A Anhang

A.1 Technische Realisierung des ersten Prototyps

Der erste Prototyp besteht aus einem in der Programmiersprache Python geschriebenen Plugin für das Change-Request-System Trac sowie einem Java-Backend, das für die Zeichnung des Graphen verantwortlich ist. Das Trac-Plugin basiert in weiten Teilen auf dem "Hello World 2"-Plugin[1] aus dem Plugin-Entwicklungs-Tutorial-Bereich der Internet-Seite "track-hacks.org"[2]. Die Methoden *get_active_navigation_item* und *get_navigation_items* sorgen für den Eintrag eines Links zur Plugin-Seite in der Hauptnavigation des Trac-Systems. Die Methode *match_request* legt fest, unter welcher URL innerhalb des Trac-Systems die Visualisierung des Prototyps abrufbar ist. Abschließend definiert *process_request* das Genshi-Template[3], das zur Darstellung verwendet wird. In diesem Fall handelt es sich um eine einfache HTML-Seite, die ein Bild von einer lokalen URL einbindet.

Hinter dieser URL verbirgt sich ein mit Hilfe der *Java API für XML-Webservices (JAX-WS)* realisierter XMLRPC-Dienst, der eine Grafik des Trac-Standard-Workflow zeichnet und in Form eines PNG-Bilds als HTTP-Antwort zurückliefert (siehe Quelltext A.1). Zur Zeichnung des Workflows wurde das *Java Universal Network/Graph Framework (JUNG)*[4] verwendet. Es erlaubt u.A. die einfache Definition eines gerichteten Graphen in Java, beschriftet Knoten und Kanten und zeichnet den Graph gemäß eines selbst definierbaren oder aus einer vorhandenen Sammlung auswählbaren Layout-Algorithmus.

```
package de.rwth.swc.ccharles.crsdb.xmlrpc;

import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Image;
import java.awt.Point;
import java.awt.image.BufferedImage;
import java.io.ByteArrayOutputStream;
import java.io.IOException;

import javax.imageio.ImageIO;

import org.apache.commons.codec.binary.Base64;
import org.apache.commons.collections15.Transformer;
```

[1] `https://trac-hacks.org/wiki/EggCookingTutorialTrac0.11` abgerufen am 30.05.2013

[2] `http://trac-hacks.org/` stellt Subversion-Hosting für von Trac-Nutzern erstellte Trac-Plugins zur Verfügung; abgerufen am 30.05.2013

