Integration von C-PARTSSIM und MATLAB/Simulink

Thomas Kuhn (kuhn@cs.uni-kl.de)

Dieser Bericht beschreibt die Integration von Simulink in unseren C-PartsSim Simulator. Simulink ermöglicht es, realistische Umgebungsmodelle mittels C-PartsSim zu simulieren, und auf diese Weise neu entwickelte eingebettete Systeme unter realistischen Bedingungen zu testen. Ferner ermöglicht es C-PartsSim, unterschiedliche Simulationskomponenten in die Simulation zu integrieren. Momentan existieren Komponenten, welche Zufallszahlen erzeugen, realistische Drahtlosnetzwerke simulieren, und verschiedene Hardwareplattformen und Geräte implementieren.

I. EINLEITUNG

Moderne eingebettete Systeme sind aus der heutigen Welt nicht mehr wegzudenken. Diese Systeme werden in der Regel auf kleinen, billigen, und deswegen ressourcenbeschränkter Hardware ausgeführt. Insbesondere bei Systemen, die in hohen Stückzahlen gefertigt werden, lassen sich durch billigere Hardware signifikant Kosten einsparen. Aus diesem Grund müssen diese Systeme nicht nur sehr sparsam mit den vorhandenen Hardwareressourcen umgehen – die Hardware muss auch genau dimensioniert werden um unnötige Kosten zu vermeiden, und gleichzeitig ein fehlerfreies Ausführen der entwickelten Systeme zu gewährleisten.

Eine weitere Eigenschaft eingebetteter Systeme ist die Interaktion mit deren Umgebung. Im Gegensatz zu Informationssystemen werden eingebettete Systeme in der Regel von der Umgebung und nicht von Nutzern getriggert. Das bedeutet, dass sich diese Systeme an die Umgebung anpassen müssen – auch wenn es Interaktion mit Nutzern gibt, stehen diese nicht im Mittelpunkt der Entwicklungsaktivitäten. Testfälle müssen daher auch das Verhalten der Umgebung akkurat simulieren. Hierzu werden Umgebungsmodelle entwickelt, welche das sichtbare Verhalten der Umgebung basierend auf zeitkontinuierlichen Berechnungsmodellen akkurat nachbilden. Hierfür gibt es verschiedene Entwicklungsumgebungen, beispielsweise SciLAB, Simulink, und ASCET, welche bereits heute auch in der Industrie eingesetzt werden.

Das Testen von eingebetteten Systemen erfordert daher nicht nur das testen der entwickelten Algorithmen – sondern auch das Bewerten des Verhaltens des fertigen Systems. Dies wird als Performanz-Evaluation bezeichnet und bezieht sowohl die Plattform auf der das System ausgeführt wird, als auch dessen Umgebung ein. Hierzu müssen alle relevanten Geräte, beispielsweise Prozessoren, Netzwerke, und die Umgebung des Systems akkurat simuliert werden. Schlussendlich ist es auch notwendig, dass der getestete Code dem Produktionscode weitestgehend entspricht – unterschiedliche Codegeneratoren und Compiler können wiederum fehlerhaftes Verhalten erzeugen, welches dann während des Testens nicht erkannt wird.

C-PartsSim ist ein Simulatorframework, welches sowohl einfaches Testen, als auch die Performanz-Evaluation eingebetteter Systeme unterstützt. Dieser Bericht beschreibt die Grundzüge von C-PartsSim, und fokussiert danach auf die Integration von Simulink, welches die Simulation der Systemumgebung ermöglicht. Andere Veröffentlichungen dokumentieren die Simulation von Plattformen und die Simulation verschiedener Hardwaregeräte (siehe [1]).

Die weitere Gliederung dieses Berichts ist wie folgt: Sektion 2 beschreibt die Komponenten von C-PartsSim. Sektion 3 dokumentiert wichtige Simulationskomponenten. Sektion 4 fokussiert auf die Integration von Plattformen, und die Integration von Simulink. Sektion 5 beschreibt ein Beispielszenario und zeigt eine C-PartsSim Konfiguration, welches dieses Simuliert.

II. C-PARTSSIM KOMPONENTEN

Abbildung 1 zeigt die Komponenten des C-PartsSim Frameworks. Die zentrale Komponente, die die Simulation steuert ist ein TCL Interpreter, welcher zuerst die Standardkonfiguration von C-PartsSim verarbeitet, und danach das Simulationsskript lädt. Dieses instanziiert zuerst die C-PartsSim Komponente und legt danach alle anderen Simulationskomponenten an. Simulationskomponenten implementieren beispielsweise den *Scheduler*, welcher Simulationsereignisse ordnet, den Zufallszahlengenerator, simulierte Knoten, Plattformen, Geräte, Netzwerke, oder die Umgebung des Systems. Durch diesen Ansatz lassen sich Simulationsszenarien sehr frei definieren, da sich nahezu beliebige Simulationskomponenten erzeugen lassen.

