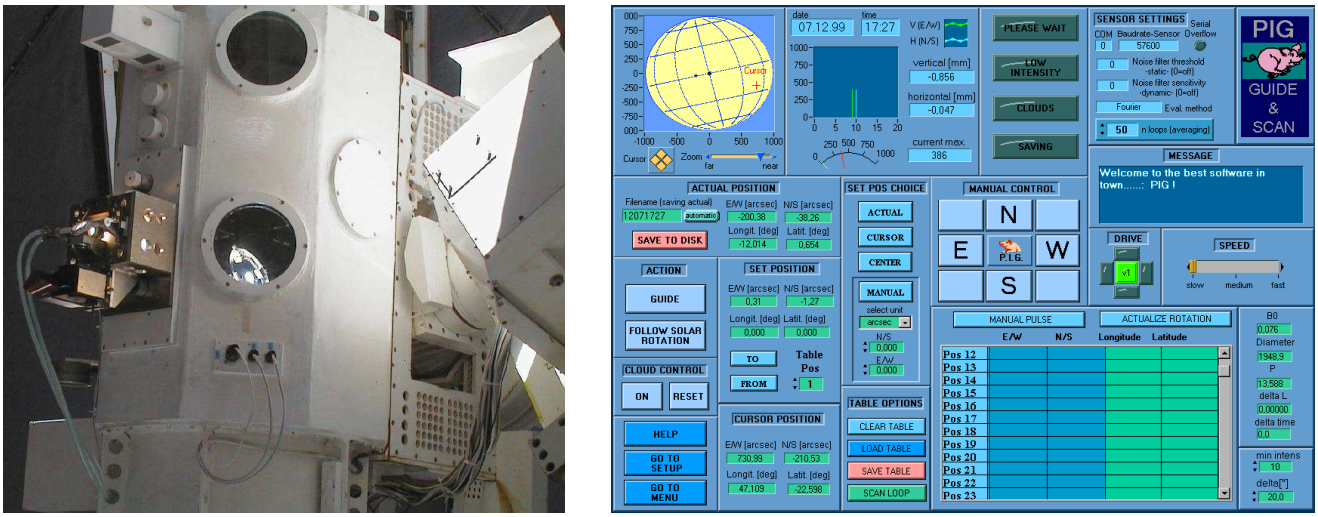
Kommando orientiertes Automatisierungssystem auf TCP-Basis – Command Server –

Van Dzung Dao[1], Gerd Küveler[1], Renzo Ramelli[2], Axel Zuber[1]

[1] Fachhochschule Wiesbaden, Am Brückweg 26, D-65428 Rüsselsheim
[2] Istituto Ricerche Solari Locarno (IRSOL), Via Patocchi, CH-6605 Locarno-Monti

Hintergrund

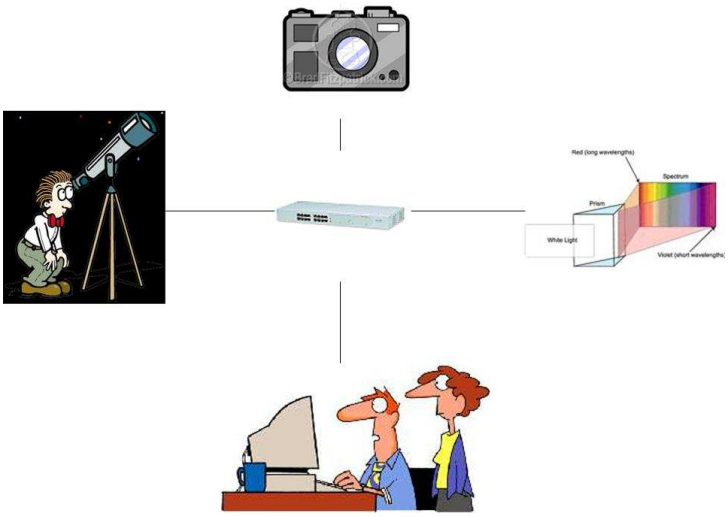
Seit mehr als zehn Jahren besteht eine Kooperation zwischen der FH Wiesbaden und dem Istituto Ricerche Solari Locarno, einem Sonnenobservatorium in der Südschweiz. Außerdem gibt es immer wieder gelegentliche Projekte mit anderen Sternwarten, z. B. auf Teneriffa. Der Beitrag der FHW besteht dabei im Bau von wissenschaftlichen Geräten und der Entwicklung von technischer Software [1]. Ein Beispiel stellt die Steuerung des Hauptteleskops dar [2 – 5].