[3] `http://genshi.edgewall.org/` abgerufen am 30.05.2013

[4] `http://jung.sourceforge.net/` abgerufen am 31.05.2013

```java
import org.apache.commons.collections15.functors.ConstantTransformer;

import edu.uci.ics.jung.algorithms.layout.FRLayout;
import edu.uci.ics.jung.graph.Graph;
import edu.uci.ics.jung.graph.SparseMultigraph;
import edu.uci.ics.jung.graph.util.EdgeType;
import edu.uci.ics.jung.visualization.VisualizationImageServer;
import edu.uci.ics.jung.visualization.decorators.
    AbstractEdgeShapeTransformer;
import edu.uci.ics.jung.visualization.decorators.
    ConstantDirectionalEdgeValueTransformer;
import edu.uci.ics.jung.visualization.decorators.EdgeShape;
import edu.uci.ics.jung.visualization.renderers.
    GradientVertexRenderer;
import edu.uci.ics.jung.visualization.renderers.
    VertexLabelAsShapeRenderer;

public class WorkflowRenderer {

    public String workflowPicture() {

        // states
        final State NEW = new State("new");
        final State CLOSED = new State("closed");
        final State ASSIGNED = new State("assigned");
        final State REOPENED = new State("reopened");
        final State ACCEPTED = new State("accepted");

        Graph<State, Transition> graph;

        VisualizationImageServer<State, Transition> vv;

        // create a simple graph for the demo
        graph = new SparseMultigraph<State, Transition>();

        graph.addVertex(CLOSED);
        graph.addVertex(NEW);
        graph.addVertex(ACCEPTED);
        graph.addVertex(ASSIGNED);
        graph.addVertex(REOPENED);

        graph.addEdge(new Transition("reopen"), CLOSED, REOPENED,
                EdgeType.DIRECTED);

        graph.addEdge(new Transition("resolve"), NEW, CLOSED,
            EdgeType.DIRECTED);
        graph.addEdge(new Transition("resolve"), ACCEPTED, CLOSED,
                EdgeType.DIRECTED);
        graph.addEdge(new Transition("resolve"), REOPENED, CLOSED,
                EdgeType.DIRECTED);
        graph.addEdge(new Transition("resolve"), ASSIGNED, CLOSED,
                EdgeType.DIRECTED);

        graph.addEdge(new Transition("accept"), NEW, ACCEPTED,
                EdgeType.DIRECTED);
        graph.addEdge(new Transition("accept"), ACCEPTED, ACCEPTED,
                EdgeType.DIRECTED);
        graph.addEdge(new Transition("accept"), ASSIGNED, ACCEPTED,
                EdgeType.DIRECTED);
        graph.addEdge(new Transition("accept"), REOPENED, ACCEPTED,
                EdgeType.DIRECTED);

        graph.addEdge(new Transition("reassign"), REOPENED, ASSIGNED,
                EdgeType.DIRECTED);
        graph.addEdge(new Transition("reassign"), ASSIGNED, ASSIGNED,
                EdgeType.DIRECTED);
        graph.addEdge(new Transition("reassign"), NEW, ASSIGNED,
                EdgeType.DIRECTED);
        graph.addEdge(new Transition("reassign"), ACCEPTED, ASSIGNED,
                EdgeType.DIRECTED);

        FRLayout<State, Transition> layout = new FRLayout<State,
            Transition>(
                graph);
        layout.setRepulsionMultiplier(1.5);
        vv = new VisualizationImageServer<State, Transition>(layout,
            new Dimension(
                600, 400));
        vv.setBackground(Color.white);

        // this class will provide both label drawing and vertex
            shapes
        VertexLabelAsShapeRenderer<State, Transition> vlasr = new
            VertexLabelAsShapeRenderer<State, Transition>(
                vv.getRenderContext());

        Transformer<State, String> state2String = new Transformer<
            State, String>() {
             public String transform(State v) {
                return v.toString();
             }
        };
        vv.getRenderContext().setVertexShapeTransformer(vlasr);
        vv.getRenderContext().setVertexLabelTransformer(state2String)
            ;

        vv.getRenderer().setVertexRenderer(
                new GradientVertexRenderer<State, Transition>(Color.
                    MAGENTA,
                         Color.WHITE, true));
        vv.getRenderer().setVertexLabelRenderer(vlasr);

        Transformer<Transition, String> transition2String = new
            Transformer<Transition, String>() {
                public String transform(Transition e) {
                    return e.toString();
                }
            };
            vv.getRenderContext().setEdgeLabelTransformer(
                transition2String);
            AbstractEdgeShapeTransformer<State, Transition> aesf = new
                EdgeShape.QuadCurve<State, Transition>();
            aesf.setControlOffsetIncrement(30);
            vv.getRenderContext().setEdgeShapeTransformer(aesf);
            vv.getRenderContext().setEdgeStrokeTransformer(
                    new ConstantTransformer(new BasicStroke(2.5f)));

            ConstantDirectionalEdgeValueTransformer<State, Transition> mv
                 = new ConstantDirectionalEdgeValueTransformer<State,
                Transition>(
                    .5, .5);
            vv.getRenderContext().setEdgeLabelClosenessTransformer(mv);


            Image image = vv.getImage(new Point(300,200), new Dimension
                (600,400));

            BufferedImage bufferedImage = new BufferedImage(600, 400,
                    BufferedImage.TYPE_INT_RGB);

            bufferedImage.getGraphics().drawImage(image, 0, 0, null);


            ByteArrayOutputStream baos = new ByteArrayOutputStream();
            try {
                ImageIO.write(bufferedImage, "png", baos);
            } catch (IOException e1) {
                e1.printStackTrace();
            }
            String base64EncodedImage = Base64.encodeBase64String(baos
                    .toByteArray());

            System.out.println(base64EncodedImage.length());

            return base64EncodedImage;

        }
}
```

