MODELLIERUNG UND KONFIGURATIONSGENERIERUNG FÜR BAUSTEINBASIERTE SATELLITENSYSTEME

M. Göller, L. Pfotzer, J. Oberländer, K. Uhl, T. Büttner, A. Rönnau, R. Dillmann
FZI Forschungszentrum Informatik, Interaktive Diagnose- und Servicesysteme
Haid-und-Neu-Str. 10-14, D-76131 Karlsruhe, Deutschland

Zusammenfassung

Bei der Entwicklung zukünftiger Satellitensysteme wächst das Interesse an einem modularen Aufbau, bringt dieser doch zwei entscheidende Vorteile mit sich: Zum einen erleichtert er das On-Orbit Servicing, da die Modularisierung zur Rekonfigurierbarkeit und somit zur vereinfachten Reparatur des Systems durch den Austausch defekter Bausteine im Orbit genutzt werden kann. Zum anderen vereinfachen sich das Design und die Auslegung von Satellitensystemen durch Standard-Bausteine und Softwareunterstützung im Designprozess.
In diesem Beitrag soll der softwareunterstützte Designprozess von modularen, auf Standard-Bausteinen basierenden Satelliten näher untersucht werden. Entscheidend hierbei sind folgende Fragestellungen: Wie kann ein gegebener Bausteinkatalog modelliert werden, und wie kann aus diesem Modell eine (pareto-)optimale Konfiguration von Bausteinen für einen Satelliten erzeugt werden?
Zunächst wird hierzu eine Bausteinhierarchie definiert, beginnend bei generischen Bausteinklassen hin zu konkreten Systembausteinen. Dabei werden Eigenschaften der Bausteine, wie z.B. benötigte oder zur Verfügung gestellte Ressourcen, stufenweise konkretisiert. Dann werden die System- und Missionseigenschaften modelliert. Hier sind beispielsweise benötigte Sensoren, aber auch benötigte Ressourcen wie Stromversorgung oder Rechenkapazität zu nennen. Zur Modellierung wird die Ontologiesprache OWL-DL verwendet. Aus dem Modell können daraufhin explizit modellierte Regeln, ergänzt durch mittels Inferenz gewonnene implizite Regeln, extrahiert werden. Diese Regeln definieren ein Optimierungsproblem, das aufgrund der komplexen und hochdimensionalen Optimierungsfunktion, die aus der Vielzahl der Komponenten der Regeln resultiert, mit Hilfe von evolutionären Algorithmen gelöst wird.
Mit diesem Verfahren wird einem Nutzer die Möglichkeit gegeben, aus einem Katalog von Standardbausteinen einen Satelliten für eine gewünschte Mission rechnerunterstützt zusammenzustellen und eine günstige, den Missionsanforderungen genügende Gesamtkonfiguration zu berechnen.

1. EINLEITUNG

Bisher sind Satellitensysteme meist monolithische, abgeschlossene und sehr spezifisch ausgelegte Raumfahrtsysteme. Eine Wartung von Satelliten im Orbit gestaltet sich äußerst schwierig, da sie sehr feingranulare Manipulationen im Inneren des Satelliten notwendig macht, wofür spezielles Werkzeug, großes Know-How und sehr weit entwickelte Manipulationsfähigkeiten notwendig sind.

Von Interesse ist daher die Erarbeitung von Konzepten, welche die Modularität von Satellitensystemen ermöglichen. Hierbei geht es nicht nur um Modularität auf (Sub-) Systemebene, vielmehr sollen Satelliten komplett aus standardisierten, gleichförmigen Bausteinen konstruiert werden. Dies hätte den Vorteil, dass künftig Satelliten aus günstigen und erprobten Standardkomponenten zusammengesetzt werden können. Der Aufbau bzw. die Modifikation eines Satelliten würde deutlich vereinfacht, so dass sogar eine Reparatur von Satelliten im Orbit möglich wird, da nicht mehr feinmotorisch Reparaturen im Inneren des Satelliten durchgeführt werden müssen, sondern defekte Bausteine als Ganzes getauscht werden könnten.

Neben dem Entwurf der Satelliten-Bausteine selbst (siehe [9]) ist die Konfiguration einer Auswahl von Bausteinen aus einem Bausteinkatalog zu einem sinnvollen Gesamtsystem notwendig. Dieser Prozess der Synthese eines Raumfahrtsystems, ausgehend von einem definierten Bausteinkatalog aus Satellitenbausteinen hin zu einer kompletten Baustein-Konfiguration, softwareunterstützt durchzuführen, soll in dem vorliegenden Beitrag näher untersucht werden. Die wichtigsten Schritte sind hierbei die folgenden:

1) Modellierung des Bausteinkatalogs in einer Wissensbasis.
2) Generierung von Regeln zur Auswahl und Anordnung von Bausteinen.
3) Auswahl von Bausteinen passend zur Mission.
4) Anordnung der Bausteine zu einer Konfiguration.

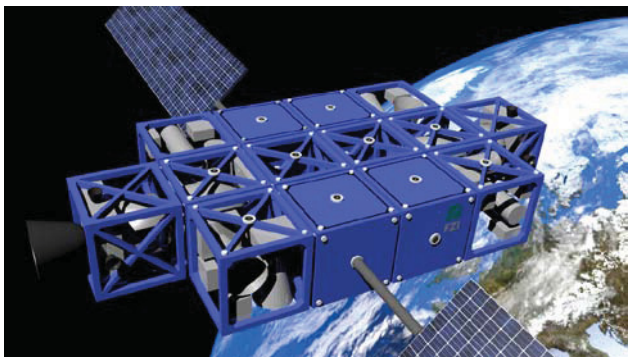

BILD 1. Beispielkonfiguration eines Raumfahrtsystems, das aus Standardbausteinen generiert wurde

Von den identifizierten Bausteinen, Bedingungen und Regeln werden einige in einer exemplarischen Wissensbasis modelliert, die dann auch zur Illustration der weiteren Schritte genutzt wird. BILD 1 zeigt ein Beispiel für eine solche Konfiguration aus Standardbausteinen.

2. SYNTHESE

Die Erstellung einer Satellitenkonfiguration ist eine hoch komplexe Aufgabe, die eine Vielzahl von Randbedingungen berücksichtigen muss. Daher soll diese Aufgabe zerlegt werden und in einer schrittweisen Synthese erfolgen. Hierzu werden mehrere Arbeitsschritte definiert, die ausgehend von der Modellierung des Bausteinkatalogs das Wissen schrittweise verfeinern. Die schrittweise Synthese soll dabei einer Reihe von Anforderungen genügen:

1) Der Bausteinkatalog soll in einer Art und Weise modelliert werden, die die Eigenschaften der Bausteine umfassend darstellen kann und gleichzeitig für den menschlichen Nutzer leicht verständlich und übersichtlich bleibt.
2) Um dem Nutzer während des Syntheseprozesses Einfluss auf den Entwurf zu geben und so auch das umfangreiche menschliche Expertenwissen zugänglich zu halten, soll der Syntheseprozess zwar automatisch erfolgen können, aber den manuellen Eingriff des Nutzers in den einzelnen Syntheseschritten zulassen.
3) Die einzelnen Schritte des Syntheseprozesses sollen unabhängig voneinander durchführbar sein. Soll zum Beispiel für ein bestehendes Raumfahrtsystem eine neue Konfiguration bestimmt werden (etwa weil sich Anforderungen oder Randbedingungen geändert haben), so muss der Syntheseprozess nicht wieder von vorn beginnen, sondern es kann direkt auf den alten – möglicherweise aktualisierten – Informationen weitergearbeitet werden. Es muss nur dann von vorn begonnen werden, wenn Änderungen am zu Grunde liegenden Bausteinkatalog erforderlich werden (z.B. weil neue Bausteintypen hinzukommen, die in der neuen Konfiguration genutzt werden sollen).

