Gutachten

Systemarchitektur KONSEQUENZ

Autor: Dr. Martin Jung
develop group BASYS GmbH

im Auftrag des Projekts

KONSEQUENZ

Änderungsübersicht

Version	Ersteller	Datu	Geänderte Abschnitte	Änderungsgrund
V0.9	Dr. Jung	2014-06-02	Alle	Neuerstellung
V1.0	Dr. Jung	2014-06-11	Alle	Rückmeldung aus der Abschlusspräsentation

Dateiname: „Gutachten Systemarchitektur KONSEQUENZ.docx“

Kontaktinformation des Autors:

Dr. Martin Jung
develop group BASYS GmbH, Bereich SEC
Am Weichselgarten 4
91058 Erlangen

Telefon: 09131 777-40
Telefax: 09131 777-444
Mail: martin.jung@develop-group.de

Inhalt

1. Einführung

In der Arbeitsgemeinschaft **KONSEQUENZ** (**KON**sortiale **S**oftware**E**ntwicklung zur **QU**alitäts- und **E**ffizienzsteigerung im **N**et**Z**betrieb) haben mehrere Netzbetreiber untersucht, in wie weit ein Konsortium seine IT-Landschaft gemeinsam, auf der Basis von Open Source-Software, erneuern kann. Im dazu vorgelegten Abschlussbericht [1] sind die Ergebnisse festgehalten. Im Rahmen der Machbarkeitsstudie ist eine Systemarchitektur skizziert worden, die einem Gutachten unterzogen wird. Die Ergebnisse des Gutachtens sind in diesem Dokument zusammengefasst.

Die erneuerte IT Landschaft soll folgendes gewährleisten:

- Der in den aktuellen Systemen zu beobachtende Vendor-lock-in durch Hersteller externer Systeme wird durchbrochen.
- Die Integration externer und interner Systeme wird verbessert, die Kopplung dabei allerdings reduziert. Die so verbesserte Modularisierung erlaubt unter anderem einen leichten Einsatz von bestehenden Open Source-Lösungen.
- Die Wartbarkeit erhöht sich durch leichte Austauschbarkeit einzelner Subsysteme. Damit steigen auch die Lebensdauer und die Evolutionsfähigkeit.
- Durch die gute Austauschbarkeit einzelner Subsysteme und den Einsatz quelloffener Systeme werden Fehlersuche und Sicherheit erleichtert.

Die Systemarchitektur liegt in einer ersten Skizze vor, an Hand der die hier dokumentierte Untersuchung durchgeführt wurde. Konkrete Anforderungen über die Skizze hinaus liegen noch nicht vor. Allerdings sind einschlägige Normen und Standards benannt und bei der Bewertung berücksichtigt (siehe auch Anhang A).

In Abschnitt 2 ist der Stand der Architektur kurz zusammengefasst. Danach wird in Abschnitt 3 ein Szenario aufgespannt, an Hand dessen die Architektur bewertet wird.

Darauf folgt die Bewertung in den vier hervorgehobenen Qualitätsdimensionen Verfügbarkeit, Integrität, Vertraulichkeit und Echtzeitfähigkeit, jeweils in den Abschnitten 4, 5, 6 und 7.

Mögliche Produkte, die für die Umsetzung der Architektur im Gespräch sind werden in Abschnitt 8 untersucht. Ein Szenario für eine prototypische Umsetzung der Architektur wird in Abschnitt 9 aufgestellt und das Gutachten schließt mit einer Liste offener Punkte in Abschnitt 10 sowie einer Zusammenfassung der Empfehlungen in Abschnitt 11.

Im Anhang sind die bei der Erstellung verwendeten Quellen (Anhang A) und eine Liste der Abkürzungen und Fachbegriffe (Anhang B) zusammengefasst.

2. Kurzvorstellung der Architektur

Die bisher grob konzipierte Systemarchitektur, auf die sich dieses Gutachten bezieht, wird in diesem Abschnitt zunächst kurz vorgestellt. Dabei werden in erster Linie die für die weitere Bewertung relevanten Details beleuchtet.

Zielsetzung des Systems ist es:

- Verschiedenste externe Systeme einzubinden und der Anwendungswelt der Netzbetreiber zusammengefasst zur Verfügung zu stellen.
- Eine Laufzeitumgebung zu schaffen, in der Dienste unterschiedlicher Hersteller und verschiedener Zielsetzung nebeneinander ausgeführt werden können ohne sich zu beeinflussen.
- Eine einheitliche Schnittstelle anzubieten, über die sich Endanwender-Software Dienste suchen und diese nutzen kann.

Die Architektur wird in unterschiedlichen Sichten präsentiert. Zunächst die logische Aufteilung des Systems im Sinne einer Schichtenarchitektur. Dann werden zwei Diagramme der Verteilungssicht gezeigt, das eine zeigt die Netzwerktopologie, das andere den Einsatz der unterschiedlichen Komponenten auf Rechnerknoten.

2.1. Logische Sicht: 4-Schichten Architektur

Die logische Sicht dient dazu, das komplexe Gesamtsystem in logisch zusammenhängende Schichten zu zerlegen. Die gewählte Schichtenarchitektur ist typisch für Client/Server Systeme mit getrennten Backend-Systemen und ist daher eine gut geeignete Wahl. Abbildung 1 zeigt diese Aufteilung des Systems in technische Schichten.

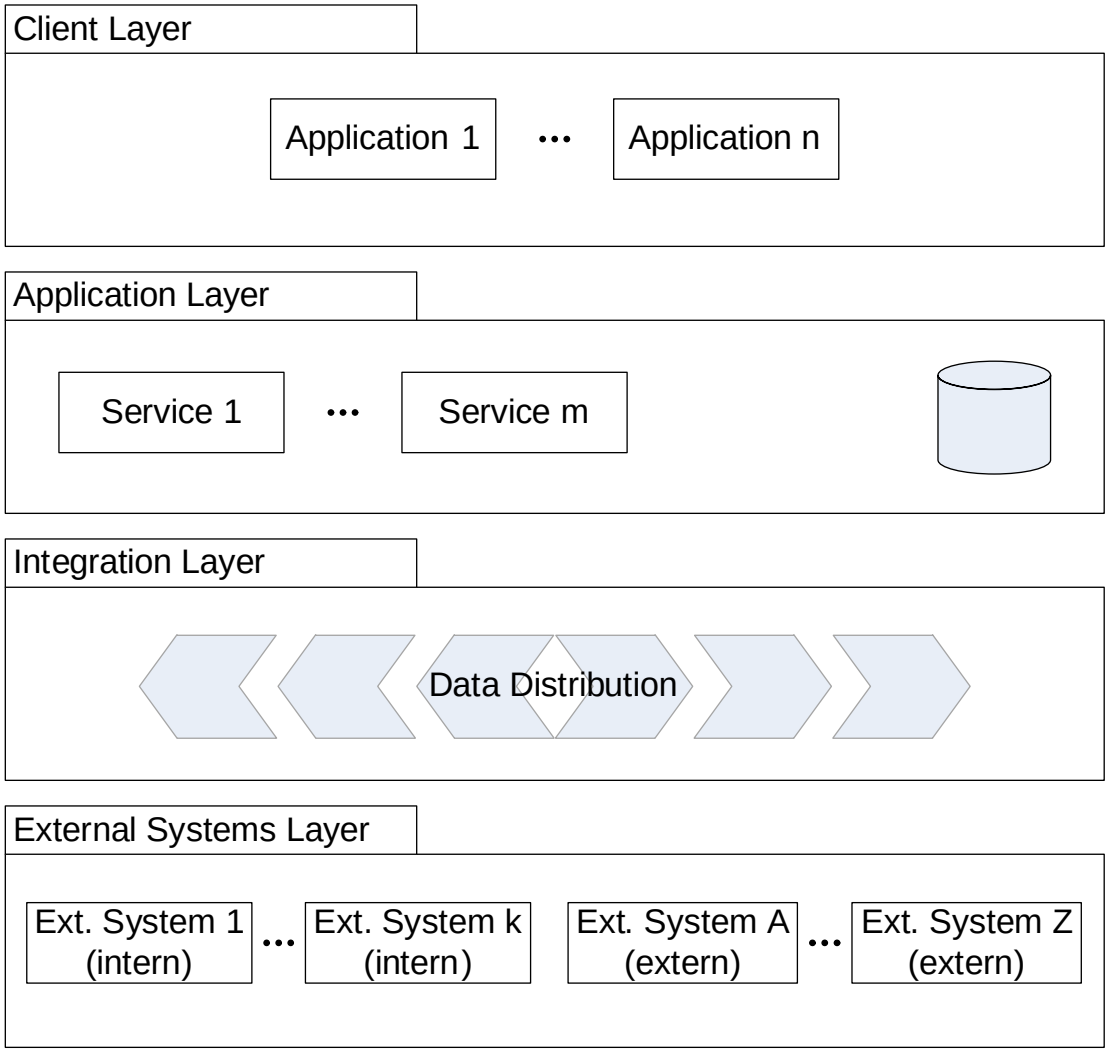

Abbildung 1: Schematische Übersicht der Systemarchitektur

Die **Externschicht** (External Systems Layer) fasst alle angebundenen externen Systeme zusammen. Externe System umfassen Datenquellen und –senken, die wesentlich für das Gesamtsystem „Netzleitung“ sind. Das sind beispielsweise intern zur Verfügung stehende System wie das Geoinformationssystem GIS oder auch externe Systeme wie z.B. eine Wetterprognose. Diese Schicht wird mit der Integrationsschicht verbunden.

In der **Integrationsschicht** (Integration Layer) werden Daten der externen Systeme empfangen, konsolidiert und zur weiteren Verarbeitung vorgehalten. Hierfür sind Datenkonversionen, Konsistenzprüfungen, Abfrage- und Benachrichtigungsmechanismen erforderlich. Die Integrationsschicht kapselt die externen Systeme vollständig vom restlichen System ab.

Die **Anwendungsschicht** (Application Layer) fasst im Wesentlichen zwei Dinge zusammen. Zum einen eine Datenbank, in der die integrierten und konsolidierten Daten aus der Integrationsschicht vorgehalten werden. Zum anderen eine Ausführungsumgebung für Funktionen aus Anwendungs- und Fachdomäne auf unterschiedlichen Abstraktionsebenen. Beispielsweise laufen hier Dienste für den Trans-

fer von Daten aus der Integrationsschicht in die Datenbank, es laufen fachliche Funktionen wie der Datendienst für den Netzzustand, aber auch komplexe Domänendienste wie eine Auswirkungsanalyse eines virtuellen Schaltauftrags. Die Anwendungsschicht bündelt somit lose gekoppelte Dienste. Sie setzt auf der Integrationsschicht auf und unterhält keine direkten Verbindungen zu den externen Systemen.