Quelltext A.1: Java XMLRPC-Dienst des ersten Prototyps

A.2 Technische Realisierung des zweiten Prototyps

Der zweite Prototyp ist wie bereits der erste Prototyp als Trac-Plugin realisiert. Das Genshi-Template bindet das Data-Driven-Documents-Sankey-Plugin (vgl.Abschnitt 2.4) zur Visualisierung des Sankey-Diagramms ein. Die Berechnung des Modells des Ticket-Status-Fluss-Graphen wird an eine Java Anwendung delegiert, die einen JAX-Webservice zum Abruf der JSON-Darstellung des Graphen implementiert. Die Java Anwendung greift über die Java SQL API und einen SqLite-JDBC-Treiber direkt auf die im lokalen Dateisystem vorhandene SqLite-Datenbank des Trac-Systems zu und extrahiert die Eigenschaft “Status” jedes Tickets und dessen Historie. Diese werden weiter zu einem Modell des Ticket-Status-Fluss-Graphen verarbeitet, das schließlich in die vom Sankey-Plugin verwendete JSON-Darstellung transformiert wird.

A.3 Java Implementierung der Sankey-Metrik

```java
package de.rwth.swc.gplcrs.controller;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.HashSet;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;
import java.util.Map.Entry;
import java.util.Set;

import javax.ejb.EJB;
import javax.ejb.Stateless;

import org.jgrapht.graph.DefaultDirectedWeightedGraph;
import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

import de.rwth.swc.gplcrs.dao.TicketChangeDaoLocal;
import de.rwth.swc.gplcrs.dao.TicketDaoLocal;
import de.rwth.swc.gplcrs.entity.Ticket;
import de.rwth.swc.gplcrs.entity.TicketChange;
import de.rwth.swc.gplcrs.sankey.graph.SankeyGraphEdge;
import de.rwth.swc.gplcrs.sankey.graph.SankeyGraphNode;
import de.rwth.swc.gplcrs.filter.Filter;
import de.rwth.swc.gplcrs.filter.FilterContext;

@Stateless
public class SankeyCalculatorBean implements SankeyCalculatorLocal {

    @EJB
    private TicketChangeDaoLocal ticketChangeDao;

    @EJB
    private TicketDaoLocal ticketDao;

    private DefaultDirectedWeightedGraph<SankeyGraphNode,
       SankeyGraphEdge> ticketChangeGraph;

    private List<Ticket> getFilteredTickets(String
       dataSourceIdentifier,
            List<Filter> filters) {

        List<Ticket> tickets = ticketDao.getAllTickets(
           dataSourceIdentifier);
        List<Ticket> filteredTickets = FilterContext.filterTickets(
           tickets,
                filters);
        return filteredTickets;
    }

    private List<TicketChange> getFilteredTicketChanges(
            String dataSourceIdentifier, String propertyName,
            List<Filter> filters) {

        List<Ticket> filteredTickets = getFilteredTickets(
           dataSourceIdentifier,
                filters);

        Set<Long> filteredTicketIds = new HashSet<Long>();

        for (Ticket filteredTicket : filteredTickets) {
            filteredTicketIds.add(filteredTicket.getTicketId());
        }

        List<TicketChange> ticketChanges = ticketChangeDao
                .getTicketChangesForProperty(dataSourceIdentifier,
                   propertyName);

        List<TicketChange> filteredTicketChanges = new ArrayList<
           TicketChange>();

        for (TicketChange ticketChange : ticketChanges) {
            if (filteredTicketIds.contains(ticketChange.getTicketId()
               )) {
                filteredTicketChanges.add(ticketChange);
            }
        }

        return filteredTicketChanges;
    }

    @SuppressWarnings("unchecked")
    private void calculateChangeGraph(String dataSourceIdentifier,
            String propertyName, List<Filter> filters) {