2.1. Der Syntheseprozess

Die im Bausteinkatalog verfügbaren Satellitenbaustein-Typen werden zunächst in einer *Baustein-Wissensbasis* modelliert. Diese enthält die relevanten Parameter aller Bausteintypen hinsichtlich ihrer Fähigkeiten und Einsatzbedingungen. Basierend auf der Wissensbasis und den Anforderungen an das Vorhandensein von Baustein-Fähigkeiten im Satelliten können dann im *ersten Optimierungsschritt* (halb-)automatisch diejenigen Bausteintypen ausgewählt und in ihrer benötigten Anzahl instanziiert werden, die nötig sind, um den Anforderungen der Mission zu genügen. Diese Auswahl an konkreten Bausteinen bildet dann die *Satelliten-Wissensbasis*. Sie enthält genau die konkreten Bausteine, die im Satelliten verwendet werden sollen. Hier ist es auch möglich, konkrete Abweichungen der Bausteine von ihren Baustein-Klassen (z.B. für Sonderanfertigungen) zu modellieren. Basierend auf der Satelliten-Wissensbasis extrahiert dann ein Reasoning-Mechanismus Regeln zur Konfiguration (d.h. der Platzierung) der Bausteine. Basierend auf der Satelliten-Wissensbasis und diesen Regeln soll dann im *zweiten Optimierungsschritt* eine gültige (pareto-)optimale Konfiguration der Bausteine gefunden werden. Zusammenfassend besteht die Konfigurationsgenerierung damit aus folgenden Teilschritten:

1) Modellierung des Bausteinkatalogs in der Baustein-Wissensbasis.
2) Definition der Missionsanforderungen an die Fähigkeiten der Bausteine.
3) Halbautomatische Auswahl eines Satzes von Bausteinen aus der Baustein-Wissensbasis und Konkretisierung der Bausteine. Durch den ersten Optimierungsschritt ergibt sich die Satelliten-Wissensbasis.
4) Schließen auf Rahmenbedingungen und Einschränkungen aus der Satelliten-Wissensbasis.
5) Generieren einer Konfiguration der Bausteine im zweiten Optimierungsschritt.

Die Darstellung in BILD 2 skizziert die wesentlichen Schritte dieses Vorgehens.

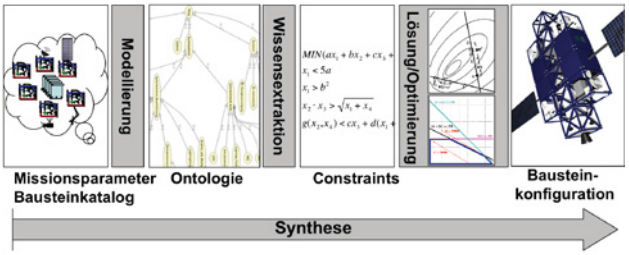

BILD 2. Syntheseschritte von Missionsparametern und Bausteinspezifikationen zur Konfiguration des Raumfahrtsystems

Bei der schrittweisen Synthese soll der Fokus möglichst auf der deskriptiven Seite, d.h. der Modellierung, liegen, weniger auf der prozeduralen Nachbereitung der Ergebnisse. Daher soll es eine Baustein-Wissensbasis geben, welche die Typen der Bausteine modelliert, und ergänzend für jedes konkret zu planende Satellitensystem eine Satelliten-Wissensbasis, in der nur die konkret verwendeten Bausteinkandidaten enthalten sind. Somit ist ein effizientes Schließen auf einer relativ kleinen Wissensbasis möglich, und die resultierenden Regeln können direkt ohne weiteres Filtern für die Konfigurationsgenerierung genutzt werden.

Im Rahmen dieses Beitrags werden die Wissensbasen in Form von Ontologien realisiert, weshalb in den folgenden Abschnitten von *Baustein-* bzw. *Satelliten-Ontologien* gesprochen wird.

2.2. Modellierung von Bausteinen und Inferenz von Regeln

Ausgangspunkt für den Syntheseprozess ist ein definierter Bausteinkatalog. Dieser enthält alle verfügbaren Baustein-Typen mit ihren Attributen wie zum Beispiel Masse und Schwerpunkt, aber auch Fähigkeiten und Einsatzbedingungen.

2.2.1. Baustein- und Komponenten-Definitionen

Als Vorstufe zur formalen Modellierung mittels der Ontologie werden zunächst die einzelnen Systemelemente des Gesamtsystems inklusive ihrer Abhängigkeiten definiert. Das entwickelte Netzwerk von Elementen und Abhängigkeiten enthält insbesondere eine Hierarchie von Bausteinen. Diese beginnt an der Wurzel bei einem generischen

Baustein und verfeinert sich dann über generische Systembausteine hin zu konkreten Bausteinen wie z.B. Strukturbausteinen oder Sensorbausteinen, die eine konkrete Kamera enthalten, an den Blättern des Baumes. Entlang der Hierarchie werden Eigenschaften der Bausteine vererbt und ebenenweise verfeinert.

2.2.2. Regel-Typen und Bedingungen

Bevor mit der Modellierung in der Ontologie selbst begonnen werden kann, müssen weiterhin die verschiedenen Typen von Regeln und Bedingungen definiert werden. Hierbei gibt es zwei Felder: Diejenigen, die nötig sind, um eine Auswahl an Bausteinen aus dem Katalog treffen zu können, welche den Anforderungen der Mission genügen, sowie diejenigen, die nötig sind, um die ausgewählten Bausteine (im Folgenden mit *BS* abgekürzt) in einer gültigen und (pareto-)optimalen Konfiguration anzuordnen. Im folgenden Abschnitt wird der Fokus auf die Regeln für die Konfigurationserzeugung gelegt, die Bedingungen für die Auswahl werden nur kurz aufgelistet. Die resultierenden Listen haben keinen Anspruch auf Vollständigkeit und sollen vielmehr exemplarisch die verschiedenen Möglichkeiten von Einschränkungen aufzeigen, die später in den Optimierungsschritten berücksichtigt werden.

2.2.2.1. Bedingungen für die Bausteinauswahl

Um die erforderlichen Bedingungen für das Treffen einer sinnvollen Auswahl an Bausteinen aus dem Bausteinkatalog zu definieren, werden die Bausteine zunächst in *funktionale Gruppen* eingeordnet:

1) *Missions-BS*: Diese Bausteine sind zwingend notwendig, um die Mission erfüllen zu können (z.B. ein Baustein, der den zentralen Sensor der Mission enthält).
2) *Betriebs-BS*: Diese Bausteine sind notwendig für den Betrieb des Satellitensystems. Hierunter fallen Bausteine, die spezielle Fähigkeiten zur Verfügung stellen (z.B. Sensoren zur Lagebestimmung).
3) *Skalierbare BS*: Diese Bausteine stellen Ressourcen zur Verfügung. Sie müssen entsprechend des Bedarfs des Gesamtsystems skaliert werden – entweder in der Anzahl oder in der Leistung pro Baustein.
4) *Struktur-BS*: Diese letzte Gruppe besteht aus passiven Bausteinen, die dem Gesamtsystem Stabilität sowie Anschlusskapazitäten für den Ressourcentransport mit besonders hoher Rate zur Verfügung stellen.

Aus diesen Überlegungen zu den Gruppen von Bausteinen resultieren die folgenden Gruppen von *Bedingungen*, die bei der Auswahl von Bausteinen berücksichtigt werden müssen:

1) *Ressourcen*: z.B. Elektrizität, Wärmeableitung, Rechenleistung, Speicher, Antrieb, Treibstoff
2) *Fähigkeiten*: z.B. spezielle Fähigkeiten der Missions-BS, Lagebestimmung, Bestimmung der Sonnenposition, Kommunikation
3) *Anzahl erforderlicher Struktur-BS*

Eine Bausteinauswahl ist *gültig*, wenn alle Fähigkeiten abgedeckt sind, und *optimal*, wenn das System hierfür minimal skaliert ist.

2.2.2.2. Regeln für die Bausteinplatzierung

Um nun die ausgewählten Bausteine zu einer gültigen und (pareto-)optimalen Konfiguration zusammenfügen zu können, müssen die Typen von Regeln, die es hierbei zu beachten gibt, identifiziert werden. Hierbei gibt es zwei Dimensionen: Zum einen können Regeln notwendig sein oder nur eine zu optimierende Empfehlung geben. Zum anderen können sie sich auf die Anzahl der benötigten Bausteine beziehen. Im Folgenden werden die Regeln entsprechend der zweiten Dimension gelistet und mit der Kennzeichnung „(*N*)otwendig“ bzw. „(*E*)mpfehlung“ versehen. Weitere Regeln können durch den Nutzer hinzugefügt werden.

