Reduktion von Entwicklungskosten durch wissensbasierte Konfiguration (Ein Fallbeispiel)

Gerhard Fleischanderl [x], Gerhard E. Friedrich [*x], Alois Haselböck [x],
Herwig Schreiner [x], Markus Stumptner [+]

[x]Siemens AG Österreich
Entwicklungszentrum für Elektronik
Erdberger Lände 26, A-1030 Wien, Austria
e-mail: firstname.{middle initial.}lastname@siemens.at

[*]Universität Klagenfurt
Institut für Informationstechnologie
Lehrstuhl für Produktionsinformatik
Universitätsstrasse 65-67, A-9020 Klagenfurt, Austria
e-mail: friedrich@ifi.uni-klu.ac.at

[+]Technische Universität Wien
Institut für Informationssysteme
Paniglgasse 16, A-1040 Wien, Austria
e-mail: mst@dbai.tuwien.ac.at

Abstract

Die wissensbasierte Konfiguration ist mit der Zielsetzung angetreten, die Entwicklung und Wartung von Konfigurationssystemen zu beschleunigen. Gleichzeitig sollen die Kosten dieser Aufgaben gesenkt und die Qualität der Benutzerinteraktion im Konfigurationsprozeß durch erhöhte Flexibilität und Erklärungsfähigkeit gesteigert werden.

Aus dem Blickwinkel von Entwicklungsabteilungen steht die Methode der wissensbasierten Konfiguration in Konkurrenz zu etablierten Software-Entwicklungsmethoden vor allem aus den Bereichen der objektorientierten Programmierung und der Datenbank-Applikationssprachen.

Für den technischen Entscheidungsträger stellt sich daher die Frage, welcher Ansatz in einem konkreten Entwicklungsprojekt zu bevorzugen ist. Wir berichten anhand von im produktiven Einsatz befindlichen Konfigurationssystemen, daß der wissensbasierte Ansatz für umfangreiche Konfigurationsprobleme tragfähig ist und gewinnbringend eingesetzt werden kann. Die Verwendung dieser Methode ermöglicht erhebliche Kostenreduktionen der Entwicklungsaufwendungen sowie eine gleichzeitige Erhöhung der Funktionalität und Flexibilität.

1 Motivation aus betriebswirtschaftlicher Sicht

Unternehmungen stehen von zwei Seiten unter steigendem Druck. Einerseits ist auch bei vielen technischen Produkten ein zunehmender Preisverfall festzustellen, andererseits sind erhebliche Kostensteigerungen zu verzeichnen. Als eine mögliche Gegenstrategie wird eine verstärkte Ausrichtung auf die Kundenzufriedenheit vorgeschlagen, ein Vorgehen, das vermehrt zur Entwicklung von anpassungsfähigen Produkten führt. Um konkurrenzfähige Preise erzielen zu können, muß diese Varianz der Produkte aber im gesamten Geschäftsprozeß (von Verkauf über Produktion bis zur Wartung) beherrscht werden. Die Konfiguration von Produkten nimmt hier eine Schlüsselposition ein. Automatisierung von Konfigurationsaufgaben, die aufgrund der geforderten Komplexität als unrentabel galten, beginnen sich unter den geänderten Rahmenbedingungen zu rechnen.

Diese Zunahme von Varianz und Komplexität der Aufgabe führt jedoch zu einer Steigerung der Entwicklungsaufwendungen von Konfiguratoren. Besonders in Bereichen, in denen Großsysteme zu konfigurieren sind, die Bereiche Kommunikationstechnik und Eisenbahnsicherungstechnik seien hier stellvertretend erwähnt, nehmen die Aufwendungen zur Entwicklung von Konfiguratoren signifikante

Größen an. Bezogen auf die wissensbasierte Konfiguration müssen im wesentlichen zwei Fragen beantwortet werden. Können wir auch für Großsysteme wissensbasierte Ansätze einsetzen, und ist es uns möglich, durch diesen Einsatz tatsächlich nennenswerte Einsparungen im Entwicklungsprozeß zu erreichen? Im folgenden werden wir anhand eines Fallbeispieles aus dem Bereich Telefonvermittlungssysteme diese Fragen beantworten.

2 Die Herausforderung: Konfiguration von Telefonvermittlungssystemen

Die Basisaufgabe eines Telefonvermittlungssystems wie EWSD (Elektronisches Wählsystem Digital) ist es, Telefonverbindungen durchzuschalten. Es werden jedoch zahlreiche zusätzliche Services geboten, wie z.B. ISDN, Online- und Broadband-Services, Videotelefonie, und Videokonferenzschaltungen. Lokale EWSD-Zentralen versorgen bis zu 600.000 Teilnehmer.

