

# Der .NET Drive

*Dr. r r o b t*  
*Chief Software Architect, Triamec AG*

**Damit die Automatisierungstechnik mit der Entwicklung der allgemeinen Software-Technik Schritt halten kann, müssen Hard- und Software als einheitliche Komponenten angeboten werden, die den Werkzeugen der allgemeinen Software-Technik zugänglich sind. Der Fokus muss von einem Feldbus-zentrierten Denken hin zu Schnittstellen für die Highlevel-Software geführt werden. Das *Triamec Automation and Motion (TAM)* Konzept präsentiert hierzu eine innovative Lösung. Es bietet die direkte Einbindung der Servo-Drives von *Triamec* in das .NET-Framework von Microsoft, eingebaute *WebService*-Fähigkeit inklusive Authentisierung und Verschlüsselung, sowie die Programmierfähigkeit der Drives in *C#/Tama*.**

Wer die Entwicklung der Automatisierungstechnik analysiert, stellt fest, dass sich der PC in Maschinensteuerungen unbestreitbar durchsetzt. Das ist clever, erlaubt es doch den Rückgriff auf Elektronik, die auch in Consumer-Anwendungen und somit in viel grösseren Stückzahlen zum Einsatz kommt. Wer diesen Hebel einsetzt, kann einen Preisvorteil schaffen und an den Anwender weitergeben. Kommt hinzu, dass die Automatisierungstechnik so auch an der rapiden Leistungssteigerung und Weiterentwicklung der Mainstream-Elektronik Teil hat. Und die Automatisierung ist durchaus noch hungrig nach mehr Leistung in der Elektronik. In Regelkreisen und gekoppelten Steuerungen gibt es höchste Ansprüche an die Geschwindigkeit bei der Verarbeitung und Übertragung von Daten.

Ganz anders die Situation der Software. Hier herrschen immer noch Konzepte aus den Anfängen der gut 15-jährigen SPS-Geschichte vor, denen die Anstrengungen aus der Anfangszeit anzumerken sind, als es galt, mit dedizierter Hardware der damaligen Technik Echtzeit-Lösungen zu realisieren. Dies betrifft Software-Schnittstellen, Programmiersprachen und Werkzeuge, mit denen Ingenieure bei der Einbindung und Inbetriebnahme von Motoren, Servo-Drives und Sensoren zu tun haben.

Die Komplexität von Automaten und Anlagen wächst stetig und verlangt eine umfassende Modellierung und saubere Abbildung der Systeme in Software, doch die dazu verfügbaren Ausdrucksmittel sind unzureichend: Die Programmiersprachen der SPS-Welt müssen schlicht als antiquiert bezeichnet werden. Für Inbetriebnahme und Konfiguration neigen Hersteller dazu, durchaus leistungsstarke Tools und ergonomisch gestaltete Wizzards anzubieten, diese bleiben aber isolierte Einzellösungen. Vielfach bieten Hersteller selbst komplette Entwicklungsumgebungen an und haben dabei keine Chance, über die Zeit mit der Entwicklung der allgemeinen Software-

Technik Schritt halten zu können.

Maschinen sind hingegen heterogen, was offene Systeme verlangt. Ein erfolgreicher Komponentenanbieter berücksichtigt die Situation seiner Kunden, die sich auf der nächsten Stufe der Wertschöpfung bewegen. Sie erarbeiten durch die geschickte Auswahl und Integration von verschiedenartigen Komponenten Alleinstellungsmerkmale, die über diejenigen der Komponenten hinausgehen. Nur so können sie sich vor ihre Konkurrenten setzen. Was für Hardware und Mechanik selbstverständlich ist, muss auch in der Software gelten. Ein Komponentenanbieter mit monolithischen, geschlossenen Lösungen und dem Anspruch zur Gesamtlösung ist hier kontraproduktiv. Vermeintlich bietet er dem Integrator Komfort, tatsächlich verbaut er ihm wichtige Möglichkeiten.