Globale Regeln: beziehen sich auf alle Bausteine.

- *Cluster*
  - **Anzahl BS-Cluster = 1** (*N*): Der Satellit *muss* zusammenhängend sein
  - **Anzahl Struktur-BS-Cluster = MIN** (*E*): Die Strukturbausteine *sollten* alle zusammenhängend sein
  - **Durchmesser des größten Struktur-BS-Clusters = MAX** (*E*): Nach Möglichkeit sollen die Strukturbausteine eine zusammenhängende Achse bilden (als „Highway“ für den Ressourcen-Transport)
- *Flüsse*
  Diese Regeln sind als Maximum-Flow-Problem rechenaufwändig zu prüfen, da sie dynamisch von den jeweils denkbaren Aktivitätszuständen der Systeme sowie Speicherkapazitäten und möglichen Lagen im Orbit abhängen. Deshalb kann es sinnvoll sein, diese Regeln nur für die fertigen Konfigurationen zu prüfen und diese bei Bedarf in einem Nachbereitungsschritt anzupassen.
  - **Stromfluss** (*N*): Sind Quellen, Senken und Leitfähigkeit des Baustein-Netzes groß genug?
  - **Wärmefluss** (*N*): Sind Quellen, Senken und Leitfähigkeit des Baustein-Netzes groß genug?
  - **Datenfluss** (*N*): Ist die Durchleitfähigkeit des Baustein-Netzes groß genug?
- *Anzahl Verbindungen zwischen Bausteinen*
  - **Anzahl gültiger BS-Verbindungen = MAX** (*E*): Vorziehen einer kompakten Konfiguration
  - **Ressourcentransport = MAX** (*E*): Bausteine mit hohem Ressourcenbedarf oder hoher Ressourcenbereitstellung sollen direkt an den „Highway“ der Strukturbausteine angeschlossen sein.
  - **Keine ungültigen BS-Verbindungen** (*E*): Es sollten keine Bausteinflächen aneinander grenzen, die keine Schnittstelle haben.
- *Baustein-Verteilung*
  - **Ein Sonnensensor in jede Richtung** (*N*)
  - **Gleiche Anzahl Sonnensensoren in alle Richtungen** (*E*)

BS-relative Regeln: beziehen sich auf eine definierte Menge spezifischer Bausteine, meist zwei Stück. Diese Regeln können durch Operatoren verknüpft werden. Zum Beispiel ist es nicht gültig, wenn ein Scheinwerfer entgegengerichtet zu einem lichtempfindlichen Sensor orientiert ist *und* sie sich gegenüber positioniert sind.

- *Abstand*
  - **Abstand zwischen BS = MIN (MAX)** (*E*): Zwei Bausteine sollen einen möglichst kleinen (gro-

ßen) Abstand haben.
    - **Mindest-/Maximal-Abstand zwischen BS** (*N*): Zwei Bausteine dürfen nicht näher als bzw. nicht weiter voneinander entfernt sein als ein definierter Abstand.
- *Orientierung*
    - **Offset Orientierung** (*N*): Bausteine müssen ein spezifisches Offset in ihrer Orientierung haben (mit Offset = 0: BS müssen die gleiche Orientierung haben).
    - **Nicht Offset Orientierung** (*N*): Bausteine dürfen ein spezifisches Offset in ihrer Orientierung *nicht* haben (mit Offset = 0: BS dürfen *nicht* die gleiche Orientierung haben).
- *Position*
    - **Offset Seite** (*N*): Bausteine müssen ein spezifisches Offset bezüglich der Seite haben, an der sie am Satelliten montiert sind (mit Offset = 0: BS müssen an der gleichen Seite des Satelliten sein).
    - **Nicht Offset Seite** (*N*): Bausteine dürfen *nicht* ein spezifisches Offset bezüglich der Seite haben, an der sie am Satelliten montiert sind (mit Offset = 0: BS dürfen nicht an der gleichen Seite des Satelliten sein).

BS-spezifische Regeln: beziehen sich nur auf einen individuellen Baustein.

- **Verdeckung des Sichtkegels** (*N*): Der Baustein erfordert, z.B. für einen bestimmten Sensor, dass dessen Sichtkegel nicht verdeckt werden darf.
- *Orientierung*
    - **Spezifische Orientierung** (*N*): Der Baustein ist (bzgl. eines definierten Koordinatensystems) in eine bestimmte Richtung auszurichten.
    - **Nicht Spezifische Orientierung** (*N*): Der Baustein darf (bzgl. eines definierten Koordinatensystems) in eine bestimmte Richtung *nicht* ausgerichtet werden.
- *Lage*
    - **Spezifische Seite** (*N*): ein Baustein soll an einer bestimmten Seite des Satelliten angebracht sein (diese muss zusammen mit dem passenden Koordinatensystem definiert werden: Satelliten-KS oder erdfestes KS).
    - **Nicht Spezifische Seite** (*N*): ein Baustein darf an einer spezifischen Seite des Satelliten *nicht* angebracht sein.
    - **Hülle** (*N*): Ein Baustein muss an der Hülle des Satelliten liegen.
    - **Nicht Hülle** (*N*): Ein Baustein darf *nicht* an der Hülle des Satelliten liegen, sondern muss im Inneren verbaut sein.
    - **Lage im System = MIN (MAX)** (*E*): Ein Baustein soll möglichst weit innen oder außen liegen.

Insgesamt kann eine Konfiguration als *gültig* angesehen werden, wenn sie alle notwendigen Regeln erfüllt, und als *optimal*, wenn dabei die Empfehlungen so gut wie möglich berücksichtigt wurden.

Ein Beispiel für einen Satz Regeln sei hier für zwei Kamerabausteine angegeben, die zusammen ein Stereokamerasystem bilden sollen. Hierzu wurde einer der beiden Bausteine (willkürlich) als Master, der andere als Slave definiert. Beim Master gelten folgende Regeln:

- Muss in eine definierte Richtung (hier: entlang der positiven X-Achse (1,0,0) im Satelliten-KS) schauen, die Rotation um diese Achse ist beliebig (-):
  → **Spezifische Orientierung (Satelliten-KS, 1,0,0, -)**
- Muss an der passenden Seite des Satelliten angebracht sein:
  → **Spezifische Seite (Satelliten-KS, 1)**

Für den Slave gelten die Regeln:

- Muss an der gleichen Seite sein wie der Master und in die gleiche Richtung schauen:
  → **Offset Seite (0, Master)**
  → **Offset Orientierung (0, Master)**
- Soll möglichst weit entfernt sein vom Master (um einen großen Basisabstand zu haben):
  → **Abstand zwischen BS (Master) = MAX**

2.2.3. Verwendung einer Ontologie zur Modellierung von Basiswissen

Der Bausteinkatalog sowie die beschriebenen Konzepte zur Modellierung der Bausteine, der Bedingungen und der Regeln sollen nun in Form einer *Ontologie* modelliert werden. Ontologien sind Formalismen, mit deren Hilfe man grundlegende Dinge (= Typen bzw. Entitäten) sowie ihre Beziehungen untereinander maschinenlesbar beschreiben kann. Eine solche formale Beschreibung hat den Vorteil, dass man später in der Lage ist, mit Hilfe von automatisierten Schlussfolgerungen (logisches Schließen, Inferenz, Deduktion, Abduktion, Induktion) Wissen zu extrahieren. Zudem steht die einmal in maschinenlesbarer Form vorliegende Basis auch in zahlreichen weiteren Anwendungen zur Verfügung. Unser Ziel hierbei ist es, die einzelnen Baustein-Typen zu modellieren, konkrete Bausteine daraus abzuleiten und schlussendlich mittels Inferenz konkrete Regeln zwischen den Bausteinen zu extrahieren, die dann bei der Konfigurationsgenerierung berücksichtigt werden.