Das Kernpaket von C-PartsSim selbst enthält keine konkreten Simulationskomponenten. Es definiert lediglich abstrakte Komponenten, welche als Basis für verschiedene Typen von Simulationskomponenten fungieren. Die folgenden Komponententypen werden von C-PartsSim unterstützt:

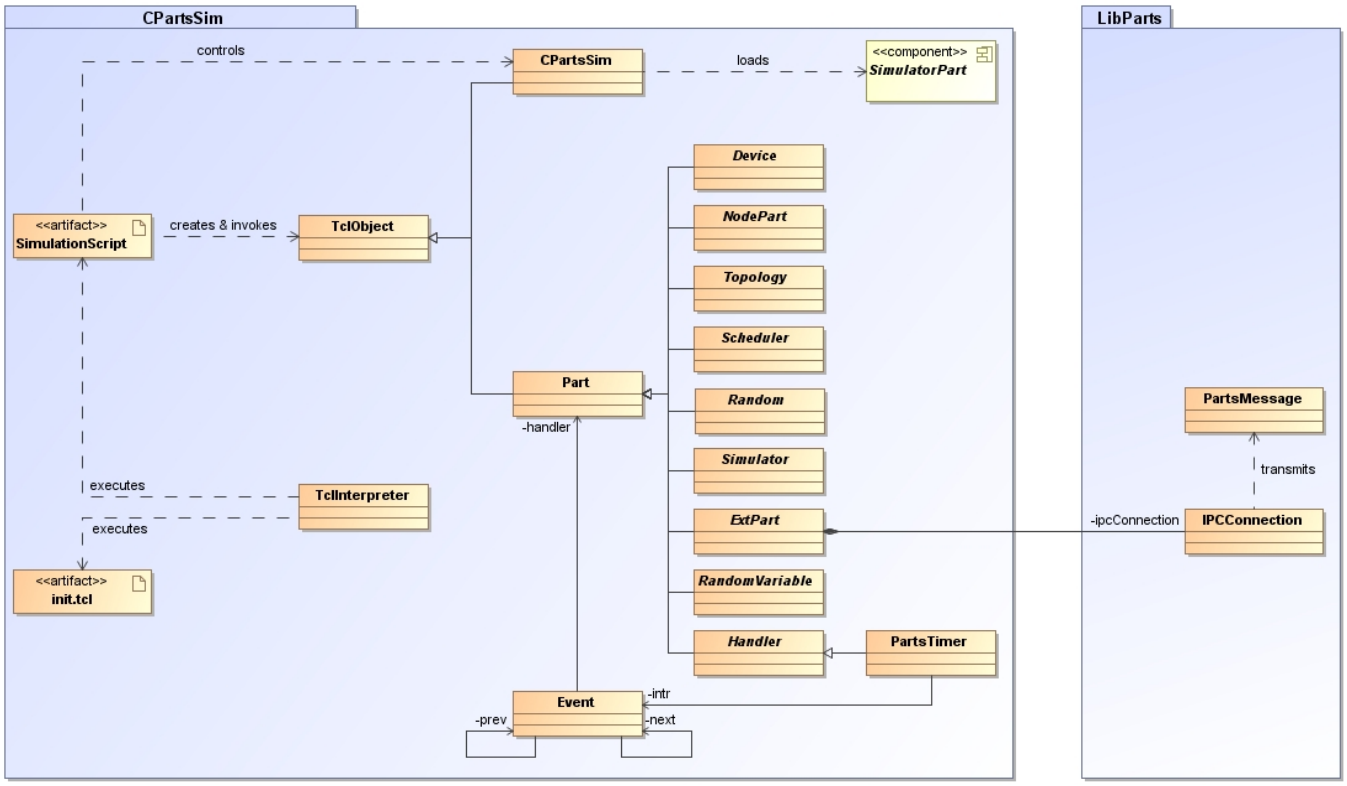

Abbildung 1: C-PartsSim Framework

- Der *Scheduler* ist die Kernkomponente des Simulators. C-PartsSim Simulationen dürfen jeweils nur eine Instanz eines *Scheduler* besitzen. Dieser regelt das Zeitverhalten der Simulation, ordnet die auftretenden Ereignisse und verteilt diese an die anderen Komponenten, und steuert, ob eine Simulation in Simulationszeit oder in Echtzeit ausgeführt wird.
- Die Komponente *Random* implementiert den Zufallszahlengenerator der Simulation. Wie die *Scheduler* Komponente, darf diese ebenfalls nur einmal in jeder Simulation existieren. Zufallsvariablen werden durch die Komponente *RandomVariable* repräsentiert. Verschiedene Implementierungen der *Random* und *RandomVariable* Komponenten erlauben es, verschiedene Mechanismen zum Erzeugen von Zufallszahlen zu verwenden.
- Komponenten vom Typ Knoten (*NodePart*) besitzen ein Verhalten. Es existieren verschiedene Typen von Knoten. Hardwareplattformen führen Modelle aus. Knoten vom Typ Hardwareplattformen modellieren Prozessoren und Speicher. Abhängig vom Detaillierungsgrad dieser Plattformen lassen sich sehr genaue, oder eher generische simulierte Hardwareplattformen erzeugen. Knoten simulieren beispielsweise auch Netzwerke, oder die Umgebung des Systems.
- Geräte sind Verbindungspunkte von Knoten. Sie repräsentieren entweder real existierende Geräte, oder virtuelle Geräte, welche nur in einer Simulation existieren. Komponenten vom Typ *Device* enthalten entweder ein definiertes Verhalten, oder nutzen einen Simulator, falls ein komplexeres Verhalten simuliert werden muss.
