal SPECT
Analysis by SPM Half Normalized (ISASHN) algorithm [2] used to assist image-guided surgery research.
First, two SPECT images are linearly registered to each other and then nonlinearly registered into MNI
(Montreal Neurological Institute) space. The registered images are then masked, smoothed, and intensity
normalized. A t-test is performed comparing these images to a healthy normal population. The resultant
image, containing a t-value at each image location, is thresholded and clustered to produce the final output.

This workflow can be implemented as a single algorithm object with its own GUI and testing protocol that sequentially calls other algorithm objects as presented in Figure 6. The algorithm object can be instantiated from our BioImage Suite VVLink gadget to connect to the BrainLAB Vector Vision Cranial system for integration into neurosurgical research [11]. With the interoperability features that this new framework provides, we can create complex workflows, such as the one presented here, using a graphical tool such as the LONI Pipeline [5].

### 4.4 Interoperability

This framework supports easy interoperability of BioImage Suite components with other software environments. For example, all command line tools (over 90 of them at this point) support the Slicer3 execution interface by providing an XML description when invoked using the `--xml` flag. This allows Slicer to scan the BioImage Suite binary directory and find all its components as plugins – see Figure 7. Similarly we can recognize other command line tools that adhere to this interface and use them as plug-ins within some of the BioImage Suite GUI applications/applets.

In addition (via the use of the `--loni` construct) BioImage Suite components output an XML description that is compatible with the LONI pipeline environment [5].

![Figure 6: Customized workflow using the unified BioImage Suite framework. Here the algorithm modules are depicted in blue (with the actual script name below it). In this workflow, the interictal and ictal SPECT are first linearly registered and output is then non-linearly registered with the MNI Template SPECT. The result of the registration is then processed using various algorithms (mask, smooth and intensity normalized). Then a t-test is performed with the mean and standard deviation from a control population. The output tmap is then thresholded and clustered to get the final output image.](image)

### 5 Conclusion

Our novel framework facilitates easy development of novel CAI algorithms that allows the developer to focus on the development of the algorithm and allows for easy creation of user interfaces and robust testing of the algorithms of multiple platforms. Additionally, customized workflow pipelines have been created by de-
Figure 7: The new framework facilitates interaction with Slicer. Figure A shows the autogenerated BioImageSuite User Interface components in Slicer. Figure B shows BioImageSuite modules being identified and loaded directly into Slicer’s user interface. Figure C shows a Slicer command line module recognized and loaded in BioImageSuite.

developers to allow for creation of complex algorithms. With this framework, we envision a more widespread adoption amongst our research group for rapid development of easy-to-use image analysis algorithms and look forward to other CAI contributions to BioImage Suite.

References


Latest version available at the Insight Journal [http://hdl.handle.net/10380/3078]

Distributed under Creative Commons Attribution License
A DICOM-based streaming service for the Digital Operating Room

Release 0.00

Adrian Vázquez, Rafael Mayoral and Oliver Burgert

July 10, 2009

ICCAS, Universität Leipzig, Semmelweisstraße 14, D-04103 Leipzig

Abstract

In the Digital Operating Room there is a need to support data streaming to create advanced integrated surgical assist systems. In this paper we propose a DICOM-based streaming mechanism which leverages the interoperability definitions offered by DICOM to offer a common interface to manage all kinds of streaming data sources, while allowing data and application-specific protocols and infrastructure for the actual data access. We have implemented the proposed solution within the ASTMA project and have shown that thanks to the flexibility in choosing an appropriate streaming protocol we can achieve the necessary streaming quality while transmitting the context information required to create valid DICOM instances. This approach ensures an early integration of streaming data with the rest of the imaging information providing for a simpler data workflow.

Contents

1 Introduction 1
2 DICOM-based implementation of the general framework 2
3 The ASTMA project implementation 4
4 Conclusions 8

1 Introduction

In surgery, the wish to tackle more complex and information-intensive tasks and to optimally use existing resources demands tighter integration of existing and future surgical assist systems [1]. This integration ultimately creates a distributed surgical assist system whose functionalities are obtained by combining the appropriate components. Such integration represents a departure from the still today common situation of isolated monolithic surgical assist systems, exchanging more-or-less independent and self-contained units of information, which are processed once the complete data set has arrived. The new class of assist system, with distributed functionalities and tighter integration, will require increased continuous data transfer and processing among its components. To handle this type of data transmission, the system will have to support streaming of continuous data.
In [2] a general framework for the integration of data streams into the Digital Operating Room (DOR) has been presented. The approach provides a two-level design in which the management and supervision of the data producing and consuming devices is independent of the actual mechanisms used to transmit that data. This approach allows for the use of infrastructure and transmission mechanisms specially adapted to the specific needs of the streamed data. The applications wishing to use streaming services are offered a consistent high-level interface to set-up and control the streamed data. Figure 1 shows the UML deployment diagram depicting this situation (taken from [2]).

In this paper we present a data streaming solution for the DOR based on the DICOM standard. This solution can be seen as an instance of the general framework presented in [2]. The mechanisms for interoperability offered by DICOM are used to implement the management layer, while appropriate protocols and infrastructure are employed for the data access layer. Already available methods in DICOM allow establishing a reference from the management layer to the specific protocols and infrastructure used for data access. We have implemented this solution in the context of the research project Automated Soft Tissue MAnipulation with mechatronic assistance using endoscopic Doppler guidance (ASTMA).

2 DICOM-based implementation of the general framework

Several technology alternatives are readily available to implement a streaming solution for distributing continuous signals in the DOR. However, in most cases the appropriate technology depends on the nature of the data being transmitted. Implementing such a system within DICOM would provide a common layer for the different technologies. Additionally, we believe that in surgery it would be of advantage to capture the data within DICOM as soon as possible, creating valid DICOM instances, to allow for easier integration of the streaming information with the rest of the imaging data into a richer patient model. Furthermore, keeping the streaming data within DICOM guarantees a smoother data workflow.

An analysis of the DICOM standard shows that it possesses the concepts necessary for implementing the general framework: (1) A client/server architecture (Service Class User/Service Class Provider); (2) A mechanism for querying and modifying capabilities of network peers (N-GET, N-SET) (3) A mechanism for reporting current status and changes (N-EVENT-REPORT) (DICOM standard part 7).
DICOM [3] offers services and objects, which are organized into Service-Object Pairs (SOP), with relationships among them to create a model of the real world corresponding to a specific service. A direct analogy with the existing Print Service Management Class (DICOM Standard part 4) can be established to guide the implementation of a Streaming Management Service Class.

The output of the Print Management Service Class is a Print Job. Such a job is a set of Film Sheets each of them containing zero or more image boxes holding an image and annotations. If we take a film sheet containing many image boxes and order them sequentially, and instead of printing them, we show them on a display one by one at a fixed rate, we will have a video stream. Figure 2 shows this analogy by relating the SOPs of the Print Management Service Class to SOPs of the proposed Stream Management Service Class. The concept of frame can easily be generalized to any kind of data yielding the ability to transmit streams of any kind.

The Stream Device has a Device Description. This description is a snapshot of the hierarchy of objects deployed in the Stream Device (figure 2). One Stream Session holds information common to all Stream Channels it manages, e.g. stream session type, number of stream channels. Each Stream Channel is defined by the network information needed for streaming, e.g. network protocol, routing scheme. One Stream Channel is related to only one Stream Container. The Stream Container’s attributes describe how to encode, decode and present each Stream Frame. Finally, the Stream Frame defines the content each transmitted frame should have which basically consists of an acquisition time stamp and the actual data, e.g. frame pixels in the case of video frame.

In general, each transmitted frame is an instance of the stream frame that defines it. It is encoded and decoded according to the attribute values specified in the stream container it belongs to. It is transmitted using the network information of the stream channel its container is related to and according to the specifications of the corresponding stream session.

---

Latest version available at the Insight Journal link http://hdl.handle.net/1926/1338
Distributed under Creative Commons Attribution License

**Figure 2** Analogy established between the DICOM print service and the proposed DICOM-based streaming service.
As it is also shown in figure 2, the Stream Session, Channel and Container, together with the Device Description build the static part of the model of the world: their values do not change during the execution of the application. Moreover the Device Description is sent from the server to the client to expose its streaming capabilities. Additionally, the device description contains the context of the transmitted frames, e.g. equipment information, transformations, annotations as required to create DICOM valid instances. On the other hand, the Frame constitutes the dynamic part of the model of the world: the frame contains the actual data changing continually over time.

The DICOM standard specifies that communication between DICOM Application Entities (AE) should be accomplished via the TCP/IP protocol (DICOM Standard part 8). In our case this is appropriate for the management channel, since the device description and other event notifications will only be exchanged sporadically. For this, we use the offered DICOM Message Service Elements (DIMSE): N-GET, N-SET, N-ACTION and N-EVENT-RPT.

However, for the data access channel, implementing the actual streaming, this protocol is not the best option. The two-layered structure of the general framework (figure 1) allows for using specific protocols, supporting real-time transfer, for the actual data access. The DICOM standard supports such concept since the introduction of supplement 106 [4]. In that supplement a mechanism for streaming very large still pictures was introduced which exploits the concept of indirection: instead of including the real pixel data, the DICOM image contains a reference to the URL where the source of the data can be found. The required data can then be streamed as needed, avoiding the transmission of a very large file. In supplement 106 it is further specified that the protocol to be used for such a streaming transmission is the JPEG 2000 Interactive Protocol.

In our case, we borrowed this idea to specify that the protocol for accessing the real data is user-defined, but must be specified as an attribute of the stream device description, specifically under the Stream Channel structure. This structure also contains an attribute for the location of the data. The client can then obtain the actual data from that source using the specified protocol.

3 The ASTMA project implementation

We have used the introduced DICOM-based streaming solution within the research project Automated Soft Tissue Manipulation with mechatronic assistance using endoscopic Doppler guidance (ASTMA).

The ASTMA system is a distributed system whose purpose is to semi-autonomously assist the surgeon during the harvesting of the internal thoracic artery (ITA) in coronary artery bypass grafting (CABG) surgery. The mechatronics component, with a combination of ultrasound probe and mono-polar blade electrode as end effector, follows the ITA, guided by intraoperatively obtained ultrasound Doppler images. Ultrasound images and control information are continuously sent over the network to a control station for processing.