Die **Bedienschicht** (Client Layer) fasst alle Endanwendungen zusammen. Beispielsweise komplexe Anwendungsprogramme, die dem heutigen System stark ähneln. Aber auch leichtgewichtige Anwendungen, die über wenig eigene Logik verfügen und lediglich eine Ein- und Ausgabeschnittstelle für die Dienste in der Anwendungsschicht darstellen sind darin enthalten. Schließlich werden in dieser Schicht auch mobile Endanwendungen einsortiert, die es aktuell noch nicht gibt, die aber sicherlich in zukünftigen Systemen eine tragende Rolle spielen.

2.2. Netzwerktopologie: Die unterschiedlichen Segmente

Abbildung 2 zeigt die vorgeschlagene Netzwerktopologie. Das Gesamtsystem wird hier in fünf Zonen geteilt, die jeweils durch Elemente der Infrastruktur voneinander klar abgetrennt sind („Firewalls“). Farblich hervorgehoben sind die Knoten, die neu hinzukommen und durch die Systemarchitektur definiert, sowie mit den bestehenden Systemen gekoppelt werden. Die erforderlichen Informationsflüsse zwischen den Netzen sind mit Pfeilen dargestellt. Der gestrichelte Pfeil steht für eine Datenverbindung die heute existiert, weil sie von den Anwendungen benötigt wird. Durch das Einführen der neuen Architektur soll dieser Informationsfluss obsolet werden.

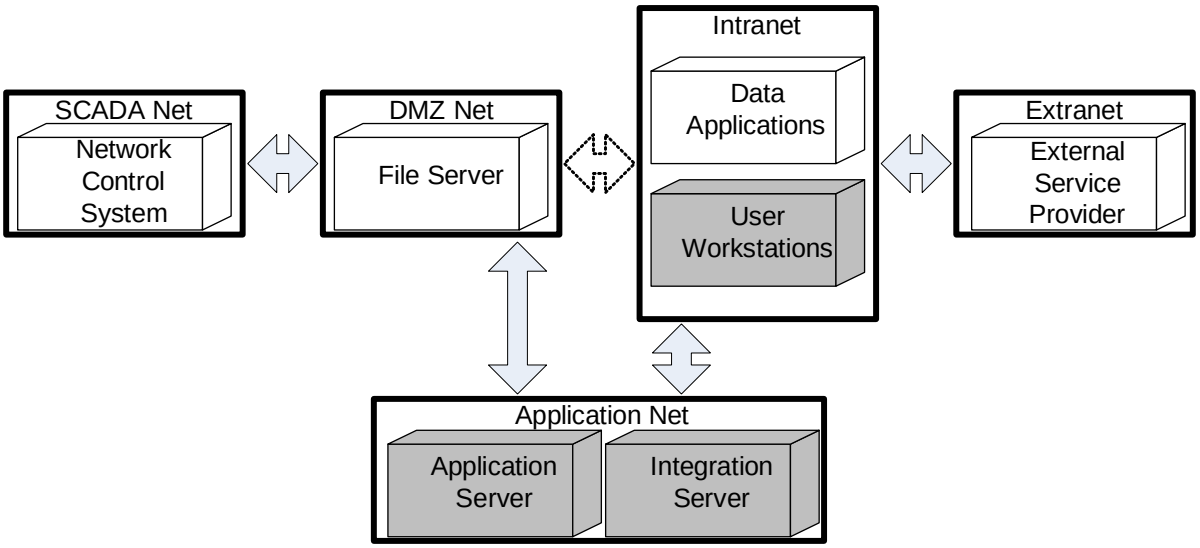

Abbildung 2: Netzwerktopologie

Im **externen Netz** (Extranet) befinden sich Datenquellen und -senken die außerhalb der Kontrolle des Systems liegen. Ein typisches Beispiel sind Prognosesysteme die Anhaltspunkte für das Lastverhalten auf dem Netz geben (z.B. Wetterdienst). Diese Systeme sind als unzuverlässig einzustufen und ankommende Daten müssen ge-

prüft werden bevor das System sie akzeptieren und berücksichtigen kann. Ausgehende Daten müssen streng überwacht werden, um die unsachgemäße Verbreitung von Information zu unterbinden. Zukünftig müssen auch Endanwendungen berücksichtigt werden, die im Extranet angebunden sind und trotzdem steuernd auf die Infrastruktur eingreifen.

Im **internen Netz der Organisation** (Intranet) befinden sich Datenquellen- und Senken die vollständig unter Kontrolle des Systems liegen („Data Applications“). Diese Anwendungen bestehen bereits jetzt. Ein typisches Beispiel ist das Geoinformationssystem GIS. Die Systeme sind bereits im Einsatz und ihre Funktion ist bekannt, die gelieferten Daten sind nicht als Gefährdung zu sehen. Wohl aber muss mit Ausfällen der Systeme und mit unvollständigen/fehlerhaften Daten gerechnet werden. Typischerweise befinden sich die Endanwendungen ebenfalls im Intranet („User Workstations“). Hierbei ist zu beachten, dass die Arbeitsplätze als frei zugänglich einzustufen sind. Somit muss dafür Sorge getragen werden, dass die Endanwendungen den Betrieb des Systems auch nicht schädigen können, weder durch unabsichtliche Fehlfunktion noch durch absichtliche Angriffe.

Im **Anwendungsnetz der Netzbetreiber** (Application Net) befinden sich die Systeme der Integrations- und Anwendungsschicht. Insbesondere muss hier für die Zukunft berücksichtigt werden, dass die Anwendungsinfrastruktur eventuell von mehreren Organisationen geteilt verwendet wird. In diesem Netz laufen die zentralen Kernkomponenten. Störungen dieses Netzes, wie beispielsweise hohe Netzwerklast oder der Ausfall einzelner Rechnerknoten, haben erheblichen Einfluss auf das Gesamtverhalten des Systems (siehe auch Abschnitt 4).

Im **Austauschnetz** (DMZ Net) findet der Datenaustausch des mit dem tatsächlich aktiv steuernden Netzleitsystem (NLS) statt. Das NLS liest Daten aus dem Austauschnetz. Die wesentliche Funktion dieses Netzsegments ist die Absicherung von Teilsystemen mit jeweils unterschiedlichen Qualitätseigenschaften. Langfristig sind Szenarien gewünscht, in denen die Dienste im Anwendungsnetz automatisch Daten an das Kontrollnetz weitergeben können.

Im **Kontrollnetz** (SCADA Net) laufen die Systeme, die tatsächliche Steuerungsfunktionen ausführen und damit das physische Versorgungsnetz beeinflussen. Diese Systeme sind als kritisch einzustufen und dürfen weder in ihrer Funktion gestört, noch mit falschen oder unvollständigen Daten versorgt werden. Die Elemente dieses Netzes sind unter allen Umständen vor Fehlfunktion zu bewahren, ihre Ausführung muss gewährleistet bleiben. Aus dem Kontrollnetz werden aktuell Daten in das Austauschnetz zurückgespielt. Von dort aus werden Sie zukünftig der Integrationsschicht zur Verfügung gestellt.

2.3. Einsatz: Komponente zu den Rechnern im Netz zuordnen

Zusätzlich zur logischen Strukturierung und zur geplanten Netzwerktopologie muss noch festgelegt werden, wie die Komponenten der logischen Struktur auf die Knoten

der Topologie verteilt und zum Einsatz gebracht werden. Von diesem Einsatz hängen die Qualitätseigenschaften des Systems maßgeblich ab. Ein *möglicher* Einsatz ist in Abbildung 3 gezeigt. Die neu zu entwickelnden Systemkomponenten sind farblich hervorgehoben.

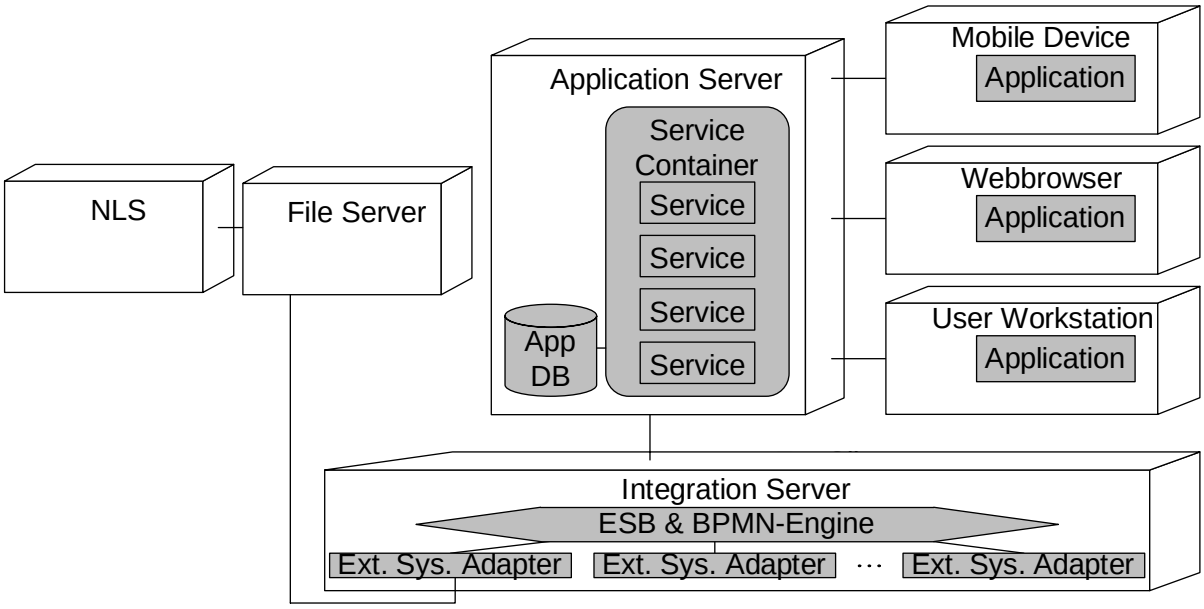

Abbildung 3: Einsatz der Komponenten

Auf dem **Fileserver** läuft keine der Komponenten des Systems, er ist lediglich eine passive Instanz und dient zum Datenaustausch mit dem NLS.

Auf dem **Application Server** ist zum einen die Anwendungsdatenbank untergebracht, zum anderen auch ein Service-Container, der den einzelnen Diensten als Ausführungsumgebung und Infrastruktur zur Verfügung steht.

Auf dem **Integration Server** läuft der Kommunikationsbus, über den die externen Systeme integriert werden. Die gesammelten Daten werden über einen Dienst im Application Server an die Anwendungsdatenbank weitergegeben. Im Kontext des ESB (Enterprise Service Bus) laufen auch die jeweiligen Adapter, die die externen Systeme mit dem Kommunikationsbus verbinden. Die Integration dieser Systeme wird per „Business Process Model and Notation“ (BPMN) gesteuert. Auf dem Integration Server läuft also auch eine Ausführungsmaschine für BPMN.