- Komponenten vom Typ *Simulator* implementieren existierende Simulatoren, welche in ein Simulationsszenario integriert werden. Beispielsweise wird der Netzwerksimulator ns-2 als Simulator integriert. Um einen Simulator in eine Simulationskomponente zu konvertieren muss dessen Simulationsmodell angepasst werden.
- Komponenten vom Typ *ExtPart* definieren allgemeine externe Simulationskomponenten, welche über Nachrichten mit C-PartsSim kommunizieren. Diese Komponenten werden ebenfalls genutzt um Simulatoren zu implementieren, welche jedoch nicht in C-PartsSim geladen werden können. Gründe hierfür sind ein inkompatibles Speichermanagement, oder das Verwenden einer anderen Programmiersprache. Auch ist es häufig nicht wirtschaftlich einen existierenden Simulator in eine vollwertige Simulationskomponente vom Typ Simulator zu konvertieren. In diesem Fall bietet sich die einfachere *ExtPart* Komponente an.
- Die Topologiekomponente (*Topology*) definiert die Räumliche Position aller an der Simulation beteiligter Knoten. Diese Komponente darf in einer Simulation ebenfalls nur einmal instanziiert werden.
- Komponenten vom Typ *Handler* definieren

Komponenten, welche auf Ereignisse reagieren. Eine vordefinierte Komponente vom Typ *PartsTimer* ermöglicht es, Ereignisse zu bestimmten Zeitpunkten auszulösen.

Zusätzliche Parts (= Teile) werden zu Beginn der Simulation durch das Simulationsskript in den Simulator geladen. Diese realisieren konkrete Simulationskomponenten – die folgende Sektion 3 beschreibt wichtige Simulationskomponenten.

III. SIMULATIONSKOMPONENTEN

Die Schedulerkomponente (siehe Abbildung 2) enthält vier verschiedene Scheduler, welche unterschiedliche Sortieralgorithmen benutzen um simulierte Ereignisse zu verwalten. Während das sichtbare Verhalten der unterschiedlichen Komponenten Äquivalent zueinander ist, sind die jeweiligen Komponenten unterschiedlich effizient bei der Verwaltung von Ereignisfolgen. Die gezeigten Simulationskomponenten basieren auf den Schedulern des ns-2 Simulators – weitere Informationen zu den jeweiligen Komponenten kann in der Dokumentation dieses Simulators [2] gefunden werden.

