Ein flexibles, selbst rekonfigurierbares Rechnersystem zur Untersuchung von Selbstorganisation in verteilten Systemen

Klaus Drechsler, Dominik Murr und Christophe Bobda

AG Selbstorganisierende Eingebettete Systeme
Technische Universität Kaiserslautern, 67655 Kaiserslautern, Deutschland
{drechsler,bobda}@informatik.uni-kl.de, dominik@murr.com

Zusammenfassung. Selbstorganisation ist eine interessante und vielversprechende Möglichkeit, um die Komplexität verteilter Systeme beherrschbar zu machen. In diesem Beitrag schlagen wir ein leistungsfähiges Rechnersystem auf Basis von rekonfigurierbarer Hardware vor, welches aufgrund seiner Flexibilität in vielen Bereichen eingesetzt werden kann. Es wird die geplante Systemarchitektur und Systemsoftware beschrieben und ein intelligentes, verteiltes Kamerasystem vorgestellt, welches wir als Anwendung mit dem vorgeschlagenen System realisieren wollen, um Selbstorganisation in verteilten Systemen näher zu untersuchen.

1 Einleitung

Aufgrund des rasanten Fortschritts in der Informationstechnologie werden zunehmend komplexe Systeme entwickelt, die aus einer Vielzahl von Einzelkomponenten bestehen. Die Integration dieser Einzelkomponenten zu einem Ganzen gestaltet sich mit zunehmender Anzahl und den dadurch möglichen und unvorhersehbaren Konfigurationsmöglichkeiten immer schwieriger.

Selbstorganisation stellt ein vielversprechendes Konzept dar, um diese Probleme in den Griff zu bekommen. In selbstorganisierenden Systemen interagieren die einzelnen Komponenten autonom und nur auf Basis lokaler Informationen miteinander, um ein gemeinsames Ziel zu erreichen. Selbstorganisation ist dann der Vorgang, bei dem sich aus den Interaktionen auf niedrigerer Ebene Muster auf globaler Ebene ergeben [2]. Hat das System dann Fähigkeiten, die die Fähigkeiten der einzelnen Komponente übersteigen, spricht man von emergentem Verhalten. Ein solches System soll nach Möglichkeit einen oder mehrere der sogenannten Selbst-X Eigenschaften besitzen. Dazu zählen beispielsweise Selbst-Schutz, Selbst-Heilung, Selbst-Konfiguration und Selbst-Optimierung. Anders ausgedrückt soll sich das System selbstständig an veränderte Bedingungen adaptieren können, um so die Verfügbarkeit und Zuverlässigkeit sicherzustellen.

Das stetige Wachstum und die Integration neuer Funktionen haben Field Programmable Gate Arrays (FPGAs) in den letzten Jahren zu einem interessanten und flexiblen Medium für die Realisierung von hochperformanter, spezialisierter Hardware gemacht. FPGAs sind frei programmierbare Logikschaltkreise, mit denen sich beliebige digitale Schaltungen realisieren lassen, die auch nachträglich

im Feld noch geändert (rekonfiguriert) werden können. Die realisierte Funktionalität des FPGAs wird durch die Konfiguration bestimmt, welche ganz oder teilweise in einem Bitstrom codiert ist und dazu verwendet wird, das FPGA zu programmieren. Mit den Kapazitäten heutiger FPGAs ist es damit möglich, so genannte selbst rekonfigurierbare Systeme zu realisieren. Ein selbst rekonfigurierbares System bezeichnet, analog zu der Auffassung in [9], ein System, das die Möglichkeit hat, seine eigene Hardwarekonfiguration zu ändern, um sich an veränderte Bedingungen zu adaptieren. Vereinfacht dargestellt besteht ein solches System aus einem festen, statischen Teil, der den restlichen, rekonfigurierbaren Teil kontrolliert und bei Bedarf reprogrammiert. Ist der Konfigurationsport des FPGA intern zugänglich, kann eine im FPGA eingebette CPU, die dann den statischen Teil darstellt, den rekonfigurierbaren Teil reprogrammieren.