Für die weitere Abhandlung sind folgende qualitative Anforderungen zu betonen:

- Das zu entwickelnde System muss eine hohe **Verfügbarkeit** garantieren. Gegenüber dem aktuellen System, das lediglich gegen Hardwareausfall abgesichert ist, muss auch Redundanz der Softwarekomponenten berücksichtigt werden. Da für das neue System tendenziell mit vergleichbaren Anforderungen an die Verfügbarkeit gerechnet wird wie sie an das Leitsystem gestellt werden, sind die Anforderungen an die Verfügbarkeit gemäß der Einstufung aus [8] mit „hoch“ bewertet.

- Das zu entwickelnde System muss eine hohe **Integrität** garantieren. Sowohl die konsolidierten Daten aus den externen Systemen, wie auch die generierten Daten für die tatsächlichen Schaltaktionen müssen in sich konsistent und valide sein. Das aktuelle System ist gegen Fremdeinwirkung geschützt, bietet aber keine logische Validierung und Konsistenzprüfung der jeweiligen Daten. Die Anforderungen an die Integrität sind gemäß der Einstufung aus [8] mit „hoch“ bewertet.
- Das zu entwickelnde System muss die **Vertraulichkeit** der Daten garantieren. Die aktuellen Systeme sind jeweils hauseigen bei einer Organisation eingesetzt. Zukünftige Szenarien beachten jedoch auch gemeinschaftlich genutzte Teilsysteme. In diesen Fällen muss dass System ungewollten Informationsaustausch über die organisatorischen Grenzen hinweg unterbinden. Die Anforderungen an die Vertraulichkeit sind gemäß der Einstufung aus [8] mit „normal“ bewertet.
- Das zu entwickelnde System muss **Zeitschranken** einhalten. Schaltempfehlungen müssen in einem bestimmten Zeitraster erstellt werden. Das aktuelle System garantiert eine Antwortzeit bei der Datenanzeige von < 2 Sekunden. Schaltanträge werden ebenfalls mit dieser maximalen Latenz weitergegeben. Insgesamt gibt es ein 15 minütiges Zeitraster, das allen Systemaktionen zu Grunde liegt, und dessen Takt einzuhalten ist.

3. Betrachtete Szenarien

Für die Untersuchung und Bewertung der Architektur wird ein typisches Nutzungsszenario betrachtet:

I. Der Benutzer sieht das gerade aktualisierte Netzzustandsbild.
II. Auf Grund einer Situation im Netz schlägt das System einen Schaltantrag vor.
III. Der Benutzer bearbeitet den Schaltantrag und markiert ihn zur Übertragung.
IV. Das System stellt den Schaltantrag auf der Leitwarte zur Verfügung.
V. Der Schaltantrag wird in der Leitstelle ausgeführt.
VI. Über die zyklische Aktualisierung des Netzzustandsbilds ist die Veränderung beim Benutzer sichtbar.

Grundsätzlich ist dies eine „Analyse-Aktion-Kontrolle“ Funktion des Systems. Ausgehend von einer analytischen Sicht auf den Zustand der tatsächlichen Welt, wird eine Aktion ausgelöst und deren Wirkung kontrolliert. Kritisch ist in solchen Interaktionen im Allgemeinen das Duo „Aktion-Kontrolle“, so auch hier. Abbildung 4 veranschaulicht an Hand des Szenarios exemplarisch, welche Datenflüsse zwischen einzelnen Elementen des Systems erforderlich sind.

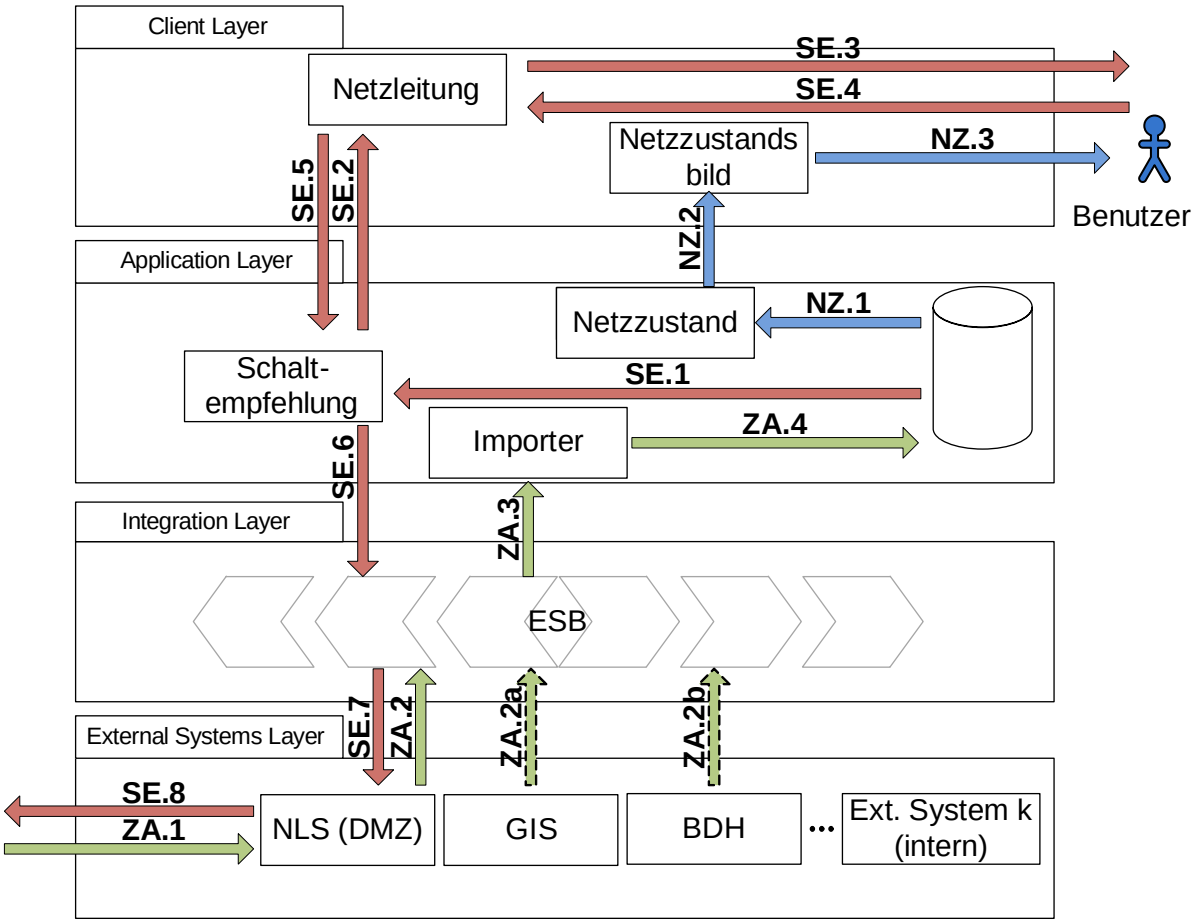

Abbildung 4: Datenfluss für das betrachtete Szenario

Bei *Schritt I* des betrachteten Szenarios handelt es sich um den „Analyse“ Komplex der Interaktion, er ist durch blaue Pfeile mit der Abkürzung „NZ für Netzzustand gekennzeichnet. Berücksichtigt wird hier nur der Pfad vom lokalen Speicher der Applikationsdatenbank bis zum Benutzer. Bei den *Schritten II bis IV* des Szenarios handelt es sich um den „Aktion“ Komplex der Interaktion, er ist durch rote Pfeile mit der Abkürzung „SE“ für Schaltempfehlung gekennzeichnet. *Schritt V* des Szenarios gehört ebenfalls zum „Aktion“ Konzept, liegt aber außerhalb des Systems und ist daher nicht dargestellt. Der *Schritt VI* des Szenarios stellt den „Kontrolle“ Komplex dar. Er ist durch grüne Pfeile mit der Abkürzung „ZA“ für Zustandsaktualisierung gekennzeichnet.

Die einzelnen Komplexe laufen in der Realität von einander unabhängig ab. In dieser Betrachtung sind sie jedoch gemeinsam dargestellt, um als Grundlage unterschiedlicher Bewertungsszenarien zu dienen. Beispielsweise kann die flüssige Aktualisierung der Netzzustandsanzeige in der Endanwendung bestimmt werden, indem lediglich NZ.1 bis NZ.3 betrachtet werden. Für die Verfügbarkeit des tatsächlich aktuellen Netzzustandsbilds müssen die Schritte ZA.1 bis ZA.4 sowie die Schritte NZ.1 bis NZ.3 betrachtet werden.

Im Einzelnen bedeuten die Schritte des Datenflusses:

NZ.1. Der Komplex wird zyklisch aktiviert, die Aktivierung ist nicht dargestellt. Als erster Schritt werden dem Dienst „Netzzustand“ auf Anfrage Daten aus der Datenbank übertragen.

NZ.2. Die Daten werden von dem Dienst „Netzzustand“ verarbeitet und an das Endanwenderprogramm „Netzzustandsbild“ übertragen.

NZ.3. Das Endanwenderprogramm bearbeitet die Daten und bringt sie dem Benutzer zur Anzeige.

SE.1. Der Komplex wird zum einen zyklisch aktiviert, kann zum anderen aber auch vom Benutzer angestoßen werden. Beide Formen der Aktivierung sind hier nicht dargestellt. Als erster Schritt werden dem Dienst „Schaltempfehlung“ auf Anfrage Daten aus der Datenbank übertragen.

SE.2. Die Daten werden von dem Dienst verarbeitet und der Dienst stellt die Notwendigkeit einer Schaltempfehlung fest. Die Schaltempfehlung wird vom Dienst berechnet und an das Endanwenderprogramm „Netzleitung“ übertragen.

SE.3. Das Endanwenderprogramm gibt die Daten an den Benutzer aus.

SE.4. Der Benutzer bearbeitet die Schaltempfehlung und signalisiert dem Endanwenderprogramm, dass die Empfehlung fertig ist und übertragen werden soll.

SE.5. Das Endanwenderprogramm prüft die vom Benutzer eingegebenen Daten und überträgt sie an den Dienst „Schaltempfehlung“.

SE.6. Der Dienst prüft die Daten erneut, auch auf Konsistenz mit dem aktuellen Netzzustand und gibt bei erfolgreicher Verifikation die Daten an die Integrationsschicht weiter.

SE.7. Die Integrationsschicht gibt die Schaltempfehlung an den Adapter für das externe System „NLS“ weiter.

SE.8. Der Adapter reicht die Schaltempfehlung an das tatsächliche SCADA System weiter. Damit ist der Komplex „Aktion“ aus Sicht des hier betrachteten Systems zu Ende. In zukünftigen Szenarien ist hier der nächste Schritt, das tatsächliche Durchführen der Schaltempfehlung, zu beachten.

