

The Operating Tool – a framework for monitoring and control of heterogeneous data and workflows

Christoph Reck, Sven Kröger

German Aerospace Center (DLR)

German Remote Sensing Data Center (DFD)

Oberpfaffenhofen, D-82234 Weßling, Germany

Email: Christoph.Reck@dlr.de, Sven.Kroeger@dlr.de

ABSTRACT

The Operating Tool (OT) is a graphical user interface to monitor and control heterogeneous data and workflows in distributed services. It has been designed as a configurable framework with a flexible set of re-usable views, which can be instantiated to communicate with the remote services over a wide range of protocols as CORBA, CIP, SOAP, LDAP, HTTP, and the file system. The implementation profits from an architecture using parameterised building blocks, based on the Model-View-Controller paradigm, that are assembled using a very simple but powerful XML configuration file syntax. Displaying and having access to the interfaces of several services within one tool enables the user to track the stages in the production process and monitor the status across service boundaries. In case of an error, the operator can interact with the process at the location that caused the problem.

Keywords: GUI, HMI, MMI, Java, Framework, DIMS, MMFI

INTRODUCTION

The Operating Tool (OT) is a graphical user interface used to monitor and control the data and workflows of the distributed services from the Data and Information Management System (DIMS)^[2] services of the national data library at the German Remote Sensing Data Center of DLR and for the European earth observation Multi-Mission Facilities Infrastructure (MMFI)^[3] at several sites of ESA. It is setup to handle many existing missions including the high throughput of the TerraSAR-X Payload Ground Segment^[13].

Driving idea was to provide a configurable framework to present views onto remote services within one tool. This is much alike modern tabbed browsers, but with additional support of other protocols than HTTP and providing rich views beyond HTML capabilities. Complex frameworks like J2EE^[6], Eclipse^[12] and Spring^[11] covering a similar domain have recently emerged and are widely adopted; in contrast the OT shines by its power and simplicity as described in this paper.

Use Cases

Typical operational tasks are:

- Monitoring and Control
Examples are status monitoring, startup and shutdown, log view, etc.
- Configuration Management
This includes allowing graphically and centrally to edit external files, as well as viewing and updating internal momentary state.
- Data Management
Examples are order follow-up, customer data entry and update, accounting data, etc.

These categories of the scenarios condense a large set of possible use cases to be handled by the Operating Tool. The OT framework is configurable with plug-ins and views to achieve these scenarios.

Requirements

The major characteristics needed for the Operating Tool are:

- Unify the regular operational tasks into one tool.
This avoids having to keep multiple tools running in parallel, and eases the learning curve to operate the system.
- Provide a similar look-and-feel to the common tasks of different subsystems.
Common functions used in all or at least several areas shall be provided as reusable components. This reduces complexity for extension developers, it increases uniformity for the operational phase, and it reduces costs by reuse of software components for current and future missions.
- Support different users, groups and roles.
The responsibility of operation and administration of the different services span several departments and each working group shall be able to view and operate strictly within its area. Diverse groups will view the same subsystems, but will act on them with different roles. The operating tool shall allow restricting the actions that the operators are allowed to perform within the views.
- Easy of configuration and extension.
Operators will change and production systems will be added in the future; the operating tool shall be able to grow with the system with new instances, interfaces, views and requirements.
- Provide support in the framework for:
 - Selection handling and communication across views
 - Geographic coordinate projections into a map view
 - Image viewer
- Remote operating of the distributed system and platform independence.
The operating tool allows remote operating of the services from a desktop machine.

Architecture

The operator tasks are characterized by a common flow: the operator interacts with the views; the views obtain the data from the model; the model from the interface adapters.

The building blocks of the Operating Tool are the views, the models, the interface adapters, and the configuration that binds them together.