Die Nutzung von FPGAs in eingebetteten System ermöglicht es, strenge Anforderungen an Platzbedarf, Rechenleistung und Flexibilität zu erfüllen. Dadurch lassen sich flexible und leistungsfähige Systeme auf kleinstem Raum realisieren. Ein Einsatzgebiet, für das eingebettete Systeme auf Basis von FPGAs aufgrund ihrer inhärenten Parallelität bestens geeignet sind, stellt die Bild- und Videoverarbeitung dar. Weitere Einsatzgebiete werden zum Beipiel in [4] beschrieben. Stattet man ein Bild- oder Videoverarbeitendes System mit einer Kommunikationsschnittstelle aus, lassen sich intelligente, verteilte Kamerasysteme entwickeln, die beispielsweise im Bereich Robotik und Personen- und Raumüberwachung eingesetzt werden können. In gewisser Weise stellt ein solches verteiltes System ein Sensornetzwerk dar. Unter einem Sensornetzwerk wird in der Literatur (z.B. in [1]) jedoch häufig der Zusammneschluss kleiner, sehr stromsparender, kostengünstiger Sensoren verstanden, die über sehr begrenzte Ressourcen (Speicher, Rechenleistung) verfügen und mit einer Batterieladung Wochen, Monate und sogar Jahre Daten (z.B. Temperatur, Helligkeit) über die Umgebung sammeln. Ein intelligentes und verteiltes Kamerasystem würde nach dieser Auffassung kein Sensornetzwerk darstellen. Ihnen gemeinsam ist jedoch, daß sie über Sensorik verfügen, um Daten der Umgebung zu sammeln und sich spontan mit anderen Sensorknoten vernetzen können. Beide können von den Konzepten der Selbstorganisation profitieren.

Ein selbst rekonfigurierbares System stellt, aufgrund seiner Leistungsfähigkeit und Adaptierbarkeit auf Hardwareebene, eine interessante Plattform zur Untersuchung von Selbstorganisation in verteilten Systemen dar. Existierende, auf dem Markt verfügbare, eingebettete Hardware-Plattformen sind jedoch aus unterschiedlichen Gründen ungeeignet. Einige der Gründe, die nicht notwendigerweise alle auf einmal auf eine gegebene Plattform zutreffen müssen, sind:

- Mangel an Rechenleistung. Dadurch lässt sich das System beispielsweise nicht für die Bildverarbeitung einsetzen.
- Mangel an Flexibilität. Dadurch ist man an ein bestimmtes Einsatzgebiet des Systems in einer bestimmten Umgebung gebunden.
- Unzureichende Systemarchitektur. Die Fähigkeiten der vorhanden Komponenten können dadurch nicht optimal genutzt werden.

– Mangelhafte Softwareunterstützung. Es sind keine, oder nur wenige, Software-Bibliotheken vorhanden, um dem Entwickler lästige Arbeit abzunehmen.

In diesem Artikel schlagen wir ein flexibles, selbst rekonfigurierbares Rechnersystem zur Untersuchung von Selbstorganisation in verteilten Systemen im allgemeinen und in einem verteilten, intelligenten Kamerasystem im speziellen, welches in Kapitel 2 beschrieben wird, vor. Kapitel 3 beschreibt die Anforderungen an das System und unsere Lösungsansätze. Kapitel 4 beschreibt den aktuellen Stand unserer Arbeit. Kapitel 5 gibt einen Überblick über verwandte Arbeiten auf diesem Gebiet und Kapitel 6 fasst noch einmal die wichtigsten Ideen zusammen.

2 Ein selbstorganisierendes Kamerasystem

Um Selbstorganisation in verteilten Systemen zu untersuchen, möchten wir ein verteiltes, intelligentes Kamerasystem auf Basis von rekonfigurierbarer Hardware realisieren, welches beispielsweise für Überwachungsaufgaben eingesetzt werden kann. Die Hauptaufgabe besteht demnach darin, Objekte visuell zu erkennen und bei Bedarf im gesamten Überwachungsbereich zu verfolgen. Die Datenmenge, die zwischen den Kameras übertragen wird, soll dabei auf ein Minimum reduziert werden, indem beispielsweise nur charakteristische (biometrische) Merkmale des verfolgten Objekts übertragen werden. Jede intelligente Kamera soll dabei die Möglichkeit haben, ihre Position zu ändern, um den eigenen Überwachungsbereich bei Bedarf erweitern bzw. verändern zu können und um den Überwachungsbereich des gesamten Systems zu optimieren. Dazu soll sich das System selbst organisieren, sodass sich diese Selbst-Optimierung herausbilden kann. Darüber hinaus sollen neue Kameras jederzeit in das System integriert werden können, um den Überwachungsbereich beispielsweise auf neue Areale auszudehnen. Dazu soll sich das System auf Netzwerk- und Anwendungsebene Selbst-Konfigurieren können. Der Einsatz von rekonfigurierbarer Hardware ermöglicht, Bilddaten sehr effizient verarbeiten und sich den Gegebenheiten und Anforderungen flexibel anpassen zu können, indem Bilder beispielsweise je nach Helligkeit der Umgebung, notwendiger Verarbeitungsgeschwindigkeit und Erkennungs-Algorithmus durch unterschiedliche Hardware-Module verarbeitet werden. Diese Hardware-Module kann die Kamera entweder aus einem lokalen Speicher nachladen oder von anderen Kameras anfordern. Die Kameras sollen sich also auf Hardwareebene an unterschiedliche Bedingungen adaptieren, indem sie sich Selbst-Rekonfigurieren. Fällt eine Kamera aus, sollen andere Kameras versuchen die entstandene Überwachungslücke durch Verändern der eigenen Position in Absprache mit den unmittelbaren Nachbarn zu schliessen und so zur Selbst-Heilung des Systems und damit zur Aufrechterhaltung der Funktionalität beizutragen.

