ITSM mit Open-Source-Software

Kombinierbar

Gabriel Jülke

Aus der geschickten Kombination von Open-Source-Projekten wie i-doit und OTRS entsteht eine flexible Anwendung zum IT Service Management, die sich auf die eigenen Bedürfnisse zuschneiden lässt.

Proprietäre ITSM-Angebote locken mit ihrem großen Funktionsumfang und der Bequemlichkeit der Interoperabilität: alle Informationen immer auf einen Blick, alles arbeitet Hand in Hand. Open-Source-Alternativen sind zwar kostengünstiger, weniger komplex und schneller auszurollen, decken aber oft nur einzelne Teilbereiche ab. Es droht ein Konglomerat aus voneinander abgeschotteten Systemen.

Hier hilft die offene Natur von Open-Source-Software. Mit den richtigen Tools, etwas Know-how und ein bisschen Programmieren entsteht eine intelligente, vernetzte ITSM-Anwendung. Wie so etwas aussehen kann, soll hier am Beispiel Prozessmanagement demonstriert werden: Open-Source-based Enterprise IT Service Management.

Die Lösung besteht aus fünf Komponenten, von denen jede eine Teilaufgabe übernimmt:

– Als Workflow-Engine zum Designen der Prozesse und Managen des Ablaufes dient die Community Edition von OTRS 6.
– Zur Dokumentation der prozesskritischen Hardware kommt die Configuration Management Database (CMDB) i-doit zum Einsatz.
– SNAG-View übernimmt die Überwachung der beteiligten Systeme und Services.
– Das Network Discovery Tool NeDi hilft bei der Inventarisierung.
– Für Statistik-Fans (und die Geschäftsleitung) erstellt der Jasper Report Server frei konfigurierbare Reports und kann dazu ebenso frei wählbare Datenquellen nutzen.

iX-TRACT

- Leistungsfähiges IT Service Management ist auch mit Open Source möglich, wenn man mehrere Projekte miteinander verknüpft.
- OTRS bildet dabei die Prozesse ab, SNAG-View oder Icinga2 überwacht die Hardware und in i-doit laufen alle Informationen zusammen.
- Dabei muss man mit einigen zusätzlichen Modulen für die Integration der Open-Source-Projekte sorgen.

OTRS verknüpft Tickets und Prozesse

OTRS ist das zentrale Element. Das optionale Modul „Process Management“ erweitert das auf Perl basierende Ticketsystem um eine mächtige Workflow-Engine, die auch komplexe Abläufe abbilden kann. Da OTRS keine Abhängigkeiten zu den anderen Tools hat, bietet es sich an, das Ticketsystem als Erstes auszurollen. So kann man zügig mit dem Erstellen der Prozesse beginnen.

Diese bestehen aus Aktivitäten, die die einzelnen Schritte repräsentieren und in OTRS über selbst definierbare Formulare abgebildet werden. Hier lassen sich sowohl vordefinierte Standardfelder als auch selbst definierte „DynamicFields“ verwenden. Verknüpft werden die Aktivitäten mit Übergängen, die sich über Filter aktivieren lassen und Übergangsaktionen ausführen. Diese legen beispielsweise mehrere Subtickets zur parallelen Abarbeitung gleichzeitig laufender Prozessschritte an, starten andere Prozesse oder verschicken Benachrichtigungen.

Über selbst geschriebene Perl-Module lassen sich eigene Übergangsaktionen zum System hinzufügen. Dazu legt man zunächst eine neue Datei im Ordner *<otrs>/Kernel/System/ProcessManagement/TransitionAction/* an. Eine Namenskonvention gibt es nicht, allerdings dient der Dateiname als Bezeichnung für die TransitionAction im Frontend, er sollte daher sinnig gewählt sein.

Das Modul muss zwingend von der Basisklasse *Kernel::System::ProcessManagement::TransitionAction::Base* erben, damit es das System als valide TransitionAction akzeptiert. Zudem sind zwei Methoden zu implementieren: Während *new* die Initialisierungslogik enthält, wird *run* vom Prozessmanager ausgeführt, wenn die Transition getriggert wird. Hier kann man das Formular der

vorangegangenen Aktivität auslesen und auf der Grundlage der Prozessinformationen eigene Aktionen ausführen, beispielsweise Funktionen anderer Systeme über APIs aufrufen. Die Antwort der Calls kann wiederum in den Prozess einfließen und die nächsten Schritte beeinflussen.