Eine große EWSD-Konfiguration besteht aus ungefähr 200 Racks (Schränken), 1.000 Rahmen, 30.000 Modulen, 10.000 Kabeln und 1.000 anderen Einheiten. In der Wissensbasis sind ca. 20 Typen von Racks, 50 Typen von Rahmen, 200 Typen von Modulen, 50 Typen von Kabeln und 50 Typen von anderen „organisatorischen“ Einheiten abgebildet. Diese geringe Anzahl an unterschiedlichen Typen konnte dadurch erreicht werden, daß Komponenten durch das Einstellen von Parametern im Konfigurationsprozeß selbst adaptiert werden können.

Module werden in Baugruppenrahmen gesteckt, welche wiederum in Racks montiert sind. Die Module und Rahmen sind durch Kabel miteinander verbunden und bilden eine Netzwerktopologie, die auf der hierarchischen physikalischen Struktur aufgesetzt ist. Zusätzlich gibt es (externe) PCs, Ventilatoren und andere Einrichtungen, die montiert und mit dem Hauptteil des Systems verbunden werden müssen.

Die Spezifikation von gültigen Konfigurationen wird durch Konfigurationsconstraints bestimmt. Typische Beispiele sind:

- Der erste Anschluß jedes Rahmens ist für Stromversorgungsmodule vorgesehen.
- Der Energieverbrauch aller in einem Rahmen montierten Module muß kleiner oder gleich der maximalen Energieabgabe des entsprechenden Stromversorgungsmoduls sein.
- Analoge und digitale Module dürfen in einem Rahmen nicht gemischt werden.
- Ist ein Durchschaltemodul in einem Rahmen gesteckt, dann muß mindestens einer der Anschlüsse contr1 oder contr2 auch mit einer Komponente des Typs DigitalController verbunden sein.

Basierend auf der Komponentenbibliothek und den Konfigurationsconstraints werden gültige Konfigurationen während eines Konfigurationsprozesses generiert. Dieser Prozeß, den wir in der Langversion näher beschreiben, wird durch den Konfigurator LAVA (Lokale Aufbauplanung und Verkabelung für AOSA) unterstützt und automatisiert. LAVA ist auf Basis des domänen-unabhängigen Konfigurationswerkzeugs COCOS (Configuration by Constraint Satisfaction [SHF94, FFH$^+$95]) implementiert, welches die effiziente Entwicklung von wissensbasierten Konfiguratoren für komponentenorientierte Systeme ermöglicht.

3 Wissensrepräsentation

Die Beispielconstraints in Abschnitt 2 zeigen wichtige Anforderungen an die Form von Wissensbasen, z. B. gibt es eine Oder-Verknüpfung im Folgerungsteil der Implikation. Weiters werden Quantoren und Gleichheit in der Formulierung der Konfigurationsconstraints von den Konfigurationsexperten verwendet. Geht man davon aus, daß dieses Wissen möglichst explizit modelliert werden soll, um Wartungsfreundlichkeit und Erklärungsfähigkeit zu gewährleisten, so müssen diese Ausdrucksmittel in der Wissensrepräsentationssprache zur Verfügung gestellt werden. Wissensrepräsentation und Schlußfolgern in LAVA entsprechen im wesentlichen den Ansätzen wie sie in [SHF94, FFH$^+$95] beschrieben wurden und stützen sich auf verschiedene Konfigurationsprojekte in den Bereichen Tonstudiosysteme, Kommunikationstechnik und Eisenbahnsicherungsanlagen.

Unser Ziel war es daher, um breite Anwendbarkeit zu erreichen, ein Wissensrepräsentationsschema zu entwickeln, das so allgemein ist, daß es für verschiedenste Anwendungsdomänen verwendet werden kann und eine explizite und direkte Modellierung dieser Domänen xuläßt. Das Ergebnis dieses Prozesses ist die Sprache ConTalk[1], welche auf der Basis unserer Erfahrung bei der Entwicklung verschiedenster Konfigurationsanwendungen in den Bereichen Telekommunikation, Kontrollsysteme und Audiostudios entstanden ist (insgesamt mehr als 100 Personenjahre).

[1] ConTalk ist eine Weiterentwicklung von LCON [SHF94], die entwickelt wurde, um den Anforderungen einer Produktivumgebung gerecht zu werden. Da ConTalk als Smalltalk-Sprachschnittstelle realisiert ist, basiert die Syntax auf der von Smalltalk.

