Konzept und Struktur des SDL Environment Frameworks (SEnF)

I. Fliege, A. Geraldy, S. Jung, T. Kuhn, C. Webel, C. Weber

Technischer Bericht 341/05

Konzept und Struktur des SDL Environment Frameworks (SEnF)

I. Fliege, A. Geraldy, S. Jung, T. Kuhn, C. Webel, C. Weber

Fachbereich Informatik, TU Kaiserslautern, Kaiserslautern
{fliege,geraldy,s_jung,kuhn,c_webel,c_weber}@informatik.uni-kl.de

Technischer Bericht 341/05

Fachbereich Informatik
TU Kaiserslautern
Postfach 3049
67653 Kaiserslautern

Konzept und Struktur des SDL Environment Frameworks – SEnF

Ingmar Fliege, Alexander Geraldy, Simon Jung,
Thomas Kuhn, Christian Webel, Christian Weber

{fliege,geraldy,s_jung,kuhn,webel,c_webe}
@informatik.uni-kl.de

AG Vernetzte Systeme
TU Kaiserslautern

25. November 2005

Zusammenfassung

SDL (Specification and Description Language) ist eine von der ITU genormte graphische, formale Beschreibungssprache für verteilte Systeme und Kommunikationsprotokolle. Um offene, in SDL spezifizierte Systeme, d. h. Systeme, die mit ihrer Umgebung interagieren, zu implementieren, ist eine Umgebungsschnittstelle erforderlich, die die Semantik des SDL-Signalisierungsmechanismus erfüllt. Bei Verwendung kommerzieller Werkzeugumgebungen sind manuelle Kodierungsschritte erforderlich, um diese Schnittstelle (auch Environment Functions genannt) bereitzustellen.

In diesem Beitrag wird das SDL Environment Framework (SEnF) vorgestellt. Dabei handelt es sich um eine erweiterbare Sammlung generischer Umgebungsschnittstellen für SDL-Systeme, die die automatische Anbindung an unterschiedliche Betriebssysteme und Hardwareplattformen unterstützt. Damit entfällt die sehr zeitaufwändige und fehleranfällige manuelle Kodierung der Environment Functions.

SEnF unterstützt gegenwärtig die Kommunikationstechnologien IEEE 802.11a/b/g (WLAN), IEEE 802.15.1 (Bluetooth), IEEE 802.15.4 (ZigBee), RS-232 (UART), und I^2C-Bus, diverse Ein-/Ausgabegeräte sowie die Betriebssysteme Windows NT/2000/XP und Linux.

1 Einleitung

Durch kommerzielle Werkzeuge, wie Telelogic Tau[1], ist es möglich, aus formal spezifizierten SDL-Systemen automatisch C-Code zu generieren und diesen nach der Kompilation auf einer Zielplattform auszuführen. Generell ist der C-Code der ein SDL-System beschreibt, abgesehen von einigen Listen und Mengenoperationen, plattformunabhängig. Die Schwachstelle stellt aber die Anbindung an die Umgebung und damit an die Zielplattform da. Zum einen müssen Funktionen der Zielplattform aufgerufen werden, um beispielsweise die Zeit in SDL darzustellen, und zum anderen ist eine Anbindung an Basistechnologien notwendig. Speziell in der Domäne der verteilten Kommunikationssysteme ist die

Anbindung von SDL-Systemen an Kommunikationstechnologien und Anwendungen erforderlich.

Unter Einsatz des SDL Environment Frameworks (SEnF) erfolgt eine automatische Anbindung von spezifizierten SDL-Systemen an die Umgebung. In SDL sind hierfür Signale vorgesehen, die an die Umgebung gesendet werden. Die Codegeneratoren von Telelogic und der Transpiler ConTraST [2] generieren aus SDL-Spezifikationen C bzw. C++-Code. Dieser kann von verschiedenen C-Compilern in ausführbare Anwendungen umgesetzt werden. Wird kein weiterer Eingriff durch den Benutzer vorgenommen, existiert jedoch keine Behandlung von Signalen an die Umgebung, d.h. diese werden verworfen, und es können auch keine Signale an das SDL-System verschickt werden.