Randnotiz: Es empfiehlt sich, aus Gründen der Wartbarkeit und Wiederverwendbarkeit die Geschäftslogik in eigenen Perl-Modulen außerhalb der Actions zu speichern. Solcher Code wird in OTRS üblicherweise unter *<otrs>/Custom/Kernel/System* abgelegt. In der TransitionAction können die Module dann über die Funktion *$Kernel::OM->Get('<Namespace>::<Module>')* geladen und verwendet werden.

Widgets sorgen für Zusatzfunktionen

Neben eigenen Übergangsaktionen lassen sich auch modulspezifische Einstellungen definieren, beispielsweise zum Hinterlegen der Zugangsdaten zu Fremdsystemen. Über eigene Widgets, die OTRS bei der Ticketerstellung oder der Ticketansicht einblendet, kann man zusätzliche Informationen und Funktionen ins OTRS bringen.

Ein Widget besteht aus drei Teilen: der Konfiguration, einem Template und der Widget-Logik. Die Konfiguration wird als XML-Datei unter *otrs/Kernel/Config/Files/XML/* abgelegt und erscheint im OTRS später in der globalen Konfiguration. Sie definiert den Widget-Namen, seine Position (Seitenleiste oder unter der Artikelanzeige), die anzuzeigenden Felder sowie Defaultwerte und referenziert die Templatedatei und das Perl-Modul mit der Widget-Logik.

Für seine Templates benutzt OTRS das Template-Toolkit von Perl mit eigenen Erweiterungen für die automatische Übersetzung und Formatierung von Texten. Hier legt man die grundsätzliche HTML-Struktur fest und definiert Platzhalter, die beim Rendering mit Daten befüllt werden. Templates laden unter *otrs/Kernel/Output/HTML/Templates/Standard/* in Dateien mit der Erweiterung *.tt*. Unterordner zur Sortierung sind natürlich möglich.

Die Geschäftslogik für das Template kommt in ein eigenes Perl-Modul, das unter *otrs/Kernel/Output/HTML/* abgelegt wird. Das Modul muss zwingend vom Basisobjekt *Kernel::Output::HTML::Base* erben und eine *run*-Methode implementieren. Sie bereitet die Daten vor und ruft anschließend das OTRS-Layoutobjekt auf. Dieses befüllt das Template über die *Output*-Methode mit den Daten.

So lässt sich einrichten, dass der Bearbeiter beim Erstellen eines Tickets für das Incident Management das defekte Gerät direkt aus einer aus den i-doit-Daten gespeisten Liste auswählen kann. Das Widget speichert die i-doit-ID in den Ticketinformationen, wodurch dann ein zweites Widget in der Ticketansicht den CMDB-Status holen und anzeigen kann.

Ist das Gerät in i-doit als überwachtes Objekt registriert, ist dort auch die ID aus dem Monitoringsystem gespeichert und wird gleich mit abgerufen. Mit dieser Information liefert ein weiteres Widget in der Ticketansicht den Monitoring-Livestatus. So kommen im Ticket die Informationen aus allen drei Tools zusammen und ergänzen sich.

Ist das Prozessmanagement ausgerollt, kann man die weiteren Tools installieren und vorbereiten. Die CMDB i-doit dokumentiert nicht nur die an den Prozessen beteiligte Hardware, sondern verwaltet

auch Nutzer, Verantwortliche und Lizenzen. Sie dient gleichzeitig als Datengrundlage für das Monitoring mit SNAG-View.

Hardware überwachen

Letzteres ist ein proprietäres Monitoringprogramm, das auf Nagios aufsetzt. Neben vielen nützlichen Visualisierungsmöglichkeiten erweitert SNAG-View Nagios um ein detailliertes Berechtigungskonzept und ein mächtiges Vererbungssystem. Außerdem gibt es bereits Kopplungsmodule für OTRS und i-doit, ein weiteres Modul bringt das Network Discovery Tool NeDi direkt vorinstalliert mit. Die Public API erlaubt es, skriptgesteuert Aktionen auszuführen.

Zusammengehörige Checks (Services) bündelt SNAG-View in Serviceprofilen. Das Vererbungssystem erlaubt es, eine Grundkonfiguration inklusive aller benötigten Serviceprofile für einen zu überwachenden Server in einer Hostgruppe zu hinterlegen. Weist man einen neuen Host einer solchen Hostgruppe zu, bekommt er alle hier definierten Konfigurationseinstellungen und Services vererbt. Hostgruppen können auch ineinander verschachtelt werden. So lässt sich eine detaillierte Konfiguration vorbereiten, die später das Erstellen einzelner Hosts automatisiert.