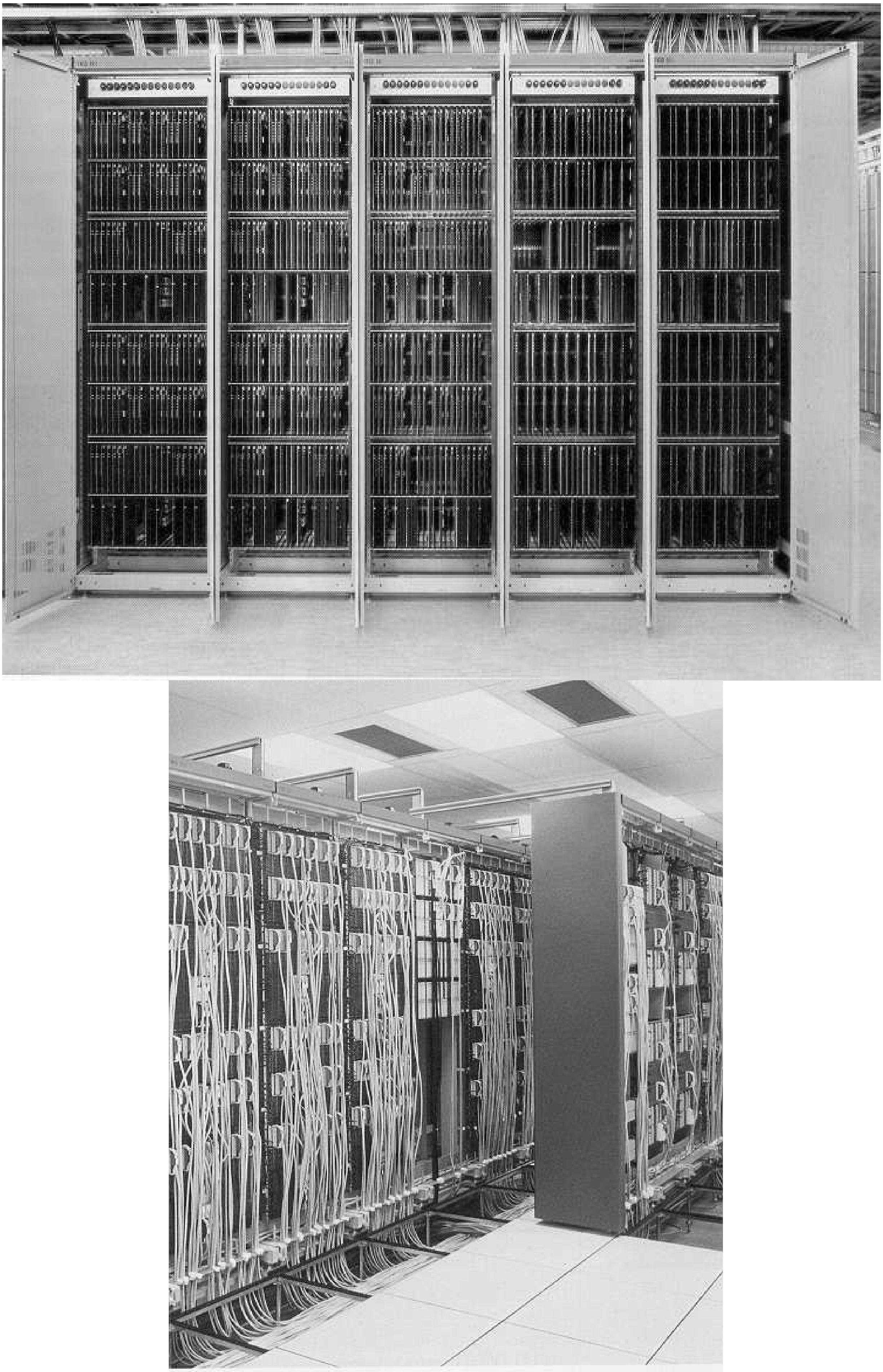


Abbildung 1: Vorder- und Rückansicht eines EWSD-Telefonvermittlungssystems

Komponentenbibliothek

Alle diese Problembereiche haben gemeinsam, daß die Komponententypen, welche zum Bau einer Konfiguration verwendet werden könnten, in der Komponentenbibliothek beschrieben sind. Da im voraus nicht bekannt ist, wieviele Komponenten für eine bestimmte Konfiguration benötigt werden, müssen die Komponenten während des Konfigurationsprozesses nach Bedarf generiert werden.