ZA.1. Der Komplex wird vom tatsächlichen Netzleitsystem ausgelöst. Dieses stellt dem Adapter für externe Systeme Daten in der Integrationsschicht zur Verfügung.

ZA.2. Der externe Adapter meldet dem Integrationsbus neue Daten des NLS. Je nach den hinterlegten BPMN-Regeln werden gegebenenfalls weitere externe Quellen abgefragt (z.B. das GIS ZA.2a oder das BDH-System ZA.2b).

ZA.3. Die von externen Systemen geschickten Daten werden zu einem konsistenten Datensatz zusammengeführt. Dieser wird anschließend an den Dienst „Importer“ übertragen.

ZA.4. Der Dienst prüft die Daten und trägt sie nach erfolgreicher Integration in die Anwendungsdatenbank ein.

Bei der Betrachtung des Szenarios spielt die Art und Weise des Zugriffs durch den Benutzer noch keine Rolle. Dieser Zugriff kann entweder durch Anwenderprogramme aus dem Netzwerksegment „Intranet“ erfolgen oder durch mobile Bediensysteme aus dem Netzwerksegment „Extranet“. Die Unterschiede zwischen den beiden Zugriffsarten werden in der folgenden Bewertung ggf. unterschieden.

4. Bewertung der Verfügbarkeit

Die Verfügbarkeit eines Systems gibt an, mit welcher Wahrscheinlichkeit eine Dienstleistung, die bei dem System angefragt wird, auch erbracht wird. Verfügbarkeit ist das Verhältnis der Zeit, die das System Dienstleitungen erbringen kann, zur Gesamtlaufzeit. Verfügbarkeit = (Gesamtlaufzeit - Ausfallszeit) / Gesamtlaufzeit. Üblicherweise wird die Verfügbarkeit in Prozent angegeben.

An Hand des in Abbildung 4 abgebildeten Szenarios sind die einzelnen an der Interaktion beteiligten Komponenten zu erkennen. Grundsätzlich sind zunächst alle Kommunikationskanten von Ausfällen bedroht und müssen in die Bewertung der Verfügbarkeit eingehen. Hier lassen sich folgende Klassen an Kommunikationsverbindungen finden:

- **Kommunikation über das Internet**: Für mobile Endanwendungen ist diese Kommunikationsart erforderlich um überhaupt mit dem System in Kontakt zu treten. Sind mobile Endanwendungen Teil des Szenarios, so ist davon auszugehen, dass die Verfügbarkeit des Gesamtsystems ausschließlich von dieser Kommunikation bestimmt wird. Alle anderen Teile sind kontrollierbar und können mit höherer Verfügbarkeit ausgestattet werden.
- **Kommunikation im Intranet über mehrere Netzwerksegmente**: Wie in Abbildung 2 zu sehen ist, wird das gesamte vom System genutzte Netzwerk segmentiert. Bei Kommunikationskanten zwischen diesen Segmenten spielen die Koppelungspunkte („Firewalls“) zwischen den Segmenten eine Rolle. Diese Bausteine (in der Regel spezialisierte Hardware) sind zumeist auf hohe Verfügbarkeit ausgelegt.
- **Kommunikation innerhalb eines Netzwerksegments**: Hier ergibt sich die Verfügbarkeit aus den Verfügbarkeiten der Endpunkte der jeweiligen Kommunikationskanten. Ein Ausfall der Netzinfrastruktur kann nicht vom System abgefangen und behandelt werden, hier müssen externe Mechanismen (Zugangskontrolle, IT-Bereitschaft) definiert werden.
- **Lose gekoppelte Kommunikation**: Die Kommunikation zum SCADA Netz erfolgt asynchron losgekoppelt über einen Dateiaustausch. Die Verfügbarkeit entspricht der Verfügbarkeit des für den Austausch verwendeten Speichers. Die Kommunikation aus dem physischen Netz zurück zum NLS Subsystem ist für das restliche System intransparent, die Verfügbarkeit entspricht der des NLS Subsystems selbst.

Neben der Kommunikation wird die Verfügbarkeit maßgeblich vom Einsatz (siehe Abbildung 3) bestimmt. Demzufolge muss berücksichtigt werden:

- **Hardware**: Die Verfügbarkeit des Systems darf nicht durch die Verfügbarkeit der einzelnen Hardwarekomponenten limitiert werden. Die Hardware muss dementsprechend redundant ausgelegt werden. Zusätzlich muss die Systemarchitektur den Softwarekomponenten vorgeben, einen Wechsel von Hard-

wareknoten zur Laufzeit transparent verarbeiten zu können. Somit ist die Verfügbarkeit des Systems abhängig von drei Dingen: Der Menge an redundant vorgehaltener Hardware, der Latenz der Software beim Umschalten auf eine andere Hardwareinstanz, und der Verfügbarkeit eines Verteilers, der ankommende Anfragen an den jeweils aktiven Hardwareknoten weitergibt.

- **Basissoftware**: In Abbildung 3 ist die Software-Infrastruktur der einzelnen Knoten nicht dargestellt. Hierunter fallen Komponenten wie die einzelnen Betriebssysteme, Treiber und ähnliche Basissoftware. Die Verfügbarkeit dieser Basis fällt in die gleiche Betrachtungsweise wie die Hardware. Ein Knoten muss als ausgefallen angesehen werden, auch wenn „nur“ die Basissoftware ausfällt (z.B. Absturz des Betriebssystems).
- **Laufzeitumgebungen**: Der Service-Container und der ESB sind Softwarekomponenten, die auf der Basissoftware aufsetzen und den Anwendungen (Services oder Protokolladapter bzw. Datenkonverter) einen Kontext aus leistungsfähigen Funktionen bieten. Laufzeitumgebungen bilden häufig auch Mechanismen wie Lastausgleich und transparente Wechsel im Fehlerfall ab. Die Verfügbarkeit der Laufzeitumgebung limitiert die Verfügbarkeit des gesamten Systems.
- **Datenbank**: Ein Ausfall der Anwendungsdatenbank führt nicht zu einem Datenausfall, da alle Daten über die externen Systeme wiederbeschafft werden können. Allerdings ist das Datenvolumen, was aus den externen Systemen in die Datenbank zurückgeholt werden muss erheblich. Das System steht während des Wiederherstellens der Datenbank nicht zur Verfügung. Daher sollte die Datenbank redundant ausgelegt werden, da ein Ausfall nicht nur massive Einbrüche in der Systemleistung verursacht, sondern auch Ausfallszeiten erzeugt.
- **Anwendungsdienste**: Die einzelnen Dienste der Anwendungsdomäne arbeiten in einem Kontext, der den Ausfall von Hardwarebausteinen verbirgt. Ist ein einzelner Dienst nicht verfügbar, so sind nur die Teile des Systems davon betroffen, die diesen Dienst direkt nutzen. Die Verfügbarkeit des Systems wird also durch die Abhängigkeitsstrukturen zwischen den Diensten beeinflusst.
- **Endanwendungen**: Die Endanwendungen stellen für alle Szenarien mit direktem Einbezug eines Endnutzers die ausschlaggebende Komponente dar. Lediglich autonom im System ablaufende Dienste (Datenprüfung, Backup, etc.) sind von dieser Komponente nicht beeinflusst. Für alle extern sichtbaren Funktionen ist die Verfügbarkeit der Endanwendung ebenfalls ein oberes Limit für die Verfügbarkeit des gesamten Systems.

Die Informationen zu dem geplanten System sind aktuell noch nicht ausreichend, um quantitative Aussagen zu treffen. Zur Übersicht hinsichtlich einer quantitativen Betrachtung sind in Tabelle 1 die Ausfallszeiten für eine Endanwendungen (8 Stunden am Tag, 5 Tage die Woche, 52 Wochen im Jahr; 2080 Stunden im Jahr) dargestellt. Für eine Serveranwendung, die rund um die Uhr in Betrieb ist, müssen die Zahlen linear hochgerechnet werden.

Verfügbarkeit in Prozent	Ausfallszeit in Stunden	Ausfallszeit in Minuten	Ausfallszeit in Sekunden
99	20,8	1248	74880
99,9	2,08	124,8	7488
99,99	0,208	12,48	748,8
99,999	0,0208	1,248	74,88
99,9999	0,00208	0,1248	7,488
99,99999	0,000208	0,01248	0,7488

Tabelle 1: Verfügbarkeit und Ausfallszeiten für eine Büroanwendung

Die Systemarchitektur in ihrem hier zu Grunde liegenden Detaillierungsgrad ist für die Erfüllung hoher Verfügbarkeitsanforderungen geeignet. Folgende Punkte geben hier den Ausschlag:

- Unterschiedliche Arten von Endanwendungen sind vorgesehen. Falls eine solche Anwendung versagt, besteht noch die Möglichkeit auf eine andere zu wechseln. Beispielsweise kann bei Ausfall eines Fat Clients auf einen Thin Client ausgewichen werden. Allerdings reduziert sich dabei eventuell der Funktionsumfang.
- Endanwendungen und Anwendungsdienste sind entkoppelt. Wenn ein bestimmter Anwendungsdienst nicht zur Verfügung steht, kann die Endanwendung auf andere Dienste zurückgreifen und so einen verringerten Funktionsumfang aufrechterhalten. Ein derartiges Verhalten muss aber bei der Implementierung der Endanwendungen berücksichtigt werden.
- Die Anwendungsdienste sind über ausgereifte Laufzeitumgebungen mit Strategien ausgestattet, um im Fehlerfall transparent auf andere Serverinstanzen bzw. andere Hardware ausweichen können.
- Die Dienste können durch ihre Laufzeitumgebung überwacht und diagnostiziert werden.
- Durch die Systemarchitektur wird insbesondere das Redundanzprinzip umgesetzt, dass unter anderem von [3] und [8] gefordert wird.
- Auch das dort genannte Prinzip der mehrstufigen Absicherung („Defence in Depth“) ist hier umgesetzt, da Ausfälle nicht nur an einzelnen Stellen identifiziert und – so weit möglich – toleriert werden.

Zu beachten ist:

- Typische querschnittliche Infrastrukturdienste, wie die Benutzerauthentifizierung, müssen wie die Dienste in den Laufzeitumgebungen zur Verfügung gestellt werden, so dass auch deren Ausfall überbrückt werden kann.
- Die extern angebundenen Systeme sind außerhalb der Kontrolle des Systems. Sollten diese Systeme ausfallen kann das System unter Zugriff auf die Anwendungsdatenbank einen (geringeren) Funktionsumfang aufrechterhalten.
- Wenn möglich sollten für die extern angebundenen Systeme QoS Vereinbarungen getroffen werden.