In den folgenden Abschnitten soll die Modellierung und die Inferenz von Wissen anhand einer beispielhaften Ontologie, die eine Auswahl an exemplarischen Bausteinen und Regeln enthält, gezeigt werden. Als Sprache zur Modellierung haben wir exemplarisch *OWL*, die *Web Ontology Language* des W3C-Konsortiums [7] gewählt. Die Modellierung soll mittels des Werkzeugs *Protégé* [6] der Stanford University durchgeführt werden. Protégé ist eine Open-Source-Anwendung und durch die Implementierung in Java plattformunabhängig. Als Reasoning-Werkzeug zur Inferenz von Wissen kommt *FaCT++* (*Fast Classification of Terminologies*) zum Einsatz [1]. Diese Kombination von Werkzeugen ist frei verfügbar und aufeinander abgestimmt.

2.2.3.1. Komponenten der Ontologie

Zunächst werden die drei Hauptfelder der Ontologie identifiziert und jeweils in Form einer eigenen Klassenhierarchie modelliert (siehe BILD 3). Diese drei Felder sind:

1) die Klassen der Bausteine selbst inklusive ihrer Eigenschaften;
2) spezielle Klassen von Parametern, die (z.B. über elektromagnetische Strahlung) Einfluss über Bausteingrenzen hinweg haben; und
3) die Regel-Klassen, die für die Platzierung der Bausteine gelten müssen.

Neben den Klassenhierarchien gibt es noch eine vierte

Hierarchie, die *Object-Property*-Hierarchie (siehe BILD 4). Diese modelliert vielfältige Relationen zwischen Klassen. Die vier Hierarchien sollen in den nächsten Abschnitten näher erläutert werden. Es sei der Leser darauf hingewiesen, dass die im Folgenden nacheinander eingeführten vier Hierarchien eng miteinander verwoben sind und daher die Definitionen einiger Begriffe Bezug auf erst später eingeführte Definitionen nehmen müssen. In BILD 10 findet sich abschließend eine Darstellung der gesamten Ontologie. Einzelheiten sind hierbei nicht identifizierbar, die hierarchische Gesamtstruktur ist aber gut erkennbar.

BILD 3. Die drei Klassenhierarchien

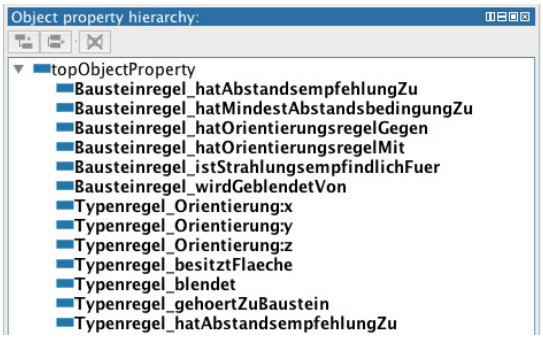

BILD 4. Auszug aus der Object-Property-Hierarchie zur Modellierung von Relationen

Relationen: Die Object-Property-Hierarchie

Die *Object-Property*-Hierarchie enthält alle Relationen, die ein Quell- und ein Ziel-Objekt betreffen bzw. verknüpfen. Objekte können hierbei alle Elemente der drei Klassenhierarchien (inkl. ihrer Individuen) sein (siehe folgende Absätze), also auch abstrakte Objekte wie z.B. Regeln. Bei der Modellierung der einzelnen Elemente der Klassenhierarchien wird definiert, welche dieser Relationen auf das jeweilige Objekt zutreffen. Zum Beispiel betrifft die Relation „strahlt“ das Baustein-Individuum „BST_TMUndTC_GLST“ (Telemetrie) sowie das Parameter-Individuum „25MHz“ der Parameter-Klasse „Strahlung“ (siehe BILD 5). Zu vielen Relationen ist es sinnvoll, eine inverse Relation zu definieren, um später die Inferenz in beide Richtungen zu erlauben. Hier ist das z.B. „wirdGestrahlt“ als Inverse zu „strahlt“.

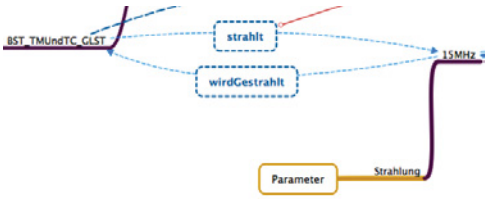

BILD 5. Die Property (Relation) „strahlt“ bezieht sich auf die Individuen „BST_TMUndTC_GLST“ und „15MHz“

Die Baustein-Klassenhierarchie

Die Baustein-Hierarchie gliedert sich in sieben Sub-Klassen, die jeweils der primären Funktion eines enthaltenen Bausteins zugeordnet sind. Dies sind Bausteine zur Datenverarbeitung, Energieversorgung, Kommunikation, Lageregelung, Sensor-BS, Struktur-BS und sonstige Bausteine in der Klasse „Nutzlast“. Der Stand der Modellierung erhebt hierbei keinen Anspruch auf Vollständigkeit. An dieser Stelle ist es lediglich das Ziel, anhand einer exemplarischen Ontologie die Möglichkeiten der Modellierung und Inferenz aufzuzeigen.

Auf den nächsten Ebenen in der Hierarchie folgen auf die Klassen die Individuen (siehe BILD 6, BILD 7). Die Trennung an dieser Stelle liegt in den Eigenheiten der Werkzeuge OWL und Protégé begründet: Das Treffen von Annahmen, d.h. die Zuordnung von Relationen zu Objekten, ist nur bei Individuen möglich, die Vererbung muss dann aber manuell, durch zusätzliche Object Properties, modelliert werden. Bei Klassen hingegen findet die Vererbung automatisch statt. Die Trennung zwischen Klassen und Individuen findet ab der entsprechenden Ebene statt, in der Annahmen getroffen werden müssen. Die oberste Ebene der Individuen enthält Typen von Bausteinen. Dies wäre zum Beispiel der Typ „Magnettorquer-Baustein_A“ in der Sub-Klasse der „Magnettorquer-BS“ in der Oberklasse der Lageregelungsbausteine. Beispiele für Annahmen, die durch das Zuordnen von Relationen zu Objekten modelliert werden, finden sich in BILD 8 und BILD 9. Hier wird definiert, dass der Magnettorquer-BS strahlungsempfindlich ist und dass der Telekommunikations-BS strahlt. Die Relationen verknüpfen diese beiden Bausteine jeweils mit einem Individuum der Klasse „Wellenlänge“. Außerdem ist definiert, dass der Telekommunikations-BS immer an der der Erde zugewandten Seite des Satelliten sein soll (Relation: hatRegel bzgl. *z*-Koordinate im Erdfesten Koordinatensystem). Mit dieser ersten Ebene der Individuen endet die Tiefe der Baustein-Ontologie.

In der Satelliten-Ontologie werden dann nur die tatsächlich im Satelliten zu verwendenden (Ober-/Unter-)Klassen und Individuen verwendet. Hier wird dann eine zweite Ebene von Individuen hinzugefügt: Konkrete, einzigartige Bausteine. In einem Satelliten kann jeder BS-Typ (Individuen-Ebene 1) nur einmal vorkommen, aber in beliebig vielen konkreten (dann einzigartigen) Instanzen (Individuen-Ebene 2).