Die Komponententypen sind in einer Vererbungshierarchie organisiert. Die Wurzel der Hierarchie ist der Typ *Component*. Die Blätter des Baumes sind die konkreten Komponententypen, die für die zu konfigurierenden Systeme erhältlich sind. Eigenschaften und Verhalten eines Komponententyps werden durch seine Attribute, Ports und Constraints definiert.

Attribute

Jeder Komponententyp besitzt eine Menge von *Attribut*definitionen (die Namen, Typ und wahlweise einen Defaultwert festlegen). Beispiele sind:

```
Frame attribute:'name' type: String const: 'F_DLU_A'.
Frame attribute: 'width' type: Number.
```

Ports

Ports dienen zum Herstellen von Verbindungen zwischen zwei Komponenten. Eine Portdefinition für eine Komponente besteht aus dem Namen des Ports, seinem Typ[2] und seinem Wertebereich (der Menge von Komponententypen, die an diesen Port angeschlossen werden könnten). Beispiele von Portdefinitionen sind:

```
Rack port: 's01' type: Port mustConn: #(F_DLU_A).
Rack port: 's02' type: Port mayConn: #(Frame).
```

In diesem Beispiel hat ein Rack zwei Ports: s01, an den ein Rahmen des Typs F_DLU_A angeschlossen sein muß, und s02, an den ein beliebiger Rahmen (F_DLU_A oder F_DLU_B) angeschlossen sein muß. Ports sind immer paarweise miteinander verbunden (oder werden explizit auf 'unverbunden' gesetzt).

Constraints

Constraints stellen ein gebräuchliches Repräsentationsschema für Konfigurationsprobleme dar. Sie sind ein natürlicher Weg, um Kompatibilitätswissen zwischen Komponenten auszudrücken. Außerdem erleichtern ihre Eigenschaften die Wartung von Wissensbasen:

- Derselbe Constraint kann sowohl für das Generieren als auch für das Prüfen einer Konfiguration verwendet werden.
- Constraints bieten reiche Repräsentations- und Argumentationsmöglichkeiten, die so erweitert werden können, daß komplexeres Konfigurationswissen natürlich auszudrücken ist.
- Einfachheit und Deklarativität des zugrunde liegenden Constraintmodells erlauben die Definition einer klaren Semantik.
- Constraints unterstützen eine klare Trennung zwischen strategischem Wissen und Problemwissen sowie Präferenzen, da Strategien und Präferenzen mittels Ordnungen von Variablen und Werten ausgedrückt werden können (z.B. einen Rahmen von links nach rechts auffüllen).

In COCOS werden Constraints immer lokal bei einer einzigen Komponente definiert, von wo aus sie mit Nachbarkomponenten in Beziehung treten und dadurch durch die gesamte Konfiguration navigiert werden kann. Hier ein Beispiel eines Constraints, der mehr als eine Komponente betrifft:

> „Die zwei an den Ports s01 und s02 eines jeden Racks angeschlossenen Rahmens müssen die gleiche Weite haben."

```
(Rack connComp: 's01') width
eq: (Rack connComp: 's02') width.
```

aComp connComp: aPort ist ein „Navigationsausdruck" und referenziert die mit dem Port aPort der Komponente aComp verbundene Komponente.

Bestimmte Spezifikationen haben die Form von Requirements. Solche „Ressourcenconstraints" können in ConTalk durch das Definieren von Constraints auf Variablenmengen formuliert werden, wie dies im folgenden Beispiel dargestellt wird:

> „Der Stromverbrauch aller Module in einem Rahmen (dies sind alle Module, die mit den Portset allSwitchingSlots verbunden sind), muß kleiner gleich dem Stromoutput des entsprechenden Stromversorgungsmoduls sein. Beachten Sie, daß das Stromversorgungsmodul in jedem Rahmen am ersten Anschluß slot1 angebracht ist."

```
((Frame ports: 'allSwitchingSlots')[3]
```

[2] Ports in COCOS sind selbst Objekte, die Attribute haben können und in eine Typenhierarchie eingebettet sind.