```

```java
        List<TicketChange> filteredTicketChanges = getFilteredTicketChanges(
                dataSourceIdentifier, propertyName, filters);

        // build ticket change map. Maps ticket ids to lists of String
        // representing the state changes
        Map<Long, LinkedList<String>> ticketChangeMap = new HashMap<Long, LinkedList<String>>();

        for (TicketChange ticketChange : filteredTicketChanges) {
            // for each ticket change: if we already have a flow for this
            // ticket, append the new state
            if (ticketChangeMap.containsKey(ticketChange.getTicketId())) {
                List<String> ticketFlow = ticketChangeMap.get(ticketChange
                        .getTicketId());
                ticketFlow.add(ticketChange.getNewValue());
                // otherwise create a new flow for this ticket id, and start
                // the chain with old value -> new value
            } else {
                LinkedList<String> ticketFlow = new LinkedList<String>();
                ticketFlow.add(ticketChange.getOldValue());
                ticketFlow.add(ticketChange.getNewValue());
                ticketChangeMap.put(ticketChange.getTicketId(), ticketFlow);
            }
        }

        // build the graph state Map. States are represented by their ticket
        // change history

        ticketChangeGraph = new DefaultDirectedWeightedGraph<SankeyGraphNode, SankeyGraphEdge>(
                SankeyGraphEdge.class);
        Map<List<String>, SankeyGraphNode> nodeMap = new HashMap<List<String>, SankeyGraphNode>();

        for (Entry<Long, LinkedList<String>> ticketChangeEntry : ticketChangeMap
                .entrySet()) {
            LinkedList<String> clonedTicketChange = (LinkedList<String>) ticketChangeEntry
                    .getValue().clone();
            while (!clonedTicketChange.isEmpty()) {
                // create node in ticketflow graph if it does not exist already
                SankeyGraphNode sankeyGraphNode = new SankeyGraphNode
                    (
                        clonedTicketChange);
                if (!ticketChangeGraph.containsVertex(sankeyGraphNode
                    )) {
                    ticketChangeGraph.addVertex(sankeyGraphNode);
                    // put the node into the node map so it can be
                        retrieved by
                    // later ticketchanges regarding the same node
                    nodeMap.put(clonedTicketChange, sankeyGraphNode);
                } else {
                    sankeyGraphNode = nodeMap.get(clonedTicketChange)
                        ;
                }
                // add ticket id to the set of tickets represented by
                    this
                // node
                sankeyGraphNode.getTicketIds().add(ticketChangeEntry.
                    getKey());

                clonedTicketChange = (LinkedList<String>)
                    clonedTicketChange
                        .clone();
                clonedTicketChange.removeLast();
            }
        }

        for (Entry<Long, LinkedList<String>> ticketChangeEntry :
            ticketChangeMap
                .entrySet()) {
            LinkedList<String> clonedTicketChange = (LinkedList<
                String>) ticketChangeEntry
                    .getValue().clone();
            while (clonedTicketChange.size() >= 2) {
                LinkedList<String> targetState = (LinkedList<String>)
                    clonedTicketChange
                        .clone();
                SankeyGraphNode targetNode = nodeMap.get(targetState)
                    ;
                clonedTicketChange = (LinkedList<String>)
                    clonedTicketChange
                        .clone();
                clonedTicketChange.removeLast();
                LinkedList<String> sourceState = (LinkedList<String>)
                    clonedTicketChange
                        .clone();
                SankeyGraphNode sourceNode = nodeMap.get(sourceState)
                    ;

                SankeyGraphEdge edge;
                if (ticketChangeGraph.containsEdge(sourceNode,
                    targetNode)) {
```

```java
                    edge = ticketChangeGraph.getEdge(sourceNode, targetNode);
                    double weight = ticketChangeGraph.getEdgeWeight(edge);
                    ticketChangeGraph.setEdgeWeight(edge, weight + 1);
                } else {
                    edge = ticketChangeGraph.addEdge(sourceNode, targetNode);
                }
                edge.getTicketIds().add(ticketChangeEntry.getKey());

            }
        }

    }

    private String toJson() {

        String jsonString;

        Map<List<String>, Integer> stateMap = new HashMap<List<String>, Integer>();
        int stateCounter = 0;

        JSONObject jsonObj = new JSONObject();

        JSONArray nodeList = new JSONArray();
        JSONArray linkList = new JSONArray();

        try {
            for (SankeyGraphNode node : ticketChangeGraph.vertexSet()) {
                List<String> state = node.getState();
                Set<Long> ticketIds = node.getTicketIds();
                JSONObject nodeObj = new JSONObject();
                if (state.get(state.size() - 1) == null) {
                    nodeObj.put("name", "null");
                } else {
                    nodeObj.put("name", state.get(state.size() - 1));
                }
                nodeObj.put("tickets", ticketIds);
                nodeList.put(nodeObj);

                stateMap.put(state, Integer.valueOf(stateCounter));
                stateCounter++;
            }

            jsonObj.put("nodes", nodeList);

            for (SankeyGraphEdge transition : ticketChangeGraph.edgeSet()) {
                JSONObject linkObj = new JSONObject();
                linkObj.put("source", stateMap.get(ticketChangeGraph
                        .getEdgeSource(transition).getState()));
                linkObj.put("target", stateMap.get(ticketChangeGraph
                        .getEdgeTarget(transition).getState()));
                linkObj.put("value",
                        ticketChangeGraph.getEdgeWeight(transition));
                linkObj.put("tickets", transition.getTicketIds());

                linkList.put(linkObj);
            }

            jsonObj.put("links", linkList);
            jsonString = jsonObj.toString();

        } catch (JSONException e) {
            jsonString = "";
        }

        return jsonString;
    }

    @Override
    public String getSankeyJson(String dataSourceIdentifier,
            String propertyName, List<Filter> filters) {

        calculateChangeGraph(dataSourceIdentifier, propertyName,
           filters);
        return toJson();
    }

    @Override
    public Long getNumberOfTicketsWithChangedProperty(
            String dataSourceIdentifier, String propertyName,
            List<Filter> filters) {

        List<TicketChange> filteredTicketChanges =
           getFilteredTicketChanges(
                dataSourceIdentifier, propertyName, filters);
        Set<Long> ticketIds = new HashSet<Long>();
        for (TicketChange filteredTicketChange :
           filteredTicketChanges) {
            ticketIds.add(filteredTicketChange.getTicketId());
        }

        return Long.valueOf(ticketIds.size());
    }
}
```