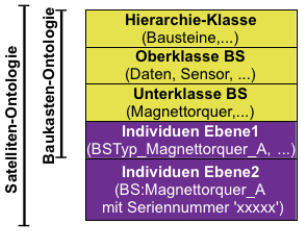

BILD 6. Baustein-Hierarchie mit Klassen (gelb) und Individuen (violett)

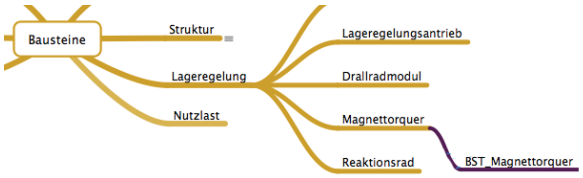

BILD 7. Auszug aus der Baustein-Hierarchie mit Klassen (gelb) und einem Individuum der Ebene 1 (violett)

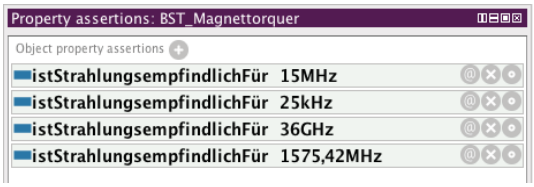

BILD 8. Definition des Bausteintyps „BST_Magnettorquer“ mit den zugehörigen Relationen aus den Object Properties

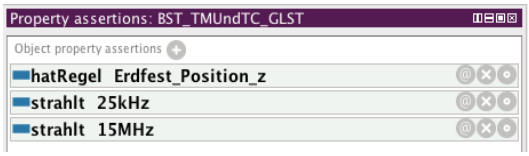

BILD 9. Definition des Bausteintyps „BST_TMUndTC_GLST“ mit den zugehörigen Relationen aus den Object Properties

Die Parameter-Klassenhierarchie

Die Parameter-Hierarchie beschreibt diejenigen Objekte, die Auswirkungen auf die Objekte der Baustein-Hierarchie haben können bzw. von ihnen verursacht werden. Ein Beispiel für ein solches Objekt ist die Parameter-Klasse „Strahlung“ mit den verschiedenen Strahlungsfrequenzen als Individuen. Diese können von Objekten der Baustein-Hierarchie emittiert werden und können empfindliche Objekte der Baustein-Hierarchie beeinflussen. Die Objekte (genauer: die Individuen) der Parameter-Hierarchie (z.B. „15MHz“, ..., „1575,42Mhz“, „Erdfeste_Position_Z“) werden durch die Relationen (Object Properties: „istStrahlungsempfindlich“, „hatRegel“, „strahlt“) an die Objekte der Baustein-Hierarchie gebunden (siehe BILD 8 und BILD 9). Hier kann wieder das bereits erwähnte Beispiel der Baustein-Hierarchie-Instanzen „BST_Magnettorquer“ und „BST_TMundTC_GLST“ angeführt werden.

Die Regel-Klassenhierarchie

Die Regel-Hierarchie enthält zwei Klassen von Objekten. Die erste Klasse modelliert die Baustein-spezifischen Regeln, die zweite Klasse die Baustein-relativen Regeln.

Im Fall der Baustein-spezifischen Regeln werden die Individuen der Baustein-spezifischen Regel-Klasse über die Relationen mit den zugehörigen Bausteinen verknüpft. Ein Beispiel hierfür ist die Verknüpfung des Individuums „BST_TMUndTC_GLST“ mit dem Individuum „Erdfeste_Position_Z“ mittels der Relation „hatRegel“ aus BILD 9. Da die Baustein-spezifischen Regeln nicht von den Relationen zwischen Bausteinen abhängen, können diese auf die beschriebene Weise direkt modelliert und später extrahiert werden. Im Fall der Baustein-relativen Regeln liegt das Wissen hingegen nur implizit vor. Es muss daher später mittels der Inferenz extrahiert werden. Deshalb ist diese Klasse vor dem Inferenzschritt leer.

2.2.3.2. Inferenz von Baustein-relativen Regeln

Als Grundlage für die Inferenz wurden, wie vorhergehend beschrieben, bei der Modellierung der Objekte der Baustein-Hierarchie entsprechende Individuen mit den Individuen der Parameter-Klasse mittels der Relationen verknüpft. Hierbei bildet sich in den Fällen, in denen implizit Baustein-relative Regeln vorliegen, eine längere Kette von Verknüpfungen, die nicht explizit modelliert wurde. Das Beispiel aus BILD 8 und BILD 9 zeigt diesen Fall: das Individuum „BST_Magnettorquer“ ist mit dem Individuum „15MHz“ verknüpft über die Relation „istStrahlungsempfindlichFür“. Genauso ist das Individuum „BST_TMUndTC_GLST“ mit dem Individuum „15MHz“ durch die Relation „strahlt“ verknüpft. Diese Kette wird in BILD 11 links durch die blauen Verbindungen dargestellt.

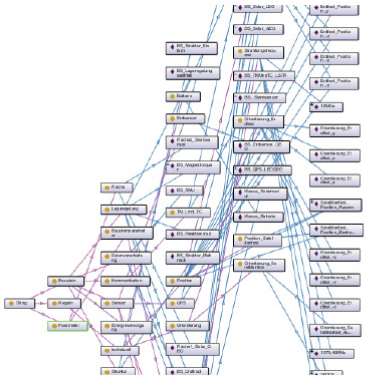

BILD 10.Darstellung eines Ausschnitts der gesamten Ontologie

Das Inferenzwerkzeug (FaCT++) bildet explizit diese Verknüpfungsketten durch logisches Schließen aus. Dabei werden automatisch passende Relationen aus der Object-Property-Hierarchie herausgesucht, welche die gefundenen Verknüpfungsketten „überspannen“. Diese Relationen werden dann als Baustein-relative Regeln interpretiert und als neue Object Properties den Individuen der Baustein-Hierarchie zugeordnet. Dieser Zusammenhang wird in BILD 11 links dargestellt.

Ein wenig komplizierter ist dieser Prozess, wenn sich die Regel nicht auf die Individuen der Baustein-Hierarchie selbst bezieht, sondern auf untergeordnete Individuen, die den Baustein-Individuen gehören (verknüpft durch die Relationen „besitzt“ oder „gehört-zu“). Dieser Fall liegt vor, wenn zum Beispiel eine Regel nur eine Seite eines Bau-

steins betrifft, weil dort beispielsweise ein Sensor platziert ist. BILD 11 rechts zeigt ein Beispiel hierfür: es gibt einen Baustein, der mit einer Strahlungsquelle im sichtbaren Spektrum (hier: Halogenstrahler) ausgestattet ist, der von der Baustein-Seitenfläche 1 aus strahlt. Ein anderer Baustein beinhaltet einen Sternensensor auf seiner Seitenfläche 2, der durch die Lichtquelle geblendet wird. Hierbei werden zuerst die neuen Relationen „blendet“ und „hatStrahlungVonFläche“ gefolgert und dann basierend darauf die Relation „hatBlendungsbedingungZu“ als neue Baustein-relative Regel abgeleitet.

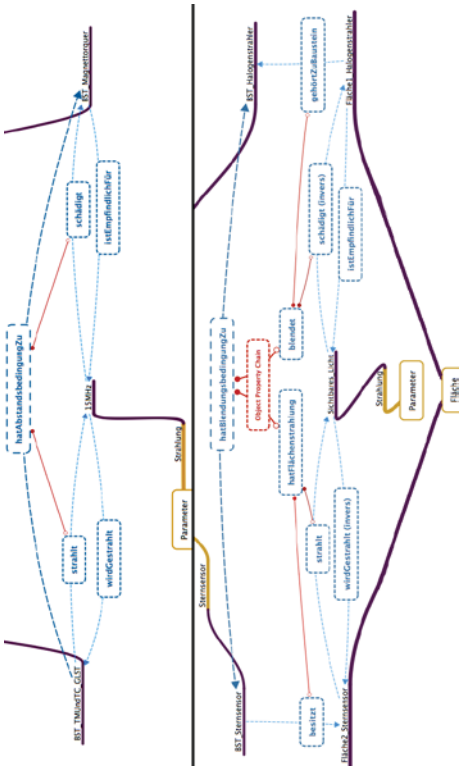

BILD 11.Links: Durch Inferenz wird die neue Relation „hatAbstandsbedingung“ identifiziert und den Individuen der Bausteinhierarchie zugeordnet. Rechts: Durch mehrfache Inferenz wird die Relation „hatBlendungsbedingungZu“, bezüglich einer spezifischen Seitenfläche der jeweiligen Bausteine, erzeugt.

In unserer exemplarischen Ontologie kommen vier Baustein-relative Regeln vor (siehe Beispiele in BILD 12):

1) **MindestabstandsBedingung**: Zwei Bausteine müssen bei der Konfigurationsgenerierung einen Mindestabstand voneinander einhalten. Es geht genau hervor, welcher konkrete Baustein von welchen anderen Bausteinen einen Abstand einhalten muss. Der konkrete nötige Abstand wird in einem Nachbearbeitungsschritt aus der Intensität der Strahlungsquelle und dem kritischen Schwellenwert des empfindlichen Bausteins berechnet.
2) **Abstandsempfehlung**: Zwei Bausteine sollen bei der Konfigurationsgenerierung einen möglichst großen Abstand voneinander einhalten.
3) **OrientierungsbedingungMit**: Zwei Bausteine müssen bei der Konfigurationsgenerierung so platziert werden, dass eine definierte Seitenfläche der Bausteine in die gleiche Richtung zeigt.
4) **OrientierungsbedingungNichtGegen**: (= hatBlendungsbedingungZu): Zwei Bausteine müssen bei der Konfigurationsgenerierung so platziert werden, dass die definierten Seitenflächen der Bausteine nicht im Sichtkegel der jeweils anderen Fläche liegen. (Die konkreten BS-Eigenschaften geben dabei den Öffnungswinkel des Sichtkegels vor.)