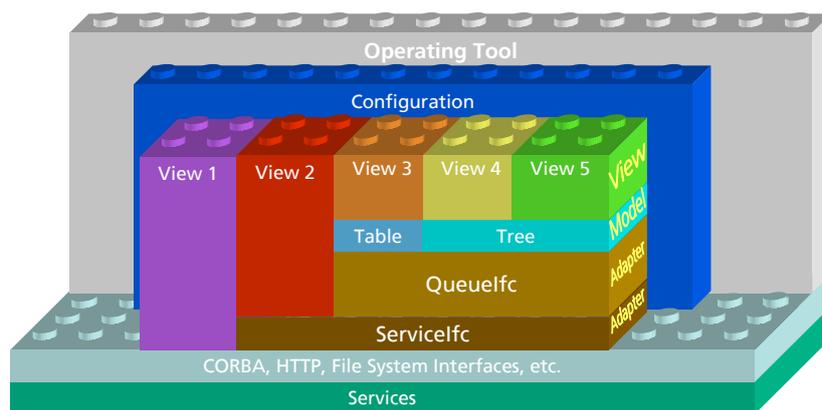


Figure 1: DIMS Operating Tool Components and Interfaces

The above figure depicts the components and interfaces as building blocks of the Operating Tool. It demonstrates how the components are assembled to construct the application. The configuration can be accessed from every layer. The views are loaded to display the data accessible via a model. The model adapter layer is not required by Operating Tool specific views (View 2). Some views can directly access external data or interfaces (View 1) - the framework does not impose any restrictions on the building blocks.