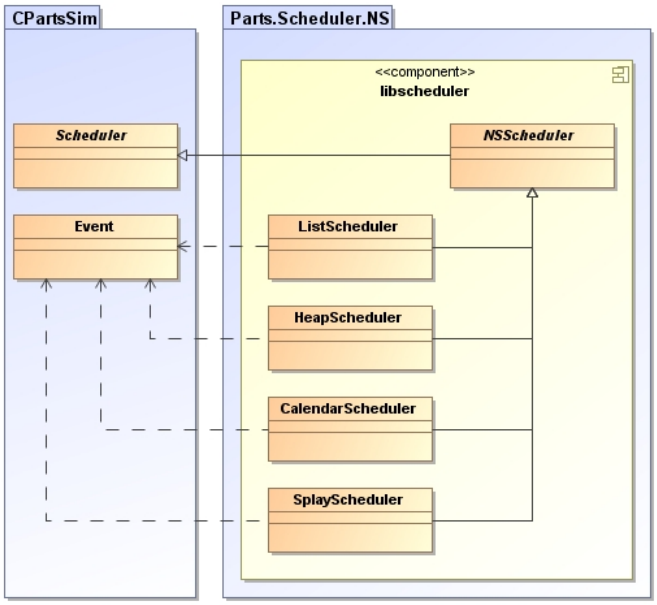

Abbildung 2: Scheduler Simulationskomponente

Das Erzeugen von (wiederholbaren) Zufallszahlen ist unabdingbar für korrekte Simulationsergebnisse. Basierend auf den Zufallszahlengeneratoren des ns-2 Simulators wurden für C-PartsSim verschiedene Zufallsvariablen definiert, welche von der Komponente *Random* kontrolliert werden. Üblicherweise wird genau eine Zufallsvariable pro Simulation erzeugt, welche wiederholbare Zufallszahlenketten für jedes Szenario liefert. Allerdings ist dies keine strikte Anforderung: Denkbar sind auch Generatoren, welche mehrere Zufallsvariablen nutzen, und Zufallsvariablen, welche keine sich bei jeder Simulation wiederholenden Ketten von Zufallszahlen erzeugen. In diesem Fall lassen sich Simulationsergebnisse nicht wiederholen. Abbildung 3 zeigt die Struktur der für die Zufallszahlen verantwortliche Simulatorkomponente *librandom*.

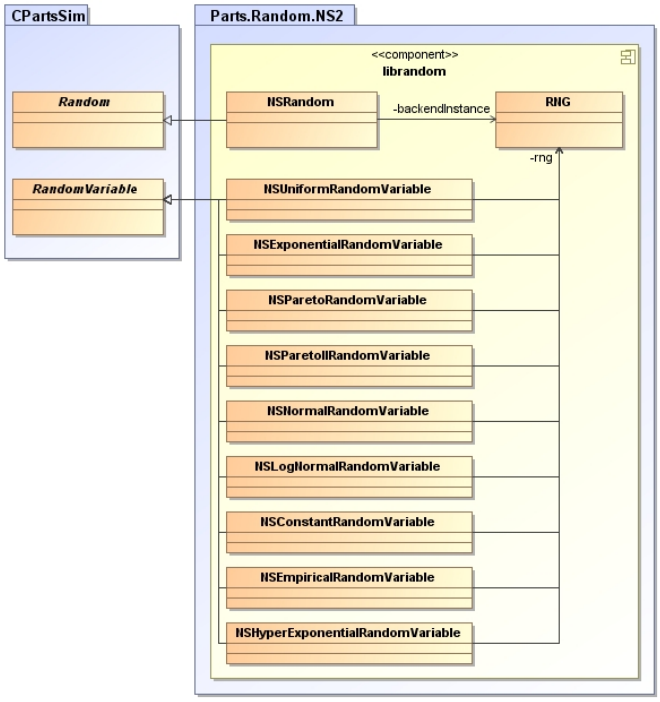

Abbildung 3: Zufallzahlengenerator

Die dritte Basiskomponente verwaltet die Topologie der Simulation. Lediglich eine einfache Implementierung dieser Komponente wurde implementiert, diese Speichert die Position jedes Knotens im Raum in einer Instanz der Komponente *NodePos*. Abbildung 4 zeigt die Simulationskomponente, welche die Topologie bereitstellt.

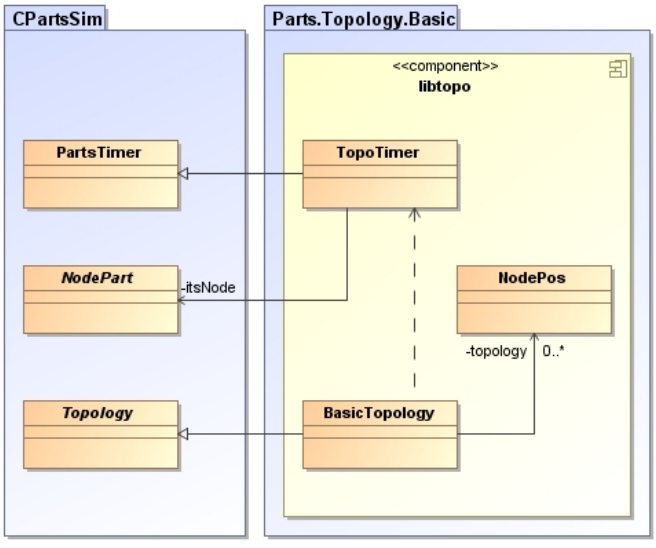

Abbildung 4: Topologiekomponente

Die drei vorgestellten Komponenten sind Kernkomponenten von C-PartsSim. Sie sind ausreichend mächtig um viele Simulationsszenarien zu unterstützen. Durch die modulare

Struktur von C-PartsSim lassen sich jedoch beliebige Implementierungen für unterschiedliche Szenarien verwenden. Auf diese Weise kann C-PartsSim an die Anforderungen der jeweiligen Szenarien angepasst werden.