Möchte man ganz bei Open Source bleiben, bietet sich Icinga2 mit seinem Web-Frontend IcingaWeb2 an. Die Pflege der Monitoringobjekte erfolgt grundsätzlich über Konfigurationsdateien, diese können aber auch fast vollständig über die */config*-API übergeben werden. Icinga2 legt die so übermittelten Konfigurationsdateien an, prüft diese und aktiviert sie automatisch, wenn keine Fehler gefunden wurden. Einziger Nachteil: Es muss immer die komplette Konfiguration übergeben werden, Updates einzelner Objekte sind so etwas aufwendiger.

Das in PHP geschriebene Web-Frontend IcingaWeb2 bietet einige Schnittstellen, um die Oberfläche mit eigenen Modulen zu erweitern und zu ergänzen. So lassen sich in die Detailansichten zusätzliche Abschnitte einfügen, die beispielsweise eine Liste der mit dem Objekt verknüpften OTRS-Tickets oder Informationen aus i-doit darstellen.

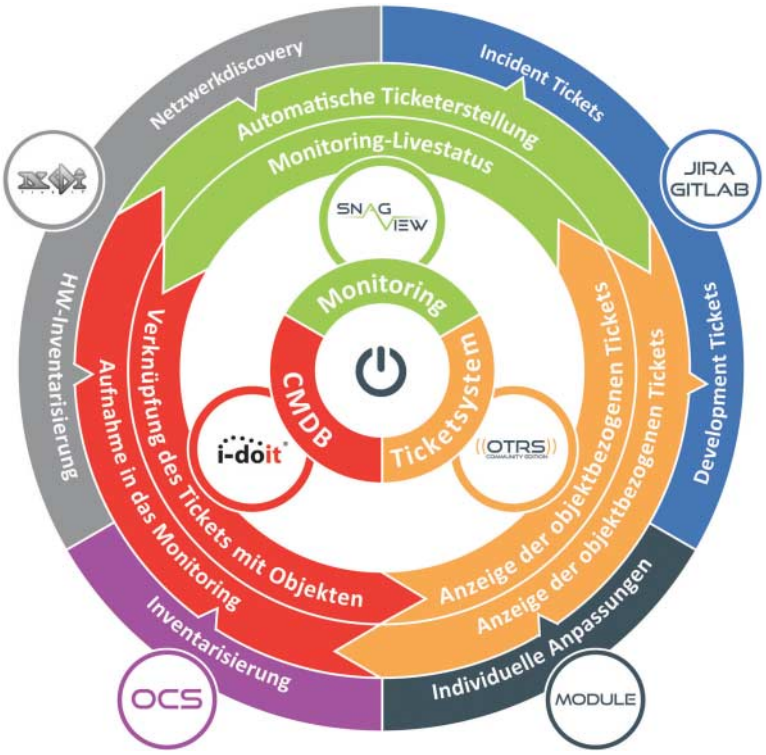
Was ist los im Netz?

Bevor man das Monitoring jetzt mit Leben füllen kann, muss die IT-Infrastruktur in i-doit dokumentiert werden. Findige Administratoren machen das natürlich nicht händisch, sondern überlassen diese Arbeit einem Network Discovery Tool wie NeDi. Die Software des Schweizer Netzwerkexperten Remo Rickli nutzt hauptsächlich die Protokolle SNMP, CDP und LLPD und hangelt sich vom Startknoten durch das gesamte Netzwerk. Die umfangreichen Scans liefern neben Stammdaten wie Hersteller, Typ und IP-Adresse auch die Netzwerktopologie und erkennen, welche Geräte an den Ports eines Switches hängen.

Neben dem Standardskript *nedi.pl*, das Server und Netzwerkgeräte wie Switches scannt, gibt es seit Kurzem auch *nodi.pl*. Dieses ist speziell für Endgeräte wie Desktop-PCs entwickelt und kann beispielsweise aus Windows-Clients die dort installierten Rollen auslesen. Zusammen kartografieren die beiden Skripte das komplette Netzwerk bis hin zu den Clients. NeDi unterscheidet dabei zwischen Servern und Netzwerkgeräten wie Switches (Devices) und Endgeräten (Nodes) und pflegt diese separat.