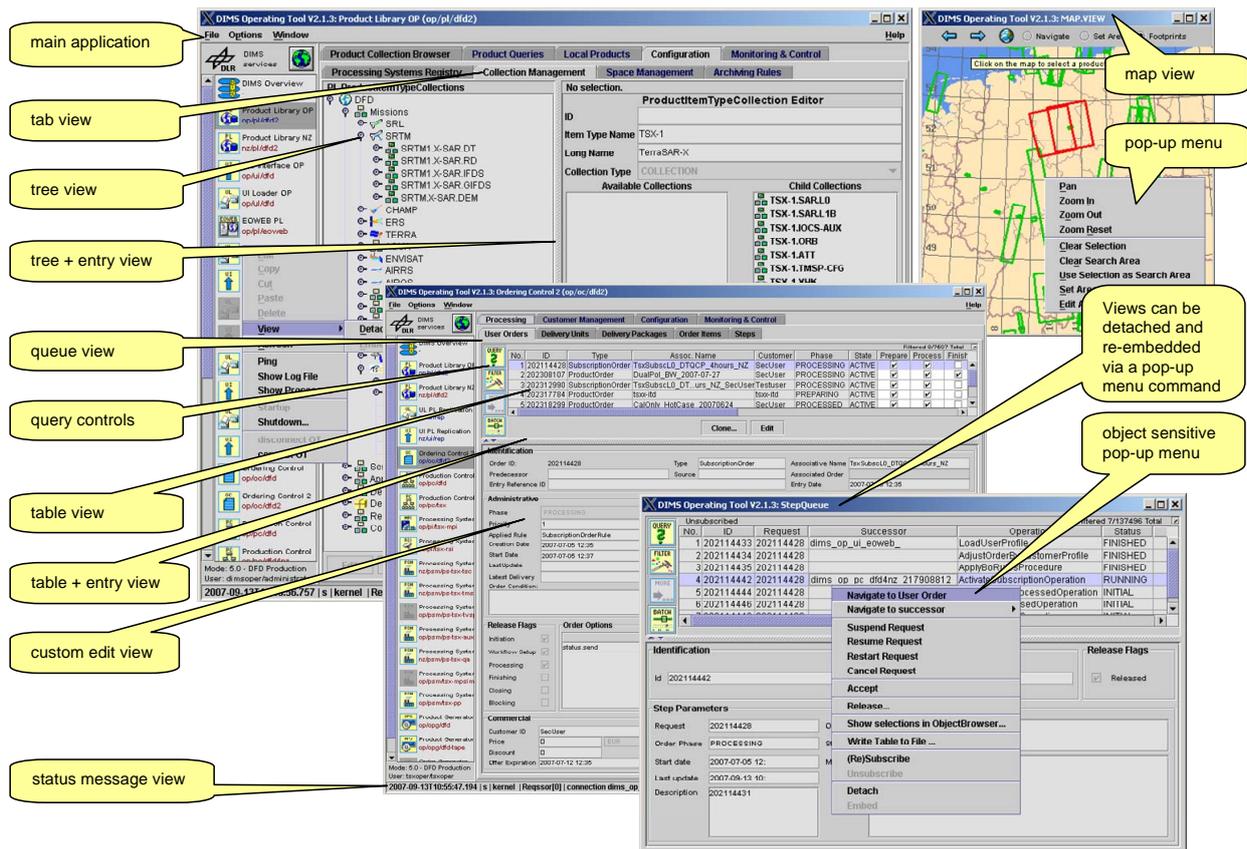


Figure 3: Views of the DIMS Operating Tool

The tool setup used for DIMS and the MMFI offers a list of icons as a handle to the target services on different computers, partially on remote sites. Selecting a target from this list will load and present the views to monitor and control the service. This approach, instead of using an own control interface for each component, integrates the views to all the services relevant to the specific user in a single application.

CONFIGURATION

An important aspect of the OT is the configuration potential. This allows the administrator of the system to reconfigure the OT in an easy way without the necessity of changing and compilation of java classes.

The first section will give a short overview of the configuration hierarchy of the OT and how it is used in practice. The second section will show the advantage of parameterized views.

The configuration hierarchy of the OT

For configuration, the OT uses a hierarchy of XML files based on a very simple but powerful schema. The following code snippet is an example for the user "tsxuser" (modified for this paper).

```

<CONFIGURATION>
  <INCLUDE NAME="/etc/common.conf.xml" />

  <INCLUDE NAME="/etc/target/pl-dfd-op.conf.xml" />
  <INCLUDE NAME="/etc/target/pl-dfd-nz.conf.xml" />
  <INCLUDE NAME="/etc/target/psm-tsx-tsc.conf.xml" />

  <STRING ID="group" VALUE="tsxuser" />
  <STRING ID="pl-dfd-op.ROLE" VALUE="user" />
  <STRING ID="pl-dfd-nz.ROLE" VALUE="user" />

</CONFIGURATION>

```

The entrance point of a configuration is normally the user- or group-configuration file as the example above, hosted on the instance site of the OT. This file defines which service instances will be displayed for the user. Also, this is the place of user specific settings.

The element "INCLUDE" directs the configuration reader to include four configuration files; a first one with common settings then three service targets. Next, the group membership is set to "tsxuser" – this is a global definition.

Last, the "pl-dfd-op" and "pl-dfd-nz" package specific roles of the user at the two product libraries are defined. These definitions override the defaults from the included target configuration template. The advantage of using templates is the possibility to narrow the settings for this user, without duplicating the target configuration files for other users and therefore avoiding the need to maintain repeated code.

The configured values are accessible within the application by the name specified in the ID field.

The next example shows the configuration file for the service target "pl-dfd-op".

```
<CONFIGURATION>

  <INCLUDE NAME="/etc/target/pl2/PL2_common.conf.xml" />

  <PACKAGE ID="pl-dfd-op">

    <INCLUDE NAME="/etc/target/pl2/PL2_instance.conf.xml" />

    <STRING ID="TITLE" VALUE="Product Library OP" />
    <STRING ID="SITE">op</STRING><!-- Oberpfaffenhofen -->
    <STRING ID="NAME">dfd2</STRING>

    <STRING ID="COLLECTIONS.ROOTS" VALUE="DFD,Missions" />
    ...
```

Instances are typically configured with the service specific common definitions and the service configuration template, followed by the instance specific information like its name and the site. Thereafter, the service specific configuration parameters are supplied, here for example the root collections to display in the collection browser.