IV. C-PartsSim Plattformen

Plattformen sind Simulationskomponenten, welche Hardwareplattformen simulieren. Die Komponente *GenericPlatform*, welche in Abbildung 5 gezeigt ist, simuliert eine allgemeine Hardwareplattform, welche keine Ressourcenbeschränkungen aufweist. Diese Plattform ist geeignet um Hardware zu simulieren, welche ausreichend dimensioniert ist um keine messbaren Einflüsse auf das Systemverhalten zu haben. Denkbar ist der Einsatz dieser Plattform auch, wenn keine konkrete Simulationskomponente für die genutzte Plattform verfügbar ist. Ebenfalls wird diese Komponente eingesetzt, um die Umgebung eingebetteter Systeme zu modellieren, da Umgebungsmodelle ebenfalls keiner Ressourcenbeschränkung unterworfen sind.

Die Plattformkomponente ist mittels dreier Pakete realisiert: Das erste Paket *Parts.Node.Extplatform* definiert eine Komponente, welche es ermöglicht beliebige externe Plattformsimulatoren in C-PartsSim einzubinden. Diese wird beispielsweise auch genutzt, um den Avrora Simulator zu integrieren, welcher eine Simulation der Mica2 und MicaZ Plattform bereitstellt (siehe dazu auch [3]). Die Komponente *ExtPlatform* leitet dazu die Kommunikation zwischen den simulierten Hardwareplattformen und den Geräten zu den jeweiligen Instanzen weiter – zu diesem Zweck wird eine nachrichtenbasierte Kommunikation genutzt.

Die generische Hardwareplattform wird durch das Paket *Parts.Node.Platform.Generic* realisiert. Diese definiert einen Knotentyp, welcher *NodePart* indirekt spezialisiert, und ein Simulator backend, welches die Simulation implementiert. Für jeden simulierten Knoten der eine generische Hardwareplattform simuliert, wird im Simulationsskript eine Instanz der Klasse *GenericPlatformNode* erzeugt. Dieser erzeugt eine Instanz von *GenericPlatform*, falls dies noch nicht geschehen ist, und anschließend eine Instanz der Klasse *SDLModel*. Diese Klasse führt das geladene Modell aus – ursprünglich wurde sie für das Laden von SDL Modellen in den Simulator konzipiert, und trägt daher noch diesen Namen. Instanzen der Klasse *SDLModel* verwalten die lokale Zeit jedes simulierten Knotens und ermöglichen es so, defekte Zeitgeber zu simulieren. Sie fungieren ebenfalls als Kommunikationsendpunkte zwischen den simulierten Modellen und C-PartsSim. Jede Instanz von *SDLModel* kommuniziert mit genau einem *ModelKernel* – dies ist Code, der von einem Modell generiert wurde, und ein kompatibles Ausführmodell implementiert. Momentan existieren solche Kernel für SDL und Simulink.

Die Integration von SDL in C-PartsSim wurde bereits in [4] beschrieben. Im folgenden wird daher auf den Kernel für Simulink fokussiert.

Sobald ein Modell geladen und gestartet wird, wird es von C-PartsSim Initial ausgeführt. Simulink Modelle implementieren ein kontinuierliches Verhalten, welches von ereignisbasierten Simulatoren nicht ausgeführt werden kann.