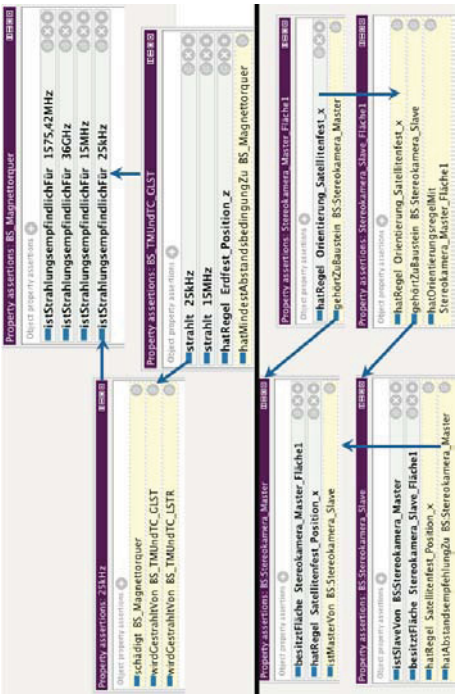

BILD 12.Inferenz von Eigenschaften und Regeln. Die automatisch abgeleiteten Eigenschaften und Regeln sind gelb unterlegt.

2.3. Konfigurationsgenerierung

Zur Generierung von (Satelliten-/Baustein-)Konfigurationen werden die ausgewählten Bausteine gemäß der zutreffenden expliziten und impliziten Regeln zu einem System zusammengefügt. In diesem vollautomatischen Schritt wird ein evolutionärer Optimierungsalgorithmus eingesetzt. Basierend auf den ausgewählten Bausteinen und den zugehörigen Regeln wird versucht, eine möglichst optimale Konfiguration zu generieren.

2.3.1. Lösen des Optimierungsproblems: Anordnen der Bausteine

Grundsätzlich bieten sich hier eine Vielzahl von Methoden an. Diese können in zwei Klassen eingeordnet werden: *deterministische Methoden* (z.B. Hill-Climbing-Varianten) und *nicht-deterministische Methoden* (z.B. Monte-Carlo-Varianten). Die deterministischen Methoden haben den Vorteil, dass sie zielgerichtet einem Gradienten folgen, um möglichst schnell ein Optimum zu finden. Ergänzt werden sie durch deterministische und nicht-deterministische Verfahren, um aus lokalen Optima entkommen zu können (z.B. Stochastic Hill Climbing, oder Simulated Annealing). Die deterministischen Methoden setzen jedoch voraus, dass die Optimierungsfunktion stetig differenzierbar ist. Aufgrund der sehr komplexen Optimierungsfunktion, die aus den extrahierten Regeln generiert wird, kann die stetige Differenzierbarkeit nicht garantiert werden. Deshalb wird in unserem Konzept auf die Monte-Carlo-Methoden

gesetzt. Der bekannteste Vertreter hierbei ist der Evolutionäre Algorithmus (siehe z.B.: [4]). Hierbei wird eine *Population* von verschiedenen Konfigurationen (*Individuen*) zufällig erzeugt. Diese werden dann iterativ anhand der Optimierungsfunktion bewertet und *mutiert*, bis die festgelegten Abbruchkriterien erreicht werden. Die zur Optimierung eingesetzten Schritte sind in den folgenden Abschnitten näher beleuchtet.

2.3.1.1. Framework und Konzept

Der evolutionäre Algorithmus ist dem biologischen Evolutions-Prinzip nachempfunden. Das Grundprinzip geht von einer Population von Individuen aus. Individuen innerhalb einer Population werden mittels einer *Fitnessfunktion* bewertet. Lediglich Individuen mit guter Bewertung können ihre Eigenschaften durch Mutation an die nächste Generation weitergeben. Durch iteratives Ausführen von Bewertung, Selektion und Mutation einer Population, dann *Generation* genannt, entwickelt sich die Population in Richtung Optimum. Die Anzahl der Individuen in der Population, die Art der Initialisierung, die Art der Mutation sowie die Strenge der Selektion bestimmen hierbei die Abwägung zwischen einer explorativen, breit gestreuten Abdeckung des Suchraums möglicher Konfigurationen einerseits, und einer zielgerichteten lokalen Optimierung einzelner Kandidaten andererseits.

2.3.1.2. Initialisierung

Bevor die iterative Optimierung beginnt, wird eine Startpopulation an Individuen erzeugt. Jedes Individuum besteht dabei aus einer Anordnung der vorgegebenen Satellitenbausteine. Diese kann rein zufällig erzeugt werden – oder es kann beim Erzeugen der Startkonfigurationen schon auf einige Regeln geachtet werden, um die Anzahl der benötigten Generationen zu reduzieren. So ist es zum Beispiel sinnvoll, alle Bausteine in einem zusammenhängenden Cluster zu positionieren. Dazu wird ein beliebiger Baustein in der Mitte des Konfigurationsraumes mit einer zufälligen Orientierung platziert. Ausgehend von diesem Baustein werden alle weiteren Bausteine aus der Liste rekursiv an einen direkten Nachbarn der bereits platzierten Bausteine angefügt. Diese Vorgehensweise ermöglicht es, ungültige Konfigurationen mit verstreuten Bausteinen am Anfang auszuschließen. Diese müssen während der Optimierung nicht betrachtet werden, und der Algorithmus konvergiert schneller zu einem (pareto-)optimalen Ergebnis. Neben der Cluster-Regel werden bei der Erzeugung von Startkonfigurationen keine weiteren Regeln überprüft, um eine zu starke Einschränkung des Suchraums vor der eigentlichen Optimierung zu vermeiden.

2.3.1.3. Fitnessfunktion

Die Fitnessfunktion zur Bewertung der Qualität der einzelnen Individuen ist essenziell für die Funktionsweise des evolutionären Algorithmus. Die hier verwendete Fitnessfunktion F wurde aus den in den vorigen Abschnitten gewonnenen Regeln abgeleitet und ist in Gleichung (1) dargestellt.

$$(1)\quad F = \alpha \cdot f_{Cluster} + \beta \cdot f_{Cluster\text{-}Struktur} + \gamma \cdot f_{Fluss} + \delta \cdot f_{Verbindung} + \varepsilon \cdot f_{Rel\text{-}Abstand} + \zeta \cdot f_{Rel\text{-}Orientierung}\ \eta \cdot f_{Rel\text{-}Position} + \theta \cdot f_{Spez\text{-}Verdeckung} + \iota \cdot f_{Spez\text{-}Orientierung} + \kappa \cdot f_{Spez\text{-}Position}$$

Es ist sehr wichtig, eine ausgeglichene und nicht zu sehr einschränkende Fitnessfunktion zu definieren, um die Optimierung möglichst auf den ganzen Suchraum auszuweiten. Aus diesem Grund kann der Einfluss der einzelnen Regeln auf die Gesamtfitness F durch die Gewichte α, β, γ, δ, ε, ζ, η, θ, ι und κ variiert werden. Die einzelnen Komponenten der Fitnessfunktion lassen sich durch entsprechende Fitness-Tests ermitteln.

2.3.1.4. Fitness-Tests

Für viele der Komponenten der Fitnessfunktion lässt sich die Qualität nicht direkt durch eine mathematische Funktion überprüfen. Für diese Komponenten müssen separate Tests entwickelt werden, die dann für jedes Mitglied der Population die jeweilige Komponente bewerten. Im Folgenden sind diese Tests aufgelistet und einige exemplarisch näher beleuchtet.