[3] allSwitchingSlots steht für ein Set von Portnamen,

```
Sum: [ :aPort | aPort connComp
          powerConsumed ]4 )
 le: (Frame connComp: 'slot1')
          powerSupplied.
```

Eine andere wichtige Art von Ausdrücken in ConTalk sind Quantoren, die z.B. eine Bedingung definieren, die sich über eine Menge von Ports erstreckt. Grundsätzlich gibt es den All- und den Existenzquantor, wobei der letztere im folgenden Beispiel demonstriert wird:

> *„Gibt es einen Anschluß für Schaltmodule in einem Rahmen, der mit einem Schaltmodul verbunden ist, dann muß mindestens einer der Anschlüsse contr1 und contr2 auch mit einer Komponente des Typs DigitalController verbunden sein.“*

```
((Frame ports: 'allSwitchingSlots')
          Exists: [ :aPort | aPort isConn ])
   Then: [ (Frame ports: #(contr1 contr2))
          Exists: [ :ctrPort | ctrPort isConn
          And:[ctrPort connComp isA:
                    DigitalController ] ].
```

Um ConTalk-Constraints leichter lesbar zu machen, kann man anwenderdefinierte Prädikate (eine Art Makros) spezifizieren, welche die Definition eines domänenspezifischen Vokabulars erlauben. Diese anwenderdefinierten Prädikate können in jedem Constraint verwendet werden, um die Formeln zu vereinfachen.

Ein wesentlicher Unterschied zwischen ConTalk und herkömmlichen Constraintsystemen ist die Darstellung des Konfigurationsproblems als *Generatives* CSP (GCSP). Ein generatives CSP erweitert das Constraintnetzwerk während der Suche um zusätzliche Variablen und Constraints, indem bei Bedarf neue Komponenten erzeugt werden. Auf diese Weise können große Systeme konfiguriert werden, ohne künstlich im voraus alle zu erwartenden Komponenten spezifizieren zu müssen.

4 Resultate

LAVA ist nun seit mehr als einem Jahr in Anwendung und ist vollständig in die EWSD-Logistikkette integriert, was die Implementierung verschiedener Interfaces erforderte, angefangen von Schnittstellen zu Anwenderprogrammen, die auf Großrechnersystemen laufen, bis hin zu Excel-Sheets. LAVA selbst läuft derzeit unter Windows NT 4.0. Da wir Smalltalk (Visualworks) als Implementierungssprache verwenden, sind COCOS und LAVA unmittelbar auch für alle Systeme verfügbar, für die Visualworks erhältlich ist, z.B. Sun Solaris und viele Unix-Plattformen.

Beim Anwender von LAVA arbeiten im Schnitt sechs Personen an Konfigurierungsaufgaben. Diese Benutzer widmen ungefähr 20% Prozent ihrer Arbeitszeit dem Konfigurieren von Anlagen. (Andere Tätigkeiten sind Organisationsaufgaben, Planungen für Ressourcen, die die Abteilung disponiert, und Besprechungen bei den Endkunden der Anlagen.)

Die Arbeitsweise mit dem Konfigurator ist interaktiv, also kein Batchdurchlauf. Die Analyse der Arbeitsprozesse ergab, daß diese Arbeitsweise für die Modularität der Anlagen und für den Arbeitsfortschritt am geeignetsten ist, weil die Benutzer zwischen den einzelnen Teilschritten Überprüfungen durchführen und Korrekturen anbringen können. In einem repräsentativen Jahr wurden ungefähr 90 (vorwiegend kleine) Anlagen neu konfiguriert, 60 (vorwiegend große) Anlagen erweitert und 140 Anlagen umgebaut. In den genannten Zahlen sind nachträgliche Umkonfigurierungen aufgrund geänderter Anforderungen nicht enthalten. Bei der Bearbeitung einer Anlage sind des öfteren noch Klärungen erforderlich, wodurch sich die Bearbeitungszeit ausdehnt.

Eine durchschnittliche EWSD-Konfiguration besteht aus ungefähr 25 Racks, 125 Rahmen, 3.500 Modulen, 1,500 Kabeln und 250 anderen Einheiten.

4.1 Einsparungen im Projektierungsprozeß

Der finanzielle Aspekt eines Entwicklungsprojekts ist für die Bewertung des Erfolges entscheidend. Im Falle von LAVA machte sich die gesamte Investition bereits innerhalb des ersten Betriebsjahres bezahlt.

Zusätzlich zu den Vorteilen, die im Konfigurationsprozeß realisiert werden konnten, wurde die Qualität der Konfigurationsergebnisse verbessert und damit die Kosten in jenen Abteilungen reduziert, welche die Ergebnisse dieses Prozesses nutzen. Während des Ladens alter EWSD-Beschreibungen wurden zahlreiche Fehler entdeckt, z.B. falsche Längencodes von Kabeln, was zu Problemen in der Montage und Produktion führte. Diese aus den Qualitätsverbesserungen des Konfigurationsprozesses resultierenden Kostenreduktionen sind in der vorhin beschriebenen Kalkulation der Kosteneinsparung noch nicht enthalten und würden die Beurteilung des Erfolgs von LAVA noch verbessern.