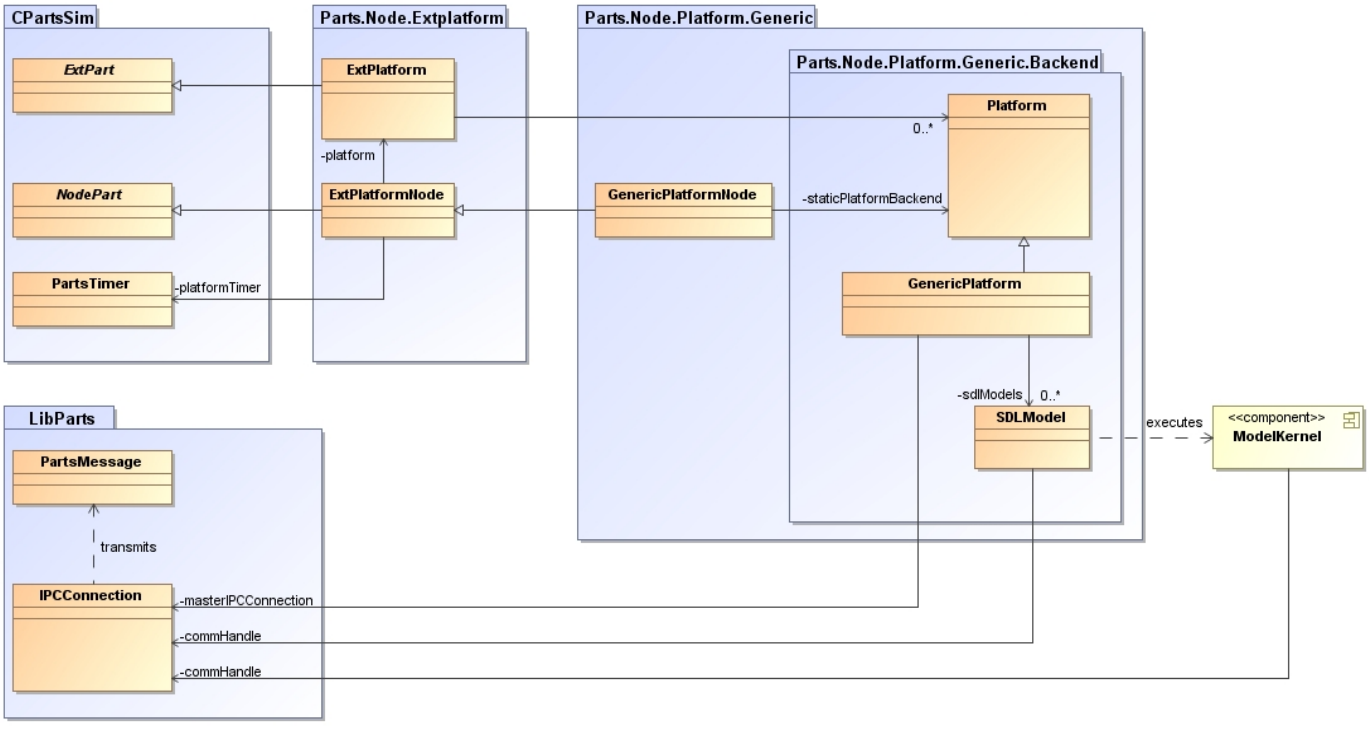

Abbildung 5: Simulationskomponente zur Simulation der generischen Plattform

Gleiches gilt auch für die Implementierung solcher Modelle in eingebettete Systeme, beispielsweise um einen Regelkreis zu kontrollieren. In diesem Fall wird das kontinuierliche Modell diskretisiert. Dies bedeutet, dass es in festen Zeitabständen ausgeführt wird, und nur zu diesen Zeitabständen eine Änderung an den Eingabewerten registriert. Abhängig von der Größe dieser Zeitabstände nimmt die Genauigkeit der Berechnung verglichen zu dem ursprünglichen, kontinuierlichen Modell mehr oder weniger stark ab. Simulink unterstützt diese diskretisierung bereits; der entsprechende Kernel muss daher in regelmäßigen Abständen, welche mit der während der diskretisierung gewählten Zeitspanne übereinstimmt aufgerufen werden, um den nächsten Rechenschritt durchzuführen.

Dies wird mittels Timer erreicht. Nachdem ein Kernel Initial aufgerufen wurde, wird er nur noch gestartet, wenn ein Ereignis aufgetreten ist. Die kann der Empfang einer Nachricht von einem simulierten Gerät sein, oder ein Timer, der vom Kernel selbst gesetzt wurde. Um einen kontinuierlichen Aufruf zu gewährleisten setzt der Kernel nach jeder Berechnung des Simulink Modells einen neuen Timer auf den nächsten Berechnungszeitpunkt. Sofern ein *Scheduler* benutzt wird der auf Simulationszeit basiert, ist so gewährleistet, dass das Simulink System immer zu der exakt vorgeschriebenen Zeit neuberechnet wird. Die Zeit, die für diese Neuberechnung benötigt wird, spielt keine Rolle bei Simulationszeit. Wird ein Echtzeitscheduler verwendet, muss die während der Berechnung vergangene Zeit allerdings berücksichtigt werden. Der implementierte Kernel ist daher nur bedingt tauglich für Echtzeitsimulationen.

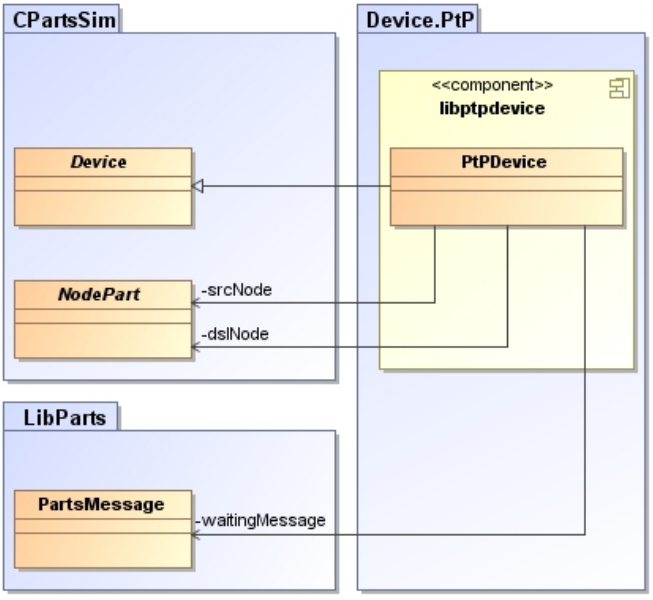

Abbildung 6: Punkt-zu-Punkt Verbindung

Die Kommunikation zwischen simulierten Knoten und dem Simulink System wird von Geräten unterstützt. In C-PartsSim wurde zu diesem Zweck eine spezielle Simulationskomponente implementiert, welche die Kommunikation über eine Punkt-zu-Punkt Verbindung realisiert. Diese Verbindung hat keine Ressourcenbeschränkungen, daher kostet eine Kommunikation über diese keine Zeit. Jedoch ist es sehr einfach möglich eine solche Beschränkung in spezialisierten Implementierungen hinzuzufügen. Abbildung 6 zeigt die Struktur der Komponente *libptpdevice*, welche die Punkt-zu-Punkt Verbindung zwischen zwei Knoten (Komponenten vom Typ *NodePart*) simuliert.