- *Cluster-Test*:
  Die Bedingung, dass Satelliten stets zusammenhängend sein müssen, wird mit dem *Flooding-Algorithmus* überprüft, der auch zur Informationsverteilung in Netzwerken eingesetzt wird. Der Algorithmus markiert ausgehend von einem Baustein iterativ alle Nachbarbausteine, bis keine direkten Nachbarn mehr gefunden werden. Noch nicht markierte Bausteine sind in einem neuen Cluster angeordnet und werden ebenfalls untersucht. Der Algorithmus endet, wenn alle Bausteine markiert sind, und gibt die Anzahl der Cluster zurück.
- *Fluss-Test*:
  Der Fluss-Test wird eingesetzt, um eine ausreichende Übertragung von Energie, Daten und Wärme zwischen BS zu gewährleisten. Dazu kann auf den *Ford-Fulkerson-Algorithmus* zurück gegriffen werden, der mittels eines gewichteten Graphen den maximalen Fluss ermittelt.
- *Verbindungs-Test*:
  Zur Vermeidung ungültiger BS-Verbindungen und zur Erzeugung kompakter Satellitenkonfiguration werden die ungültigen und gültigen Verbindungen zwischen allen Bausteinen gezählt und über die Anzahl an Bausteinflächen zu einem Fitnesswert normiert.
- *Abstands-Test*:
  Beim Abstands-Test werden die Manhattan-Distanzen zwischen zwei Bausteinen berechnet und an Hand der minimalen und maximalen Schwellenwerte die Fitness bestimmt.
- *Orientierungs-Test*:
  Zur Bestimmung der BS-spezifischen und BS-relativen Orientierungsfitness werden für jeden Baustein die Winkeldifferenzen für alle drei Raumachsen aufsummiert und normiert.
- *Verdeckungs-Test*:
  Zur Vermeidung von Abschattungen (z.B. eines Sensor-BS) kann für alle Bausteinflächen mit Sensoren ein kegelförmiger Bereich mit einem Öffnungswinkel zwischen 0° und 180° definiert werden. Innerhalb dieses Bereichs dürfen sich keine anderen Bausteine befinden. Zur Überprüfung dieser Verdeckungsbedingung wird für jeden BS die Entfernung entlang der Kegel-Mittelachse zum Kegel-Ursprung berechnet. Mit Hilfe dieser Entfernung und des Öffnungswinkels kann der Kegel-Radius ermittelt werden. Ist der orthogonale Abstand des BS zur Kegel-Mittelachse kleiner als der Kegel-Radius, so liegt der BS im Sichtbereich und die Fitness verschlechtert sich entsprechend.

2.3.1.5. Mutation

Mutationen werden eingesetzt, um die Individuen einer Population so zu verändern, dass Optima gefunden, lokale Optima aber überwunden werden können. Es werden die Mutationsoperationen *Move*, *Rotate* und *Swap* eingesetzt, um neue Individuen zu erzeugen.

Mit Hilfe der *Move*-Operation wird ein einzelner Baustein von seiner aktuellen Position an eine ausgewählte freie Position verschoben. Zur Bestimmung dieser freien Position wird zufällig ein Satelliten-BS ausgewählt und für eine zufällige direkte Nachbar-Position überprüft, ob diese frei ist und ob der zu verschiebende Baustein auf Grund seiner Größe und Orientierung dort platziert werden kann. Sollte das nicht der Fall sein, wird die nächste Nachbar-Position überprüft. Findet sich keine Position für den gewählten BS, werden so lange die Nachbar-Positionen von anderen Satelliten-BS überprüft, bis eine geeignete Position gefunden wurde.

Die *Rotate*-Operation rotiert einen Baustein um eine zufällig ausgewählte Achse im Konfigurationsraum, unter Berücksichtigung seiner Abmessungen. Eine Rotation kann nur dann durchgeführt werden wenn die entsprechenden Positionen an der Ziel-Orientierung nicht belegt sind. Für den Fall, dass eine Rotation nicht möglich ist, wird eine andere Achse bzw. ein anderer Rotationswinkel gewählt. Der Rotationswinkel kann auf Grund der quadratischen Grundform und der gitterförmigen Anordnung der Bausteine in 90°-Schritten variiert werden.

Das Vertauschen zweier Bausteine, mit dem Ziel die Satellitenkonfiguration zu mischen ohne die äußere Geometrie zu verändern, ermöglicht die *Swap*-Operation. Ein zufällig ausgewählter Baustein kann mit einem zweiten zufällig bestimmten Baustein getauscht werden, wenn beide exakt dieselbe Größe haben. Ist der erste Baustein kleiner als der zweite, wird nach einem alternativen Tauschkandidaten gesucht. Für den Fall, dass der erste Baustein größer ist als der zweite, wird versucht, durch Hinzufügen von Nachbarbausteinen die für das Tauschen benötige Größe zu erhalten. Sollte dies nicht möglich sein, werden zufällig zwei neue Kandidaten bestimmt.

Der eigentliche Ablauf einer Mutationsphase stellt sich wie folgt dar: Pro Mutationsphase können mehrere Mutationsoperationen auf ein Individuum angewendet werden. Die Anzahl der Mutationen pro Individuum ergibt sich zufällig aus der Anzahl an Bausteinen *n*, wobei mindestens eine und höchstens *n* Mutationsoperationen durchgeführt werden. Für jede Mutation wird zufällig bestimmt, auf welchen Baustein sie angewendet wird. Anschließend wird über eine Zufallsvariable *op* mit Werten im Intervall [0, 1] und den entsprechend dimensionierten Schwellenwerten *prop_move*, *prop_rotate* und *prop_swap* ermittelt, mit welcher der drei Operationen *Move*, *Rotate* oder *Swap* mutiert wird. Gute Resultate konnten mit Werten von jeweils 1/3 erzielt werden. Es ist zu beachten, dass die Mutationsoperationen möglichst allgemein gehalten wurden und die Auswahl der Bausteine sowie die Anzahl der Mutationen probabilistisch variiert werden, um den Suchraum nicht zu sehr einzuschränken. Damit erhöht sich die Wahrscheinlichkeit, dass der Algorithmus in Richtung eines globalen Optimums konvergiert.

Zusätzlich zu den drei Mutationsoperationen, die der zielgerichteten lokalen Optimierung dienen, da sie neue Individuen aus den vorhandenen Individuen erstellen, gibt es noch sehr stark zufällige Operationen, die völlig neue Individuen erzeugen, um die Exploration des Suchraums zu verbessern. Eine konstante Anzahl an Individuen pro Population während der gesamten Optimierung wird dadurch erreicht, dass in jedem Evolutionsschritt die am schlechtesten bewertete Hälfte der Population gelöscht wird. Die ersten 80% der nun fehlenden Individuen werden durch Mutationen der besten Individuen aufgefüllt. Die noch verbleibende Lücke wird nun mit zufällig neu erzeugten Individuen geschlossen. Das Verhältnis zwischen Exploration und lokaler Optimierung kann mit Hilfe eines Gewichts bestimmt werden. Es ist möglich, das Verhältnis im Verlaufe der Optimierung zu verschieben. So bietet es sich an, zu Beginn den Suchraum stärker zu explorieren und die Explorationsquote mit steigender Fitness der Individuen nach und nach zu reduzieren.

2.3.1.6. Abbruchkriterium

Das Abbruchkriterium definiert, ab wann eine Lösung als akzeptabel angesehen wird bzw. keine Verbesserung mehr zu erwarten ist. Neben einer maximalen Anzahl an Iterationen i_{max} zur zeitlichen Begrenzung wird die Fitness der besten *n* Individuen geprüft. Da die Bewertung der Fitness unabhängig von der Anzahl der Bausteine ist, kann über einen Schwellenwert f_{min} ermittelt werden, ob die Fitness klein bzw. gut genug ist, um ein (pareto-)optimales Ergebnis zu erhalten. Bei unseren Tests mit dem Satelliten aus BILD 13 hat sich gezeigt, dass i_{max} auf 500 Iterationen begrenzt werden kann. Zudem kann der Algorithmus vorzeitig beendet werden, wenn die besten 10% der Individuen mit einer Fitness kleiner als $f_{min} = 1$ bewertet wurden.