Figure 3 shows The ASTMA system with its main components: 1) the control station; 2) the combination of ultrasound probe and mono-polar blade electrode, mounted on a universal medical robot.

Latest version available at the Insight Journal link http://hdl.handle.net/1926/1338
Distributed under Creative Commons Attribution License
From the streaming point of view the mechatronics system can be seen as a provider of video and control streams. The control station plays the role of stream consumer; it receives both stream types to combine them in a way that guarantees temporal consistency. Furthermore DICOM secondary capture image and multiframe objects can be stored and retrieved.

The component diagram of the ASTMA system is shown in figure 4. As mentioned before, the StreamServer component is deployed on the mechatronics system, whereas the StreamClient is deployed on the control station. ASTMA implements three main interfaces: Management, Video Access and Control Access. The management interface defines a new DICOM Service Class called “DICOM Stream Management Service Class”. It is responsible of the communication establishment and negotiation between the ASTMA stream user and the ASTMA stream provider as well as of the status reports (it corresponds to the management channel in figure 1). The Video Access interface guarantees the ultrasound Doppler video stream, while the Control Access interface maintains the stream of control frames (they correspond to the data access channel in figure 1). It is worth mentioning that the video data is actually captured via a video converter (Imaging Source converter). Currently, no ultrasound device can provide an image stream with the additional context required. Thus in ASTMA only DICOM secondary captured objects can be created from the video data stream.
As an example of using the DICOM-based streaming within the ASTMA system, we now discuss in detail the important use case “get video stream”: a correctly set-up and functioning process (the primary actor) in the ASTMA control station wishes to obtain the video stream offered by the ASTMA stream provider. Figure 5 shows the UML communication diagram modelling this use case. As a precondition, it is assumed that the client component is aware of the available streaming servers on the network (for this, discovery services such as DICOM-supported DNS Service Discovery could be used).

Figure 4 ASTMA system component diagram.

As an example of using the DICOM-based streaming within the ASTMA system, we now discuss in detail the important use case “get video stream”: a correctly set-up and functioning process (the primary actor) in the ASTMA control station wishes to obtain the video stream offered by the ASTMA stream provider. Figure 5 shows the UML communication diagram modelling this use case. As a precondition, it is assumed that the client component is aware of the available streaming servers on the network (for this, discovery services such as DICOM-supported DNS Service Discovery could be used).

Figure 5 Communication diagram of the ASTMA use case “get video stream data”.

The main steps of the use case are steps four and five. In step four, the client requests the device description (DD) of the server using the DIMSE message N-GET(DD). In step five, the server sends back the DD codified as a DICOM dataset. An excerpt of the codified device description is shown in figure 6. Once the DD is received, the ASTMA client process analyzes it (step six), requests a subscription to the video channel (step seven) and starts getting the video frames (step 11) encoded as Video Frame DICOM.
datasets. The transmission of the actual data frames takes place directly between the VideoStreamSource and the VideoStreamSink over the data access channel.

The analysis of the capabilities of the StreamServer is done on the basis of the obtained device description offered by the mechatronics system. As mentioned before, in the case of the ASTMA project, the server maintains two streams: a video stream and a control stream. Figure 6 shows, for the ASTMA project, the most important attributes contained in the description. The attribute StreamSessionType determines the type of data contained in the stream (VIDEO or DEVICE_CONTROL). This would be the first attribute checked by the client to decide which stream session to use. Focusing on the video stream and further down in the hierarchy (compare figures 6 and 2), the structure VideoStreamChannel includes details regarding the protocol for data access (StreamProtocol) and the URL of the data source (ProviderURL). The structure SCVideoContainer informs the client of the video format employed (FrameFormat), the encoding (TransferSyntax) and of the resolution (Rows, Columns). Finally, VideoFrame represents the static structure of the transmitted frames, it lets the client know that each frame will contain an acquisition time stamp (AcquisitionDateTime) followed by the actual pixels.

<table>
<thead>
<tr>
<th>VideoStreamSession</th>
<th>ControlStreamSession</th>
</tr>
</thead>
<tbody>
<tr>
<td>NumberOfChannels: 1</td>
<td>NumberOfChannels: 1</td>
</tr>
<tr>
<td>StreamSessionType: VIDEO</td>
<td>StreamSessionType: DEVICE_CONTROL</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VideoStreamChannel</th>
<th>ControlStreamChannel</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChannelNumber: 1</td>
<td>ChannelNumber: 1</td>
</tr>
<tr>
<td>ProviderURL: server400</td>
<td>ProviderURL: server400</td>
</tr>
<tr>
<td>UserURL: client464</td>
<td>UserURL: client505</td>
</tr>
<tr>
<td>RoutingScheme: UNICAST</td>
<td>RoutingScheme: UNICAST</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SCVideoContainer</th>
<th>ASTMAControlFrame</th>
</tr>
</thead>
<tbody>
<tr>
<td>FrameFormat: DICOM</td>
<td>FrameFormat: DICOM</td>
</tr>
<tr>
<td>TransferSyntax: Little Endian</td>
<td>TransferSyntax: Little Endian</td>
</tr>
<tr>
<td>Rows: 576</td>
<td>Rows: 576</td>
</tr>
<tr>
<td>Columns: 720</td>
<td>Columns: 720</td>
</tr>
<tr>
<td>StreamSource: BROADCAST</td>
<td>StreamSource: BROADCAST</td>
</tr>
<tr>
<td>CalibratedFrame: YES</td>
<td>CalibratedFrame: YES</td>
</tr>
<tr>
<td>PixelSpacing: 0.111 0.111</td>
<td>PixelSpacing: 0.111 0.111</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VideoFrame</th>
<th>TimeStampData</th>
</tr>
</thead>
<tbody>
<tr>
<td>AcquisitionDateTime:</td>
<td>AcquisitionDateTime:</td>
</tr>
<tr>
<td>PixelData:</td>
<td>PixelData:</td>
</tr>
</tbody>
</table>

Figure 6 Excerpt of the ASTMA stream server description. Only the most relevant attributes are shown.

Using this proposed scheme we have been able within the ASTMA system to transmit video and control streams with the desired quality to allow for detection (using image processing) and following of the ITA. Additionally, since the required context information is exchanged during the transmission of the device description and is available to the client before the actual data are streamed, we can directly create valid DICOM instances thus simplifying the data workflow and making tighter integration possible.

Latest version available at the Insight Journal link http://hdl.handle.net/1926/1338
Distributed under Creative Commons Attribution License
4 Conclusions

In the Digital Operating Room there is a need to support data streaming to create advanced integrated surgical assist systems. In this paper we propose a DICOM-based streaming mechanism which leverages the interoperability definitions offered by DICOM to offer a common interface to manage all kinds of streaming data sources, while allowing data and application-specific protocols and infrastructure for the actual data access.

We have shown the feasibility of such an approach by implementing this solution on the distributed ASTMA system. The ASTMA system relies on the real-time transmission of video and control streams using RTP [5] and UDT [6] protocols over a conventional Ethernet network. Both streams are managed through the same management channel based on a common device description and on DIMSE messages.

Further work in the ASTMA implementation of the streaming solution will involve having more channels per session, for example offering different video resolutions or different data sampling frequencies, and making the client dynamically change channels during the execution to test the server internal administration of resources.

Acknowledgements

The authors would like to thank Dr. Werner Korb for his support in the ASTMA project and Arun Voruganti for his assistance with the details of video handling.

ICCAS is funded by the German Federal Ministry of Education and Research (BMBF) and the Saxon Ministry of Science and Fine Arts (SMWK) within the scope of the Unternehm Region Program with grant numbers 03 ZIK 031 and 03 ZIK 032. The ASTMA project is financed by the German Research Foundation with the grant number DFG KO3554/1-1.

References

matVTK - 3D Visualization for MATLAB

Erich Birngruber\textsuperscript{1,4}, René Donner\textsuperscript{1,2}, and Georg Langs\textsuperscript{3,1}

\textsuperscript{1} Computational Image Analysis and Radiology Lab, Department of Radiology, Medical University of Vienna, Vienna, Austria, erich.birngruber@meduniwien.ac.at.
\textsuperscript{2} Institute for Computer Graphics and Vision, Graz University of Technology, Graz, Austria.
\textsuperscript{3} Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, USA.
\textsuperscript{4} Pattern Recognition and Image Processing Group, Vienna University of Technology, Vienna, Austria. *

Abstract. The rapid and flexible visualization of large amounts of complex data has become a crucial part in medical image analysis. In recent years the Visualization Toolkit (VTK) has evolved as the de-facto standard for open-source medical data visualization. It features a clean design based on a data flow paradigm, which the existing wrappers for VTK (Python, Tcl/Tk, Simulink) closely follow. This allows to elegantly model many types of algorithms, but presents a steep learning curve for beginners. In contrast to existing approaches we propose a framework for accessing VTK’s capabilities from within MATLAB, using a syntax which closely follows MATLAB’s graphics primitives. While providing users with the advanced, fast 3D visualization capabilities MATLAB does not provide, it is easy to learn while being flexible enough to allow for complex plots, large amounts of data and combinations of visualizations. The proposed framework will be made available as open source with detailed documentation and example data sets.

1 Introduction

Fueled by the increased interest in medical imaging research in recent years the importance of powerful frameworks covering feature computation, registration and segmentation as well as visualization have risen considerably. Researchers depend on efficient ways to formulate and evaluate their algorithms, preferably in environments facilitating rapid application development.

The Visualization Toolkit (VTK) \cite{1} has proven to deliver high quality, efficient visualization for a variety of fields including general purpose GUIs \cite{2} as well as medical imaging \cite{3,4}. Similarly, the Insight Toolkit (ITK) \cite{5} has provided a framework for medical image registration and segmentation. It features

\* This work has been supported by the Austrian National Bank Anniversary Fonds project 12537 COBAQUO.
Fig. 1. The figure shows examples for some of the graphical primitives available through matVTK: (a) Points showing different colors, sizes and opacities, connected with a line plot (b) matVTK’s equivalent to MATLAB’s built-in `quiver` function (c) Volume rendering of medical data including axes with labels, a grid and annotations for individual points.