Quelltext A.2: SankeyCalculatorBean.java

A.4 Ticket-Daten-Modell: SQL-Skript zur Erzeugung des Datenbank-Schemas

```
-- -----------------------------------------------------
-- DROP tables
-- -----------------------------------------------------

-- disable foreign key contraint checks while dropping tables, so
    drop order does not matter
SET foreign_key_checks = 0;

DROP TABLE IF EXISTS TICKETCHANGE;
DROP TABLE IF EXISTS TICKET;

DROP TABLE IF EXISTS TICKET_PROPERTY;

-- reactivate foreign key constraint checks
SET foreign_key_checks = 1;

-- -----------------------------------------------------
-- Table TICKETCHANGE
-- -----------------------------------------------------

CREATE TABLE TICKETCHANGE (
  ID INTEGER AUTO_INCREMENT NOT NULL,
  DATASOURCEIDENTIFIER VARCHAR(255),
  TICKETID BIGINT,
  CHANGETIME BIGINT,
  PROPERTYNAME VARCHAR(255),
  OLDVALUE VARCHAR(255),
  NEWVALUE VARCHAR(255),
  PRIMARY KEY (ID)
)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;

-- -----------------------------------------------------
-- Table TICKET
-- -----------------------------------------------------

CREATE TABLE TICKET (
  ID INTEGER AUTO_INCREMENT NOT NULL,
  DATASOURCEIDENTIFIER VARCHAR(255),
  TICKETID BIGINT,
  CHANGETIME BIGINT,
  SUBJECT VARCHAR(255),
  URL VARCHAR(255),
  PRIMARY KEY (ID),
  CONSTRAINT UI_TICKET UNIQUE (DATASOURCEIDENTIFIER, TICKETID)
)
ENGINE = InnoDB
```

```sql
DEFAULT CHARACTER SET = utf8;

-- ---------------------------------------------------------
-- Table TICKET_PROPERTY
-- ---------------------------------------------------------

CREATE TABLE TICKET_PROPERTY (
  TICKET_ID INTEGER NOT NULL,
  PROPERTYNAME VARCHAR(255) NOT NULL,
  PROPERTYVALUE VARCHAR(65535) NOT NULL,
  PRIMARY KEY (TICKET_ID, PROPERTYNAME),
  CONSTRAINT FK_TICKET_PROPERTY_TICKET_ID
    FOREIGN KEY (TICKET_ID)
    REFERENCES TICKET (ID)
)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;
```