Beim Beenden des Optimierungsvorgangs erhält der Konstrukteur eine Reihe von Vorschlägen für die Satellitenkonfiguration jeweils zusammen mit den Bewertungen der einzelnen Zwangsbedingungen. Hier besteht nun die Möglichkeit, diese zu übernehmen, sie als Ausgangspunkt für eine weitere Optimierung zu benutzen oder sie manuell zu prüfen und zu überarbeiten.

2.3.1.7. Laufzeitbetrachtung

Zur Laufzeitmessung des Optimierungsalgorithmus wurde ein Rechner mit einem Intel Core i7-920 mit 2,67 GHz (vier Prozessorkerne mit Hyperthreading) verwendet. Der Algorithmus wurde sowohl bei der Berechnung der Fitness-Tests als auch bei der Durchführung der Mutationsoperationen mit der Programmierschnittstelle *OpenMP* [8] parallelisiert. Die durchschnittliche Laufzeit zur Berechnung einer Konfiguration im 3D-Raum verdoppelt sich etwa alle fünf Bausteine mit gleichzeitig ansteigender Anzahl an Regeln.

# BS (# Regeln):	15 (33)	20 (44)	25 (48)
Laufzeit:	7,9 s	24,0 s	51,9 s

3. FAZIT

In den ersten Abschnitten dieses Beitrags wurde beschrieben, wie ein definierter Bausteinkatalog mittels der Ontologiesprache *OWL* mit dem Werkzeug *Protégé* mo-

delliert werden kann. Hierfür ist es zunächst nötig, eine Bausteinhierarchie zu entwerfen und die benötigten Regeln für die Bausteine zu identifizieren. Nach der Modellierung kann dann mit Hilfe von Inferenzwerkzeugen wie dem hier verwendeten *FaCT++* implizites Wissen automatisch abgeleitet werden. Abschließend kann das Wissen des Modells exportiert werden, wobei die exakten Daten von der Anwendung abhängen: Sollen Bausteine für einen Satelliten ausgewählt werden, werden Bausteintypen mit ihren Fähigkeiten und Anforderungen exportiert. Sollen Bausteine zu einer Konfiguration kombiniert werden, müssen die konkreten Bausteininstanzen exportiert werden, zusammen mit den sie betreffenden Regeln. Anschließend wurde ein Konzept basierend auf einem evolutionären Algorithmus dargelegt, mit dessen Hilfe die ausgewählten Bausteine unter Berücksichtigung der identifizierten Regeln zu einer Konfiguration zusammengefügt werden können. Außerdem wurden Ergebnisse einer exemplarischen Implementierung vorgestellt.

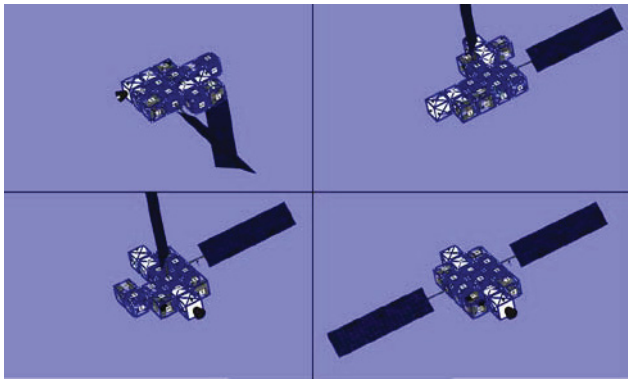

BILD 13. Optimierung der Konfiguration durch den evolutionären Algorithmus. Von links oben nach rechts unten: 1, 100, 200 Iterationen, Endergebnis nach 319 Iterationen.

4. AUSBLICK

Zwei Planungsschritte rahmen den beschriebenen Schritt der Konfigurationserzeugung ein: die halbautomatische Auswahl von Bausteinen für die Konfiguration, sowie die Erstellung eines Plans, um eine Startkonfiguration in eine Zielkonfiguration umzubauen.

1) *Bausteinauswahl*: Bei der Konfigurationsgenerierung wurde ein zweistufiges Verfahren vorgeschlagen. Im ersten Schritt werden die für die Mission benötigten Bausteine ausgewählt. Hierbei werden sowohl die Missionsbedingungen als auch etwaige Einschränkungen der Bausteine selbst berücksichtigt. Anschließend wird die Versorgung mit Ressourcen (Elektrizität, Antrieb) und die Struktur dimensioniert. Auch dieses Vorgehen kann softwareunterstützt halbautomatisch ablaufen. Gelöst wird das dadurch definierte Constraint-Satisfaction-Problem durch geeignete Algorithmen wie *Constraint-based Local Search*, *Backtracking* oder *Tabu Search*. Der zweite Schritt folgt dann wie im Beitrag beschrieben.
2) *Rekonfiguration*: Um ein Raumfahrtsystem tatsächlich in eine generierte Konfiguration zu überführen, muss der Umbau geplant werden. Die Erstellung eines Sequenzplanes, der einzelne Arbeitsschritte enthält, um von einer Start- zu einer gewünschten Ziel-Konfiguration zu gelangen, stellt ein Planungsproblem dar. Es bietet sich an, dieses mittels der *Action Description Language* (ADL) [2] und der *Planning Domain Definition* Language (PDDL) [3] zu modellieren. Das Planungsproblem wird über seinen Start- und seinen Zielzustand definiert. Hier werden jeweils diejenigen Bedingungen angegeben, die in diesen Zuständen erfüllt sein müssen bzw. nicht erfüllt sein dürfen. Die einzelnen möglichen Aktionen werden durch Vor- und Nachbedingungen definiert. Das so beschriebene Problem kann dann mit verschiedenen Planungsalgorithmen und Heuristiken wie zum Beispiel dem LAMA-Planer [5], einem der Sieger der International Planning Competition der International Conference on Automated Planning and Scheduling 2008, gelöst werden.

5. DANKSAGUNG

Gefördert durch die Raumfahrt-Agentur des Deutschen Zentrums für Luft- und Raumfahrt e.V. mit Mitteln des Bundesministeriums für Wirtschaft und Technologie aufgrund eines Beschlusses des Deutschen Bundestages unter dem Förderkennzeichen 50RA1007.

6. REFERENZEN

[1] D. Tsarkov, I. Horrocks: *FaCT++ Description Logic Reasoner: System Description.* In: U, Furbach, N. Shankar (Hrsg.), Proceedings of the Third International Joint Conference on Automated Reasoning, Lecture Notes in Computer Science 4130:292–297, 2006.

[2] E. Pednault: *ADL: Exploring the Middle Ground Between STRIPS and the Situation Calculus*. In: R. Brachman, H. Levesque, R. Reiter (Hrsg.), Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning, S. 324–332, 1989.

[3] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, D. Wilkins: *PDDL – The Planning Domain Definition Language*. Yale Center for Computational Vision and Control, Tech Report CVC TR-98-003/DCS TR-1165, 1998.

[4] S. Russell, P. Norvig: *Artificial Intelligence: A Modern Approach*. Prentice Hall Inc., New Jersey, 2003.

[5] S. Richter, M. Westphal: *The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks*. In: A. Darwiche, S. Zilberstein (Hrsg.), Journal of Artificial Intelligence Research 39:127–177, 2010.

[6] J. Gennari, M. Musen, R. Fergerson, W. Grosso, M. Crubézy, H. Eriksson, N. Noy, S. Tu: *The Evolution of Protégé: An Environment for Knowledge-Based Systems Development*. In: E. Motta, S. Wiedenbeck (Hrsg.), International Journal of Human-Computer Studies 58(1):89–123, 2003.

[7] P. Patel-Schneider, P. Hayes, I. Horrocks (Hrsg.): *Web Ontology Language* (OWL). W3C Recommendation, http://www.w3.org/TR/2004/REC-owl-semantics-20040210/, 2004.

[8] OpenMP Architecture Review Board: *OpenMP Application Program Interface*, Version 3.1. http://www.openmp.org/mp-documents/OpenMP3.1.pdf, Juli 2011.

[9] J. Weise, K. Brieß, A. Adomeit, H.-G. Reimerdes, M. Göller, R. Dillmann, D. Nölke: Ein Bausteinkonzept für Wartungsfreundliche und Rekonfigurierbare Satelliten; Deutscher Luft- und Raumfahrtkongress, 2011