Eingesetzt werden könnte dieses Kamerasystem beispielsweise an Flughäfen und Bahnhöfen, um verdächtige Personen zu identifizieren und zu verfolgen, oder in Altenheimen und Kliniken, um Patienten zu überwachen.

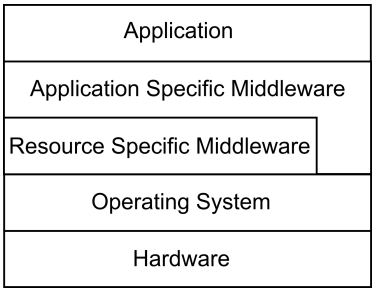

Abb. 1. Das Rechnersystem als Schichtenmodell

3 Das Rechnersystem

Abbildung 1 zeigt das gesamte Rechnersystem als Schichtenmodell bestehend aus der Hardware, dem Betriebssystem, einer ressourcenspezifischen Middleware, einer anwendungsspezifischen Middleware und der Anwendungsschicht. Die Anforderungen an das System ergeben sich größtenteils aus dem geplanten Einsatzgebiet und aus der in Kapitel 2 beschriebenen Anwendung und werden im folgenden diskutiert.

Rechenleistung Bei der Bild- und Videoverarbeitung fallen üblicherweise sehr große Datenmengen an, die enorme Anforderungen an die Rechenleistung eines eingebetteten Systems stellen. Einige Werte sind beispielhaft in Tabelle 1 aufgelistet. Bei Videodaten und angenommenen zehn Bildern pro Sekunde muß das System für die aufgelisteten Auflösungen und Farbtiefen bereits Datenmengen von 3 bis 120 Megabyte pro Sekunde bewältigen. Dazu ist leistungsfähige Hardware erforderlich.

FPGAs können diese Anforderung aufgrund ihrer inhärenten Parallelität sehr gut erfüllen. Die Daten werden dabei durch einen oder mehrere Hardware-Module, mit denen das FPGA konfiguriert werden kann, verarbeitet. Gegenüber der Verarbeitung durch Software auf einem Mikrocontroller ergibt sich der Vorteil, daß keine zusätzliche Interpretation durch eine CPU stattfindet. Dadurch lässt sich das System bei gleicher Leistung langsamer Takten, oder die Leitung bei gleichem Takt erhöhen.

Speicher Es wurde eben erwähnt, daß anfallende (Bild-)daten durch einen oder mehrere Hardware-Module verarbeitet werden können. Nicht immer ist es jedoch möglich, die Daten sofort zu verarbeiten, sodass sie zunächst zwischengespeichert werden müssen, um zu einem späteren Zeitpunkt verarbeitet zu werden. Das ist beispielsweise immer dann erforderlich, wenn ein Bild als Ganzes verarbeitet werden soll und nicht nur partiell, oder um Zwischen- bzw. Endergebnisse, die während der Verarbeitung entstehen, abzuspeichern. Daraus ergibt sich die Anforderung, jedem Hardware-Modul bei Bedarf Speicher zuordnen zu können.

Auflösung	Farbtiefe [Bits pro Pixel]	Datenmenge [Byte pro Bild]
640x480	8	307200
640x480	16	614400
800x600	8	3840000
800x600	16	7680000
1024x768	8	6291456
1024x768	16	12582912

Tabelle 1. Anfallende Datenmengen bei verschiedenen Auflösungen und Farbtiefen

Um diese Anforderung zu erfüllen, bietet es sich an, mehrere physikalisch unabhängige Speicher vorzusehen. Sie sollten unabhängig voneinander sein, weil sich ansonsten mehrere Hardware-Module einen gemeinsamen Speicher, und somit die verfügbare Speicherbandbreite, teilen müssten, was die Effizienz negativ beeinflussen würde.

Kommunikation Um Selbstorganisation in verteilten Systemen untersuchen zu können, ist es erforderlich, daß ein System mit anderen Systemen über eine Kommunikationsschnittstelle interagieren kann. Je nachdem, welche Anforderungen beispielsweise an Bandbreite, Zuverlässigkeit und Übertragungsmedium gestellt werden, bieten sich unterschiedliche Kommunikationsstandards, wie zum Beispiel CAN, ZigBee, Bluetooth oder Wireless LAN, an. Um in unterschiedlichen Umgebungen eingesetzt werden zu können, muß das System die Flexibilität besitzen, unterschiedliche Kommunikationstechnologien verwenden zu können.