A data-flow driven design paradigm which models algorithms as filters which are concatenated and transform source (input) data into results (output). It is implemented in C++, providing high throughput, and provides wrappers for selected scripting languages. Recently, the project VTK Edge [6] is actively investigating the use of GPU acceleration for complex visualization tasks which cannot be modeled in OpenGL.

While VTK’s data-flow approach in combination with C++ is very flexible its utilization imposes a steep learning curve on the user. Furthermore, in medical imaging and computer vision MATLAB is one of the major development platforms, especially in the academic field. MATLAB itself provides unique capabilities for rapid application development, but severely lacks state of the art 3D visualization features. There are no volume rendering methods in MATLAB, and basic operations like isosurfaces are very slow. Thus, for visualizing medical data along with meta-data like segmentation results, classification probabilities or the like, external toolboxes have to be used. Recently, a Simulink based approach [7] was proposed which automatically wraps VTK’s functionality in Simulink blocks. While this allows to graphically structure a VTK plot in MATLAB, it still requires the user to get familiar with the internal concepts of VTK. The user’s knowledge of MATLAB’s built-in plot commands is not exploited. For ITK there exists a wrapper, matITK [8], which forgoes ITK’s complexity by providing simple MATLAB MEX commands for the most commonly used functionality. This gives MATLAB users an effective tool for performing most operations directly from within MATLAB without having to worry about data conversion, data-flow formulations and result structures.
**Contribution** Because of VTK’s relevance to the computer vision community using MATLAB, we propose a framework to model VTK’s functionality in a way which is similar to MATLAB’s graphics concept, minimizing learning efforts by the user while providing VTK functionality in a flexible manner. The proposed solution is available as open source and we are working towards making it available as an additional wrapper distributed with future VTK versions. matVTK provides a rapid integration of VTK functionality in MATLAB. It is a necessity for the analysis of large and complex data, and the visualization of algorithm outputs, in the medical image analysis domain.

The paper is structured as follows: In Sec. 2.1 we outline the data-flow paradigm employed by VTK along with its internal properties which are relevant for our approach. Sec. 2.2 discusses how MATLAB’s MEX interfacing concept can be used to interact with external libraries. In Sec. 3 we detail our approach with the presented in Sec. 4, followed by conclusion and outlook in Sec. 5.

2 Foundation & Methods

In the following the technical basis for our approach is outlined: VTK’s internal design and the MEX interface provided by MATLAB. Based on these components we will explain the matVTK framework in detail in Sec. 3.

2.1 VTK

VTK is a visualization library written in C++. VTK can be used to draw geometric primitives such as lines and polygon surfaces as well as render volume visualizations. Furthermore it allows fine grained control of the combination of these primitives in a scene.

The first thing noticeable to the programming end user is the data-flow based “Pipes and Filters” design pattern, used to concatenate various data processing methods. To be able to combine different and multiple filters and preprocessing steps all these algorithms share a common interface, providing their fundamental functions `SetInput(in)` and `GetOutput()`. The `SetInput` method accepts the input data, the `GetOutput` method provides the processed result. The return value of `GetOutput` can then again be used as input to another algorithm. At the front of such a filter chain there is always a reader or general source that provides the initial data. The last link in the chain is a sink that either renders the output on the screen or saves it into a file. However, this concept is more sophisticated than simply handing over the whole data set from filter to filter. On the contrary, the filter pipe helps to compute multiple steps at once and time stamps in the pipe allow to only recompute the parts of the pipeline affected by changes in source code or parameters. This allows for an economical memory footprint and fast visualization even if the underlying data changes during scene rendering.

Although VTK provides a clean and well performing code base, the first time or casual user may be overwhelmed with the software design concept and the complexity of the large API.
2.2 MATLAB MEX Interface

MATLAB can be extended using the MEX API, which is the name of the C/C++ and Fortran API for MATLAB. The API can be used to manipulate workspace data, the workspace itself or use the MATLAB engine in external applications.

The following focuses on the properties relevant to implementing a framework which integrates VTK into MATLAB. When thinking about a MATLAB framework and plotting / rendering, two important properties come to mind: First memory management should exhibit a small footprint – however the MATLAB principle of non-mutable input data must not be violated. Secondly, we need to maintain an internal state over several function calls, which is accomplished using a handle based approach.

MATLAB comes with its own memory management component and therefore its own implementations of alloc() and free() functions. This way the software can ensure to release allocated memory even in the case of an error during a function call, when the program flow does not reach calls to free(). This means that any handle implementation must respect these cleanup calls in order to destruct/deallocate correctly in case of such an event.

As the MEX API does not provide functions to implement handles returned to the MATLAB workspace [9] was used which provides a C++ template implementation for returning an object pointer back to the MATLAB workspace. Furthermore the code includes checks to verify that a correct handle was passed before using the pointer.

A particular problem arises on Unix platforms using the X11 window system, as MEX functions are intended to be used as single threaded and blocking calls. Because the API does not allow for interaction with the window management, this causes two problems: The MATLAB GUI is blocked during the use of a user interactive window. Even more important, secondly, closing the window via the GUI shuts down the complete MATLAB session. This does not occur on windows platforms. We are uncertain whether a solution for this problem exists, as in depth research showed this to be a problem deeply rooted in internal X11 design.

3 matVTK Framework

In this section, based on the above overview of the underlying environment, the proposed approach of providing a visualization API which closely follows MATLAB’s plotting concepts is detailed. The three main building blocks are 1. The pipeline handle which establishes the rendering window, 2. the config interface that manages the parameters of the individual plots, and 3. the graphics primitives that provide the actual plotting functionality analogous to standard MATLAB commands.

The Pipeline Handle The VTK handle forms the core of the framework. Its main purpose is to keep track of the data sets currently used in a scene. The second most important task is the management of the render window that is used
to interactively display a scene. Additionally, it controls global components that cannot be kept in a single function or that are relevant for multiple functions. The handle is either generated automatically, or can be acquired by `handle = vtkInit()`. It can automatically delete itself and all its dependencies in case MATLAB is closed or all MEX memory is freed using `clear mex`. Handles can be used implicitly, or explicitly to be able to work with multiple scenes. On each of the various function calls, if no explicit handle is given, it is checked whether there exists a default handle or not. In the latter case it is automatically created and reused in subsequent calls. When using an explicit handle it must be created and destroyed by the user. Furthermore it must be handed to each plot function as the first argument.

The Configuration Interface VTK offers fine grained control of parameters for basically three steps: filters, scene components (actors) and the global scene (i.e. camera settings). For this reason a large amount of setters and getters exist in various places of VTK’s class hierarchy. For the uninitiated VTK user this is complex and opaque. This is why we decided to implement a flexible configuration interface that can model the complex VTK design. The first, and implementation wise simple approach is using a MATLAB struct. It is a data structure that consists of named fields and values (e.g., `config.opacity = 0.3`). Constant configuration parameters can be easily reused by the user. Alternatively, the typical MATLAB concept for the configuration of parameters - a list of string/value pairs - is also supported, as well as a combination of the two. When using both, the MATLAB style can be used to override settings in the configuration struct.

Graphics Primitives MATLAB uses the `plot` function for different kinds of primitives - lines and points. For our framework this did not seem feasible, which is why we decided to use the prefix `vtkplot` and a specific suffix for points, lines etc. Currently the framework supports primitives for points (represented as small spheres), lines, polygon meshes, vector fields, tensors, volumes, cut planes through volumes and isosurfaces. Also provided are functions gaining a better overview of the scene, such as functions for labels, legends and plot titles. These primitives can be easily combined to create complex scenes as we will show in the following section.

Additional Functionality As the simple combination of graphical primitives does not always meet the users needs, there are special features available including scene export. Screenshots of a scene can be created and saved as PNG images, including the support for high resolution images for printing. The `vtkShow` function, used to display the scene window, returns the camera settings at the moment of closing the window, which can then be reused as configuration parameters to restore or reproduce global scene settings over multiple plots. Another valuable feature is the ability to cut away certain parts of the scene. For this operation a box widget – i.e. six perpendicular planes – or a single plane widget are available. The cropping operation can be applied to scene primitives of the users choosing and therefore provide the best possible insight into the displayed data. Fig. 4 (a) shows an overview of the interaction of the matVTK
components among each other as well as their interaction with the MATLAB userspace.

4 Scenarios

In the following we demonstrate the proposed framework and the plot types available. All types of plots can be arbitrarily combined within one plot.

- In Fig. 1 (a) the ability to easily render primitives with different properties is shown. The level of detail for rendering spheres / tubes can be chosen to allow for either exact representation or coarser approximation which lets the user plot up to several ten thousand 3D points at interactive frame rates.
- Fig. 1 (b) demonstrates the superiority when plotting vector fields, color coding and scaling the arrows according to vector length
- Fig. 1 (c) shows the labeling available for individual points in space as well as for the axes. Grids with arbitrary spacing as well as orientation widgets (box, arrows) are available.
- The superposition of meshes onto the volume is shown in Fig. 2 (a). Rendering this view from the MATLAB command line takes below one second and the resulting view can be rotated and cropped interactively even on medium class hardware.

Fig. 2. (a) Volume rendering combined with a mesh representing a segmentation result (b) Additionally showing scalar values mapped onto the mesh and planes cut through the volume. Note how the cropping only affects the volume and the cut planes, but not the mesh (see code example in Fig. 3).
v = vtkplotvolume(volume, 'builtin1');
vtkplotmesh(vertices, faces, vertexLabel)
p = vtkplotcutplanes(volume, planePoints, planeVecs);
vtkcrop(v, 'plane')
vtkcrop(p)
vtkshow('backgroundColor', [1 1 1])
vtkdestroy

Fig. 3. matVTK code example for Fig. 2 (b)

– Finally, Fig. 2 (b) shows the possibility to map scalar values onto the surface of a mesh and display additional views of the volume along cut planes, while cropping the whole volume (with or without the other actors). The cut planes / the crop box can be set both programmatically as well as interactively.