Quelltext A.3: SQL-Skript zur Erzeugung des Datenbank-Schemas (createDatabase.sql)

A.5 Ticket-Daten-Verarbeitung: Message-Driven-Bean zum Empfang einer TicketJournalMessage

```java
[...]
@MessageDriven(mappedName = JMSConfig.TOPIC, activationConfig = {
                @ActivationConfigProperty(propertyName = "destinationType", propertyValue = "javax.jms.Topic"),
                @ActivationConfigProperty(propertyName = "destination", propertyValue = JMSConfig.TOPIC),
                @ActivationConfigProperty(propertyName = "acknowledgeMode", propertyValue = "Auto-acknowledge"),
                @ActivationConfigProperty(propertyName = "messageSelector", propertyValue = "messageType = 'TicketJournalMessage'") })
public class TicketJournalMessageReceiver extends AbstractMessageReceiver {
[...]
        @Override
        public void receiveMessage(BaseGplcrsMessage message) {

                TicketJournalMessage ticketJournalMessage = (TicketJournalMessage) message;
                List<Ticket> ticketJournal = ticketJournalMessage.getTicketJournal();
                String dataSourceIdentifier = ticketJournalMessage
                                .getDataSourceIdentifier();
                Long ticketId = ticketJournalMessage.getTicketId();

                // clear if exists
        if (ticketDao.ticketExists(dataSourceIdentifier, ticketId)) {
            // ticket
            Ticket ticketToDelete = ticketDao.getTicketById(
                        dataSourceIdentifier, ticketId);
            ticketDao.deleteTicket(ticketToDelete);

            // and its ticket changes
            List<TicketChange> ticketChangesToDelete = ticketChangeDao
                        .getAllTicketChangesOfTicket(dataSourceIdentifier, ticketId);
            for (TicketChange ticketChangeToDelete : ticketChangesToDelete) {
                ticketChangeDao.deleteTicketChange(ticketChangeToDelete);
            }
        }

        // build ticketchanges
        Ticket currentTicketState = null;
        for (Ticket newTicketState : ticketJournal) {
            if (currentTicketState != null) {
                ticketMessageHelper.createTicketChanges(dataSourceIdentifier,
                            currentTicketState, newTicketState);
            }
            currentTicketState = newTicketState;
        }

        // create a final Ticket State after parsing through the log, that
        // is managed by the entity manager
        // previousTicketState should never be null, since the DataSourceBeans
        // always send at least one ticket in a ticketJournalMessage.
        // It could be null due to bugs or if we receive a bogus/malicious
        // message
        if (currentTicketState != null) {
            entityFactory.createTicket(dataSourceIdentifier,
                        currentTicketState.getTicketId(),
                        currentTicketState.getSubject(),
                                             currentTicketState.getUrl(),
                                             currentTicketState.
                                                 getChangeTime(),
                                             currentTicketState.
                                                 getProperties());
                    }

                    // Count the same message once only
                    if (!isRedelivered()) {
                            loadProgress.incProcessedCurrentTicket();
                    }

          }
}
```