Es bietet sich an, eine Standard-Schnittstelle vorzusehen, die es ermöglicht, handelsübliche Module an das System anzuschliessen. Der Universal Serial Bus (USB), wie er auch in handelsüblichen PCs zum Einsatz kommt, ist zur Erfüllung dieser Anforderung eine gute Wahl, da die oben genannten Kommunikationsstandards kostengünstig in Form sogenannter USB-Dongles käuflich erwerbbar sind. Dadurch wird ein Höchstmaß an Flexibilität bei gleichzeitig geringem Aufwand erreicht.

Sensorik Sensoren sollen direkt und ohne den Umweg über Standard-Schnittstellen, wie USB oder FireWire, mit den Hardware-Modulen verbunden werden können. Dadurch wird der direkte und effiziente Transport der Daten von der Quelle zur verarbeitenden Einheit möglich und Protokolloverhead der Standard-Schnittstellen wird vermieden.

Um dieser Anforderung gerecht zu werden, müssen generische Ein- und Ausgänge, die direkt mit den Hardware-Pins des FPGAs verbunden sind, vorgesehen werden.

Betriebssystem Das Betriebssystem sollte vorhandene Ressourcen, wie zum Beispiel Rechenkapazität, Speicher und Schnittstellen verwalten und dem Entwickler eine gewohnte Entwicklungsumgebung bieten, um lange Einarbeitungszeiten zu vermeiden und neue Ideen schnell realisieren zu können.

Linux, respektive μCLinux, ist ein geeignetes Betriebssystem, um diese Anforderungen zu erfüllen. Es setzt die Flexibilität, die die Hardware ermöglicht, auf Betriebssystemebene fort. Die Verfügbarkeit einer Vielzahl von Gerätetreibern, Dateisystemen, Netzwerkprotokollen und Software-Bibliotheken trägt dazu bei, eigene Ideen schnell umsetzen zu können, ohne sich mit Nebensächlichkeiten beschäftigen zu müssen. Ein weiterer Vorteil ist, dass Linux bereits für eine grosse Anzahl von Hardware-Plattformen portiert wurde. Dadurch erreicht man einen weiteren Grad an Flexibilität. Es ist somit nicht nur möglich, die Hardware um neue Peripherie zu erweitern, sondern die komplette Hardware kann ausgetauscht werden, ohne auf die gewohnte Softwareumgebung verzichten zu müssen. Andere Gründe, die für Linux sprechen, sind die umfangreiche Dokumentation und die Quelloffenheit. Linux ist darüber hinaus ein sehr guter Ansatz für die Entwicklung von rekonfigurierbaren Rechnersystemen [10] und wurde in diesem Zusammenhang auch erfolgreich eingesetzt [11,3].

Middleware Die Middleware soll von lokalen und im Netzwerk verteilten Ressourcen abstrahieren und die Komplexität der zugrunde liegenden Kommunikation vor den Anwendungen verbergen und diese Funktionalität den darüber liegenden Schichten als Dienst zur Verfügung stellen.

In unserem System wird die Middleware in einen ressourcenspezifischen und einen anwendungsspezifischen Teil aufgeteilt. Die ressourcenspezifische Middleware wird Aufgaben, wie z.B. Netzwerkorganisation, Überwachen der Umgebung, Analysieren der Umgebung, Service/Resource Discovery, Sicherheit, Fehlertoleranz und Rekonfiguration übernehmen. Dienste wie das Überwachen (Monitor) und Analysieren der Umgebung (Analyze) sind wichtige Bestandteile in autonomen Systemen [6] und bilden das Fundament, auf dem selbstorganisierende Systeme aufbauen. Die anwendungsspezifische Middleware wird eine Laufzeitumgebung für eine Klasse von Anwendungen bereitstellen und von den darunter liegenden Schichten abstrahieren.

3.1 Systemarchitektur

Die Systemarchitektur unseres Rechnersystems, welches die in in Kapitel 3 beschriebenen Anforderungen erfüllen kann, ist in Abbildung 2 dargestellt. Das Herzstück bildet ein Virtex-4 FPGA der Firma Xilinx. An das FPGA sind drei SDRAM-Speicherbausteine (16 bis 48 Megabyte), ein Universal Serial Bus (USB) On-The-Go (OTG) Controller der Firma Philips, ein Konfigurations-PROM und Flash-Speicher (Speicherkarten bis 2 Gigabyte) angeschlossen.

Der USB OTG Controller ermöglicht es, bis zu drei USB Ports realisieren zu können, wovon ein Port als OTG Controller Port konfiguriert werden kann. Das bedeutet, dass das System nach außen hin entweder als Peripherie oder als Host auftreten kann.