Summarizing the above pattern, the Operating Tool configuration is structured as follows:

- user configuration file including the group configuration, possibly narrowing the operator role
- group file including the target configuration files and defining the default role for the group and setting some values
- target configuration files with the service specific definitions

Administering this structure requires minimal effort in defining new users, groups or targets.

Extending the Operating Tool with additional views is outlined next. These configuration files are typically part of the OT core software and define the appearance of the views. Assembling of the OT views is done with the "OBJECT" configuration element and the desired parameters. Objects may be simple values like a string (e.g. a label) or complex like a pop-up menu of a view. These changes or extensions are achieved without the necessity of a recompilation.

Using Parameterized Views via Configuration

The advantage of using parameterized views is reusability of code and ease of customization by configuration, therefore fast development of new applications. The next code snippet shows how the "Processing" tabbed view is assembled, followed by the example showing the differences in the configuration of the "Production Requests" and "Steps" views.

```
<OBJECT ID="PROCESSING" CLASS="de.dlr.dfd.dims.ot.view.TabbedView">
  <CONSTRUCTOR>
    <STRING REF="..CONTENTS" />
  </CONSTRUCTOR>
  <METHOD NAME="setName">Processing</METHOD>
  <HASH ID="CONTENTS">
    <STRING NAME="Production Requests" REF="*.PdR.VIEW" />
    <STRING NAME="Steps" REF="*.StepQueue.VIEW" />
  </HASH>
</OBJECT>
```

The next two figures show the rendered views in this "TabbedView".

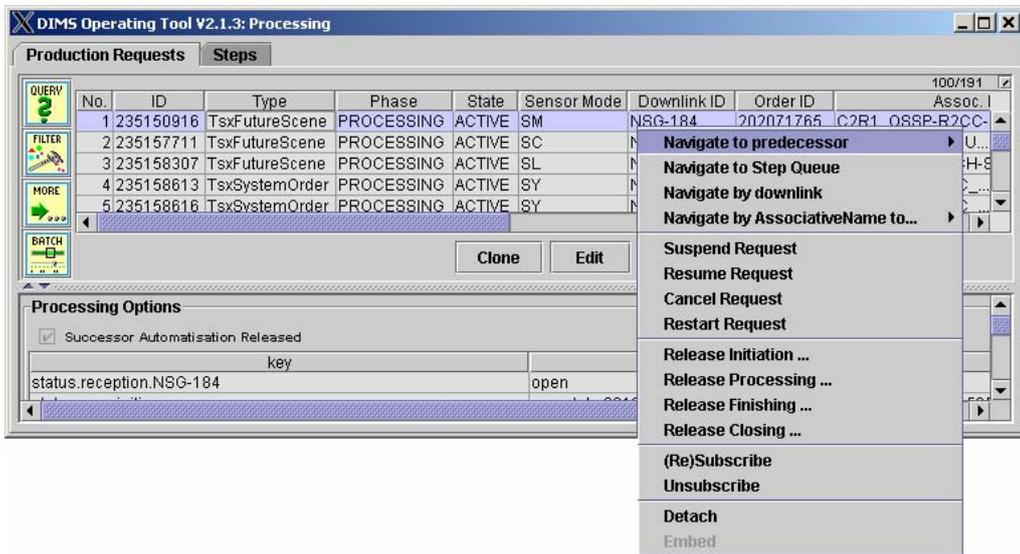


Figure 4: Example of the "Production Request" queue entry view with its pop-up menu