Fig. 3 shows the MATLAB code to create a plot, using an implicit handle, as only one plot is needed. First the user plots a 3D matrix as volume, using the builtin color map “builtin1”. Next a polygon surface is plotted using its vertex coordinates and a triangulation. The labels in vertexLabel are used to map scalar data to the surface. The function vtkplotcutplanes creates several planes at the given points, using the normal vectors planeVecs. vtkcrop uses the handles returned from the plotting functions, to decide which parts of the scene can be clipped with the interactive crop widget. After displaying the assembled scene with vtkshow the resources are freed with vtkdestroy. The call vtkshow also demonstrates the configuration interface, setting ’backgroundColor’ to the RGB value of white.

**Performance** Fig. 4 (b) shows a comparison of rendertime for cubic volumes with increasing sidelength. The graph shows pure MATLAB and matVTK performance for extracting iso surfaces from a chest CT dataset, sampled at 100×100×100 to 250×250×250. The time for the entire visualization is up to 60 secs. for a pure MATLAB implementation, and up to 4 sec. for the matVTK command (up to 15× faster). The main reason for this is the fast implementation of VTK and its use of OpenGL hardware accelerated graphics rendering.

5 Conclusion and Outlook

We propose an approach to wrap VTK’s main capabilities using an easy to use, efficient framework for the MATLAB programming environment. It provides MATLAB users with state of the art 3D volume rendering and visualization features while retaining MATLAB’s ease of use. Even complex medical visualizations can be assembled in few lines of code, without knowledge about VTK internal data-flow paradigm. The framework exposes VTK’s most relevant features while being easily extendable.
Future work will focus on two areas, namely the inclusion of additional visualization and interaction features and to improve the internal structure of our framework. While user interaction in the VTK window with the output being forwarded to MATLAB is definitely possible and is already being used in several cases, it is not yet fully covered by the framework. Saving widget states to return to a previous visualization will be added, as well as animations and movie export functionality. Handles, which are currently used only internally, will be exposed to the user to be able to selectively remove parts of the scene or control their visibility. Streaming visualization output via state of the art codes is another topic which is currently investigated.

The framework is a valuable tool in the design and application of complex algorithms to medical imaging data, the interactive navigation and data exploration, and can serve as a basis for a high level interface design.

References

1. Visualization Toolkit (VTK), Kitware: http://www.vtk.org/
5. Insight Segmentation and Registration Toolkit (ITK): http://www.itk.org/
6. VTKedge: http://www.vtkedge.org/
An open-source real-time ultrasound reconstruction system for four-dimensional imaging of moving organs

Release 1.00

Danielle F. Pace\textsuperscript{1,2}, David G. Gobbi\textsuperscript{3}, Chris Wedlake\textsuperscript{1}, Jan Gumprecht\textsuperscript{4}, Jonathan Boisvert\textsuperscript{5}, Junichi Tokuda\textsuperscript{4}, Nobuhiko Hata\textsuperscript{4} and Terry M. Peters\textsuperscript{1,2}

August 16, 2009

\textsuperscript{1}Imaging Research Laboratories, Robarts Research Institute, London, ON, Canada
\textsuperscript{2}Biomedical Engineering Graduate Program, The University of Western Ontario, London, ON, Canada
\textsuperscript{3}Atamai Inc., Calgary, AB, Canada
\textsuperscript{4}Surgical Planning Laboratory, Brigham and Women’s Hospital, Boston, MA, USA
\textsuperscript{5}Institute for Information Technology, National Research Council Canada, Ottawa, ON, Canada

Abstract

Four-dimensional imaging is a necessary step in computer-assisted interventions (CAI) performed on moving surgical targets, such as those located within the beating heart or influenced by respiratory motion. Ultrasound (US) is often the best imaging modality for this application, as it provides good contrast and spatial resolution while remaining simple to integrate into the operating room. However, conventional 2D US imaging is often insufficient for preoperative planning and surgical guidance, real-time 3D ultrasound imaging using matrix array probes has a smaller field of view and lower spatial resolution, and 3D volumes created using 3D US reconstruction suffer from motion artifacts. As an alternative, gated 4D ultrasound reconstruction using a tracked 2D US probe is a promising technology. In this paper, we present SynchroGrab4D to the open-source community. SynchroGrab4D performs 3D and 4D US reconstruction in real-time while visualizing the output volume(s) within an OpenIGTLink-compliant CAI system, such as 3D Slicer. Also included are VTK classes that perform ECG-gating and that interface with several commercial tracking systems, as well as a “4D Imaging” module in 3D Slicer. Our open-source imaging system can be used to collect the 4D image data required for computer-assisted surgical planning and intraoperative guidance for a variety of moving organs.

Contents

1 Introduction 2
2 Real-time 4D ultrasound reconstruction 3
1 Introduction

Initial development of computer-assisted intervention (CAI) systems focused on the orthopaedic and neurosurgical fields, since assuming that the organ of interest (whether a bony structure or the brain encased within the skull) is a rigid body made superimposing preoperative images onto the intraoperative patient much simpler. The success realized in these surgical subspecialties has paved the way for today’s interest in more complicated procedures, including surgery performed on soft tissue and on moving organs. Focusing on mobile tissue in particular, investigators are making inroads in all aspects of CAI, including diagnosis, modeling and preoperative planning, image registration and intraoperative guidance. For example, systems are under development for minimally-invasive therapy performed on the beating heart [9] and for respiratory motion compensation during both lung tumour radiotherapy [8] and liver tumour ablation [1]. In virtually all CAI systems for moving organs, four-dimensional imaging (which yields a time series of 3D volumes depicting the organ’s deformation over time) is required at some point in the surgical workflow.

Ultrasound (US) has sufficient soft tissue contrast and spatial resolution for many applications while remaining low-cost and ubiquitous within operating rooms. US is therefore often the imaging modality of choice as an alternative to magnetic resonance imaging (MRI), computed tomography (CT) or fluoroscopy, especially for intraoperative imaging. Ultrasound probes conventionally image in two dimensions, but 2D information is rarely enough to complete an interventional task. Two alternatives exist for volumetric US imaging of moving organs: real-time 3D (RT3D) ultrasound imaging with a matrix array probe [13] and performing gated 4D ultrasound reconstruction by tracking a standard 2D US probe. 4D US reconstruction extends the well-established 3D US reconstruction technique [4] and generates a time-series of 3D US volumes by using gating to collect multiple 2D US frames acquired at the same point in the organ’s motion cycle and compositing them into 3D volumes using the probe’s pose as measured by the tracking system. When imaging moving organs, 4D US reconstruction with gating is needed instead of 3D US reconstruction, as volumes generated using the latter will typically contain significant motion artifacts. Compared to RT3D ultrasound, 4D US reconstruction offers advantages with respect to field of view and spatial resolution, uses widely available equipment and does not suffer from difficulties in streaming 3D data from closed US machines.

An advanced form of US reconstruction is real-time reconstruction, where the 2D US frames are inserted into the output volume(s) as they are acquired, enabling interactive visualization of the incremental reconstruction and ensuring that the volume(s) contain no “holes” and cover all of the region of interest. This is especially valuable when performing 4D US reconstruction, which is more technically challenging compared to 3D US reconstruction due to the extremely slow probe speed required. For our application in minimally-invasive beating-heart surgery, we have developed and validated a real-time 4D US reconstruction system [10] based on our group’s previously described real-time 3D US reconstruction approach [6].

3D and 4D ultrasound reconstruction has been integrated into commercial ultrasound systems for quite some time, but the inaccessibility of the resulting raw image data makes these systems unsuitable for research (as opposed to clinical) use. Some highly-specialized open-source software can perform both non-gated and
gated ultrasound reconstruction, including the well-known Stradx system [15], but usually cannot perform subsequent image registration, segmentation, advanced visualization (such as volume rendering) or data transfer for integration into surgical guidance systems. In contrast, 3D Slicer [11] is an actively growing open-source application that bundles an extensive collection of medical image processing and image-guided therapy modules. However, despite current research interest, 4D imaging and image processing algorithms are currently limited in 3D Slicer and in other open-source medical imaging packages.

In this paper, we present SynchroGrab4D, open-source software for the interactive acquisition and visualization of reconstructed 3D and 4D ultrasound data using 3D Slicer. We take advantage of several open-source software solutions, including the Visualization Toolkit (VTK) [12], the OpenIGTLink network transfer protocol [14], and a previously described open-source 3D US reconstruction system [2]. As part of our approach, we also describe VTK classes for gating and a “4D Imaging” module in 3D Slicer. Although our particular focus is on 4D US imaging of the beating heart using ECG-gating, our software is easily generalizable to imaging other cyclically moving organs, for example using respiratory gating.

2 Real-time 4D ultrasound reconstruction

SynchroGrab4D performs both 3D (non-gated) and 4D (gated) ultrasound reconstruction and interactively visualizes the results in 3D Slicer. It can be used with both freehand and mechanical scanning, supports multiplanar probes (those with a rotating imaging plane) and implements both prospective and retrospective gating. As our real-time 3D US reconstruction approach has been previously described [6, 2], here we focus on 4D reconstruction. An example 4D ultrasound dataset of a beating-heart phantom reconstructed using SynchroGrab4D is shown in Figure 1, while an implementation of our algorithm within the AtamaiViewer framework [7] has been used to image the beating heart in both porcine and human subjects (Figure 2a, 2b).

Figure 2c illustrates the 4D US acquisition procedure. The operator moves the US probe once per cycle while SynchroGrab4D simultaneously uses gating to divide each cycle into \( N \) phases. The 2D US frames at the beginning of each phase are identified and are inserted into the appropriate output volume using their corresponding pose from the tracking system. In our experience in intraoperative imaging of the beating heart, the acquisition takes approximately four minutes and can be integrated into the surgical workflow quite easily. The accuracy of any US reconstruction approach depends on the quality of the spatial calibration (which determines the transformation between the origin of the US fan in the 2D US frames to the sensor mounted
Figure 2: (a,b) Reconstructed 4D ultrasound dataset representing the beating heart of an intraoperative porcine subject (acquired using a tracked multiplanar 2D transesophageal probe). Gating is performed using the subject’s ECG signal; for this case, a 3D volume was reconstructed (a) at the R-wave and (b) midway between successive R-waves. The dashed line highlights the differences between the image volumes due to the beating heart’s motion over the cardiac cycle; (c) The 4D ultrasound reconstruction process (illustrated with a rotational acquisition approach; \( N = 3 \)).

on the US probe) and temporal calibration (which determines the lag time between 2D US frames and their corresponding transforms matrices from the tracking system). We have shown that our reconstruction technique has RMS errors in object localization measuring approximately 1.5 mm for 3D reconstruction and 2.6 mm for 4D reconstruction [10], which is within range for the spatial calibration technique used [5].