Im Konfigurations-PROM wird ein minimales System, bestehend aus einer Softcore-CPU und einem Bootloader, untergebracht. Der Bootvorgang wird weiter unten beschrieben. Programmiert wird das Konfigurations-PROM über die JTAG-Schnittstelle.

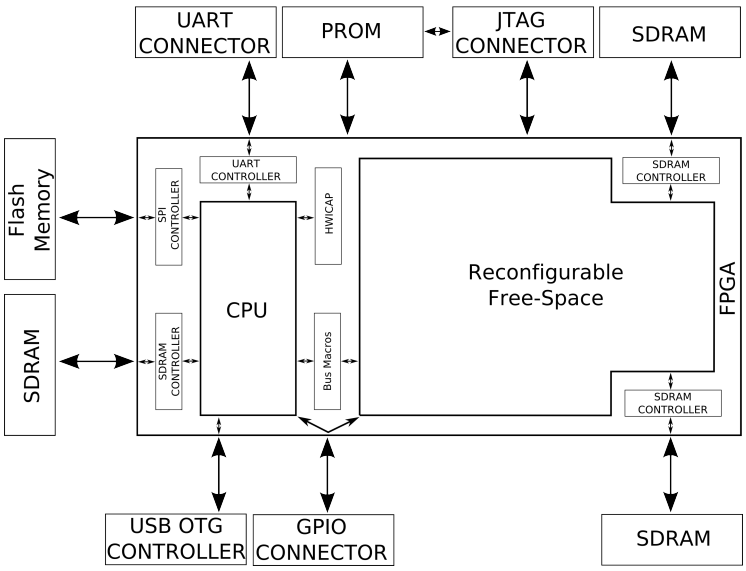

Abb. 2. Die Systemarchitektur für ein flexibles, selbst rekonfigurierendes Rechnersystem

Der Flash-Speicher beinhaltet ein Abbild des Betriebssystems, nachladbare Hardware-Module, sowie Treiber und andere System-Ressourcen. Nachdem das Betriebssystem hochgefahren ist, kann der Flash-Speicher wie eine Festplatte angesprochen werden.

Die Einteilung in PROM- und Flash-Speicher spiegelt die Dynamik des Systems wieder. Die CPU und der dazugehörige Bootloader werden üblicherweise nicht regelmässig geändert. Es darf also etwas aufwendiger sein, diesen Teil zu ändern, ohne den Arbeitsfluß wesentlich zu beeinträchtigen. Das Abbild des Betriebssystems, die Treiber und die Hardware-Module werden mit einer hohen Wahrscheinlichkeit des öfteren geändert und dementsprechend einfach sollte dieser Vorgang auch sein. Das bedeutet auf unser System bezogen, dass neue Dateien bequem am PC mit einem kostengünstigen Card-Reader auf den Flash-Speicher übertragen werden können.

Das Hochfahren des Systems erfolgt in drei Schritten:

1. Das FPGA wird mit der Softcore-CPU und dem Bootloader aus dem PROM konfiguriert.
2. Der Bootloader lädt das Abbild des Betriebssystems aus dem Flash-Speicher in den Hauptspeicher und springt anschliessend zum Einsprungspunkt des Betriebssystems.
3. Das Betriebssystem startet und initialisiert das ganze System.

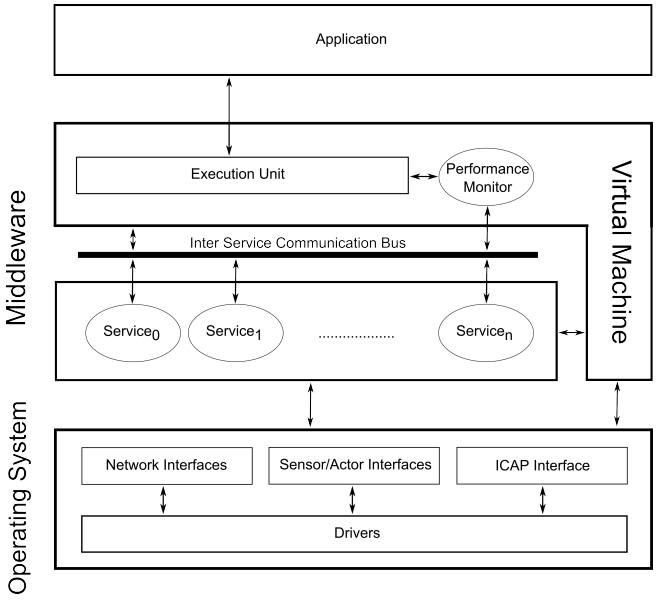

Abb. 3. Die Systemsoftware im Detail

Anschliessend lässt sich das System entweder Interaktiv über ein Terminal-Programm bedienen, oder eine Anwendung wird automatich gestartet und ausgeführt. Ausserdem besteht jederzeit die Möglichkeit, Hardware-Module nachzuladen. Dazu wird die ICAP (Internal Configuration Access Port)-Schnittstelle verwendet, welche eine selbstinitiierte Rekonfiguration des FPGAs ermöglicht.