Sensorbox links am Teleskop und grafische Benutzerschnittstelle einer Sonnenteleskop-Steuerung

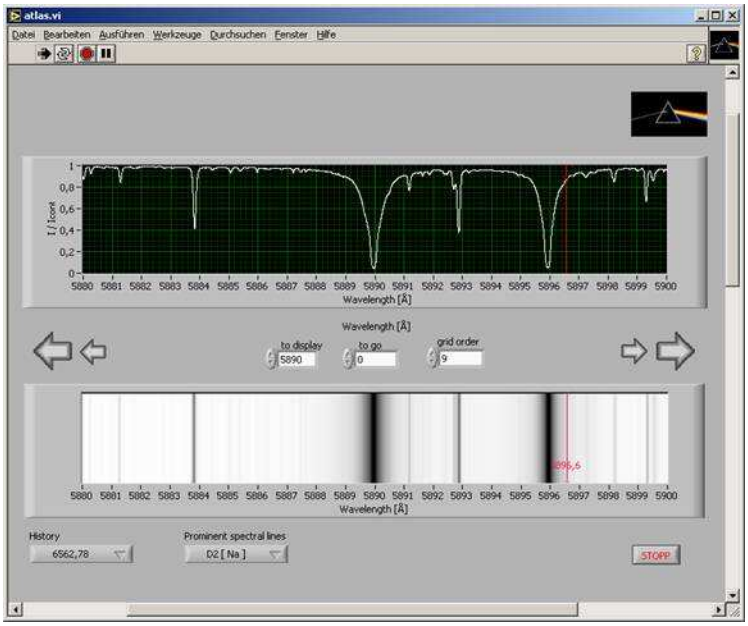
Problematik

Eine Sternwarte ist ein Forschungslabor mit vielen unabhängigen Einzelgeräten, z. B. Teleskopen, Spektrografen, CCD-Kameras, usw.



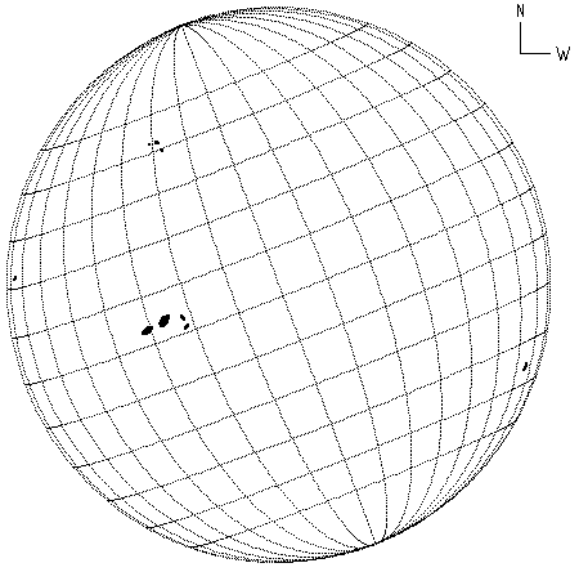
Typische Geräte einer Sternwarte

Jedes Gerät ist oder wird separat automatisiert und besitzt seine eigene grafische Oberfläche (GUI). Das Beispiel zeigt das GUI des Haupt-Sonnenspektrografen, mit dessen Hilfe der Astrophysiker einen bestimmten Spektralbereich durch Anklicken einer Wellenlänge im eingeblendeten Spektralatlas einstellen kann.