Besonders schön ist, dass das Tool sich die gefundenen Geräte merkt und so bei regelmäßigen Scans neue Geräte erkennt und seine Datenbank aktualisiert: die perfekte Datengrundlage für die IT-Dokumentation. Die Daten lassen sich über den Zugriff auf die NeDi-MySQL-Datenbank direkt nutzen.

Neben den Perl-Skripten zur Geräte-Identifikation besitzt NeDi auch ein in PHP geschriebenes Frontend, um die gewonnenen Daten anzuzeigen, Maps zu generieren und Konfigurationen vorzunehmen. Alternativ zum direkten Datenbankzugriff gibt es den Endpunkt */query.php*, über den SQL-Statements per API-Call ausgeführt werden können.



Aus der Kombination verschiedener Open-Source-Tools ergibt sich eine flexible ITSM-Anwendung (Abb. 1).

Alle Informationen an einer Stelle

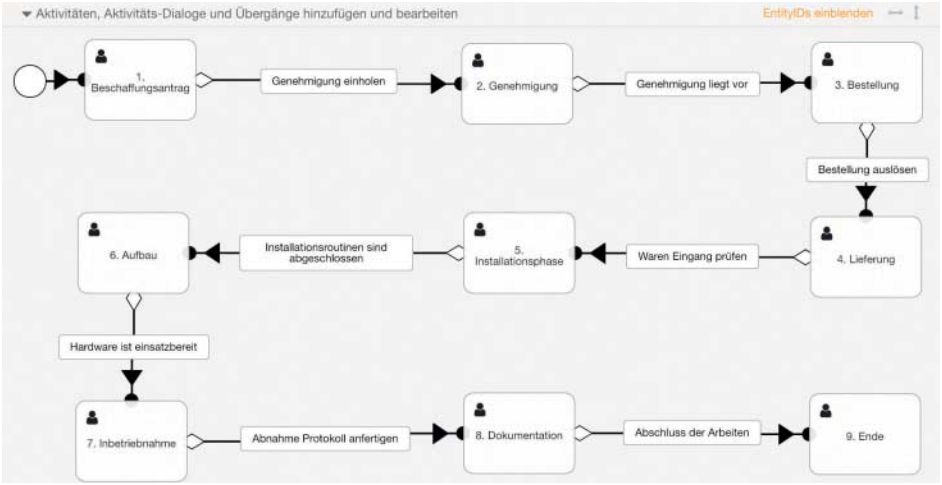
Die NeDi-Datenbank dient als Grundlage für die CMDB i-doit. Auch hier lassen sich sowohl die Oberfläche als auch die Hintergrundprozesse mit eigener Logik erweitern. Hierzu gibt es im Backend eine Vielzahl von Event-Hooks, über die auf nahezu jede Aktion im Frontend reagiert werden kann. synetics, das Unternehmen hinter i-doit, bietet eine gute Onlinedokumentation (siehe ix.de/ix1903082).

Ein eigenes Modul kann den Zugang zur NeDi-Datenbank konfigurieren und eine Importmaske implementieren, die alle gefundenen Objekte als Configuration Items (CI) in i-doit anlegt. Mit ein wenig mehr Mühe lassen sich sogar Filter erstellen, um nur einen Teil der NeDi-Geräte zu importieren. Ganz Findige schreiben sich einen eigenen API-Endpunkt, über den der Import angestoßen wird.

Per Cronjob über ein Skript getriggert können so automatisch alle von NeDi neu erkannten Geräte und Updates regelmäßig in i-doit übernommen werden. Aus NeDi lassen sich detaillierte Daten nutzen, sodass i-doit sogar die aktuelle Port-Belegung von Switches abbilden kann. Sind die in i-doit verfügbaren Zustände und Objekttypen nicht ausreichend, lassen sich eigene definieren. So kann man die CIs um eine neue Kategorie „Monitoring“ erweitern, die die Hostgruppen-Zuordnung definiert und das Monitoring aktiviert oder deaktiviert.