3.2 Systemsoftware

In Abbildung 3 ist das Schichtenmodell aus Abbildung 1 detailierter dargestellt (ohne Hardware).

Das auf Linux basierende Betriebssystem bildet die Grundlage aller weiteren Schichten. Gerätetreiber im Kernelspace regeln die Low-Level-Zugriffe auf Hardware-Komponenten, während Anwendungen im Userspace über eine Schnittstelle (z.B. Sockets) darauf zugreifen.

Die ressourcenspezifische Middleware besteht aus einer Menge von Diensten, die auch untereinander kommunizieren können. Damit können die Dienste nicht nur von der darüber liegenden Schicht in Anspruch genommen werden, sondern auch von anderen Diensten auf derselben Schicht. Linux bietet die Möglichkeit Dienste im Hintergrund ausführen zu lassen. Ein Dienst ist in diesem Fall ein eigenständiges Programm und die ressourcenspezifische Middleware würde aus einer Menge solcher Programme bestehen. Die Kommunikation, im Abbildung 3 als *Inter Service Communication Bus* bezeichnet, könnte in diesem Fall über einen IPC-Mechanismus (z.B. Shared Memory, TCP/IP) realisiert werden.

Die anwendungsspezifische Middleware ist im Bild als virtuelle Maschine implementiert, die aus einer Ausführungseinheit und einem Leistungsmonitor besteht. Der Leistungsmonitor überwacht die Ausführung der Anwendung und bewertet ihre Leistung anhand von Metriken. Die Ergebnisse dieser Bewertung können entweder von anderen Diensten abgefragt werden, oder ihnen aktiv mitgeteilt werden, um bestimmte Aktionen zu veranlassen.

4 Aktueller Stand

Zum gegenwärtigen Zeitpunkt sind wir mit dem Aufbau der grundlegenden Infrastruktur beschäftigt. Die Hardware wird derzeit nach dem in Kapitel 3.1 beschriebenen Konzept realisiert.

Auf der Software-Seite wird ein Monitoring-Dienst für die ressourcenspezifischen Middleware entwickelt, welcher die Aufgabe hat, lokale Nachbarn zu überwachen und Merkmale wie beispielsweise Auslastung, Verbindungsqualität, verfügbare Dienste und Position der Kamera zu protokollieren. Diese Informationen können von anderen Diensten auf derselben Schicht, oder von Anwendungen auf der darüber liegenden Schicht abgerufen werden. Jedem System im Netzwerk ist es damit möglich, Entscheidungen auf Basis von lokalen Informationen zu treffen. Damit wird der Grundstein für ein selbstorganisierendes System gelegt.

Im Bereich Reconfigurable Computing sind wir zur Zeit in der Lage einen FPGA sich selbst partiell rekonfigurieren zu lassen. Dabei kommt ein Linux-System zum Einsatz, dass mittels des von Williams beschriebenen Gerätetreibers [10] auf die ICAP-Schnittstelle zugreift. Die dazu benötigten Bitströme können bereits über das Betriebssystem von entfernten Rechnern in einem TCP/IP basierten Netzwerk geholt werden.

Darauf aufbauend soll nun als Beispielanwendung ein einfaches System zur Selbstorganisation in einem Netzwerk dieser Einheiten erstellt werden. Mindestens zwei Dienste werden benötigt, gleich ob es sich um einen minimalistischen Sensor- oder leistungsstärkeren Rechenknoten handelt.

Zum einen wird ein Modul benötigt, das aufgrund gegebener Fakten abschätzen soll, wie aufwändig eine bestimmte Operation für den Knoten ist. Auf Anfrage soll dieser „Händler“ einen Kostenvoranschlag liefern, für den es dem betreffenden Knoten möglich ist die angefragte Aufgabe zu erledigen.

Zum anderen bedarf es als Gegenstück zum „Händler“ eines „Entscheiders“, der den Kostenvoranschlag des eigenen wie auch Angebote entfernter Knoten berücksichtigt, um das beste Vorgehen einzuleiten.

Der „Händler“ kennt die Fähigkeiten seines Knotens und die Kosten für Kommunikation mit Nachbarn. In Verbindung mit dem Anfragetupel für Aufgabe K, $A_K = \{Quelle, Ziel, Antragsteller, Datenmenge, Aufgabe\}$ kann er nun einen Kostenvektor für K berechnen: $C_K = \begin{pmatrix} Z_K & E_K & P_K \end{pmatrix}^T$. Dabei ist Z die insgesamt benötigte Zeit, E die benötigte Energie und P der gesamte Platzbedarf, der z.B. hinsichtlich der belegten Menge an rekonfigurierbaren Blöcken angegeben werden kann. Z_K ergibt sich aus $\sum Z_{K_{module}}$, z.B. also aus Z_{komm}, der Zeit, die für die Kommunikation benötigt wird, oder Z_{trans} für den Datentransfer.