V. Beispielszenario "Inverses Pendel"

Basierend auf den vorgenannten Simulationskomponenten wird nun das Beispielszenario aus Abbildung 7 beschrieben. Das gezeigte Szenario besteht aus einem Inversen Pendel, welches von einem Wagen stabil gehalten wird. Drei Sensoren wurden an dem Pendel angebracht: einer für die Position des Wagens, einer für dessen Geschwindigkeit, und einen für den Winkel des Pendels. Ein Knoten sammelt diese Werte und sendet diese an einen Empfänger. Der Empfänger berechnet neue Stellgrößen für die Geschwindigkeit des Wagens und kommuniziert diese drahtlos an den Wagen.

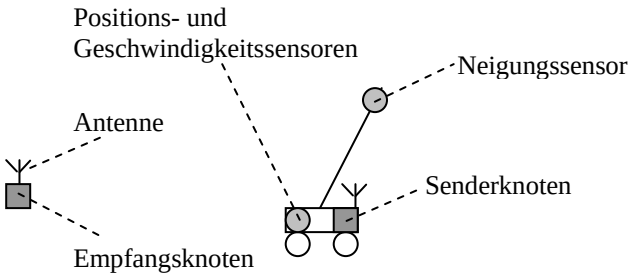

Abbildung 7: Simuliertes Szenario

Die Kommunikation der Knoten untereinander ist drahtlos, hierfür wird ein IEEE 802.15.4 Netzwerk, basierend auf CC2420 Tranceivern simuliert. Diese Simulation wird mittels des ns-2 Simulators, der als Simulationskomponente in C-PartsSim integriert ist, realisiert. Die Hardwareplattform der Knoten ist ein Intel iMote, der mit einem XScale Prozessor ausgestattet ist. Da sowohl ausreichende Prozessor, als auch ausreichende Speicherressourcen verfügbar sind, werden diese mittels der generischen Plattform simuliert.

Die Umgebung des Systems ist der Pendel; der Zustand des Pendels wird mittels Simulatoren gemessen. Ein Umgebungsmodell, welches das Verhalten des Pendels modelliert und die jeweiligen Sensorwerte zu jeder Zeit berechnet wurde mit Simulink erzeugt, und in die Simulation eingebunden.

Um dieses Szenario zu simulieren wurde ein Simulationsskript entwickelt, welches die in Abbildung 8 gezeigten Instanzen von Simulationskomponenten erzeugt.