Vom Monitoring in die CMDB

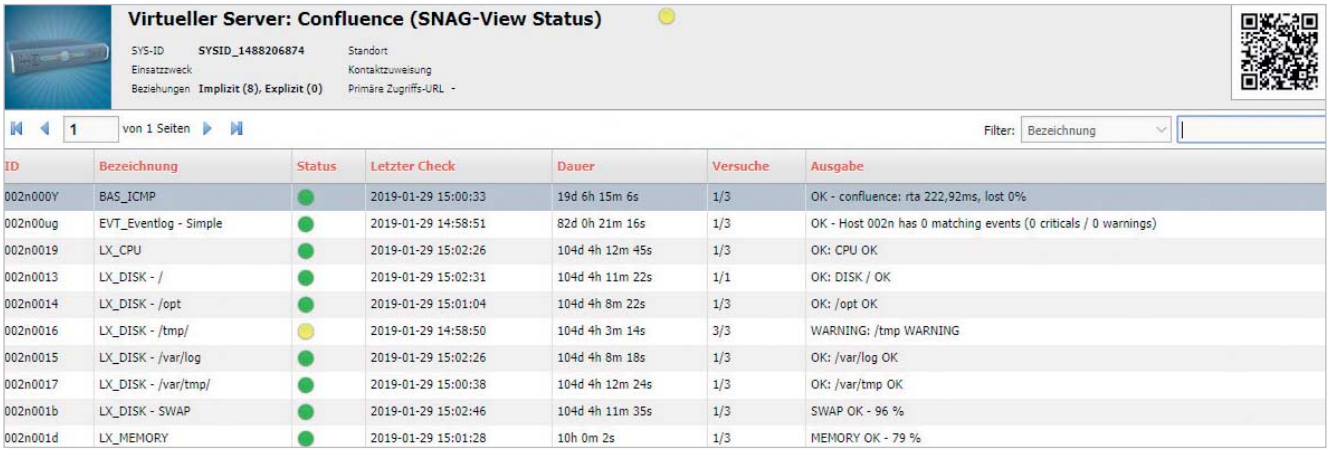
Die Hostgruppen in SNAG-View werden zu diesem Zweck auf i-doit-Gruppen gemappt, diese lassen sich dann ganz i-doit-konform den CIs zuordnen. Ein selbst geschriebenes Modul liest das Mapping in i-doit aus und überträgt die gewünschte Hostgruppe einfach bei der Erstellung oder beim Update an den SNAG-View-Server, dieser kümmert sich dann um das Anlegen der Objekte.



Der Prozess zur Bestellung eines Servers (Abb. 2)

SNAG-View liefert dabei eine Host-ID zurück, über die sich das CI direkt mit dem Host verknüpfen lässt. Nun kann i-doit den Livestatus des Monitorings anzeigen. Auch eine Liste der überwachten Services lässt sich in der Detailansicht einbinden. Damit ist es nicht mehr nötig, ins Monitoring zu wechseln, um den aktuellen Status abzufragen. Über eine weitere zusätzliche Kategorie „OTRS“ lassen sich alle verknüpften Tickets anzeigen. Nun kommen alle verfügbaren Informationen in i-doit zusammen.

Die ITSM-Lösung ist im Grunde fertig, wäre da nicht die Geschäftsleitung, die monatliche Berichte haben möchte. Doch auch hier gibt es ein passendes Open-Source-Tool: Der Jasper Report Manager von Jaspersoft ist eine sehr mächtige Java-Report-Engine. Mithilfe des kostenlosen Jaspersoft Studio kann sich jeder mit ein bisschen Einarbeitung eigene Reports zaubern. Um an die Daten zu kommen, lassen sich APIs als Quellen definieren, aber auch direkte SQL-Querys sind möglich. Zugegeben ist die Lern-



i-doit zeigt den Status des Servers an (Abb. 3).

kurve für die Jasper Reports recht hoch, doch der Aufwand lohnt sich.

In der Theorie mag das noch etwas vage klingen, daher soll nun ein praktisches Beispiel die Fähigkeiten der bidirektionalen Schnittstellen zeigen – die Bestellung eines neuen Servers. Der Prozess ist simpel und geradlinig; der Einfachheit halber wurde die Möglichkeit, Schritte zu wiederholen, weggelassen (siehe Abbildung 2).

Das initiale Formular ist ein Beschaffungsantrag, der auch die grundsätzlichen Anforderungen des neuen Servers definiert. Beim Abschicken des Formulars weist der Prozess das Ticket automatisch einem Vorgesetzten zu, der den Antrag prüft und genehmigt.