SynchroGrab4D is written in C++ using VTK 5.2, uses OpenIGTLink to send images and transform matrices to 3D Slicer 3.4 and is fully cross-platform (the source code is included in this submission and a simplified class diagram is provided as a supplemental figure). Seven threads perform the following tasks: (1) grabbing and timestamping 2D US frames; (2) retrieving and timestamping transform matrices from the tracking system; (3) performing automatic phase detection from an incoming ECG signal; (4) performing US reconstruction by updating pixels within the output volume(s); (5) buffering the output volume(s) for transfer to 3D Slicer; (6) sending OpenIGTLink messages of the output volume(s); and (7) sending OpenIGTLink messages of the transform matrices (optional). Synchronization between the 2D US frames and the transform matrices from the tracking system is accomplished using the aforementioned timestamps and the temporal calibration results, while the lag in receiving the ECG signal is assumed to be negligible.

Video capture 2D US frame grabbing, timestamping and buffering is based on the vtkVideoSource-derived VTK classes, which have been refactored to provide a clearer representation for a single frame, the circular buffer of frames and the frame grabbing mechanism (vtkVideoFrame2, vtkVideoBuffer2 and vtkVideoSource2, respectively). Although we used VTK, please note that frame grabbing has recently been added to IGSTK as well [3]. SynchroGrab4D supports frame grabbing with Video-for-Windows and Matrox (Montreal, QC, Canada); adding support for Linux and the open-interface Sonix RP ultrasound system (Ultrasonix, Vancouver, BC, Canada [2]) is in progress. Although the original VTK frame grabbing classes always output the most recently acquired frame, in the new classes we provide the ability to specify a “desired” timestamp so that the filter output is the frame whose timestamp is closest to the desired timestamp.

Ultrasound probe tracking SynchroGrab4D’s tracking classes are from Atamai Inc. (Calgary, AB, Canada) and store transform matrices with their timestamps in a circular buffer as they are retrieved from one of the

Latest version available at the Insight Journal [http://hdl.handle.net/10380/3083] Distributed under Creative Commons Attribution License
Figure 3: (a) `vtkSignalBox` uses the duration of the previous cycle to predict the current phase (int output), which may differ from the true phase (dashed lines). Once the next cardiac cycle has begun, the true beginning of each phase can be calculated retrospectively; (b) 4D US reconstruction: the arrow indicates the current time (at the predicted beginning of phase 3). Prospective gating uses the current timestamp (solid green line, compare to the true timestamp shown by the dashed green line). The solid navy line shows the timestamp used by retrospective gating.

following tracking systems: NDI’s Aurora, Polaris or Certus (Waterloo, ON, Canada), Ascension’s Flock of Birds (Burlington, VT, USA) or Clarion Technology’s Micron Tracker (Toronto, ON, Canada). We also provide a tracking simulator (`vtkFakeTracker`) for testing when a tracking system is not available.

Gating An integral part of SynchroGrab4D is the gating subsystem, which currently operates on ECG signals but could be easily modified to perform other forms of gating, such as respiratory gating. Our ECG-gating software performs automatic R-wave detection from patient ECG signals and splits the measured cardiac cycle period into a user-specified number of equally-spaced phases. Prospective gating provides real-time numerical output estimating the current cardiac phase and heart rate, while retrospective gating can determine the true time at which each phase began in the previous cycle (Figure 3a). The base class, `vtkSignalBox`, is an ECG-gating simulator with a user-specified cardiac cycle period. The derived class `vtkHeartSignalBox`, which was originally designed for use with a beating-heart phantom (The Chamberlain Group, Great Barrington, MA, USA) that outputs a voltage pulse at the beginning of each cardiac cycle, gates based on a 5V pulse arriving at the machine’s parallel port. The second derived class, `vtkECGUSBBox`, reads a patient ECG signal (amplified to the volt range) over a USB connection. Automatic R-wave detection can then be accomplished by applying a user-specified threshold. We have used both `vtkHeartSignalBox` and `vtkECGUSBBox` to interface with clinical anesthesiology equipment within the operating room. The following code snippet illustrates the use of these classes:

```cpp
vtkSignalBox* sb = vtkSignalBox::New(); sb->SetNumberOfPhases(5); // phase shift at 0%, 20%, 40%, 60% and 80% R-R sb->Start(); int phase = sb->GetPhase(); // current phase as an integer in range [0,4] float rate = sb->GetBPMRate(); // current heart rate in beats per minute double curr = sb->GetTimestamp(); // timestamp (TS) of most recent measurement double ret = sb->CalculateRetrospectiveTimestamp(3); // TS at 60% R-R for the previous cycle sb->Stop();
```

4D ultrasound reconstruction Ultrasound reconstruction in SynchroGrab4D is performed by two classes: `vtkFreehandUltrasound2` implements real-time 3D ultrasound reconstruction as described in [6], while the derived class `vtkFreehandUltrasound2Dynamic` incorporates prospective and retrospective gating for real-time 4D ultrasound reconstruction. In the most basic configuration,
vtkFreehandUltrasound2Dynamic has \( N \) image outputs (where \( N \) is the number of phases of the associated vtkSignalBox) each depicting the beginning of a cardiac phase. SynchroGrab4D can also reconstruct a user-specified subset of phases to allow greater flexibility in the points of the cardiac cycle that are represented by the output volumes. The 4D ultrasound reconstruction algorithm proceeds as follows:

If gating subsystem shows a change from phase \( i \) to phase \( j \) and phase \( j \) corresponds to an output volume:

1. Get gating timestamp \( t_g \): Prospective gating – use most recent timestamp from the gating subsystem. Retrospective gating – find actual timestamp at which phase \( j \) began in the previous cycle (Figure 3b).
2. Get 2D US frame whose timestamp \( t_f \) is closest to \( t_g \): Prospective gating – decide between most recent frame and the next frame. Retrospective gating – retrieve frame with best timestamp from frame buffer.
3. Calculate transform matrix timestamp \( t_t \) corresponding to \( t_f \): subtract lag from temporal calibration.
4. Interpolate the transform matrix for timestamp \( t_t \) from the transform matrix buffer.
6. Insert 2D US frame into output volume for phase \( j \) using transform matrix, splatting with nearest-neighbor or trilinear interpolation and compositing frames with alpha blending or compounding [6].

Since variations in patient heart rate can introduce additional artifacts in 4D US reconstruction, one might not want to insert frames acquired during erratic heart beats into the output volumes. SynchroGrab4D can determine an expected heart rate by calculating the mean heart rate over a user-specified duration, while ensuring that the heart rate is relatively stable over the measurement time. During the reconstruction, frames acquired when the heart rate differs too much from the expected value are rejected. If using prospective gating, frames tagged for insertion must be saved in memory until the next R-wave, at which point the actual heart rate of when they were acquired can be checked.

Once the reconstruction is finished, hole filling [6] is performed to fill in voxels that did not intersect with a 2D US frame during the reconstruction. The output volume(s), a calibration file describing the parameters used and (optionally) the contents of the transform matrix buffer are then saved. Users can also save the inserted 2D US frames as bitmap images and/or their corresponding timestamps.

3 User interface and communication with 3D Slicer

SynchroGrab4D is designed as a command-line application that sends reconstructed US volumes and/or the transform matrices from the tracking system to a computer-assisted intervention system over a network connection, rather than being integrated within a particular CAI system. Command-line options allow the user to specify a calibration file containing the reconstruction parameters, the type of video grabbing, tracking and gating hardware that is to be used, and the server IP and port number. During real-time reconstruction, the reconstructed volume(s) can be sent at a user-specified rate over the network connection using the OpenIGTLink protocol [14] for interactive visualization within a CAI system. Alternatively, the user can choose to delay the image transfer until after the reconstruction’s completion if the transfer speed is insufficient. The transform matrices from the tracking system can also be sent using the OpenIGTLink protocol to a second port, allowing the US probe’s position and orientation to be visualized by the CAI system as well.

We have focused on visualizing the reconstruction results within 3D Slicer, although SynchroGrab4D can send image data and/or transform matrices to any system supporting the OpenIGTLink protocol. 3D Slicer’s “OpenIGTLink IF” module can manage the incoming image volumes; for 4D US reconstruction their names in the MRML data tree are annotated with the phase that they correspond to. If the user elects to send the reconstructed volume(s) over the network connection during the reconstruction, then the incremental results

Latest version available at the Insight Journal [http://hdl.handle.net/10380/3083]
Distributed under Creative Commons Attribution License
can be visualized within 3D Slicer and the operator can ensure that the reconstructed volume(s) contain all of the desired image data. A new “4D Imaging” module that visualizes 4D image datasets in 3D Slicer is also under active development. The reconstructed 3D volumes making up the four-dimensional dataset can be added to a “4D bundle” within 3D Slicer along with their relative timestamps, and the 4D Imaging module can then display the volumes as a cine loop to visualize the motion of the organ of interest.

4 Discussion and conclusions

Our open-source 3D and 4D ultrasound imaging system performs real-time US reconstruction and visualizes the reconstruction’s progress while imaging any cyclically-moving organ, including the beating heart and organs that are influenced by respiratory motion. 4D US image data can be used as input for CAI systems for diagnosis, image-based modeling and atlas building, and preoperative planning. Our 4D ultrasound reconstruction system is especially useful for intraoperative imaging, as intraoperative MR or CT is often infeasible while fluoroscopy and X-ray do not provide 4D data. Within the operating room, 4D ultrasound images can be used as part of the registration process between preoperative images and the intraoperative patient and for surgical guidance within augmented reality systems. Since our software provides a clear link to 3D Slicer, any of these tasks can be performed using the variety of tools within the 3D Slicer suite.