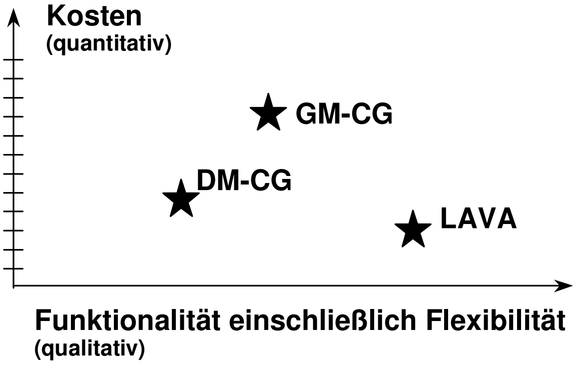
---

die alle jene Anschlüsse bezeichnen, wo Schaltmodule angeschlossen werden können.

[4] In ConTalk definieren die sogenannten, in eckige Klammern gefaßten *blocks,* namenlose Ausdrücke, ähnlich den Lambda-Expressions in LISP. Z.B. evaluiert der Ausdruck

```
aSet Sum:  [ :x | <block body>]
```

den <block body> für alle Elemente der Menge aSet; die Laufvariable x wird der Reihe nach an alle Elemente von aSet gebunden.



Zusätzliche Funktionalität von LAVA im Vergleich zu GM-CG:

- **Interaktive Konfiguration**
- **Graphisches Benutzerinterface**
- **Erklärungen**
- **Automatische Kabelkonfiguration**
- **Wiederverwendung von Baugruppen**
- **Ausdrucksstarke Beschreibungssprache**
- **Erweiterung von bestehenden Konfigurationen**
- **Unterstützung des Urladens von Altbeständen**

DM- & GM-CG: Konfiguratoren, tabellenbasiert, prozedurales Wissen
LAVA: Konfigurator, constraintbasiert, deklaratives Wissen

Abbildung 2: Vergleich von Konfiguratoren

4.2 Ersparnis in der Entwicklung

Obwohl für den Erfolg eines Projekts der Gewinn ein dominanter Faktor ist, ist für die zukünftige Entwicklung von Konfiguratoren die interessantere Frage, ob die verwendete Technologie Vorteile verglichen mit anderen Ansätzen mit sich bringt. Wir können auf eine Historie zurückblicken, in der LAVA die dritte Anwendung war, die im Bereich der automatisierten Konfiguration von EWSD-Systemen entwickelt wurde.

Abbildung 2 zeigt einen Vergleich zwischen LAVA und den beiden früheren Konfiguratoren DM-CG und GM-CG in Bezug auf Kosten und Funktionalität (einschließlich der Flexibilität für den Anwender während des Konfigurationsvorganges).

Im Gegensatz zum Kostenvergleich ist ein aussagekräftiger Vergleich von Funktionalität in unserem Fall nur qualitativ möglich. DM-CG vs. GM-CG waren spezielle EWSD-Konfiguratoren (im Gegensatz zum generellen Konfigurationswerkzeug COCOS), wobei GM-CG eine bedeutend höhere Flexibilität in der Wartung von Konfigurationswissen bietet. GM-CG erlaubt die Modellierung von Konfigurationswissen mittels propositionaler Logik erweitert um Defaulteigenschaften. Komplexe Constraints, die Verbindungen, Attributwerte, Prädikate und Quantifizierung verwenden, konnten jedoch nicht formuliert werden.

Die Hauptvorteile von LAVA gegenüber GM-CG sind in Abbildung 2 aufgelistet. Ein zusätzliches Ergebnis des LAVA-Projekts ist ein erweitertes Konfigurationswerkzeug, das für die Realisierung weiterer Konfiguratoren verwendbar ist.

Wenn man alle diese zusätzlichen Ergebnisse betrachtet, ist es bemerkenswert, daß verglichen mit GM-CG eine Kostenreduktion von 66% erzielt werden konnte. Selbst wenn wir auch den für das vorhergehende COCOS Forschungsprojekt verwendeten Aufwand mit einbeziehen (einschließlich der Forschung in theoretischen Themen der Constraint Satisfaction), erzielen wir noch immer eine Kostenreduktion von ungefähr 50%.

Im Vergleich zu DM-CG ist die Funktionalität von DM-CG ein Bruchteil von jener von LAVA, nichtsdestotrotz sind die Kosten von LAVA um 30% geringer als jene von DM-CG.