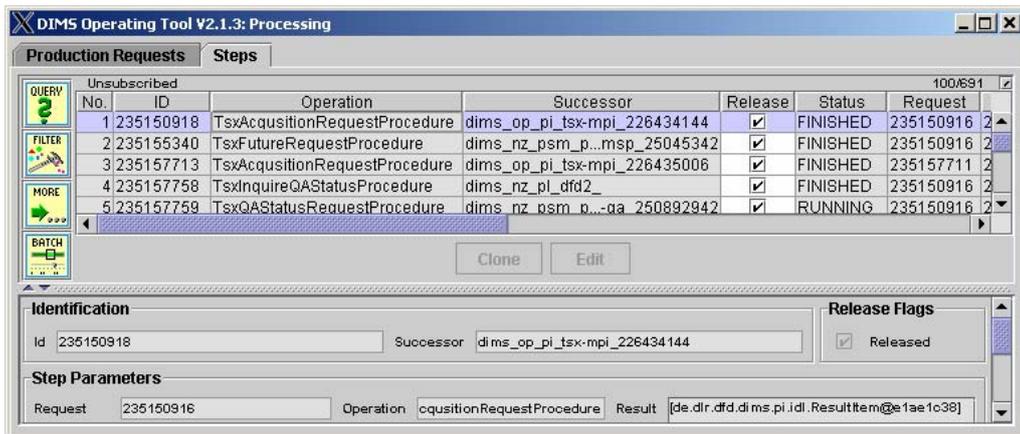


Figure 5: Example of the "Steps" queue entry view

The code snippets below highlight the differences in the configurations:

Production Requests	Steps
<pre> <OBJECT ID="PdR"> <INCLUDE NAME="QueueEntryView.conf.xml" /> <STRING ID="QUEUE_NAME" VALUE="RequestQueue" /> <STRING ID="ENTRY_NAME" VALUE="Request" /> <STRING ID="ENTRY_CLASS" VALUE="de.dlr.dfd.dims.pi.idl.PdRId1" /> <OBJECT ID="ONLINE" REF="TRUE" /> ... <MAP ID="COLUMNS"> <STRING NAME=" No. " >#</STRING> <STRING NAME=" ID " >id</STRING> <STRING NAME=" Type " >type</STRING> ... </MAP> <OBJECT ID="EDITVIEW" CLASS="de.dlr.dfd.dims.ot.targets.pi.PdREditView"> ... </OBJECT><!-- PdR --> </pre>	<pre> <OBJECT ID="StepQueue"> <INCLUDE NAME="QueueEntryView.conf.xml" /> <STRING ID="QUEUE_NAME" VALUE="StepQueue" /> <STRING ID="ENTRY_NAME" VALUE="Step" /> <STRING ID="ENTRY_CLASS" VALUE="de.dlr.dfd.dims.gen2.idl.StepId1" /> <OBJECT ID="ONLINE" REF="FALSE" /> ... <MAP ID="COLUMNS"> <STRING NAME=" No. " >#</STRING> <STRING NAME=" ID " >id</STRING> <STRING NAME=" Operation " >Operation</STRING> ... </MAP> <OBJECT ID="EDITVIEW" CLASS="de.dlr.dfd.dims.ot.targets.pc.StepsEditView"> ... </OBJECT><!-- StepQueue --> </pre>

The included "QueueEntryView" template instantiates the appropriate class implementing the view and defines the required default values. The instance specific values are configured after the inclusion of the template. You can observe the differences that control the appearance:

- The object "QUEUE_NAME" tells the class *queue entry view*, which queue is accessed for the input. The example shows "RequestQueue" and "StepQueue"
- Further, their input objects are of different classes – this is defined in the object "ENTRY_CLASS"
- Setting the "ONLINE" state to "FALSE", tells the view to work in a “unsubscribed” mode, as shown on the top left of the queue view
- The object "COLUMNS" defines the order, the title and the value of the columns. The third column of the *Production Requests* view is "Type", but in the view *Steps* it is "Operation".
- The entry view to be used is defined by the "EDITVIEW" parameter.

These parameters are sufficient to customize the class which implements the *queue entry view*. Additional parameters allow configuring items like filters or pop-up menus.

The advantages of this approach are:

- The developer can setup new views in a very fast way
- Rearranging or renaming of the columns or adding of further columns is very simple
- Specific groups of users can override column definitions to fit their requirements, e.g. hide fields or show additional parameters.

Navigation between Views

Displaying and having access to the interfaces of several services within one tool enables cross-service navigability. This permits the user to track the stages in the production process and monitor the status across service boundaries. In case of an error, the operator can interact with the process at the location that caused the problem.