- Für alle Systemkomponenten müssen Rückfallebenen vorgesehen werden, die bei Ausfall eine Grundfunktionalität absichern.
- Die Kommunikation sollte weitestgehend über dedizierte Netze erfolgen, so dass sich hohe Netzlast nicht auf die Verfügbarkeit der Netzkommunikation auswirken kann.
- Aus Aufwandsgründen ist es empfehlenswert, in der endgültigen Architektur unterschiedliche Klassen der Verfügbarkeit vorzusehen. Im Zukunftsszenario, in dem auch die SCADA-Funktionen von diesem System getragen werden, liegt es auf der Hand, dass die tatsächlichen Steuerfunktionen hochverfügbar ausgelegt werden müssen. Im Gegensatz dazu können Simulationsrechnungen und Prognosen mit einer geringeren Verfügbarkeit umgesetzt werden.

5. Bewertung der Integrität

Integrität bezeichnet im Kontext von informationsverarbeitenden Systemen die Korrektheit und Unversehrtheit von Informationen. Dabei ist insbesondere darauf zu achten, dass dies sowohl für die verarbeiteten Nutzdaten gilt, als auch für die ausführbaren Daten, die Programme.

Vom Blickwinkel der Systemarchitektur sind hinsichtlich der ausführbaren Programme folgende Aspekte zu beachten:

- **Unversehrtheit und Korrektheit von Basissoftware und Laufzeitumgebungen**: Durch die Wahl von Open Source-Produkten als Basis der Systemarchitektur können diese inspiziert werden, bevor sie in die jeweiligen Binärformate übersetzt werden. Wichtiger ist aber die Kontrollfunktion der Open Source Communities hinsichtlich der Korrektheit. Zusätzlich werden häufig Prüfsummen von den Herstellern angegeben, um die Unversehrtheit festzustellen.
- **Unversehrtheit und Korrektheit der Dienste**: Durch die Systemarchitektur muss festgelegt werden, dass Dienste nur nach eingehender Prüfung in die Laufzeitumgebung aufgenommen werden dürfen. Zur Prüfung der Korrektheit bieten sich Reviews und/oder Abnahmetests an. Dabei ist allerdings nur eine Prüfung der Dienste gegen die Spezifikation möglich, diese muss also gesondert geprüft werden. Ein so überprüfter Dienst kann digital signiert werden und somit seine Unverfälschtheit sichergestellt werden.
- **Unversehrtheit und Korrektheit der Endanwendungen**: Die Integrität von Endanwendungen kann über ähnliche Mechanismen gewährleistet werden wie bei den Diensten, durch eine Korrektheitsprüfung, gefolgt von der digitalen Signierung der Anwendung.

Grundsätzlich ist für Programme *vor deren Ausführung* sicherzustellen, dass sie korrekt sind. Über eine geeignete Testumgebung kann die Korrektheit auch untermauert, aber nicht abschließend bewiesen werden. Zur Laufzeit muss lediglich deren Unverfälschtheit geprüft werden.

Hinsichtlich der Daten sind folgende Aspekte zu beachten:

- **Vollständigkeit von Datensätzen**: Ein Aspekt der Korrektheit von Daten ist ihre Vollständigkeit. Um diese beurteilen zu können, wird ein Datenschema benötigt, das vorgibt, welche Informationen in einem Datensatz vorliegen müssen.
- **Konsistenz von Datensätzen**: Ein weiterer Aspekt der Korrektheit von Daten ist die Stimmigkeit eines Datensatzes. Um diese prüfen zu können, wird zusätzlich zu dem Datenschema noch eine Menge von Konsistenzregeln benötigt. Diese Regeln beziehen sich auf die Abhängigkeiten von Informationen in

einem Datensatz oder auch auf die Beziehung zwischen mehreren Datensätzen, beispielsweise auf die unverfälschte Reihenfolge.

- **Unversehrtheit von Daten**: Sind Vollständigkeit und Konsistenz eines Datensatzes geprüft, so muss deren Unversehrtheit auf dem Transportweg gesichert werden. Hier kommen, wie bei den ausführbaren Programmen auch, digitale Signaturen zum Einsatz.

Für Daten ist also während der Laufzeit an allen Punkten, an denen Daten verarbeitet werden*, vor der jeweiligen Berechnung* die Korrektheit zu prüfen, wohingegen die Unverfälschtheit *nur an Übergabepunkten* kontrolliert werden muss.

Im Kontext der Integrität müssen zwei Aspekte gesondert betrachtet werden: Die Integrität wird durch fehlerhafte Verarbeitungsschritte verletzt, beispielsweise geht die Integrität eines Datensatzes durch fehlerhafte Berechnungen eines Programms verloren. Derartige Integritätsverletzungen können durch Prüfung auf Basis eines Datenschemas und durch Validierungsregeln aufgedeckt, aber nicht vollständig ausgeschlossen werden. Zudem können in einer Laufzeitumgebung auch unterschiedliche Varianten eines Dienstes parallel ausgeführt werden, um Ergebnisse vergleichen und damit verifizieren zu können (sofern man unabhängige Implementierungen der Dienste zur Verfügung hat).

Davon getrennt zu betrachten ist die absichtliche Verletzung der Datenintegrität durch Angreifer. Hier werden entweder Programme oder Daten verfälscht, um eine fehlerhafte Funktion hervorzurufen. Die Erkennung solcher Verletzungen lässt sich durch den Einsatz geeigneter Prüfsummenverfahren verbessern, siehe dazu auch [5].

Grundsätzlich sollte bei hohen Integritätsanforderungen jede Systemkomponente eigene Prüfungen vornehmen. Zusätzlich müssen die Daten an allen Übergabepunkten digital signiert werden. Bei Betrachtung des Anwendungsszenarios aus Abschnitt 3 zeigt sich, dass dementsprechend viele Prüf- und Signaturschritte notwendig sind. Diesem Aufwand kann man durch zwei Mechanismen begegnen. Zum einen bildet man mehrere Integritätsklassen unter den Systemfunktionen und verhindert so, dass tatsächlich für alle Systemfunktionen der größtmögliche Aufwand betrieben wird. Zum anderen kann man die Dienstspezifikationen entsprechend dem „Design by Contract“ Verfahren erweitern und damit die aufrufenden Komponenten dazu zwingen, bestimmte Integritätsbedingungen selbst zu erfüllen. Beide Mechanismen können in der Systemarchitektur festgelegt werden. Dabei stellen die einzelnen Systemkomponenten die Stellen dar, an denen Konsistenzprüfungen durchgeführt werden, die Netzwerktopologie bzw. die Interaktionen stellen die Übergabepunkte und somit die erforderlichen Signaturschritte dar.

Die vorgeschlagene Systemarchitektur ist in der Lage, Integrität von Daten und Programmen zu gewährleisten. Folgende Punkte geben den Ausschlag:

- Die Endanwendungen sind von den Serverdiensten entkoppelt und ermöglichen so definierte Prüf- und Übergabepunkte.
- Das NLS als kritisches externes System ist durch Netzwerktopologie und logische Architektur vor Zugriffen geschützt, die nicht durch Integritätsmechanismen abgesichert sind.
- Eine eigene Integrationsschicht kann die Datenintegrität prüfen.
- Die Systemfunktionen sind in einzelne Dienste zerlegt, die jeweils für sich geprüft und signiert werden können.
- Die Laufzeitumgebung kann Endanwendungen, ankommende Daten und auch Dienste gegen Signaturen prüfen.
- Durch die Systemarchitektur wird das in [3] genannte Prinzip der mehrstufigen Absicherung („Defence in Depth“) umgesetzt, die Integrität von Daten wird auf jeder Hierarchieebene des Systems unabhängig von den anderen Schichten geprüft.
- Durchgängiger Einsatz von digitalen Signaturen setzt die von den Normen geforderte Absicherung gegen Verfälschung von Daten durch.
- Das ebenfalls in [3] genannte Prinzip der Minimierung von Informationszugang („Need to Know“) wird von der Systemarchitektur bei Verwendung eines standardisierten Datenmodells mit zugehörigen Zugriffsregeln umgesetzt.

Zu beachten ist:

- Ein gemeinsames Datenmodell auf Basis des CIM [6] muss erarbeitet und der Architektur zu Grunde gelegt werden. Damit erst können strukturelle Integritätsprüfungen zentral umgesetzt werden.
- Auf Basis des Datenmodells sollten Verfahren zur semantischen Integritätsprüfung entworfen und in der Architektur verankert werden.
- Die Spezifikationen der einzelnen Systemfunktionen sollten durch ein Architekturboard geprüft werden. So können auch wirksame Reviews und Tests durchgeführt werden.
- Die Qualifikation der Produkte sollte in einem unabhängigen Gremium „IT-Board“ institutionalisiert oder ausgelagert (siehe z.B. [7]) werden.
- Aus Aufwandsgründen ist es empfehlenswert, in der endgültigen Architektur unterschiedliche Klassen der Integrität vorzusehen. Beispielsweise ist die Datenintegrität der Schaltempfehlungen unbedingt einzuhalten, die von Prognosedaten nicht zwingend.
- Bei der Auswahl der Verfahren zur Integritätssicherung müssen Laufzeitaufwände an Hand einer realitätsnahen Testumgebung geprüft werden.
- Zur Prüfung der Korrektheit wird eine dem realen Einsatz ähnliche Testumgebung benötigt.

6. Bewertung der Vertraulichkeit

Vertraulichkeit bezeichnet die Eigenschaft eines Systems, nach bestimmten Regeln Daten und Funktionen ausschließlich berechtigten Systemen beziehungsweise Personen zugängig zu machen. Das Sicherstellen von Vertraulichkeit in IT-Systemen stützt sich auf drei Mechanismen:

Authentifizierung: Alle Akteure, die mit dem System interagieren, müssen eindeutig identifizierbar sein.

Autorisierung: Um Vertraulichkeit zu wahren, muss auf Basis der Identität eines Akteurs entschieden werden, ob für Zugriff auf Dienste bzw. Daten erlaubt ist oder nicht.

Schutz der Kommunikationskanäle: Immer wenn Daten transferiert bzw. Dienste angeboten werden muss gewährleistet sein, dass die Kommunikation im System sowie zwischen Akteur und System vor dem Zugriff Dritter geschützt ist.

Die Systemarchitektur muss für alle drei Aspekte geeignete Mechanismen vorhalten. Zunächst ist erforderlich, dass eine Interaktion ausschließlich über einen Authentifizierungsdienst begonnen werden kann. Typischerweise werden hierzu in den Endanwendungen Mechanismen zur Identitätsabfrage verankert. Die serverseitigen Systemkomponenten kontrollieren die angegebenen Daten. Da ein Szenario der hier diskutierten Systemarchitektur den Einsatz einer Systeminstanz für mehrere Organisationen vorsieht muss allerdings auch eine Authentifizierung der Dienste untereinander und der Dienste gegenüber der Integrationsschicht vorgesehen werden. Insofern müssen Authentifizierungsdienste für alle aktiven Komponenten (Akteure, Dienste, extern angebundene Systeme) vorgesehen werden und an allen Interaktionspunkten (Endanwendungen, Application Server, Integration Server) geprüft werden.