4.3 Vorteile in der Wartung

Die Wartungskosten hängen von der Häufigkeit und vom Umfang der Änderungen in der Komponentenbibliothek ab. Zum gegenwärtigen Zeitpunkt ist eine solide quantitative Metrik für das Messen dieser Menge nicht verfügbar. Im Gegensatz zu direkten Veränderungen der Komponentenbibliothek kann die Funktionalität des Systems auch durch andere Anforderungen verändert werden, z.B. Unterstützung neuer Fileformate. Wir betrachten jedoch nur Veränderungen der Komponentenbibliothek, da diese Veränderungen die für uns interessanten sind, sowohl aus wirtschaftlicher Sicht als auch für die Arbeiten auf dem Gebiet der Konfigurationstechniken.

Der Vergleich anhand konkreter Änderungswünsche zeigt, daß bei DM-CG und GM-CG ein gewisser Prozentsatz an kleineren Änderungswünschen zwar mittels Anpassung der Tabelleninformation befriedigt werden konnten, daß aber zahlreiche Veränderungen des Produktkataloges Änderungen des DM-CG- und des GM-CG-Programmcodes auslösten, was zu größeren Wartungsaufwendungen führte. In LAVA konnten alle diese Wünsche durch kleinere Veränderungen in der Wissensbasis erfüllt werden. Außerdem ist die LAVA-Wissensbasis kompakter. Änderungen in der Komponentenbibliothek, welche eine Änderung an vielen Punkten in DM-CG und GM-CG auslösen, verursachen nur einen einzigen Änderungspunkt in LAVA.

Weiters verglichen wir die Fähigkeiten von COCOS mit einem anderen wissensbasierten Konfigurationssystem, das sich im Zustand der praktischen Anwendung befindet. Dieses System verfolgt den regelbasierten Ansatz [BO89], wobei durch framebasierte Repräsentationsfähigkeiten Verbesserungen eingebracht wurden. Wiederum zeigte unsere Bewertung ähnliche Ergebnisse. Einfache Konfigurationsconstraints neigen zu umfangreichen Beschreibungen in der Regelbasis. Durch die Repräsentationsformalismen von COCOS konnte eine mehr als fünffache Reduktion für Teile der Wissensbasis erzielt werden. Veränderungen in der Komponentenbibliothek lösen häufig viele Veränderungen in der Wissensbasis des regelbasierten Systems aus, was für die untersuchten Fälle in COCOS vermieden werden könnte.

4.4 Organisatorische Vorteile

LAVA macht das Wissen über EWSD-Konfiguration explizit. Daher können neue Mitarbeiter im Vertrieb die essentiellen Eigenschaften von EWSD schneller und in einer besser strukturierten Form lernen, und das Konfigurationswissen bleibt erhalten, so daß zu einem späteren Zeitpunkt direkt darauf zugegriffen werden kann.

5 Verwandte Arbeiten

Es gibt zahlreiche vielversprechende Vorschläge (welche teilweise in wirtschaftlich sehr erfolgreichen Werkzeugen integriert sind), wie Entwicklung, Wartung und Funktionalität von Konfiguratoren verbessert werden können. Obwohl solche Werkzeuge in verschiedenen Konfigurationsdomänen anwendbar sind, hat die tatsächliche Projekterfahrung gezeigt, daß sie in Bezug auf Ausdrucksstärke und Effizienz verbessert werden müssen: Dies gilt auch für weniger komplexen Aufgaben, als wir im EWSD-Bereich antreffen.

Im industriellen Sektor sehen wir derzeit drei gängige Ansätze zum Lösen von Konfigurationsproblemen. Neben dem constraintbasierten Ansatz ist der ressourcenbasierte Ansatz [HJ91] ein verbreitetes Konzept, um verschiedenste Formen von Konfigurationswissen auszudrücken. In unserem Ansatz gehen wir über dieses Konzept hinaus, so daß komplexe Strukturbeziehungen entsprechend der Sicht des Fachexperten dargestellt werden können. Ein weiterer vielversprechender Ansatz zur Konfiguration sind Description Logics, welche von einer umfangreichen Grundlage theoretischer Forschung profitieren können. Ihre Anwendung ist in [WWB+93] für die Konfiguration von Telekommunikationssystemen beschrieben, die jedoch bedeutend kleiner als unsere Anwendungsdomäne sind. Auch bei diesem Ansatz stellt sich die Frage, wie komplexe Strukturbeziehungen aus Sicht des Anwenders möglichst einfach dargestellt werden können.