In our experience, the visual feedback enabled by real-time 4D US reconstruction is invaluable when attempting to acquire reconstructed US volumes of the highest possible quality. Designing a real-time US reconstruction system presents several challenges though, as limitations in both memory and processing speed restrict the number of output volumes, their dimensions and the time difference between them. Since memory for all of the output volumes is allocated at the beginning of the reconstruction, there is a practical limitation to the number and size of the output volumes. Also, SynchroGrab4D will drop frames if the time difference between phases of interest is less than the time required to insert a frame into an output volume. Finally, SynchroGrab4D is currently implemented such that 3D US volumes are sent over the network connection in their entirety, restricting the rate at which they can be updated within the 3D Slicer scene.

Several components of SynchroGrab4D, namely the video grabbing, tracking and gating subsystems, are of general interest as their modular design makes them easily extensible to work with currently unsupported hardware. A variety of ultrasound machines can be used by specifying the appropriate parameters for extracting the US fan from the 2D US frames. SynchroGrab4D’s automatic rotation detection for multiplanar US probes is currently customized for grabbing frames from a Philips Sonos 7500 or 2500 scanner with a Matrox Meteor or Morphis card, but once again this can be modified for different ultrasound machines.

In conclusion, we have presented SynchroGrab4D, an open-source 3D and 4D ultrasound imaging solution that can interface with 3D Slicer for real-time visualization and subsequent processing. Our US reconstruction software can be used to generate 4D image data of moving organs using equipment that is easily integrated into the operating room, while the included ECG-gating classes are of general interest. Given the current development of CAI systems for moving organs, the ultrasound volumes reconstructed using our system have a variety of applications to support diagnosis, modeling and intraoperative surgical guidance.

5 Acknowledgments

The authors would like to thank John Moore, Dr. Andrew Wiles, Dr. Daniel Bainbridge, Dr. Gerard Guiraudon, Dr. Doug Jones and Diego Cantor. This work was supported by CIHR, NSERC, ORDCF, OIT and CFI, as well as NA-MIC, NAC, BIRN, NCIGT and the Nedo Intelligent Surgical Instruments Project.
References


A generic computer assisted intervention plug-in module for 3D Slicer with multiple device support

András Lassó¹, Junichi Tokuda², Siddharth Vikal¹, Clare M Tempany², Nobuhiko Hata²,
Gabor Fichtinger¹

July 10, 2009

¹Queen’s University
²Brigham & Women’s Hospital

Abstract

Various frameworks and toolkits have been proposed for rapid development of computer assisted intervention (CAI) software. In this paper, we investigate how the open-source 3D Slicer application framework can be used for this purpose. We defined common requirements for CAI software to evaluate and enhance 3D Slicer for interventional applications in general. 3D Slicer is found to be an appropriate end-user deployment environment for CAI software, as its built-in functions fulfill many requirements and missing functionalities can be conveniently added. We describe the implementation of a CAI software based on extending core 3D Slicer functions. Three enhancements are described in detail: the management of workflow, DICOM image transfer, and multiple views. The resulting software fulfills general CAI requirements and supports two different MRI-guided prostate biopsy systems (each with a different imaging mode, robotic manipulator and calibration method), thereby demonstrating the usability of Slicer for implementing different CAI applications.

Contents

1 Introduction 2
2 Typical CAI application workflows 3
3 Generic CAI software requirements 5
4 Implementation of a CAI application in 3D Slicer 6
5 Conclusion 9
1 Introduction

Development of computer assisted intervention (CAI) systems is a rapidly evolving field, with lots of new methods, devices, and applications appearing day by day. CAI software components have to be built on reusable toolkits, frameworks, and open interfaces to keep up with the pace of changes.

To date, a number of free open-source software frameworks and applications have been developed in the research community of image-guided therapy. A group from Georgetown University [1] has been developing the Image-Guided Surgery Toolkit (IGSTK) a component-based software framework for image-guided surgery applications. The framework is built on top of Visualization Toolkit (VTK) and Insight Segmentation and Registration Toolkit (ITK) and it provides a set of application programming interfaces (APIs) for various functionalities required in image-guided therapy such as visualization, image transfer, and tracking device connectivity. Its modular architecture allows users to rapidly prototype navigation software for their clinical application and validate it. NaviTrack proposed by Von Spiczak et al. [2] is based on the similar concept, but provides Extended Markup Language (XML) interface that allows the users to configure data flow pipeline without coding. The SIGN framework [3] is another toolkit for rapid development of image-guided navigation software, with support for various device interfaces and workflow management. The Surgical Assistant Workstation (SAW, [4]) provides a framework targeted for the daVinci surgical robot, but its generic design allows its usage with other telesurgical robot systems. Besides those software frameworks specialized for image-guided therapy, several medical image processing and visualization software applications are extended for image-guidance of surgical procedures. Medical Imaging Interaction Toolkit (MITK, [5]) is also a framework build on top of VTK and ITK and can be extended for image-guided therapy application with MITK-IGT component. Another approach is to interconnect medical image processing and visualization software with existing surgical navigation systems. Papademetris [6] and his group developed a network protocol to interconnect their research software, BioImage Suite, with a commercial neurosurgical navigation system (VectorVision Cranial, BrainLab Inc.). The underlying idea for this work is to investigate state-of-art image processing and visualization techniques, which are not available in the conventional navigation system in the operating room without any modification to the clinical system.

3D Slicer [7] is one of those free open-source medical image visualization and processing applications that have been investigated for surgical navigation. 3D Slicer was used as prototype software for neurosurgery [8], prostate [9] and liver biopsy and treatment [10]. It offers functionalities useful for surgical applications, including various image processing and visualization techniques as well as network interface (OpenIGTLINK, [11]) for imaging and therapeutic device connectivity. Those functionalities are provided as plug-in modules, which users can fully control from 3D Slicer’s integrated graphical user interface. For data management, 3D Slicer provides its own scene graph architecture called Medical Reality Modeling Language (MRML), where all data e.g., images, models and transforms, are accessed from those modules. This centralized data handling mechanism allows the users to perform complex tasks without writing code, by combined use of the many elementary processing tools available in the integrated environment.

However, the flexibility in choice of functionalities and data often misleads the users and let them fail to follow a certain workflow. Therefore, our challenge here is to provide a ready-to-use integrated graphical environment that strikes the balance between usability and flexibility of clinical configuration. Moreover, a few key CAI software features are not yet supported by the current 3D Slicer core, such as the ability to...
receive images directly from imaging devices through DICOM network transfer and the possibility to display information on multiple screens in both the control room and operating room at the same time.

In this paper, we describe our new 3D Slicer plug-in that can be used for implementing a wide range of CAI systems. Our previous efforts focused primarily on the mechanical engineering aspects of the CAI system development and did not provide vendor-independent strategy for software architecture and integration. The engineering contribution of our current work is the software architecture and tools to implement CAI software in 3D Slicer in general, and specifically for MRI-guided prostate intervention.

2 Typical CAI application workflows

In this section we describe two different MRI guided prostate intervention systems that we built previously and specify a common workflow for them. The workflow represents well a typical CAI application, containing calibration, planning, targeting, verification steps and supporting image guidance and robot control.

**MRI-compatible robot system for transperineal prostate intervention**

This system uses an actuated MRI-compatible manipulator to insert needles into the prostate through the perineum. Instant feedback is provided during the needle insertion by displaying near real-time two-dimensional MR images that are taken in a plane aligned to the current needle position. The three main components of the system are the needle placement robot, a closed-bore whole-body 3T MRI scanner, and 3D Slicer with a custom module, as the user interface for the entire system ([12], Figure 1). All components are connected to one another via Ethernet, and communicating with each other using OpenIGTLink and DICOM protocols.

![Figure 1](image). A robot for transperineal prostate biopsy and treatment (left) and its system configuration (right). Pneumatic actuators and optical encoders allow operating the robot inside a closed-bore 3T MRI scanner.

The system has six states, corresponding the phases of the clinical workflow. In *start-up* phase system initialization is performed, including setup of software and hardware connections, sterile needle
installation, and starting position adjustments. During the planning step 3D MR images are acquired, loaded into the prostate intervention module in 3D Slicer, and finally the target positions are defined on them. In calibration phase the robot coordinate system is registered to the image coordinate system by determining the position of the robot on the MR image from the Z-shaped fiducial that is attached to the robot base. In targeting state, for each planned target the robot is moved automatically to the desired position, while 2D images are acquired continuously to monitor the needle insertion. Manual and emergency states are used for positioning the robot by low-level motion commands, and stopping all robot motion for exceptional situations.

**MRI-compatible robot system for transrectal prostate intervention**

The system uses an MRI-compatible manipulator to insert needles into the prostate through the rectum ([13], Figure 2). MRI images are acquired before the needle insertion to define the targets, and after the insertion for verification. The end-effector is currently operated manually, by setting the targeting parameters that the navigation software has computed.

![Figure 2. A robot for transrectal prostate biopsy and treatment (left) and its system configuration (right).](image)

The navigation software has four states. In calibration phase the robot coordinate system is registered to the image coordinate system, by locating on the MR image four markers that are attached to the manipulator. In the segmentation step a 3D model of the prostate can be created from an MR image for a better visualization of the relative positions of the targets, the needle, and the prostate. In the targeting step the user can define target positions and get the targeting parameters for a selected target. In the verification state an MR image is taken while the needle is inserted, and the distance of its actual visible position from the planned position is computed.

**Generic CAI system workflow**

By analysing the workflows of the above described two interventional systems, we can define a common workflow, which is applicable to both these two systems, and also to wide range of similar CAI systems.

- Start-up: In this state the necessary configuration parameters are set and software and hardware connections are established and tested.
- Planning: The intervention plan is created, describing what tools to be used and how, specifying target areas, positions, trajectories etc. If the planning procedure is lengthy or complex and information (images, measurements, other data) are already available pre-operatively then the
planning is carried out before the intervention, while adjustments and simple planning entirely can be performed during the intervention.

- Calibration: In this phase the devices are calibrated and coordinate systems of interventional tools (such as the robotic manipulator), patient, and imaging device are registered to each other. This is completed as near as possible to the time of the intervention (i.e., right before the intervention or even during the intervention if that is practically achievable), to minimize the chance of any changes that could alter the calibration or registration.

- Targeting: The manipulator is moved to each planned position and the necessary operation (such as biopsy or seed placement) is performed. The users may monitor the progress of the operation by observing instant, continuous feedback of the system through displayed images and other measured parameters (tracked position, etc.).