Hinsichtlich der Autorisierung muss durch die Systemarchitektur ein Regelsystem vorgesehen werden, mit dem sich Zugriffsberechtigungen pro Interaktionspartner definieren lassen. Diese Berechtigungen müssen vor jeder Dienstanfrage bzw. jedem Datentransfer abgefragt werden (und der Interaktionspartner muss authentifiziert sein). Dabei unterscheidet man Zugriffe auf Dienste, auf Datentypen und auf Dateninhalte. Zugriffsschutz auf Dienste und auf Datentypen gehört in der Regel zum Funktionsumfang von Middleware. Zugriffsschutz auf Inhaltsebene bedeutet z.B. bei einer generellen Zugriffserlaubnis auf Kundendaten diejenigen Kundendaten zu sperren, denen nicht der zugreifende Akteur als Betreuer zugewiesen ist. Solche Funktionen müssen entweder als spezielle Anforderungen bei der Auswahl der Produkte berücksichtigt werden, oder die entsprechenden Dienste müssen spezifiziert und im Rahmen der Realisierung der Architektur implementiert werden.

Auch wenn beide bisher diskutierten Aspekte gewährleistet sind, muss der Kommunikationskanal vom System zum Interaktionspartner in zweierlei Hinsicht abgesichert

sein. Die Daten dürfen nicht beobachtbar sein (Abhören) und sie dürfen nicht verfälscht werden. Dies wird üblicherweise durch sog. „End to End“ Verschlüsselungskomponenten realisiert, die die Daten für alle Verarbeitungsstellen zwischen Sender und Empfänger nicht lesbar machen und deren Unversehrtheit (incl. korrektem Sender und Empfänger) überprüfbar machen.

Im Rahmen der Authentifikations- und Autorisierungsfunktionen muss von der Systemarchitektur zudem ein Mechanismus vorgesehen werden, bestehende Berechtigungen schnell und wirksam zu widerrufen.

Die Systemarchitektur in ihrem hier zu Grunde liegenden Detaillierungsgrad ist für die Erfüllung von Vertraulichkeitsanforderungen geeignet. Folgende Punkte geben hier den Ausschlag:

- Authentifizierung ist eine Standardfunktion in nahezu allen Produkten, die für die Systemelemente Application Server und Integration Server in Frage kommen. Zudem ist typischerweise innerhalb einer Organisation eine Unterstützung für Authentifizierung bereits vorhanden, z.B. bestehende Authentifizierungssysteme.
- Unterstützung für Autorisierungssysteme stehen es als fertige Komponenten frei (Open Source) und kommerziell zur Verfügung. Deren Verwendung muss aber durch die Systemarchitektur ohne Ausnahme durchgesetzt werden.
- Der Schutz von Kommunikationsverbindungen ist auf dem heutigen Stand der Technik ebenfalls realisierbar. Hier stehen entweder die Standardbibliotheken von Programmiersprachen (z.B. das SSL Paket in Java) zur Verfügung, oder Open Source Komponenten, oder COTS („Commercial Off The Shelf“, fertige Komponenten von Drittanbietern).
- Durch die Systemarchitektur wird das in [3] genannte Prinzip der mehrstufigen Absicherung („Defence in Depth“) umgesetzt, die Identität von Interaktionspartnern kann bei jeder Interaktion geprüft werden.
- Das ebenfalls in [3] genannte Prinzip der Minimierung von Informationszugang („Need to Know“) wird von der Systemarchitektur bei Verwendung eines standardisierten Datenmodells mit zugehörigen Zugriffsregeln umgesetzt.

Zu beachten ist:

- Insbesondere beim Zugriff über mobile Endanwendungen über das Internet sind in besonderem Maß Schutzmechanismen zu definieren. Hier sind auch Angriffsszenarien zu spezifizieren und mögliche Abwehrmechanismen zu erarbeiten und zu testen.
- Die Zugriffskontrollfunktionen (Access Control Lists, ACL) sind anwendungsspezifisch und müssen jeweils mit den Diensten zusammen entwickelt werden. Allerdings müssen grundlegende Schutzregeln (z.B. „Kein Zugriff auf IT-Management-Funktionen von außerhalb des Application Net“) bereits in der Systemarchitektur definiert werden.

- Bei der Auswahl der Verfahren zur Sicherung der Vertraulichkeit müssen Laufzeitaufwände an Hand einer realitätsnahen Testumgebung geprüft werden.
- Falls externe Systeme (beispielsweise bestehende LDAP Systeme) zur Umsetzung von Vertraulichkeitsanforderungen herangezogen werden, so müssen aus den Anforderungen an die Architektur QoS (Quality of Service, Dienstgüte) Anforderungen für diese externen Systeme abgeleitet werden.
- Ein gemeinsames Datenmodell auf Basis des CIM [6] muss erarbeitet und der Architektur zu Grunde gelegt werden. Wesentliche ACLs auf Typenebene können so von der Systemarchitektur vorgegeben werden.

7. Bewertung der Echtzeitfähigkeit

Hinsichtlich der Bewertung der Echtzeitfähigkeit ist zunächst noch der Begriff "Echtzeitfähigkeit" zu definieren (siehe [2]). Unterschieden werden:

- **Harte Echtzeitanforderungen**: Das Überschreiten der spezifizierten Antwortzeit ist ein Versagen des Systems. Ein System, das harte Echtzeitanforderungen erfüllt, liefert das Ergebnis *immer* innerhalb der spezifizierten Zeitschranken.
- **Weiche Echtzeitanforderungen**: Das Überschreiten der spezifizierten Antwortzeit ist schlecht, aber kein Versagen des Systems. Solche Überschreitungen treten mit sehr geringer Wahrscheinlichkeit auf. Häufig liegt der Mittelwert der Antwortzeiten deutlich unter der spezifizierten Zeitobergrenze, aber einzelne Abweichungen können nicht (oder nur mit nicht vertretbarem Aufwand) ausgeschlossen werden.
- **Feste Echtzeitanforderungen**: Das Überschreiten der spezifizierten Antwortzeit stellt die Berechnung ad absurdum. Das Ergebnis ist mit Ablaufen der Zeitschranke irrelevant. Im Wesentlichen gilt hier das gleiche wie bei weichen Echtzeitanforderungen: Vereinzelte Überschreitungen der Zeitschranken können toleriert werden und sind kein Versagen des Systems.

Der Unterschied zwischen weicher und fester Echtzeitanforderung besteht darin, dass die Berechnung im zweiten Fall mit der Zeitschranke abgebrochen werden kann, wohingegen die Berechnung bei weichen Echtzeitanforderungen noch fortgeführt wird und noch brauchbar ist.

Ein verbreitetes Missverständnis ist hierbei, dass es nicht auf die *Dauer* des Zeitintervalls ankommt, sondern auf dessen *exakte Einhaltung*. Ein Beispiel für eine harte Echtzeitanforderung ist: „Das Kommunikationssystem mit einer Weltraumsonde muss garantieren, dass ein Funksignal spätestens 5 Stunden, 2 Minuten und 10 Sekunden nach dem Start der Sonde abgesendet wird.“. Wenn diese Anforderung nicht erfüllt wird geht die Mission verloren, das System hat aber viel Zeit für die dahinter liegenden Berechnungen. Eine weiche Echtzeitanforderung ist beispielsweise „In 90% der Fälle muss die Systemantwort am GUI des Benutzers spätestens nach 2 Sekunden sichtbar sein.“

Das aktuelle System garantiert eine Antwortzeit < 15 Minuten, was dem Zeitraster des Netzleitsystems entspricht. Dieses Zeitraster im Sinne einer *festen Echtzeitbedingung* einzuhalten ist mit Aufwand machbar. Die zukünftige Systemarchitektur müsste in diesem Fall sicherstellen, dass (bei geeigneter Hardwareausstattung) die Antwortzeit zu 99,9% unter 15 Minuten bleibt.

Aus technischer Sicht sind viele Elemente im System, die keine Echtzeit-Garantien anbieten. Angefangen vom TCP/IP Protokoll (nichtdeterministisches Routing, typischerweise kein QoS auf der Leitung) über die typischen Betriebssysteme (clientsei-

tig Windows, iOS, Android, … mit nichtdeterministischem Scheduling) bis zu den Ausführungsumgebungen und Anwendungen (teilweise dynamisch interpretierte Programme, ggf. mit Just-in-time Übersetzung). Die Einhaltung der angegebenen Zeitschranken im Sinne einer harten Echtzeitbedingung ist mit der vorgeschlagenen Architektur technisch **nicht möglich**. Von dieser Interpretation der Anforderung ist somit dringend abzuraten, es gibt auf Basis der vorliegenden Informationen auch keinen Grund für harte Echtzeitanforderungen!

In Kombination mit den Anforderungen an die Integrität (siehe Abschnitt 5) muss allerdings durch die Systemarchitektur vorgesehen werden, dass zu dem Zeitpunkt der Datenübergabe an externe Systeme ein konsistenter Datensatz vorliegt. Das muss durch die Systemarchitektur mittels unterbrechbarer Umschaltung zwischen einem Datensatz und dem nächsten gültigen Datensatz realisiert werden. Ein solcher Mechanismus ist beispielsweise in Abbildung 5 dargestellt. Im Rahmen von SQL-Datenbanken ist ein solches Verfahren auch mit dem sog. „ACID“-Prinzip (Atomic, Consistent, Isolated, Durable) unter dem Stichwort „Atomic“ schon verankert.