## *Ein Hebel für die Software*

Nun ist man zunächst versucht, die technischen Unzulänglichkeiten der Entwicklungswerkzeuge als zweitrangig gegenüber der Leistungsfähigkeit des damit erstellten Produkts anzusehen. Aber nicht nur, dass man vielen Lösungen die Grenzen der Werkzeuge ansieht, mit denen sie erstellt wurden. Wichtiger ist die Tatsache, dass der Anteil der Software bei der Entwicklung mechatronischer Maschinen 50% und mehr der Kosten beträgt. Da ist es durchaus erlaubt, die etablierten Entwicklungskonzepte in Frage zu stellen.

Warum es nicht genauso machen wie bei der Hardware? Also Werkzeuge und Standards aus der allgemeinen Software-Technik einsetzen. Die Automatisierungstechnik soll sich auf ihre originären Softwareanteile konzentrieren, die sie bereitstellt und die in der allgemeinen IT-Welt nicht zu finden sind. Wenn es ihr gelingt, ihre Systeme an diese Werkzeuge heranzuführen, dann kann hier eine ähnliche Hebelwirkung wie bei der Hardware genutzt werden.

Nehmen wir zum Beispiel die Programmiersprachen. Wenn man einen Drive in einer modernen Hochsprache wie C# oder Java programmieren könnte, stünden sofort für viele Aufgaben Werkzeuge zur Verfügung: Integrierte Entwicklungsumgebung, UML-Modellierung, Versionierung, Datenbankbindung, GUI-Design, Build-Prozess, Test, Deployment, Lizenzierung, Upgrading, Dokumentation und Metrik-Analyse des Sourcecodes, Projektplanung, Schulung etc. Und wohlgedemert bei jeder dieser Aufgaben nicht eine vom Komponentenanbieter vorgewählte Lösung, sondern jeweils eine Vielzahl, aus der der Integrator frei wählt.

Das klingt einsichtig, bedarf aber noch einer weiteren Analyse, nämlich dessen, was die originären Aufgaben von Automatisierungs-Software sind. Aus einer IT-Sicht leistet Automatisierungs-Software die Anbindung industrieller Peripheriegeräte sowie deren isochrone Regelung. Zum ersten Teil gehört, Aktuatoren zu kommandieren und Sensoren auszulesen. Sämtliche Geräte müssen in Betrieb genommen und parametrisiert, teilweise auch optimiert werden. Das kann, wie z.B. bei den Reglern motorischer Achsen, eine nicht-triviale Aufgabe sein. Der zweite Teil, die isochrone Datenverarbeitung, ist anzutreffen in den Regelkreisen eines Antriebs oder in der Signalverarbeitung eines Sensors, in der Kopplung von motorischen Achsen, sowie in der zeitlich definierten Reaktion auf Ereignisse. Hier sind die Anforderungen gross, und hier liegt der Ressourcen hunger bei Datenverarbeitung und -transport begründet.

### ***Die Trennung von "schnell" und "isochron"***

Es ist wichtig, den feinen Unterschied zwischen "schnell" und "isochron" zu rekapitulieren. Wenn in einer Fahrsequenz eine Achse von A nach B fährt, zum Stillstand kommt, und dann weiter von B nach C, dann soll die zweite Bewegung zwar "schnell" an die erste anschliessen, um keine wertvolle Prozesszeit zu verlieren, aber sie muss nicht "isochron" sein. Das heisst, es entsteht keine Funktionsbeeinträchtigung oder gar Gefährdung, wenn die zweite Bahn einmal nicht innerhalb einer definierten Zeit beginnt. Wenn hingegen zwei Achsen miteinander gekoppelt fahren, müssen sie "isochron", also in gleichen, zyklischen Zeitabständen, ihre Bahndaten austauschen. Auch schon wenn eine Achse auf einen Endschalter reagiert, muss das zwar nicht zyklisch, aber in definierter Zeit geschehen, was hier ebenfalls als isochron bezeichnet sei.

Das "isochrone" Rechnen ist Hardware-nah und eng mit

den Regelkreisen verbunden. Die Anforderungen wachsen hier mit der benötigten lokalen Regelgüte. Das "schnelle" Rechnen hat hingegen mit der logischen Steuerung der Maschine zu tun. Die Herausforderung hier ist, die wachsende Komplexität der Gesamtmaschine zu beherrschen.