- Verification: In this step the difference between the plan and the actual result of the intervention is compared. This information may be used for quality assurance and may trigger adjustments in the intervention plan.

- Manual Control: This state is necessary for all systems where the manipulator is motorized, to allow simple direct control the device in case of emergency or for testing purposes.

Some states that were present in the previously developed prostate intervention systems are removed because they can be considered to be part of these six states (segmentation is part of planning; emergency is part of manual control).

Although there is a natural order of states (Start-up, Planning, Calibration, then repeated Targeting and Verification), there may be a need to adjust the plan or calibration during the intervention, so generally any transition between the states shall be allowed at any time. It also means that any of the steps shall be allowed to be performed during the intervention.

3 Generic CAI software requirements

*Functional requirements* define what the software is supposed to do. They can be defined by analyzing the generic CAI workflow described in the previous section. In the Start-up state the software has to display controls on the user interface to configure the system and display status information. In the planning phase the software shall be able to load, display, analyze medical images and add planning information (such as target point positions). During Calibration and Targeting states the software shall be able to receive images, communicate with external devices (such as robot, MRI scanner) and visualize the images, tools. In Verification state receiving of the images and visualization is required. For Manual Control communication with external devices shall be supported.

Additionally, *non-functional requirements*, such as external interface requirements, performance requirements, design constraints, software system attributes shall be defined for the software ([14]). There are no strict constraints for external interfaces, however it is strongly recommended to use industry standard and/or open interfaces whenever it is possible, to avoid the overhead of implementing custom protocols. Performance (speed, response time, etc.) requirements and design constraints are not critical for prototyping software, so we do not define any generic requirement for them. However, there are important attributes that shall be considered if the software is to be used on patients: the safety, robustness and usability of the system shall be “good enough” to be efficiently used during interventions. Exact
definition of what is good enough is a complex topic in itself (as discussed in e.g., by Di Maio et al. [15]), and shall be analyzed specifically for each application, but the following requirements are generally applicable. The software shall be able to recover from its failures by transitioning to a safe state where the procedure can be continued from, preferably by restoring a previous valid state. All significant information, such as user actions, state transitions, computation inputs and results shall be logged during the procedure to allow analysis of usage patterns, pro-active discovery of potential errors and investigation of failures after they have happened.

4 Implementation of a CAI application in 3D Slicer

The implementation of our MRI-guided prostate intervention software using 3D Slicer is presented in this section. First, suitability of 3D Slicer as a basis for CAI software is discussed, then an overview of our software implementation is provided. Finally, three important extensions of the Slicer basic functionalities are described.

Suitability of 3D Slicer for implementing CAI software

Most of the functional requirements, such as the required image enhancement, analysis, segmentation, visualization features are nearly all implemented in the basic 3D Slicer application. Missing functionalities can be usually added easily, thanks to the modular architecture of Slicer and the rich feature set and flexibility of the underlying algorithm libraries (VTK, ITK).

3D Slicer stores all information that describes its state in an MRML scene, and this data can be saved at any time. By default Slicer saves this data only on user request, but to allow recovery from a software failure by reloading a previous state, this information can be saved automatically. An automatic save may be triggered by specific operations, such as the completion of important steps (e.g., as calibration or modification of the intervention plan), and before complex or resource-intensive tasks (e.g., loading or processing of image data).

Logging capabilities are already built into 3D Slicer, the CAI software just had to use this infrastructure by inserting code that reports all important actions to the logger during execution.

3D Slicer provides an integrated graphical user interface that allows the launching any of the numerous single-purpose software modules, in any order, on almost any data. All data, such as images, transforms and models are shared by all modules, this way the user can process, combine, analyze data by using different modules. However, as this process is not guided, and cannot be easily automated in Slicer, going through complex workflows takes time and careful, undivided attention of the user. This is hard to guarantee during an interventional procedure, therefore the workflow and data flow management in Slicer needs improvement for CAI use.

Image guided navigation software require extensive capabilities for receiving images from other system components. 3D Slicer supports communication through the OpenIGTLink interface, which is targeted for efficient real-time communication with external devices and can be used to transfer any kind of information, such as images, position tracking information, or controlling commands. This is protocol works very well for those components that support it. However, the most widely supported method for medical image transfer through networks is DICOM [16]. The DICOM Working Group 24 has been even
compiling surgical workflow models to determine the standard for integrating information about patient equipment and procedure [17]. Unfortunately, the current version of 3D Slicer does not have the capability to receive images directly through DICOM network transfer. An external application can be used to receive the images and then the images can be loaded manually into Slicer, but this manual procedure takes time and is error-prone. An extension shall be developed for CAI applications with that Slicer can directly connect to imaging devices and quickly and reliably receive the acquired images.

CAI applications generally require display both in the control room with lots of detailed information and controls, and a simplified display in the operating room with just the most important information in a very well visible format. As 3D Slicer has no built-in support yet for managing multiple views on different displays, this has to be implemented as an extension for CAI applications.

**Software implementation overview**

3D Slicer is designed so that new functionalities can be easily added by implementing add-on modules. There are different types of Slicer modules, we chose to implement our CAI software in a “Loadable module”, because this integration method provides full access to Slicer internals (data structures, user interface, etc.).

The software runs on a separate workstation and communicates with the imaging device through Ethernet connection, using OpenIGTLink and DICOM protocols (Figure 3).

![Figure 3. Architecture of the 3D Slicer based CAI software. Navigation software runs on a workstation that communicates with the imaging device and robot through standard communication protocols.](image)

We created Slicer MRML node classes for storing all of our custom data: one node for storing all common data (configuration, OpenIGTLink connections, etc.) and a separate node for each robot type (one for transperineal, one for transrectal). We store images and target positions in standard Slicer nodes (vtkMRMLScalarVolumeNode and vtkMRMLFiducialListNode) and maintain references to them in our custom nodes.
**Workflow management**

Our CAI software provides a wizard interface for workflow management. The wizard interface consists of pages corresponding to clinical steps as we described in section 2, and shows only one page at a time in the graphical user interface. This allows hiding unnecessary control widgets from the GUI, minimizing the risk of unwanted operation by the user. The steps are associated with states of software system, which defines the behavior of software components. The step transitions based on the operator’s input on 3D Slicer and is notified to other software components through the network using OpenIGTLink protocol.

The user can configure the workflow, by ordering the existing components and defining MRML nodes shared by the components. To reuse the huge amount of useful software resources in 3D Slicer, it is important to incorporate the existing modules in the workflow interface. The mechanism to embed other modules into our workflow interface depends on the type of the modules. There are mainly two different types of plug-in mechanisms are used: a standard loadable (a.k.a. dynamic) module is loaded as a shared object or dynamic link library in 3D Slicer and has full control of 3D Slicer; Command Line Interface (CLI) is a mechanism to call a command-line program passing parameters as arguments and image data as files. Since a dynamic module creates its own graphical user interface (GUI) in certain area of the main 3D Slicer window, it is hard to embed the GUI in our wizard interface and restrict user input so that the users do not select wrong data. Thus, instead of displaying the module’s original GUI, our CAI interface displays downsized GUI widgets that receive the user’s input and internally call methods defined in the module. For CLI, the wizard displays forms for the users to input parameters and call the command with specified parameters. Data consistency is maintained by setting attributes of critical data so that they are cannot be manually edited in other modules (they are hidden).

**Image transfer management**

Our CAI software system offers two types of image transfer: DICOM image transfer from a PACS database or imaging device and real-time image transfer for monitoring procedures. Since OpenIGTLink interface is already available for transferring real-time image data, we provide proxy server software that converts DICOM image transfer to OpenIGTLink. The proxy server is implemented in two components. One component is listening at a TCP port to receive the images through DICOM transfer protocol and stores any acquired image in a file in a specific directory. The other component monitors the directory and sends any newly acquired image through OpenIGTLink connection. The advantage of this separation that the functionality of the first component is already available in some DICOM toolkits, so there is no need to implement it. We used the DCMTK [18] toolkit as DICOM receiver. Once the proxy software receives an image from through DICOM, it notifies 3D Slicer that the image is available for transfer with examination, series and image identifiers. The user can request the proxy to start transferring the image. For real-time image transfer, an OpenIGTLink interface has to be installed in the imaging scanner. We have developed a prototype of OpenIGTLink interface for GE’s MRI scanner. The interface bypasses image database in the scanner system and push images directly to 3D Slicer through OpenIGTLink. 3D Slicer updates the display immediately after it receives the image. The interface can control the imaging plane based on the transform matrix or quaternion received from 3D Slicer. Thus it is possible to acquire images from the plane parallel to the needle, by sending needle position and orientation measured by mechanical encoders on the robotic system.
Multiple views management

Displaying information in the control room and in the operating room is implemented by creating a viewer window that is independent from the Slicer main window and so can be displayed on a secondary monitor (or projected into the operating room, Figure 4). The secondary viewer shall be flexible enough to display 2D slices, 3D objects and/or other simple text (messages, measurement results, etc.). The 3D viewer in Slicer (vtkSlicerViewerWidget) can display all these information, so we created a custom top-level window class and inserted an instance of the 3D viewer widget into that. The custom top-level window is implemented so that if a secondary monitor is available then it is displayed there (filling the full screen), otherwise it is displayed as a pop-up window that can be moved anywhere on the screen. If we simply used a second instance of the 3D viewer in the new window, the same information is displayed in both the Slicer main window and the secondary window, because the viewer widget instances share all inputs. This is good, because there is no need to redefine what and how should be displayed in the secondary view. However, certain information must not be shared between multiple viewers, such as the camera (because using the same camera in different windows can lead to infinite loop of updates and therefore software lock-up), and some other data (such as targeting parameters) may be required on one display only or display in a different style (e.g., text with larger, more legible font). To resolve these problems we created a 3D viewer widget that is subclassed from the original Slicer 3D viewer class, but some properties, such as the reference to the camera object and visibility of certain objects are overridden. Multiple camera and viewer support is already in Slicer's roadmap, and whenever it will be available, this secondary monitor implementation can be greatly simplified.

Figure 4. Example of a main 3D Slicer window (left) and secondary window (right). Compared to the primary window, the secondary window shows image and tools in larger size and additional targeting information (in the top right corner).