Grafische Benutzeroberfläche einer Spektrografen-Steuerung

In der Regel erfordern Messprojekte jedoch den gleichzeitigen Einsatz unterschiedlicher Geräte. Beispielsweise könnte es das Projekt erfordern, an zahlreichen Gitterpunkten im Koordinatennetz der Sonne jeweils mehrere Spektren verschiedener Spektralbereiche mit einer CCD-Kamera aufzunehmen.



Sonne mit Großkreisen

Für vollautomatische Messabläufe unter flexibler Nutzung verschiedener Geräte sind GUIs jedoch ungeeignet.

Lösung

Für diesen Fall sollten die einzelnen Geräte-Steuerprogramme skriptfähig und in der Lage sein, über eine Remote-Schnittstelle Kommandos entgegen zu nehmen. Ein Skript ist ein textbasiertes Interpreter-Programm, das es erlaubt, andere Programme in flexibler Weise aufzurufen und mit Parametern zu versorgen. Zu diesem Zweck haben wir eine besonders für Automatisierungs-Anwendungen geeignete Skriptsprache entwickelt [6, 7].

Die Skriptfähigkeit erreicht man am ehesten durch die Entkopplung von Oberfläche und eigentlicher Steuerung. Unser Lösungsansatz folgt einem 5-Stufen-Konzept für die Automatisierungs-Software:

1. layer:	gui(s)	remote task(s)	
2. layer:	command		command-server
3. layer:	dispatcher		
4. layer:	executer		
5. layer:	device control software		

Tasks und Layers des *Command Servers for Automation* (CSA)

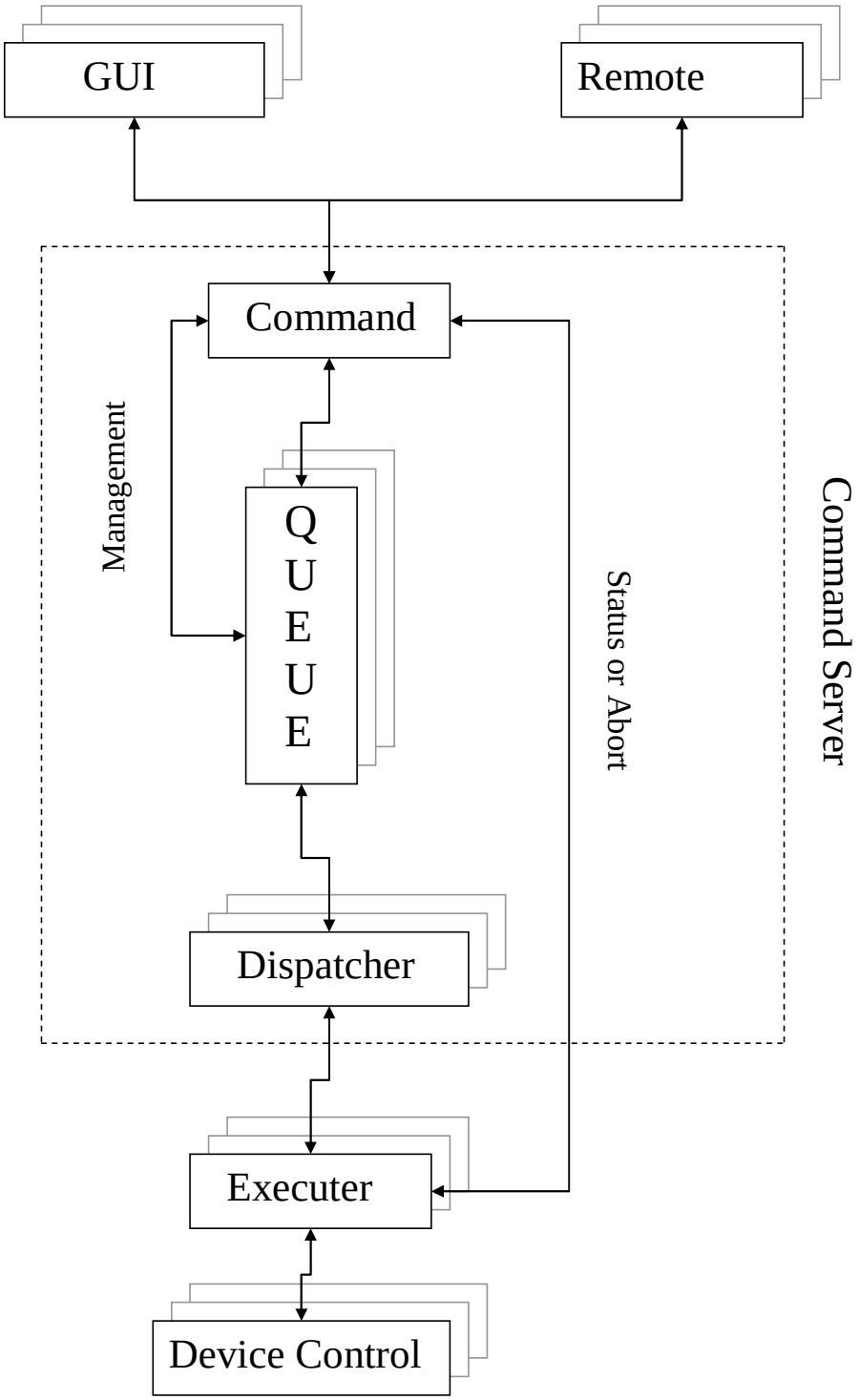
Die eigentliche Gerätesteuerung ist ganz unten angesiedelt. Der *executer* ist ein Protokollumsetzer, der nur dann benötigt wird, wenn die *device control software* nicht speziell für das hier beschriebene System geschrieben wurde (z. B. ein „altes“ oder gekauftes Programm). Alle Oberflächen, egal ob grafisch oder textorientiert, sind völlig von der Steuerungsschicht getrennt. Dazwischen sitzt der c*ommand server*, der eigentliche Kern des vorliegenden Projekts.