Quelltext A.4: Auszug aus TicketJournalMessageReceiver.java

A.6 Ticket-Daten-Aktualisierung: XMLRPC-Dienst und Trac-Plugin

```
package de.rwth.swc.gplcrs.xmlrpc;

import java.io.IOException;
import java.io.InputStream;
import java.util.Date;
import java.util.Map;
import java.util.Properties;

import javax.jms.JMSException;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.xml.bind.JAXBException;

import org.apache.commons.lang3.StringUtils;
import de.rwth.swc.gplcrs.entity.Ticket;
import de.rwth.swc.gplcrs.facade.GplcrsFacadeLocal;
import de.rwth.swc.gplcrs.jms.MessageSenderImpl;
import de.rwth.swc.gplcrs.jms.message.MessageFactoryLocal;

public class GplcrsXmlRpcService {

    private static final String EJB_COMMON_MODULE_BASENAME = "ejb-
        common";
    private static final String EJB_CORE_MODULE_BASENAME = "ejb-core"
        ;

    private GplcrsFacadeLocal facade;

    private MessageSenderImpl messageSender;

    private MessageFactoryLocal messageFactory;

    private void lookupEjbs() throws IOException, NamingException {

        // load the version from properties file
        Properties props = new Properties();
        InputStream is = getClass().getResourceAsStream("xmlrpc.
            properties");
        try {
            props.load(is);
        } finally {
            is.close();
        }

        // construct jndi module names
        String projectVersion = props.getProperty("projectVersion");
        String ejbCommonModuleName = EJB_COMMON_MODULE_BASENAME + "-"
                + projectVersion;
        String ejbCoreModuleName = EJB_CORE_MODULE_BASENAME + "-"
                + projectVersion;

        // lookup session beans in jndi
        InitialContext ic = new InitialContext();
        facade = (GplcrsFacadeLocal) ic.lookup("java:app/" +
            ejbCoreModuleName
                + "/GplcrsFacadeBean");

        messageSender = (MessageSenderImpl) ic.lookup("java:app/"
                + ejbCommonModuleName + "/MessageSenderImpl");

        messageFactory = (MessageFactoryLocal) ic.lookup("java:app/"
                + ejbCommonModuleName + "/MessageFactoryBean");

    }

    public String update(String dataSource, Integer ticketId, String
        subject,
            String url, Date changeTime, Map<String, Object>
                ticketValues)
            throws IOException, NamingException, JMSException,
                JAXBException {

        lookupEjbs();

        final String usage = "Usage: ticket.update(Data Source(string
            ), Ticket Id(int),  Subject(string), Url(string), Change
            Time(dateTime.iso8601), Properties(struct))";

        // Input parameter checking
        if (dataSource == null) {
            throw new IllegalArgumentException("Data Source must not
                be null. "
                    + usage);
        }

        if (ticketId == null) {
            throw new IllegalArgumentException("Ticket Id must not be
                 null. "
                    + usage);
        }

        if (subject == null) {
            throw new IllegalArgumentException("Subject must not be
                null. "
                    + usage);
        }

        if (url == null) {
            throw new IllegalArgumentException("Url must not be null.
                 " + usage);
        }

        if (changeTime == null) {
            throw new IllegalArgumentException("Change Time must not
                be null. "
                    + usage);
        }

        if (ticketValues == null) {
            throw new IllegalArgumentException("Properties must not
                be null. "
                    + usage);
        }

        if (!facade.getDataSourceIdentifiers("").contains(dataSource)
            ) {
            throw new IllegalArgumentException("Data Source " +
                dataSource
                    + " does not exist.");
        }

        Ticket ticket = new Ticket();
        ticket.setDataSourceIdentifier(dataSource);
        ticket.setTicketId(Long.valueOf(ticketId.longValue()));
        ticket.setSubject(subject);
        if (StringUtils.isNotBlank(url)) {
            ticket.setUrl(url);
        }
        ticket.setChangeTime(Long.valueOf(changeTime.getTime()));

        for (String propertyName : ticketValues.keySet()) {
            Object propertyValue = ticketValues.get(propertyName);
            // carry over string properties, only. effectively
                ignoring Date
            // properties changetime and time
```

```java
            // if there are other Date properties that are interesting in
            // analysis, this might pose a problem
            if (propertyValue instanceof String) {
                ticket.setProperty(propertyName, (String) propertyValue);
            }
        }

        messageSender.sendMessage(messageFactory.createTicketMessage(
                dataSource, ticket));

        return "Ticket " + ticketId + " updated.";
    }
}
```