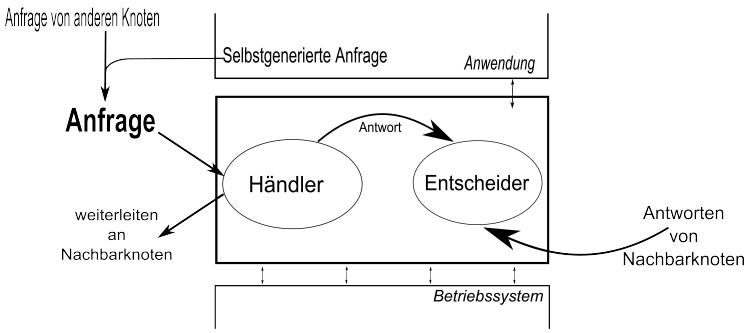

Abb. 4. Zusammenwirken der Dienste „Händler“ und „Entscheider“

Die Summanden ermittelt jeder Knoten lokal für sich, sodass individuell z.B. auch ein Z_{init} addiert werden könnte, um eine nötige Initialisierungsphase eines speziellen Knotens zu berücksichtigen. E_K sowie P_K werden analog ermittelt.

Der Anwendung im anfragenden Knoten ist es so möglich, mit dem Gewichtungsvektor G Prioritäten hinsichtlich des Energie-, Zeit- oder Platzbedarfs zu setzen: $A_K = \left(Z_K \;\; E_K \;\; P_K \right)^T \cdot \vec{G}$

Eine systemweite Metrik wird nötig, um die Angebote der Händler tatsächlich vergleichen zu können. Der Energiebedarf eines Hardwaremoduls, dass dynamisch nachgeladen werden soll, kann z.B. bei dessen Erstellung gemessen oder simuliert werden.Die Energie, die eine Kommunikation über eine bestimmte Route durchs Netzwerk erfordert, kann mittels eines „energiebewussten“ Routing-Protokolls schon bei Anfragestellung abgeschätzt werden. Ebenso wie die für Kommunikation benötigte Zeit.

Der „Entscheider“ eines Knotens erhält die Angebote anderer Knoten sowie das seines eigenen Händlers, wählt das billigste aus und setzt mit einer Bestätigung an den Angebotsersteller und der Datenübertragung die Verarbeitung der Daten in Gang. Unter Umständen wird noch ein Aufhebungssignal an die anderen Händler gesendet, sollte ein akzeptables Angebot sehr schnell eingetroffen sein und nicht auf weitere gewartet werden. Abbildung 4 gibt eine Übersicht zu den Vorgängen.

Entsteht nun in einem Knoten der Bedarf nach der Lösung eines Problems, so formuliert der Entscheider zunächst ein Anfragetupel A_K. Hier kann der Parameter $Aufgabe$ auch eine Aufgabenliste sein, die der Entscheider, wenn möglich, aufteilen und in mehreren Anfragen an verschiedene „Zulieferer“, insbesondere auch den eigenen Händler, senden kann. Die angesprochenen Händler erstellen aufgrund lokaler Information ein Angebot, dass sie dem Anfragesteller zurücksenden. Dieser nimmt das für ihn günstigste durch Bestätigung an den entsprechenden Knoten an und sendet nach Rückmeldung die zu berechnenden Daten. Die zuletzt erfolgreich in Anspruch genommenen Dienste, geholte Bitströme oder

einfach auch besonders effizente Routen werden bis zu einem gewissen Grad lokal gespeichert, um wiederkehrende Anfragen schneller bearbeiten zu können und die Ausfallsicherheit des Gesamtsystems zu erhöhen.

5 Verwandte Arbeiten

Die Intel Mote (iMote) Plattform wurde aufgrund von Untersuchungen über Anwendungsgebiete von Sensornetzwerken entwickelt. Den Ergebnissen dieser Untersuchungen zufolge besteht Bedarf an höherer Rechenleistung, mehr Speicher, einer verbesserten Zuverlässigkeit der drahtlosen Übertragung, Sicherheit und reduzierten Kosten. Ein iMote besteht aus einer CPU auf Basis eines ARM7TDMI Prozessorcores, einem Bluetooth-Transceiver für die Kommunikation und generischen Input/Output Steckverbindern für Erweiterungen, 64 kbyte Speicher und 512 kbyte Flash-Speicher. Als Betriebssystem wurde das für Sensorknoten-Hardware beliebte TinyOS [5] portiert [7].