Figure 4 depicts the pop-up menu of the “Production Requests” queue entry view. The configuration allows defining menu commands and actions^[9]. The OT framework has a navigation action class that can be attached via configuration to tables and forward the selection to a class that implements the *NavigationTarget* interface. The menu of the above example is configured as follows:

<pre> <MAP ID="MENU"> <OBJECT NAME="Predecessor" REF="...NAVIGATE_PREDECESSOR_ACTION" /> <OBJECT NAME="Step" REF="...NAVIGATE_Step_ACTION" /> <OBJECT NAME="Successor" REF="...NAVIGATE_SUCCESSOR_ACTION" /> <OBJECT NAME="AssocName" REF="...NAVIGATE_AssocName_ACTION" /> <STRING NAME="separator1" /> <STRING NAME="Suspend Request" >queueCommand:suspendOrder</STRING> ... </MAP> ... </pre>

```

<OBJECT ID="NAVIGATE_Step_ACTION" CLASS="de.dlr.dfd.dims.ot.control.NavigationHandler">
  <CONSTRUCTOR>
    <STRING>Navigate to Step Queue</STRING><!-- label -->
    <STRING REF="..NAVIGATION_SOURCE" /><!-- defined below -->
    <STRING REF="*.StepQueue.NAVIGATION_TARGET_PdO" />
  </CONSTRUCTOR>
</OBJECT>

<OBJECT ID="NAVIGATION_SOURCE"
  CLASS="de.dlr.dfd.dims.ot.targets.gen2.QueueNavigationSource">
  <CONSTRUCTOR>
    <OBJECT REF="..TABLEVIEW" /><!-- defined in QueueEntryView.conf.xml -->
    <STRING REF="id" /><!-- ID field of the table object -->
  </CONSTRUCTOR>
</OBJECT>
...

```

The above example demonstrates how menus and actions are defined. The 4 entries on the top are the navigation actions, referencing the definitions below. The other menu commands are handled by the queue view controller.

The "Navigate to Step Queue" action is created after the menu definition using the configuration object creation syntax. It instantiates a "NavigationHandler" with the parameters: label, navigation source, and navigation target. The "NAVIGATION_SOURCE" definition uses a reference to the table from the included QueueEntryView.conf.xml template. The step "NAVIGATION_TARGET" definition is part of the step queue entry view.

The navigation menus work as follows:

1. When the operator clicks the "Navigate to Step Queue" menu item, the "NavigationHandler" action is activated,
2. "NavigationHandler" obtains the selections from the "NAVIGATION_SOURCE" and passes them to the "NAVIGATION_TARGET" (which is part of the controller of the queue behind the Step queue view),
3. The "NAVIGATION_TARGET" queries the Step queue for the entries that have the specified request IDs and displays the results
4. The "NAVIGATION_TARGET" also instructs the Step view to be placed upfront.
5. The operator can now perform actions on the steps belonging to this request (see an example of the "StepQueue" menu in Figure 3).

Navigations can be configured to cross service boundaries, "Navigate by Associative Name to ..." in Figure 4 being an example. Navigation targets are registered in lookup tables and can therefore be discovered and addressed automatically.

CONCLUSION

The Operating Tool is a flexible application framework with a rich set of features.

Common operating tasks as monitoring and control, including startup and shutdown, viewing logs, handling data lists and queues, etc. are accomplished without having to code new programs.

New services can be made accessible with few lines in a configuration file. This enables prototyping and assembling of operating applications. Tasks spanning multiple distributed services can be operated centrally from within one tool.

DIMS demonstrates the successful use of Java, XML, and CORBA in a large production quality system that can be efficiently administered via a homogenous operating tool.