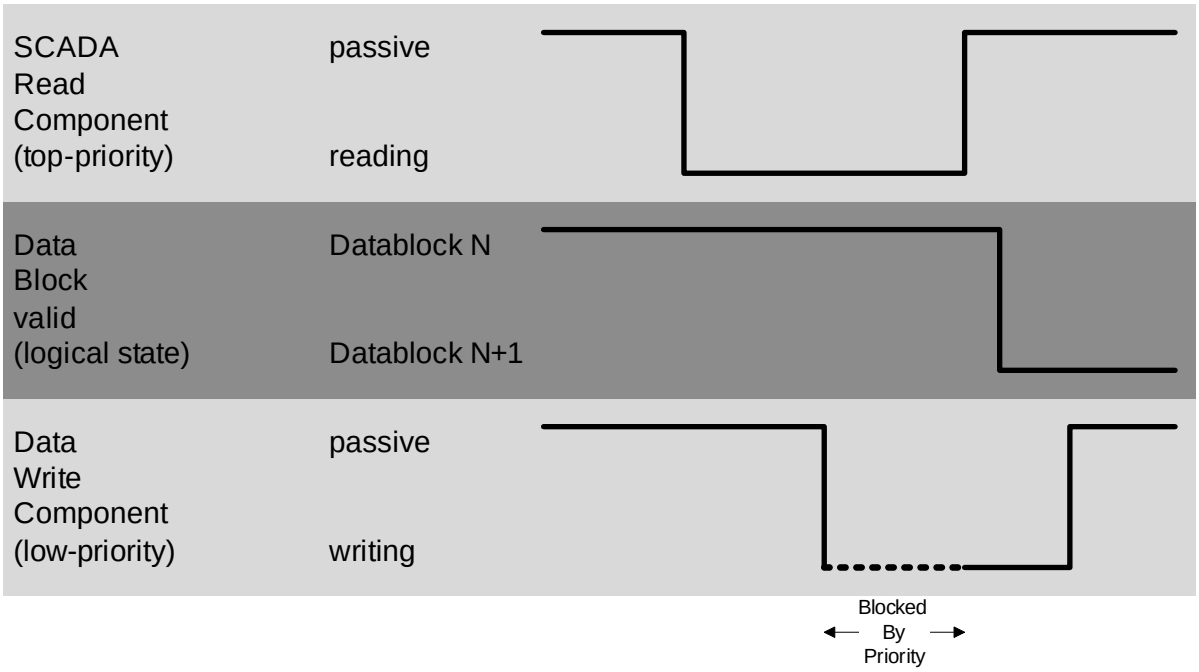

Abbildung 5: Sicherstellen integrer Daten bei festen Echtzeitanforderungen an die lesende Komponente

Die Systemarchitektur in ihrem hier zu Grunde liegenden Detaillierungsgrad ist für die Erfüllung von weichen bzw. festen Echtzeitanforderungen geeignet. Nicht aber für harte Echtzeit. Folgende Punkte geben hier den Ausschlag:

- Viele der Systemkomponenten haben dynamische Natur, und die Netzwerkverbindungen sind ohne QoS. Es ist unmöglich/unrealistisch diese Komponenten für harte Echtzeitanforderungen zu ertüchtigen.
- Die Zusicherung integrer Daten im Kontext fester Echtzeitanforderungen lässt sich über den Datenverteiler (DMZ) in Richtung NLS aus Architektursicht absichern.

Zu beachten ist:

- Die tatsächlichen Echtzeitanforderungen müssen während der Ausarbeitung der Systemarchitektur ermittelt werden. Auch deren Auswirkung auf die angebundenen externen Anwendungen muss bestimmt werden.
- Der kritische Pfad, auf den sich die Echtzeitanforderungen beziehen, muss auf einen Minimum begrenzt werden.
- Die Anforderungen an die Performance, die häufig mit dem Begriff „Echtzeit“ verbunden werden, müssen während der Ausarbeitung der Systemarchitektur ermittelt werden.
- Die Kommunikations- und Berechnungsaufwände müssen baldmöglichst mit Hilfe eines vertikalen Prototyps abgeschätzt werden.

8. Mögliche Produkte

In einer Vorstudie werden Technologien und Open Source Produkte empfohlen, die Grundlage der Umsetzung sein können. In diesem Abschnitt werden diese Technologien und Produkte beleuchtet. Durch die Wahl quelloffener Produkte mit aktiven Entwicklergemeinschaften werden eine Vielzahl der Anforderungen aus [3] und [8] durch die Systemarchitektur abgebildet. Einer der Vorzüge ist das aktive Management der Produkte durch die Entwicklergemeinschaften. Fehler werden von einer breiten Masse von Anwendern der Produkte gesucht bzw. gemeldet. Wie mit dem jeweiligen Problem umzugehen ist kann man so baldmöglichst entscheiden.

Für die Endanwendungen kommen als Basisplattform Windows, iOS und Android in Frage. Sowohl native Anwendungen wie auch leichtgewichtige Web-Anwendungen sind vorgesehen. Die Technologien sind demnach:

- Native Anwendungen ("Fat Clients"): C# .net (Windows), Java (Windows, dort eclipse RCP, Android auf mobilen Systemen), Objective C (iOS)
- „Thin Clients“: HTML5, CSS, JavaScript (insbesondere AngularJS)

Für die Laufzeitumgebungen von Application Server und Integrationsserver kommen mehrere Produkte in Frage. Hier gelistet sind aus bereits genannten Gründen nur Open Source Produkte. Auf Basis der bekannten Anforderungen ist auch keine eindeutige Empfehlung auszusprechen, insofern stellt die Auswahl unten keine abschließende Sammlung dar.

- Application Server: JBoss, Tomcat, Glassfish
- Integration Server & Workflow: JBoss, Apache ODE, Talend, Camunda
- Datenbank: PostgreSQL, MySQL

Für alle Produkte ist bekannt, dass sie im missionskritischen Bereich großer Anwendungssysteme eingesetzt werden. Im Rahmen der Detaillierung der Architektur müssen spezifische Anforderungen an die Produkte erarbeitet werden.

Eine genauere Aussage zur Eignung der Produkte ist auf Basis der Anforderungen nicht möglich, hierzu ist ein vertikaler Prototyp notwendig (siehe auch Abschnitt 11).

9. Performanceabschätzungen, Prototypen- und Einsatzszenarien

In diesem Beispiel werden Performanceabschätzungen angestellt. Auf Basis der vorliegenden Informationen müssen außerordentlich viele Annahmen getroffen werden. Somit sind die hier getroffenen Aussagen rein spekulativer Natur und geben eher einen Rahmen an, in dem Abschätzungen verfeinert werden müssen. Dazu sind die Implementierung eines vertikalen Prototyps und die damit angestellten Messungen unumgänglich.

Des Weiteren werden Konfigurationen von Hardwareausstattung für das System untersucht. Hierbei wird allerdings auf Grund der wenigen Informationen nur eine Minimalausstattung für den Prototypenbetrieb betrachtet.

Zu Grunde gelegt werden die abgeschätzten Datenmengen und Aktualisierungszeiten in der Integrationsschicht wie in Tabelle 2 dargestellt. Die Zahlen sind aus einer internen Abschätzung des Konsortiums entnommen. In den letzten beiden Zeilen sind die Datenmengen für die zyklische Aktualisierung angegeben und die Datenmengen, die bei einem asynchron auftretenden Event transferiert werden müssen:

Externes System	Datenmenge	Aktualisierungszeit
GIS	227 MB	24 Stunden
BDH	65 MB	24 Stunden
NLS	1,4 MB /20 MB	0,25 Stunden/Event
ZFA	37 MB / 37 MB	12 Stunden/Event

Tabelle 2: Zyklisch zu aktualisierende Datenmengen

Im schlechtesten Fall werden also ca. 350 MB von den externen Systemen gemeldet, müssen durch die Integrationsschicht geleitet werden und in die Anwendungsdatenbank geschrieben werden (Szenario: ZA.1 bis ZA.4). Bei einem heute üblichen Gigabit-Ethernet beträgt die Durchsatzrate an Nettodaten (Nutzdaten ohne Kontrollinformation) ca. 100 MB pro Sekunde. Zu beachten ist bei dieser groben Abschätzung auch der zusätzliche Datenaufwand, der durch Datenformate (z.B. XML) oder Metainformationen (Signatur, Sequenzkontrolle, etc.) entsteht. Nimmt man an, dass Application Server und die Integrationskomponente auf unterschiedlichen Rechnern laufen sieht man in Abbildung 4, dass für den Datenfluss zwei Netzwerktransfers geleistet werden müssen. Ohne die Laufzeitkomplexität der Verarbeitungsschritte (z.B. Signatur- bzw. Datenprüfschritte) zu kalkulieren besteht also bereits eine Verzögerung von ca. 7 Sekunden von der Datenquelle (NLS) bis in die Applikationsdatenbank. Dort müssen die Daten auch tatsächlich persistent abgelegt werden, hier ist die Schreibgeschwindigkeit der Datenbank entscheidend (Ein grober, kontextfreier Richtwert: eine postgreSQL Datenbank verarbeitet je nach Hardwareausstattung zwischen 500 und 3000 Transaktionen pro Sekunde). Selbst wenn man beide Systemkomponenten auf einen Rechner bringt beträgt die durch Kommunikation verursachte Verzögerung bereits ca. 3,5 Sekunden. Der Transfer von großen Datenmengen wie z.B. die Updates aus dem GIS und BDH können in typische Ruhezeiten des

Systems verlagert werden um die Wahrscheinlichkeit einer Datenspitze im laufenden Betrieb zu minimieren.

Der sicher günstigste Fall ist, dass nur die durch ein Ereignis ausgelösten Daten übertragen werden müssen. Laut Tabelle 2 sind das 20 MB. Für die zwei Netzwerktransfers fallen somit 0,4 Sekunden Verzögerung an. Für eine typische Reaktionszeit, die vom Nutzer noch nicht als störend empfunden wird (< 2 Sekunden) bleiben also für Berechnungen ca. 1,6 Sekunden Zeit. Da die Laufzeitkomplexität der Algorithmen zum jetzigen Zeitpunkt unbekannt ist, kann keine Aussage getroffen werden ob das System diese Zeit einhalten kann bzw. welche Anforderungen sich damit an die Hardware ergeben.

Wie viele Daten an den Benutzer und somit an die Endanwendungen weitergegeben werden (Szenario: NZ.1 bis NZ.3) ist unklar, dieser Pfad kann also in der Abschätzung nicht mit berücksichtigt werden. Das für die Endanwendungen notwendige Datenvolumen wird sich sehr stark an den jeweiligen Diensten bzw. Endanwendungen orientieren und damit stark schwanken. Das wird diese Abschätzung weiter erschweren.

Bereits an diesen beiden Betrachtungen kann man ablesen, dass ohne Daten aus einem Prototypenbetrieb keine sinnvolle Abschätzung getroffen werden kann. Nach Abbildung 4 müssen mittels eines vertikalen Prototyps die Schritte ZA.1 bis NZ.3 (reines Update der Datenanzeige) und die Schritte SE.1 bis NZ.3 (vollständiger Analyse-Aktion-Kontrolle Zyklus) gemessen werden.