In [9] wird ein System bestehend aus zwei Platinen, die miteinander verbunden werden können, beschrieben. Eine Platine ist mit einem FPGA, 512 kbyte Speicher, generischen Input/Output Steckverbindern und einer Ansteuerung für einen Schrittmotor ausgestattet. Die andere Platine beinhaltet einen Mikrocontroller zusammen mit einem Bluetooth-Transceiver, womit es möglich ist, das FPGA über Bluetooth neu zu konfigurieren.

In [8] wird eine modulare Architektur für drahtlose Sensorknoten beschrieben, die aus vier Schichten, jeweils eine für Verarbeitung, Stromversorgung, Kommunikation und Sensorik, besteht. Jede dieser Schichten kann ausgetauscht werden, wodurch man die gesamte Hardware an die jeweilige Anwendung anpassen kann. Bisher realisiert wurde eine kombinierte Mikrocontroller/FPGA-Platine für die Verarbeitung, eine Bluetooth-Platine für die Kommunikation, zwei Sensor-Platinen mit Sensoren für u.a. Temperatur, Helligkeit und Feuchtigkeit und eine Platine für die Stromversorgung, welche über ein externes Netzteil versorgt wird.

6 Zusammenfassung

In diesem Artikel haben wir das Konzept eines flexiblen Rechnersystems auf Basis eines FPGA vorgestellt, welches aufgrund der hohen Integrationsdichte ermöglicht, komplexe Anwendungen auf dem Chip zu realisieren, die sich auf Hardwareebene selbst rekonfigurieren können. Dieses System soll in Zukunft als Basis für ein verteiltes, intelligentes Kamerasystem verwendet werden, welches zur Raum- und Personenüberwachung und -verfolgung eingesetzt werden kann und als Plattform zur Untersuchung von Selbstorganisation in verteilten Systemen dienen.

Literaturverzeichnis

1. Bharathidasas, A., and Ponduru, V. A. S. Sensor networks: An overview. Technical report, Dept. of Computer Science, University of California at Davis, 2002.
2. Camazine, S., Deneubourg, J.-L., Franks, N. R., Sneyd, J., Theraulaz, G., and Bonabeau, E. *Self-Organization in Biological Systems.* Princeton University Press, 2003.
3. Donato, A., Ferrandi, F., Redaelli, M., Santambrogio, M. D., and Sciuto, D. Caronte: A complete methodology for the implementation of partially dynamically self-reconfiguring systems on fpga platforms. In *FCCM´05: 13th IEEE Symposium on Field-Programmable Custom Computing Machines* (Los Alamitos, CA, USA, April 2005), vol. 00, IEEE Computer Society, pp. 321–322.
4. Garcia, P., Compton, K., Schulte, M., Blem, E., and Fu, W. An overview of reconfigurable hardware in embedded systems. *EURASIP Journal on Embedded Systems* (2006), Article ID 56320, 19 pages.
5. Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D. E., and Pister, K. S. J. System architecture directions for networked sensors. In *Architectural Support for Programming Languages and Operating Systems* (2000), pp. 93–104.
6. Kephart, J. O., and Chess, D. M. The vision of autonomic computing. *Computer 36*, 1 (2003), 41–50.
7. Kling, R. M. Intel mote: An enhanced sensor network node. In *Workshop on Advanced Sensors, Structural Health Monitoring, and Smart Structures* (Kanagawa, Japan, 2003).
8. Portilla, J., de Castro, A., de la Torre, E., and Riesgo, T. A modular architecture for nodes in wireless sensor networks. *Journal of Universal Computer Science 12*, 3 (2006), 328–339. `http://www.jucs.org/jucs_12_3/a_modular_architecture_for`.
9. Upegui, A., Moeckel, R., Dittrich, E., Ijspeert, A. J., and Sanchez, E. An fpga dynamically reconfigurable framework for modular robotics. In *ARCS´05: 18th International Conference on Architecture of Computing Systems, Workshops* (Innsbruck, Austria, March 2005), U. Brinkschulte, J. Becker, D. Fey, C. Hochberger, T. Martinetz, C. Müller-Schloer, H. Schmeck, T. Ungerer, and R. P. Würtz, Eds., VDE Verlag, pp. 83–89.
10. Williams, J., and Bergmann, N. Reconfigurable linux for spaceflight applications. In *MAPLD´04: 7th Military and Aerospace Programmable Logic Devices Conference* (Washington DC, USA, September 2004).
11. Williams, J. W., and Bergmann, N. Embedded linux as a platform for dynamically self-reconfiguring systems-on-chip. In *ERSA´04: Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms* (Las Vegas, Nevada, USA, June 2004), T. P. Plaks, Ed., CSREA Press, pp. 163–169.