6 Resümee

Unsere Anwendung zeigt, daß ausdrucksstarke Wissensrepräsentationssprachen in umfangreichen Domänen erfolgreich verwendet werden können. Für diese Domänen trägt eine ausdrucksvolle Wissensrepräsentation wesentlich zur Verbesserung der Wartung von Konfiguratoren bei. Durch die Verwendung eines reichhaltigen Repräsentationsformalismus kann das Konfigurationswissen direkt, ohne aufgrund von fehlenden Spracheigenschaften notwendige Umwege, spezifiziert werden.

Eine wichtige Frage ist natürlich, wie zufriedenstellende Effizienz erreicht werden kann. In unserem Fall war es ausreichend, Constraint-Satisfaction-Algorithmen und eine Aufteilung nach funktionalen Gesichtspunkten auszunutzen sowie genügend Aufwand bezüglich der Verfeinerung von Algorithmen und Datenstrukturen zu betreiben. Sehr einfache Heuristiken (wie z.B. Port- und Komponententyp-Ordnungen) reichen, um die Suche zu steuern. Die Rolle der Constraints ist es, den Suchprozeß durch das Eingrenzen der Anzahl von Auswahlmöglichkeiten zu unterstützen, wobei anwendungsunabhängige Heuristiken (wie z.B. Forward checking) benutzt werden.

Wie unsere Erfahrung zeigt, gilt dies auch für andere Domänen, an denen wir arbeiten. Jedoch wäre sowohl aus anwendungsorientierter als auch aus wissenschaftlicher Sicht von Bedeutung, ein besseres Verständnis der Eigenschaften von Konfigurationsdomänen zu erlangen, die solch ein günstiges Verhalten zeigen.

Aus dem Blickwinkel der wissenschaftlichen KI-Gemeinschaft glauben wir, einen Schritt vorwärts gemacht zu haben, indem wir gezeigt haben, daß Constraint Satisfaction eine vielversprechende Technik für reale umfangreiche Konfigurationsanwendungen ist.

Aus dem Blickwinkel von Software-Entwicklern, die persönlich für das Erreichen von Kosten-, Zeit- und Qualitätsanforderungen verantwortlich sind, ist es notwendig, bereits vom Beginn des Projekts an die richtigen Technologien und Werkzeuge zu verwenden, da Kosten- und Zeitbeschränkungen typischerweise zu stark sind, um die Technologie während eines Projektes zu ändern. Aus diesem Grund werden in Entwicklungsprojekten neue Technologien nur mit größter Vorsicht angenommen. Wir haben gezeigt, daß constraintbasierte Techniken erfolgreich in produktiven Systemen angewendet werden können, wobei wir Systeme konfigurieren, die zu den komplexesten derzeit in Betrieb stehenden elektronischen Systemen gehören.

Die Anwendung dieser Technik aus dem Bereich der wissensbasierten Konfiguration hat sich in Bezug auf die Senkung von Kosten und Durchlaufzeiten in der Softwareentwicklung und Wartung positiv ausgewirkt. Gleichzeitig konnte die Funktionalität von Konfiguratoren erhöht werden. Wir haben damit zum Ziel beigetragen, eine in der KI anerkannte Technologie wie die Constraint Satisfaction auf eine breitere Anwendungsbasis in der Softwareentwicklung zu stellen.

Bibliographie

BO89 Virginia E. Barker and Dennis E. O'Connor. Expert systems for configuration at Digital: XCON and beyond. *Comm. ACM*, 32 (3): 298-318, 1989.

FFH+95 Gerhard Fleischanderl, Gerhard Friedrich, Alois Haselböck, and Markus Stumptner. Knowledge-based Configuration of Switching Systems. In *Proc. 15th Intl. Switching Symposium (ISS-95),* pp. 158-162, Berlin, Germany, April 1995.

HJ91 M. Heinrich and E.W. Jüngst. A resource-based paradigm for the configuring of technical systems from modular components. In *Proc. 7th CAIA,* pages 257-264, February 1991.

SHF94 M. Stumptner, A. Haselböck, and G. Friedrich. COCOS - A Tool for Constraint-Based, Dynamic Configuration. In *Proceedings of the 10th IEEE Conference on Artificial Intelligence Applications (CAIA),* pages 373-380, San Antonio, Texas, March 1994.

WWB+93 Jon R. Wright, Elia S. Weixelbaum, Karen Brown, Gregg T. Vesonder, Stephen R. Palmer, Jay I. Berman, and Harry G. Moore. A Knowledge-Based Configurator that Supports Sales, Engineering, and Manufacturing at AT&T Network Systems. In *Proc. 5th IAAI,* AAAI Press, 1993.