Zwischen den verschiedenen Programmen, die das Modell realisieren, existiert jeweils eine TCP-Schnittstelle. Über die IP-Adresse des Rechners, auf dem die Anwendung läuft, und die Port-Adresse der Anwendung selbst, zusammen als Socket bezeichnet, ist ein weltweiter Zugriff möglich. Das ist unter anderem für die Steuerung wissenschaftlicher Messungen mit womöglich weltweit verteilten Geräten über eine zentrale Prozedur von großem Interesse, insbesondere dann, wenn man die Prozedur von einem beliebigen Ort per Web-Browser oder *telnet* starten kann. Natürlich sind Sicherheitsaspekte zu beachten. Alle Kommandos und Rückmeldungen basieren auf ASCII-Strings.

Der command server

Alle „Oberflächen“ der an einem Forschungsprojekt beteiligten Steuerprogramme, egal ob GUI, textbasierte Konsole (z.B. *telnet*) oder Skript, kommunizieren nicht direkt mit ihrer zugehörigen Steuersoftware sondern ausschließlich mit dem *command server*. Jede neue Anwendung muss bei ihm mit ihrer Socketadresse angemeldet werden. Es werden dann als eigener Thread eine *Queue* (Befehlswarteschlange) und

ein *dispatcher* angelegt. Die weitere Kommunikation erfolgt dann über die zurück gelieferte *application number*.



Der *command server* leitet nur gültige Befehle an eine Anwendung weiter und sorgt dafür, dass ein Kommando erst dann weiter geleitet wird, wenn das vorherige mit Report als beendet gemeldet wurde. Ausnahmen sind Abbruch- und Status-Request-Kommandos. Das Warteschlangen-Prinzip sorgt dafür, dass ggf. auch mit mehreren Oberflächen gleichzeitig auf eine Anwendung zugegriffen werden kann, ohne dass dabei Konflikte entstehen.