Eine typische SPS-Lösung macht diese Unterscheidung nicht. Sie rechnet sowohl "isochrone" als auch "schnelle" Aufgaben. Neben Bahnplanung und Regelkreisen wird sie auch für die logischen Abläufe eingesetzt. Letztere sind aber besser in der Highlevel-Software auf dem PC untergebracht, wo es adäquatere Möglichkeiten gibt, die steigende Komplexität zu beherrschen.

Wenn man mechatronische Maschinen charakterisiert, findet man typischerweise nur wenige bis einige "isochrone" Anforderungen. Sie finden sich in der Regelung und Kopplung der Antriebs-Achsen, sowie in wenigen digitalen und analogen Ein- und Ausgängen. "Schnelle" Anforderungen sind dagegen bei einem grossen Teil der Ein- und Ausgänge zu finden.

Die "isochronen" Anforderungen bei Achsen sind hart. Wie L. Kucera aufzeigt<sup>1</sup>, ist es von entscheidender Bedeutung, Latenz- und Abtastzeit in der Regelung und Kopplung von Achsen zu minimieren. Die Regelkreise müssen daher nahe bei den Achsen, sprich in den Servo-Drives, sein, und dürfen nicht über den Kommunikations-Link geschlossen werden.

Die Achskopplung, die wegen der räumlichen Trennung zwangsläufig über den Kommunikations-Link geschlossen wird, verlangt viel Bandbreite und ein effizientes Übertragungsprotokoll.

### ***Der richtige Systemschnitt***

Der sich hier abzeichnende Systemaufbau wird noch durch eine grundlegende Beobachtung zu den Feldbussen gestützt. Es soll hier die These vertreten werden, dass diese Kommunikations-Links der Automatisierungstechnik zu Unrecht im Mittelpunkt der Betrachtung stehen. Die verschiedenen Allianzen und Standards, für die mit Eifer geworben wird, versprechen dem Anwender Unabhängigkeit bei der Gerätewahl und Universalität bei der Anbindung.

Weder das eine noch das andere bewahrheitet sich in der Praxis. Man denke beispielsweise an das Unterfangen, bei einer bestehenden Maschine den Servo-Drive-Hersteller

<sup>1</sup> "Regelung hochdynamischer Antriebssysteme mit ausgeprägten mechanischen Resonanzen", Dr. L. Kucera, Kongressvortrag SPS/IPC/Drives 2005

zu wechseln. Erstens sind Feldbusse alles andere als Plug-and-Play-fähig. Und zweitens zieht sich der Aufwand für Integration, Inbetriebnahme, und Konfiguration von der Feldbus-Parametrisierung bis weit hinauf in die Highlevel-Software. Dieser Aufwand ist in der Regel beträchtlich. Die Standardisierung schafft also bestenfalls Synergien für die Komponentenhersteller untereinander, der Situation der Integratoren hilft sie kaum. Ihnen ist viel mehr geholfen, wenn die Hardware-Komponente auch komplett mit der entsprechenden Software-Komponente angeboten wird, die sie in ihrer Highlevel-Software einbinden.

Wird bei der Diskussion um Feldbusse also der Systemschnitt an der falschen Stelle gemacht? Treten die Komponentenanbieter zu Hardware-lastig auf? Die praktischen Erfahrungen drängen diesen Eindruck leider auf. Die Mechatronik versteht aber Software als Maschinenelement, also einen wesentlichen Bestandteil. Der Systemschnitt muss daher weiter oben liegen, in der PC-seitigen Software. Hier muss der Komponenten-Charakter zu sehen sein, und der Kommunikations-Link muss dagegen bis zur Unsichtbarkeit in den Hintergrund treten.