Sobald das Ticket zurückkommt, kann die Hardware bestellt werden. Im Bestelldialog sind dabei ein paar Informationen zum neuen System zu hinterlegen. Während der neue Server unterwegs ist, übermittelt eine eigene TransitionAction die Daten im Hintergrund an i-doit, wo bereits das Configuration Item für den neuen Server angelegt wird. Status: „bestellt“.

Sobald die Lieferung eingetroffen ist, wird sie geprüft und aufgesetzt. Im Hintergrund benachrichtigen beide Schritte i-doit, sodass der Status des CI über „geliefert“ zu „getestet“ wechselt. Sobald der neue Server in Betrieb genommen ist, folgt der nächste große Schritt: Im Formular wird der Verantwortliche für die Wartung des Servers hinterlegt und der Prozess abgeschlossen.

Im Hintergrund passieren dabei mehrere Dinge: In i-doit wechselt der Status auf „in Betrieb“, dabei wird der benannte Verantwortliche gesetzt. Ein zweiter API-Call aktiviert den selbst geschriebenen i-doit-Endpunkt, der das Monitoring startet. Der Event-Listener triggert dabei SNAG-View und legt so einen neuen Host zur Überwachung an. Die Hostgruppe, die dem Server zugewiesen werden soll, ist über die i-doit Gruppe definiert, die direkt im OTRS-Prozess ausgewählt wurde.

Wie alles zusammenkommt

Über die Schnittstellen verknüpfen sich nun i-doit CI und SNAG-View-Host, was in i-doit die Anzeige des Livestatus und der Serviceliste ermöglicht. In SNAG-View wird der Host als „aus dem i-doit synchronisiert“ markiert, was bestimmte Felder von der Bearbeitung sperrt, damit hier nicht jemand aus Versehen etwas kaputt macht.

Stellt das Monitoring eine Anomalie fest, legt es über seine OTRS-Schnittstelle automatisch ein Incident-Ticket an. Dieses ist sowohl mit dem Host in SNAG-View als auch mit dem CI in i-doit verknüpft und zieht alle Informationen zusammen. Das Ticket enthält den Livestatus, die Informationen aus i-doit, den Schnellzugriff auf hinterlegte Notfallpapiere und so weiter. Nimmt sich ein Mitarbeiter das Ticket zur Bearbeitung, benachrichtigt OTRS SNAG-View und setzt ein Acknowledgement. Damit weiß das Monitoring, dass das Problem erkannt wurde, sich jemand darum kümmert und es nicht weiter eskalieren muss. SNAG-View aktualisiert nun regelmäßig das Ticket mit den neuesten Check-Ergebnissen.

Geschlossen werden muss das Ticket auch nicht unbedingt von Hand. Stellt SNAG-View fest, dass das Problem behoben wurde, kann es das Ticket selbstständig mit einer abschließenden Recovery-Meldung schließen und damit dem Bearbeiter signalisieren, dass seine Bemühungen erfolgreich waren.

Fazit

Durch geschicktes Erweitern und Kombinieren entsteht aus mehreren unabhängig voneinander arbeitenden Programmen eine über bidirektionale Schnittstellen intelligent interagierende Umgebung. Die eigenen TransitionActions in den OTRS-Prozessabläufen automatisieren programmübergreifende Funktionen und verteilen Informationen, die sonst händisch gepflegt werden müssten.

Darüber hinaus sind alle Informationen zu einem Objekt in allen drei Hauptprogrammen (OTRS, i-doit, SNAG-View) konsolidiert verfügbar. Ticketlisten, CI-Informationen und Livestatus sind direkt einsehbar, weiterführende Informationen nur einen Link entfernt. Dieses Konstrukt lässt sich problemlos mit weiteren Open-Source-Tools wie GitLab oder dem alternativen Inventarisierungstool OCS Inventory NG erweitern.

Das alles setzt natürlich voraus, dass man sich mit den Tools, der Plug-in-Entwicklung und den Datenbanken beschäftigt und nicht davor zurückschreckt, selbst etwas Code zu schreiben. Wer den Aufwand scheut, findet spezialisierte Dienstleister, die einem beim Aufsetzen einer Open-Source-basierten IT-Service-Management-Lösung mit professionellem Support hilfreich zur Seite stehen können. (odi@ix.de)

Gabriel Jülke

arbeitet seit sechs Jahren bei der Sector Nord AG und pflegt und entwickelt dort unter anderem die Monitoringlösung SNAG-View.

Quellen

[1] Add-ons für i-doit entwickeln: ix.de/ix1903082