5 Conclusion

In this paper we presented a new computer aided intervention software application, based on the open-source 3D Slicer framework. We analyzed the most common requirements of CAI systems and then utilized and extended the 3D Slicer core to comply with these requirements. The developed software can already be used to perform interventions with different MR-guided prostate biopsy systems, and we believe that the developed concepts and extensions can serve as a good basis for many other CAI software implementations.
Reference


[18] DCMTK - DICOM Toolkit, version 3.5.4 (2005-12-20), http://dicom.offis.de/dcmtk

Latest version available at the Insight Journal link http://hdl.handle.net/10380/3075
Distributed under Creative Commons Attribution License
This document describes the surgical robot console system based on 3D Slicer for image-guided surgery. Considering the image-guided surgery, since image workstations have complex User Interface (UI) and extra functions, it is supposed that such UI is not suitable for surgeon who is the robot operator. The proposed robot console is designed as a simple UI for the robot operator, which can display the endoscopic video image, the sensor data, the robot status and simple images for guiding the surgery. Therefore, we expect that the surgeon can concentrate on the operation itself by utilizing the robot console. On the other hand, since the robot console system is based on 3D Slicer, the robot console can use the abundant image operation functions. Moreover, it can use the flexibility of tool connectivities by using the OpenIGTLink protocol. In addition, since the video image is captured by using the multifunctional library OpenCV, we can expect the extensibility about the function of the proposed system.

Latest version available at the Insight Journal [http://hdl.handle.net/10380/3089]
Distributed under Creative Commons Attribution License

Contents

1 Introduction 2

2 System and Components 3
   2.1 Components of the surgical robot system 3
   2.2 3D Slicer as the base system of the robot console 3

3 Robot Console 4
   3.1 Introduction of OpenCV 4
   3.2 Integration test 5
1 Introduction

In this paper, for image-guided surgery, we propose a robot console system based on 3D Slicer [8], which is a well-known open source application for medical image processing. We are developing a surgical robot system. It is mainly composed of a master-slave robot and a viewer system for the robot operator who is the surgeon him/herself. The slave robot is an end effector with an endoscope that has a balloon type active touch sensor [5] and a controlled suction tube. We focus on an image-guided surgery application [2] that uses the robot. Such surgery needs helpful images for guiding the surgical operation. These images are provided by image workstations. However, the surgeon, who is the robot operator, cannot always watch the image workstation because he/she should watch the video image obtained by the endoscope during the robot operation. In addition, the image workstation has a complex UI for operating the master robot [7] because it has a lot of functions for trial and error of the image processing. However, the surgeon needs only the result of the image processing in order to obtain useful guidance for the operation. The trial and error of the image processing should be conducted by the radiologist using the image workstation.

On the other hand, the surgeon needs to pay attention to the data of the sensor and robot status obtained every moment because such data has an important role to decide the next operational tactics. If the sensor data are displayed on the complex UI of the image workstation, it is not easy for the surgeon who operates the robot watching the endoscopic image to check the data. We consider that a simple interface to confirm such information easily is necessary for the robot operator. To satisfy those requirements, we propose a robot console system where the user interface is composed of the endoscopic video image on which is overlaid...
2 System and Components

2.1 Components of the surgical robot system

Figure 2 shows a schematic diagram of the proposed surgical robot system. It is composed of 5 parts: master-slave robot, robot console, image workstation, 3D motion sensor, and log supervision server. Each component is connected by ethernet. We assume that the proposed system is for the surgical operation by one or two surgeons, a radiologist and practical nurses. One surgeon operates the master-slave robot watching the robot console. The robot status including position and orientation of the end effector and the sensor data, which are displayed on the robot console, are obtained from the master-slave robot and 3D motion sensor. The images for guiding the surgery are obtained from the image workstation, which is operated by the radiologist. The image workstation is a non-commercial navigation system such as 3D Slicer or virtual endoscopy NewVes. A commercial navigation system, such as Brainlab’s VectorVision or Aze’s Virtualplane, can be used as optional or backup systems of the image workstation. The operation process and history, including robot motion and sensing data, are recorded by the log-supervision server.

2.2 3D Slicer as the base system of the robot console

As shown in Figure 2, the robot console can collect all the data and display them as the useful information for the surgical operation. However, it should not be just an information viewer. It is required that the robot console can complete the surgical operation even if the image workstation has failed. For realizing this kind of robustness (fault tolerance), it is necessary that the robot console has some image processing capabilities. For satisfying such an important request, we decided to construct the robot console based on 3D Slicer.

the sensor data, robot status and the images for guiding the surgery, as shown in Figure 1.
Figure 3: Data flow in the integration test. Each sensor data is transmitted on ethernet by utilizing the OpenIGTLink protocol. In this case, the robot console and each sensor system become client and server, respectively. The image data of the endoscope is captured by OpenCV.

3D Slicer is an open source application for medical data processing. Many functions, including volume rendering, registration, segmentation, tractography of the medical data, are provided as modules of 3D Slicer [1]. Therefore, if we construct the robot console based on 3D Slicer, we expect that we can use those many functional modules for medical imaging also with the robot console. This is one of the primary reasons to make 3D Slicer the base of the robot console. In addition, we focus on the point that each functional module has been tested and works in real situation [4]. These results become important reasons to choose it as the base system from the viewpoint of robustness.

Of course, 3D Slicer has flexible connectivity options. Especially, by using the OpenIGTLink [6] protocol, which is a simple but extensible data format, we can connect the software and devices such as surgical navigation software and tracking device, and also robotic device.

3 Robot Console

As a way of developing the robot console based on 3D Slicer, we decided to develop it as a 3D Slicer module. 3D Slicer has no support for video image capturing. Of course, by utilizing OpenIGTLink, which can handle not only text data but also image data, the captured image data can be shown on the 3D Slicer UI. However, considering the delay of the video image, we should capture the endoscopic video image on the local hardware. For satisfying this requirement, we introduce OpenCV [10].

3.1 Introduction of OpenCV

OpenCV is an open source and cross platform library of programming functions mainly aimed at real time computer vision. This library includes camera calibration and image tracking functions. If the operating
system of the platform is Linux, specific capture boards are treated easily by using the V4L (Video for Linux) library and OpenCV detects it automatically. In addition, we can say that the constructed system can maintain strong portability because OpenCV also is an open source software.

3.2 Integration test

Figure 3 shows the schematic diagram between the part of the master-slave robot and the robot console on the integration test. Each sensor data is transmitted on ethernet by utilizing OpenIGTLink protocol. Since the sensor data are transmitted from the sensor system, the role of the robot console and each sensor system become client and server, respectively. The operating system of the robot console system is Ubuntu Linux 8.04 (Kernel 2.6.24). The hardware specification is Intel Core2 Duo 2.13GHz, 3.0GB memory. The base system of the robot console is composed of 3D Slicer Version 3.4 1.0 and OpenCV 1.0.0. The server system of the sensor is composed of LEPRACAUN-CPU and LEPRACAUN-IO of GENERAL ROBOTIX, Inc. (Renesas, Inc. RISC CPU SH-7785, 600MHz and ARTLinux 2.6 operating system). The OpenIGTLink is installed by CMake 2.6 [9] for a cross compile environment. The video image of the endoscope is captured by utilizing the OpenCV library through the V4L.

Since the robot console was based on 3D Slicer, it worked well as the client by using the OpenIGTLink protocol. The basic module used in the integration test was adopted as the 3D Slicer module which was named ‘OpenCV’.


3.3 Design of the robot console

Since the UI of the robot console is built by Visualization Tool Kit (VTK) [11] in 3D Slicer, we can treat image data for guiding the surgery and text data on the captured video image easily by using texture on polygon data of OpenGL.

The comparison of the UI between the proposed robot console and the 3D Slicer is shown in Figure 4. The upper figure shows 3D Slicer. The lower figure shows the proposed robot console. The UI of the robot console is more simple than that of 3D Slicer. The left pane of the robot console is the main view pane which shows mainly the captured video image. On the other hand, the right pane is the optional one which shows the overlook view of the 3D Slicer. If we use this module in full screen mode, the optional pane will be useful to confirm the position of the end effector. The sensor data and the robot status can be rendered over the captured video image data. The image for guiding the surgery can be also rendered easily over the video image with semitransparent overlay. Since the main information is displayed on the main view pane, we expect that the surgeon can concentrate on the surgical robot operation without turning his eyes away.

3.4 Extension of the function

Since we use the OpenCV library, we can add the second camera easily only by specifying the camera number. Therefore, by utilizing a stereo camera, two windows and two monitors, we can obtain a 3D view, which is important for brain surgery.

On the other hand, for making the end effector avoid nervous tissues, we utilize a virtual fixture [3] for the surgical robot operation. The virtual fixture gives artificial walls by the controlled force to support the operation of the end effector. The surgeon can achieve smooth operation to move the end effector without
3.4 Extension of the function

(3D Slicer)

Figure 4: Comparison of user interface between 3D Slicer and the proposed robot console, which is released as a module of 3D Slicer. The left pane can display the image of the area of the tumor for guiding the surgical robot operation over the video image. The overlook view can also show the position of the tumor as the useful information.
touching the artificial virtual wall by the controlled force. Therefore, if the virtual wall of the complex shape is rendered over the video image of the endoscope, we expect that it will become a useful guide for the smooth operation.

4 Conclusion

We propose a robot console system based on 3D Slicer for image-guided surgery. By introducing OpenCV, the robot console has a simple UI which displays the captured video image of the endoscope. In addition, it displays the images for guiding the surgery, with the sensor data and the robot status overlaid on the video image data for the surgeon to be able to concentrate on the robot operation.

Future work includes the construction of useful functions to guide the robotic surgery on the robot console system.

Acknowledgement

Part of this was funded by NEDO P08006 "Intelligent Surgical Instruments Project", Japan.

This publication was made possible by Intelligent Surgical Instruments Project (NEDO, Japan).

N.H. was supported by Grant Number 5U41RR019703, 5P01CA067165, 1R01CA124377, 5U54EB005149 from NIH. Its contents are solely the responsibility of the authors and do not necessarily represent the official views of the NIH. Part of this study was funded by them.

References