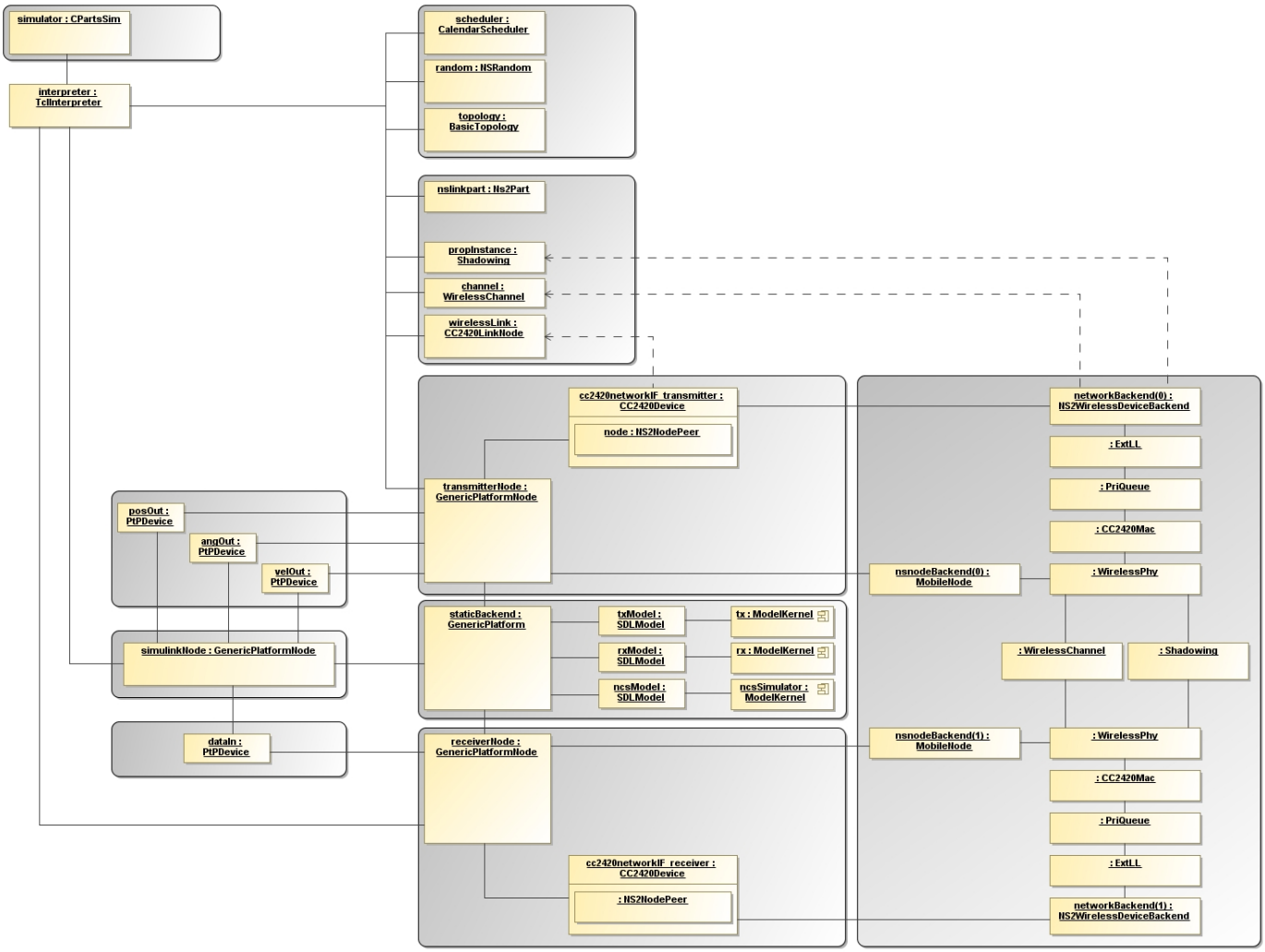

Abbildung 8: Instanzen des simulierten Szenarios

Der obere Teil von Abbildung 8 zeigt die Instanzen der Kernkomponenten CPartsSim, CalendarScheduler, NSRandom und BasicTopology. Der nächste grau hinterlegte Block zeigt die Komponenteninstanzen, die die drahtlose Kommunikation simulieren. Der Sender, der Empfänger, sowie die simulierte Umgebung werden durch die Instanzen von GenericPlatformNode simuliert. Sender und Empfänger sind mit jeweils einem Gerät vom Typ CC2420 verbunden, welcher den Zugang zu einem drahtlosen Netzwerk simuliert, sowie über punkt zu punkt Verbindungen (PtPDevice Instanzen) mit dem Ungebungsmodell. Ebenfalls sichtbar sind die drei Instanzen der ModelKernel Komponente, welche das Verhalten der drei Knoten implementieren. Die Komponenten tx und rx sind SDL-Systeme, welche mittels SEnF, das SDL Environment Framework, an den Simulator angebunden sind. Simulink Modelle werden über ein ähnliches, einfacheres Framework mit C-PartsSim verbunden, welches direkt in den Kernel einkompiliert ist.

VI. Zusammenfassung

Die in diesem Bereicht beschriebene Methode dokumentiert die Integration von Simulink als Simulationskomponente in C-PartsSim. Verwendet wurde Simulink in dem gezeigten Beispiel um die Umgebung des Systems zu modellieren. Denkbar sind auch andere Anwendungsszenarien, beispielsweise das Modellieren realistischer Ausbreitungsmodelle.

Referenzen

[1] T. Kuhn, R. Gotzhein: Model-driven Platform-specific Testing through Configurable Simulations. Fourth European Conference on Model Driven Architecture Fondations and Applications, ECMDA, Berlin, Germany, 2008.

[2] Information Sciences Institute, University of California: The Network Simulator ns-2. http://www.isi.edu/nsnam/ns (valid in 2009).

[3] T. Kuhn, P. Becker. A Simulator Interconnection Framework for the Accurate Performance Simulation of SDL Models. In: Gotzhein, R., Reed, R. (Eds.), System Analysis and Modeling: Language Profiles, Lecture Notes in Computer Science 4320, Springer, 2006.

[4] T. Kuhn, A. Geraldy, R. Gotzhein, F. Rothländer: ns+SDL – The Network Simulator for SDL Systems. In A. Prinz, R. Reed, J.Reed (Eds.) SDL 2005 - Model Driven, proceedings of the 12th International SDL Forum. Lecture Notes in Computer Science 3530, Springer, Grimstad, Norway, 2005.