Insgesamt nimmt der *command server* den einzelnen Gerätesteuerungs-Anwendungen viele Standard-Probleme ab, so dass die Erstellung neuer Anwendungen deutlich erleichtert wird. Der *command server* wurde in Java unter Windows programmiert und ist somit sehr gut auf andere Betriebssysteme portierbar.

Zusammenfassung

Das *command server* Prinzip bietet folgende Vorteile:

- Standard-Mechanismen wie Kommando-Prioritäten und Warteschlangen, Reportwesen, jederzeitige Abbruch- und Status-Kommandos müssen nicht mehr für jede Anwendung neu erfunden werden.
- Einheitliche Kommunikationsschnittstelle für alle an einer Automatisierungsaufgabe beteiligten Instanzen (z. B. verschiedene weltweit verteilte Sternwarten, die ein gemeinsames Beobachtungsprogramm durchführen).
- Einfache Kommandostruktur, die von jeder denkbaren Schnittstelle wie Konsole (z. B. *telnet),* Shell-Skript oder GUI ansprechbar ist.
- Ein zentraler Server, der alle verteilten Aktivitäten koordiniert und überwacht.
- Einheitliche Kommunikationsstruktur zwischen Kommandoebene und Geräte-Steuersoftware.
- Sicherheitsmechanismen (Passwort-Schutz, Parameterüberprüfung).
- Betriebssystem-Unhabhängigkeit des *command servers* (Windows, UNIX/Linux).

Die Anwendung des *command servers* auf andere Laboratorien oder auf industrielle Anwendungen ist möglich.

Literatur

[1] *M. Bianda, R. Ramelli, A. Feller, J.O. Stenflo, G. Küveler*: Instrumental developments at the Gregory-Coudé Telescope at IRSOL, in: F. Kneer, K. G. Puschmann, A. D. Wittmann (eds.), Proceedings of the Workshop on „Modern Solar Facilities – Advanced Solar Science“, Göttingen, Germany (2007).

[2] *Küveler, G, Wiehr, E. Thomas, D., Harzer, M., Bianda, M.Sütterlin, P., Epple, A., Weisshaar, E.:* Automatic Guiding of the Primary Image of Solar Gregory Telescopes. Solar Physics 182, 247-255(1998).

[3] *Küveler, G, Wiehr, E., Bianda, M.:* Eine sensorgestützte Computersteuerung für Sonnenteleskope. atp - Automatisierungstechnische Praxis 42(2000) Heft 7, S. 50-54.

[4] *Küveler, G., Wiehr, E., Bianda, M.:* Eine Computersteuerung und Nachführ-Automatik für Sonnenteleskope, in: Jamal, R., Jaschinski, H.: Virtuelle Instrumente in der Praxis. Begleitband zum Kongress VIP 2002. Heidelberg Hüthig 2002, S. 47-51.

[5] *Küveler, G., Wiehr, E., Bianda, M.*: Automatic Guiding of Solar Gregory Telescope. in: Kneer, F., Wiehr, E. , Wittmann, A.D. (eds.), From the Gregory-Coudé. Telescope to GREGOR: a development from past to future. Astron. Nachr. Vol. 324, No. 4, p. 308 (2003).

[6] *Küveler, G., Zuber, A.*: Eine Prozedursprache zur Steuerung vollautomatischer Prozesse. atp – Automatisierungstechnische Praxis, 46 (2004) Heft 8, S. 20-23.

[7] *Küveler, G., Zuber, A.:* Eine Skriptsprache zur Remote-Steuerung von LabVIEW-Modulen, in: Jamal, R., Jaschinski, H.: Virtuelle Instrumente in der Praxis. Begleitband zum Kongress VIP 2005. Heidelberg Hüthig 2005, S. 310-317.