Man stelle sich das Beispiel des Servo-Drive-Wechsels nun noch einmal vor. Mit dem Herstellerwechsel kommt, weil Teil der Systemkomponente, auch ein anderer Kommunikations-Link, d.h. im PC wird eine andere Kommunikationskarte benötigt. Das ist aber nicht dramatisch, denn für das neue Drive-System gibt es eine Programmierschnittstelle direkt für die Highlevel-Software. Die Einbindung des Motors von der Feldebene bis dorthin, welche im ersten Fall noch nötig war, entfällt jetzt. So kann man von einer echten Komponente sprechen. Von einer Herstellerabhängigkeit ist hingegen nichts zu spüren, statt dessen fallen weniger Entwicklungskosten bei der Integration an.

Wenn man sich diese Sichtweise zu eigen macht, erscheint auch das vielfach propagierte "TCP/IP bis auf die Feldebene" von zweifelhaftem Nutzen. Erstens kommt es nicht gratis, man bezahlt für die zusätzliche Hardware in jedem Teilnehmer. Zweitens beansprucht es Bandbreite und Protokoll-Overhead in einem Bereich, wo es nichts zu verschenken gibt. Und drittens ist der Anwendungsfall falsch überlegt. Was nützt der Temperatursensor mit eingebautem Webserver zur Parametrisierung? Das ist keine Komponente, sondern eine isolierte Komplettlösung. Der Sensor wird aber in einen Regelkreis mit anderen Komponenten eingebunden, und diesen möchte man parametrisieren und überwachen. Dazu muss der Sensor in

seinem Kontext präsentiert werden. Das ist die Mehrwertleistung des Integrators. Er benötigt eine Programmierschnittstelle in der Highlevel-Software. Die Präsentationsschicht gestaltet er, und zwar für die verschiedenartigsten Komponenten in einem einheitlichen Stil. Dazu verwendet er beispielsweise wiederum eine GUI-Bibliothek von einem darauf spezialisierten Anbieter.

Eine Anbindung von Bürogeräten wie Druckern oder Webcams, die gerne genannt wird, ist bereits gelöst, hierzu müssen Feldbusse nichts leisten. Vision-Systeme wiederum haben selbst hohe Ansprüche an Datentransferaten und kommen bereits mit eigenen Link-Lösungen. Wenn es also überhaupt Bedürfnisse für TCP/IP-Peripherie an einem Maschinensteuerungs-PC gibt, dann sind diese mit dem Onboard-Ethernet des PCs oder mit einer Ethernet-Karte (für unter 15€ erhältlich) leicht zufrieden zu stellen. Ein TCP/IP-Huckepack in einem Echtzeit-Link ist hingegen mit Leistungsbeschränkung in der Maschinensteuerung teuer erkaufte. Fazit dieser Überlegungen: Ethernet für den Feldbus ja, denn hier kommt wieder der Kostenvorteil allgemeiner IT-Bauteile zum Tragen, aber TCP/IP-Protokoll nein.

Ein gut durchdachter Systemaufbau macht also die richtigen Systemschnitte, um der Arbeitsteilung von Komponentenanbieter und Systemintegrator gerecht zu werden. In mechatronischen Komponenten bilden Mechanik, Elektronik und Software eine Gesamteinheit. Insbesondere bildet die Software ein tragendes Element, und ist nicht austauschbares Beiwerk. Der Schnitt wird nicht am Kommunikations-Link gemacht, sondern weiter oben in Form eines APIs für die Highlevel-Software. Der Kommunikations-Link stellt eine definierte Funktionalität, die Echtzeit-Fähigkeit, bereit, tritt aber für den Integrator völlig in den Hintergrund. Dieser wird befähigt, die Geräte verschiedener Komponentenanbieter durch deren jeweilige APIs anzusprechen. Der Kommunikations-Link ist dabei transparent.

### ***Das TAM-Konzept***

*Triamec* hat die Servo-Drive Serie *TS-100* und das *Triamec Automation and Motion (TAM)* Konzept konsequent nach dieser Philosophie entworfen. Das zugehörige Software Development Kit, das *TAM SDK*, liefert direkt eine Objekt-orientierte Hardware-Abstraktion für das .NET-Framework von Microsoft. Damit ist der Anschluss an das derzeit modernste und umfassendste Software-Framework hergestellt, das zudem in der kommenden

Windows-Generation zu einem festen Bestandteil des Betriebssystems wird. Das API bildet einen Drive vollständig mit Parametern und Methoden ab.