REFERENCES

- [1] Kiemle S., Mikusch E., Göhmann M., The Product Library – A Scalable Long-Term Storage Repository for Earth Observation Products, DASIA 2001, Nice, 2001.
- [2] S. Kiemle, E. Mikusch, C. Bilinski, B. Buckl, D. Dietrich, S. Kröger, C. Reck, A.-K. Schroeder-Lanz, M. Wolfmüller, Data Information and Management System for the DFD Multi-Mission Earth Observation Data. Ensuring Long-term Preservation and Adding Value to Scientific and Technical Data, PV 2005, Edinburgh UK, 21.-23.11.2005, Digital Curation Center Conference Proceedings, The Royal Society Edinburgh, (2005)
- [3] Pinna, G-P., Mikusch, E., Bollner, M., Puin, B., Earth Observation Payload Data Long Term Archiving - the ESA's Multi-Mission Facility Infrastructure. Ensuring Long-term Preservation and Adding Value to Scientific and Technical Data, PV 2005, Edinburgh UK, 21.-23.11.2005, Digital Curation Center Conference Proceedings, The Royal Society Edinburgh, (2005)
- [4] S. Kiemle, E. Mikusch, C. Bilinski, B. Buckl, D. Dietrich, S. Kröger, C. Reck, A.-K. Schroeder-Lanz, M. Wolfmüller, Data Information and Management System for the DFD Multi-Mission Earth Observation Data. Ensuring Long-term Preservation and Adding Value to Scientific and Technical Data, PV 2005, Edinburgh UK, 21.-23.11.2005, Digital Curation Center Conference Proceedings, The Royal Society Edinburgh, (2005)
- [5] Jason Cai, Ranjit Kapila, and Gaurav Pal, HMVC: The layered pattern for developing strong client tiers, JavaWorld, July 2000: http://www.javaworld.com/javaworld/jw-07-2000/jw-0721-hmvc_p.html
- [6] J2EE Design Patterns > Model-View-Controller Architecture, Sun Microsystems, 2001, http://java.sun.com/blueprints/patterns/j2ee_patterns/model_view_controller
- [7] Introducing Swing Architecture, Sun Microsystems, 2001, http://java.sun.com/products/jfc/tsc/articles/getting_started/getting_started2.html
- [8] Java Web Start 1.0.1, Sun Microsystems, 2001, <http://java.sun.com/products/javawebstart/index.html>
- [9] How to Use Actions, Sun Microsystems, 2007, <http://java.sun.com/docs/books/tutorial/uiswing/misc/action.html>
- [10] The SwiXML software <http://www.swixml.org/>
- [11] Spring Framework, <http://www.springframework.org/>
- [12] Eclipse - an open development platform, <http://www.eclipse.org/>
- [13] German TerraSAR-X Radar Satellite Mission homepage at the German Aerospace Center http://www.dlr.de/tsx/start_en.htm

Christoph Reck

Christoph graduated from the Technical University in Munich, Germany in 1988 with a Diploma in electrical engineering. His work at DLR involved hardware and software design for the D-2 Space Shuttle experiments, ERS1&2 ground segment data decoding, MarsExpress onboard compression for the HRSC camera and the data decoding in the ground segment, as well as other missions. Operator and user interfaces were always a key issue of the designs. This was followed by software analysis, design and implementation for the ISIS, INFEO and DIMS user and operator interfaces. Throughout his career Christoph based his work on standards and design patterns to achieve a large scale of reusability and efficiency. The current work as a senior system engineer involves the evolution of the tools for operations of the DLR National Remote Sensing Data Library and contributions to user services of DLR and ESA Multi-mission Facility Infrastructure and EO User Services including HPC and Grid computing.

Sven Kröger

Sven graduated from the University of Hamburg, Germany in 2002 with a Diploma in computer science with a major in qualitative spatial knowledge. During his time as research assistant at the University of Konstanz, Germany (2003-2005) he was engaged in knowledge management and collaborative e-learning. Since 2005 Sven works at the German Remote Sensing Data Center of DLR. Among the evolution of the Operating Tool Sven is DIMS system engineer of the TanDEM-X (TerraSAR-X add-on for Digital Elevation Measurement) project.