Für eine einfache Implementierung des Prototyps kann man folgenden grundsätzlichen Aufbau verwenden:

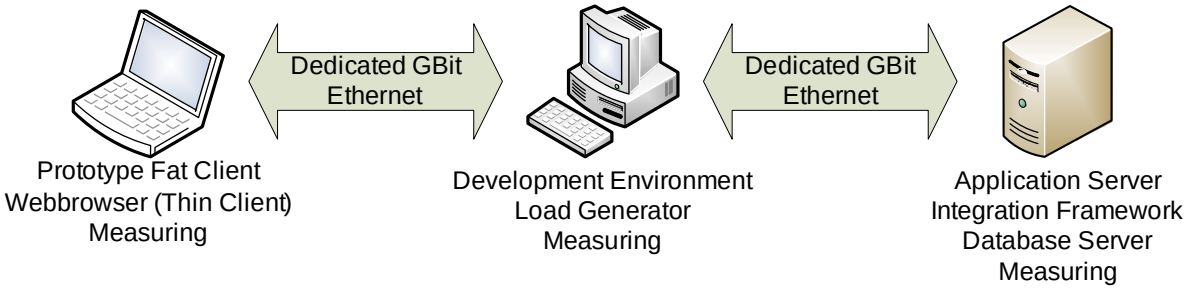

Abbildung 6: Hardwareausstattung Prototyp

Als Testhardware werden drei Rechner verwendet (siehe Abbildung 6):

- Ein **Endnutzersystem** auf dem eine prototypischer Fat Client läuft und Webbrowser (ggf. unterschiedliche), die für einen Thin Client verwendet werden können. Hier werden Antwortzeiten und der Datendurchsatz gemessen. Hierfür wird ein handelsüblicher Client-PC, beispielsweise ein Büro-Notebook verwendet.
- Auf einem der **Entwicklungsrechner** wird im Rahmen des normalen, entwicklungsbegleitenden Tests ein Lastgenerator erzeugt, der das System mit

Daten und Anfragen beschickt. Die jeweilige Datenmengen und Anfrageintervalle sind parametrierbar, um unterschiedliche Lastszenarien prüfen zu können. Hierfür wird ein normal in der Entwicklung verwendeter Rechner verwendet.

- Ein **Server** wird verwendet, auf dem als virtuelle Maschinen die drei neu zu entwickelnden Systemkomponenten gemeinsam zum Einsatz gebracht werden. Hierfür wird spezielle Hardware eingesetzt, die insbesondere über viel Arbeitsspeicher verfügt. Zusätzlich werden Messverfahren auf dem Server installiert. Je nach Ausstattung können die Virtuellen Maschinen auch auf mehrere physikalische Rechner verteilt werden.

Gemessen werden können und sollen bei diesem Einsatzszenario:

- Antwortzeit in der Endanwendung
- Datenvolumina zwischen Endanwendung und den Diensten
- Laufzeitverhalten der Algorithmen in den Diensten, sowie in den Prüf- bzw. Signaturschritten
- CPU Auslastung des Serverrechners
- Arbeitsspeicherauslastung des Serverrechners
- Performance der Datenbank
- Auslastung des ESB und Performance des Service Containers
- Datenvolumina zwischen den einzelnen virtuellen Maschinen
- Anteiliger CPU- und Arbeitsspeicherauslastung der virtuellen Maschinen

Dabei wird eine wesentliche Vereinfachung getroffen: Die Netzwerkkommunikation auf Serverseite zwischen dem Application Server und beispielsweise dem Datenbankserver wird im Prototypen **innerhalb** der Servermaschine durchgeführt. Die Messverfahren müssen dementsprechend darauf ausgelegt sein, auch den Datenverkehr zwischen den einzelnen virtuellen Rechnern zu protokollieren.

Für den vorgeschlagenen Prototypenaufbau muss der Serverrechner hohe Leistung aufweisen. Eine exakte Dimensionierung ist mangels konkreter Anforderungen nicht möglich. Einige der in 8 genannten Produkte spezifizieren jeweils für sich grundlegende Hardwareanforderungen. Die kontextfreie Betrachtung dieser Hardwareanforderungen ergibt, dass ein Serverrechner mit 64bit CPU, > 3 GHz Prozessortakt und > 32 GByte Arbeitsspeicher *vermutlich* für den Prototypen ausreichen wird.

Unabhängig von den Hardwareanforderungen bietet es sich an, alle möglichen Systemkomponenten in virtuellen Maschinen abzubilden um einfach Vergleichsdaten zu sammeln. So würde man eine virtuelle Maschine mit dem Application Server JBoss und eine weitere mit dem Application Server Glassfish vorhalten um mit einem Testszenario leicht beide Alternativen bewerten zu können.

10. Offene Punkte

Folgende offenen Punkte lassen sich aus dieser Analyse zusammenfassend hervorheben. Alle davon sollten im Rahmen einer prototypischen Implementierung untersucht werden.

1. Die Anforderungen an die Systemarchitektur müssen detailliert werden.
2. Es ist nur eine grobe Abschätzung der zu behandelnden Datenmengen verfügbar. Hinsichtlich der Daten, die an Endanwendungen transferiert werden müssen gibt es noch keine Abschätzung.
3. Unbekannt ist die Laufzeitkomplexität der Algorithmen zur Datenaufbereitung. Damit lassen sich nicht ohne weiteres Anforderungen an Hardware bzw. Laufzeitumgebungen ableiten.
4. Ein durchgängiges Safety- und Security-Konzept ist für das Gesamtsystem noch nicht aufgestellt. Dazu müsste untersucht werden, welche Teile der Systemarchitektur welche Zusagen garantieren (z.B. könnte der Application Server Authentifizierung und Autorisierung garantieren).
5. Die Interoperabilität der unterschiedlichen Produkte miteinander und die bei der Zusammenarbeit anfallenden Protokolldaten sind noch unbekannt. Insbesondere ist auch der Verbindungsprotokolloverhead unbekannt, der ggf. für erhebliche zusätzliche zu transferierende Daten sorgt.
6. Eine Strategie für die Persistenz, für Backup-Speicherungen und für das Protokollieren von Normalbetrieb sowie von Fehlerfällen muss ausgearbeitet werden.
7. Eine Strategie für ein planmäßiges Austauschen von Software-Komponenten im laufenden Betrieb (Applikations-Updates) muss erarbeitet und getestet werden.

11. Empfehlungen

Zum Abschluss des Gutachtens sind hier Empfehlungen zum weiteren Vorgehen aufgelistet.

- Bei Betrachtung des Systemarchitekturentwurfs fällt unmittelbar eine logische Zweiteilung auf:
  - der Teil des Systems, der Daten aus den externen Systemen bezieht und
  - der Teil des Systems, der auf den Daten operiert und die Nutzerinteraktion verarbeitet.

  Es ist empfehlenswert, zwischen diesen beiden Teilsystemen die Trennung und die notwendigen Schnittstellen detailliert auszuarbeiten. So kann die weitere Arbeit an der Systemarchitektur aufgeteilt und damit vereinfacht werden.
- Die Anforderungen an die Systemarchitektur müssen detailliert und erweitert werden. Das sollte mit Hilfe von Qualitätsszenarien und von strukturierten Analysen (z.B. ATAM, siehe [9]) getan werden. Dabei können auch alternative Architekturen untersucht werden, beispielsweise die Verwendung eines nachrichtenbasierten Systems vor dem ESB aus Lastgründen.
- Im Sinne einer Risikominimierung sollte schnellstmöglich eine Prototypenimplementierung aufgebaut und die offenen Punkte an diesem Prototyp geklärt werden.

Anhang A. Quellen

[1] Machbarkeitsstudie „Konsortiale Softwareentwicklung auf der Basis von Open-Source-Software“, osbf.eu/fileadmin/2013_Okt_KSE_Studie_gesamt_final.pdf, Oktober 2013

[2] Heinz Wörn, Uwe Brinkschulte: Echtzeitsysteme. Grundlagen, Funktionsweisen, Anwendungen. Springer, Berlin u. a. 2005,

[3] Whitepaper Anforderungen an sichere Steuerungs- und Telekommunikationssysteme, BDEW

[4] Anforderungen an sichere Steuerungs- und Telekommunikationssysteme – Ausführungshinweise zur Anwendung des BDEW Whitepaper, BDEW

[5] BAnz AT 20.02.2014: „Bekanntmachung zur elektronischen Signatur nach dem Signaturgesetz und der Signaturverordnung (Übersicht über geeignete Algorithmen)

[6] “Energy management system application program interface (EMS-API)”, IEC 61970

[7] “Software Qualifizierung“, http://www.tuvasi.com/de/dienstleistungen/produkte/software-qualifizierung

[8] „IT-Grundschutz Kataloge – 13. Ergänzunglieferung 2013“, Bundesamt für Sicherheit in der Informationstechnik

[9] „Architecture Tradeoff Analysis Method“, Software Engineering Institute, Carnegie Mellon University, http://www.sei.cmu.edu/architecture/tools/evaluate/atam.cfm

Anhang B. Glossar

ACID	Prinzip in der Datenbankwelt, das für Transaktionen die Eigenschaften Atomarität, Konsistenz, Isolation und Dauerhaftigkeit zusichert.
ACL	Access Control List, regelt typischerweise die Zugriffsmöglichkeiten von Aktoren auf Ressourcen
BDH	Betriebsmitteldatenhaltung, IT-System zur Verwaltung von Betriebsmitteln
BPMN	Business Process Model and Notation, Spezifikationskonzept für Geschäftsprozesse, zusammengesetzt aus Datenmodell und Notation
COTS	Commercial Off The Shelf, Softwarekomponenten, die bereits fertig von Drittanbietern zur Verfügung stehen.
Design by Contract	Entwurfsverfahren, das Systeme in einzelne Dienste zerlegt und diese Dienste im Sinne von „Verträgen“ mit klaren Vor- und Nachbedingungen in Beziehung setzt
ESB	Enterprise Service Bus, Softwareprodukt, das die Integration von Systemen über die Verschaltung von Diensten ermöglicht
GIS	Geoinformationssystem, IT-System zur Verwaltung geographischer Daten
Kopplung	Metrik zwischen Subsystemen bzw. Komponenten. Je stärker die Änderung einer Komponente andere Komponenten beeinflusst, umso höher ist die Kopplung
Layer	Gliederungseinheit einer Architektur: Ein Layer teilt ein System entlang der logischen Dimension auf. Beispielsweise in Presentation-Layer, Business-Layer, und Persistence-Layer
LDAP	Lightweight Directory Access Protocol, Protokoll eines Verzeichnisdienstes, der z.B. Authentifizierungsdaten vorhält
NLS	Netzleitsystem, IT-System zur Steuerung von Netzinfrastruktur
QoS	Quality of Service, Oberbegriff für Qualitätsanforderungen an die Dienstleistung eines Systems, z.B. Antwortzeit
SCADA	Supervisory Control and Data Acquisition, Computersysteme zur Steuerung technischer Abläufe
Tier (engl.)	Gliederungseinheit einer Architektur: Ein Tier teilt ein System entlang der technischen Dimension auf. Beispielsweise in Client-Tier, und Server-Tier.
Verfügbarkeit	Maß für die Wahrscheinlichkeit, dass zu einem bestimmten Zeitpunkt ein (per Schnittstelle) zugesicherter Dienst auch zur Verfügung steht.