Es bietet zusätzlich eine Server-Fähigkeit, um über **Web-Services** auch von Remote-Clients das System ansprechen zu können. Das API macht nach aussen keine Unterscheidung zwischen lokaler und abgesetzter Benutzung. Alle Geräte und ihre Funktionen sind universell adressierbar. So können Applikationen lokal entwickelt und lediglich per Konfigurationsänderung als Remote-Clients eingesetzt werden.

Dabei wirkt wieder der Hebel durch den Rückgriff auf vorhandene, etablierte Technologie: Die **WebService-Kommunikation** kann zusätzlich **zugangsgesichert und verschlüsselt** werden. Das geschieht als Add-On per Konfiguration, das heisst der Programmierer muss sich während der Entwicklung nicht damit beschäftigen. Es kommen keine Speziallösungen zum Einsatz, sondern die aus anderen IT-Anwendungen bekannten Public-Private-Key-Techniken mit Zertifikaten.

Intern, und für den Anwender transparent, kommuniziert das **TAM API** über den Ethernet-basierten **Tria-Link** mit den Drives. Der **Tria-Link** ist ein doppelter Tokenring mit einem schlanken Protokoll, das höchste Leistung in Bezug auf Transfer und Latenz bietet. Er ist mittels einer PCI-Karte an den PC angebunden.

In einem **TAM-System** sind die "isochronen" und "schnellen" Rechenteile klar aufgeteilt. "Isochrones" Rechnen gibt es ausschliesslich in den Drives und auf der PCI-Karte, und diese kommunizieren über den **Tria-Link** isochron miteinander. Der PC ist ausschliesslich für das "schnelle" Rechnen zuständig. Das **TAM SDK** benötigt daher keine Echtzeiterweiterung des Betriebssystems. Es bleibt für den Maschinenentwickler weitgehend "aus dem

Weg". Er kommt erst in seiner Entwicklungsumgebung für das .NET-Framework damit in Berührung.

### **Programmieren in C#/Tama**

Die isochrone Rechenumgebung der Drives ist nicht allein der Firmware vorbehalten. Sie ist, und das ist industrieweit einmalig, ebenfalls aus der .NET-Entwicklungsumgebung heraus programmierbar. Programmiert wird in der Programmiersprache **Tama**, die eine Teilmenge der .NET-Sprache C# ist. Ein **Tama**-Programm ist also immer ein korrektes C#-Programm. **Tama** ist reduziert auf den prozeduralen Teil der Sprache C#, was den typischen Aufgaben im isochronen Rechnen angemessen ist. Dazu gehören Koppelfunktionen für elektronische Getriebe, Homing-Abläufe bei der Initialisierung von Achsen, oder andere isochrone Reaktionen der Maschine.

Durch die Kompatibilität zu C# lässt sich **Tama**-Code in Projekten gleich wie dieser behandeln, das heisst organisieren, editieren, versionieren, mit XML-Kommentaren versehen, und er ist Analyse-Tools für C# Code zugänglich, die z.B. Metriken extrahieren. Der C#-Compiler wird bei **Tama**-Programmen eingesetzt, um von seinen mächtigen syntaktischen und semantischen Fehleranalysen zu profitieren, die Entwicklern eine hohe Produktivität erlauben. Dazu gehört auch das integrierte Hilfe-System. Der Einarbeitungsaufwand ist so minimiert, die Lernmittel gehen weit über das hinaus, was ein Anbieter der Automatisierungstechnik sonst bieten könnte.

**Tama**-Programme werden schliesslich durch den **Tama**-Compiler von **Triamec** in ein Binärformat überführt, das in der **Tama Virtual Machine (TamaVM)** auf dem Drive ausgeführt wird. Der Code wird dort zyklisch aufgerufen, wodurch der isochrone Charakter entsteht.



Triamec AG  
Hinterbergstr. 28  
CH-6330 Cham

Tel. +41-41-747 40 40  
[www.triamec.com](http://www.triamec.com)