Quelltext A.5: GplcrsXmlRpcService.java

```python
# gplcrs plugin

from trac.core import *
from trac.config import Option
from trac.util.text import empty
from trac.ticket.api import ITicketChangeListener

import xmlrpclib
import socket

class GplcrsPlugin(Component):
    implements(ITicketChangeListener)

    GPLCRS_CONFIG_SECTION = "gplcrs"

    gplcrsRestUrl = Option(GPLCRS_CONFIG_SECTION, "rest_url", "",
        doc="Rest Url of gplcrs. E.g. ’http://localhost:8080/gplcrsRest/’. Note the final slash. Required setting.")

    gplcrsDataSource = Option(GPLCRS_CONFIG_SECTION, "data_source", "",
        doc="Name of the gplcrs Data Source to update with ticket change notifications. E.g. ’TRAC - http://localhost:8000/trac’. Required setting.")

    def __init__(self):
        self.baseUrl = self.config.get("trac","base_url")

    # ITicketChangeListener Interface
    def ticket_created(self, ticket):
        """Notifies a gplcrs system about a ticket creation"""
        self.__updateOrCreateTicket(ticket)

    def ticket_changed(self, ticket, comment, author, old_values):
        """Notifies a gplcrs system about a ticket change"""
        self.__updateOrCreateTicket(ticket)

    def ticket_deleted(self, ticket):
        """NOP when a ticket is deleted."""

    def __updateOrCreateTicket(self, ticket):
        ticketValues = {}
        for propertyName in ticket.values.keys():
            # do not copy empty values, they mean "not set" and are not marshallable
            if (type(ticket.values[propertyName]) != type(empty)):
                ticketValues[propertyName] = ticket.values[propertyName]
        # construct an url to the ticket, if a base url is set
        url = ""
        if (self.baseUrl != ""):
            url = self.baseUrl + "/ticket/" + str(ticket.id)
        self.__sendTicketUpdate(ticket.id, ticket.values['summary'], url, ticket.values['changetime'],  ticketValues)

    def __sendTicketUpdate(self, id, subject, url, changetime, values):
        """sends a ticket update to gplcrs XML RPC"""

        if (self.gplcrsRestUrl == ""):
            self.log.warn("option rest_url not set")
            return
        if (self.gplcrsDataSource == ""):
            self.log.warn("option data source not set")
            return
        xmlRpcServer = xmlrpclib.ServerProxy(self.gplcrsRestUrl, use_datetime=1, allow_none=1)
        try:
            rpcResult = xmlRpcServer.ticket.update(self.gplcrsDataSource, id, subject, url, changetime, values)
            self.log.debug("rpcResult = " + rpcResult)
        except socket.error, (value,message):
            self.log.error(str(value) + ": " + message);
        except xmlrpclib.Fault as err:
            self.log.error("A fault occurred")
            self.log.error("Fault code: %d", err.faultCode)
            self.log.error("Fault string: %s", err.faultString)
        except xmlrpclib.ProtocolError as err:
            self.log.error("A protocol error occurred")
            self.log.error("URL: %s", err.url)
            self.log.error("HTTP/HTTPS headers: %s", err.headers)
            self.log.error("Error code: %d", err.errcode)
            self.log.error("Error message: %s", err.errmsg)
```

Quelltext A.6: Trac-Plugin gplcrs.py