Telelogic bietet mit dem *Targeting Expert* allerdings die Möglichkeit, bei der Codeerzeugung automatisch eine C-Datei zu generieren, in welcher Funktionen (*XInEnv* und *XOutEnv*) existieren, die eingehende und ausgehende Signale behandeln. Initial sind die einzelnen Abschnitte für die entsprechenden Signale leer und müssen vom Entwickler manuell ergänzt werden. Die vom *Targeting Expert* erzeugte Datei (component_env.c) bildet die Grundlage des SEnFs zur Interaktion mit dem SDL-System.

Im 2. Kapitel wird auf den grundsätzlichen Aufbau und die Ausführung von SDL-Systemen eingegangen. Anschließend wird im 3. Kapitel eine Einführung in die Ausführungssemantik und Interaktion mit der Umgebung gegeben. Kapitel 4 befasst sich kurz mit der Anbindung an den Simulator NS2 [5].

2 Implementierung von SDL-Systemen

2.1 Implementierungsstruktur

Aus vollständig spezifizierten SDL-Systemen kann durch die Codegeneratoren von Telelogic automatisch C-Code generiert werden. Dabei werden aus Systemen und Packages entsprechende C- und Header-Dateien erzeugt. Diese Dateien beschreiben die Struktur, die Variablen, die Transitionen und Signale des spezifizierten SDL-Systems. Das Versenden von Signalen, Process-Scheduling usw. ist in drei Dateien der Laufzeitumgebung im Telelogic-Verzeichnis beschrieben.

Die erzeugten Dateien bestehen neben dem C-Code aus einer Vielzahl von #defines und Macros für den Präcompiler. Aus diesem Grund ist der Code auch sehr schwer lesbar und zu debuggen, ermöglicht aber die Generierung effizienten Codes. Somit stehen drei Arten von Dateien zur Verfügung:

- Aus dem SDL-System automatisch generierte Dateien
- Die vorgegebene Laufzeitumgebung für SDL-96 [3] (SDL-Verhalten)
- Eine Umgebungsdatei zur Interaktion von SDL mit der Umgebung

Mit Hilfe des generierten Makefiles und einem C-Compiler kann ein ausführbares Programm generiert werden. Wenn die automatisch generierte Umgebungsdatei (component_env.c) durch eine C-Datei ersetzt wird, in welcher für vordefinierte

Signale an die Umgebung Code existiert, kann für jedes Signal ein entsprechendes Verhalten in der Umgebung ausgelöst werden und auch Signale an das SDL-System verschickt werden.

2.2 Die Laufzeitumgebung ConTraST

In der Arbeitsgruppe Vernetzte System wurde der Transpiler ConTraST [2] entwickelt, der eine Umsetzung von SDL96 in die Programmiersprache C++ vornimmt. Dabei wird auch die Anbindung von SDL-Systemen an eine Umgebung realisiert.

In der Laufzeitumgebung existieren Methoden, die der Signatur von Telelogic tau ähneln. Somit ist es auch hier möglich, Systeme an das SEnF zu koppeln. Die Umgebung wird in der Laufzeitumgebung nach Z.100 F3 [4] als eigenständiger *SdlAgent* mit entsprechendem Programm realisiert und kann damit die Interaktion mit der Umgebung realisieren. Die Anbindung des SEnF wird auch beim ConTraST durch einfaches Linken der entsprechenden Dateien realisiert.

2.3 Einbettung des SEnF in SDL-Systeme

Ausgangspunkt des SEnFs ist die Datei *SEnF.c*, welche die automatisch generierte Umgebungsdatei *component_env.c* ersetzt. Diese Datei implementiert den C-Code für eingehende und ausgehende Signale des SDL-Systems. Durch SDL-Signale, die an die Umgebung gesendet werden können, werden mit Hilfe des C-Präprozessors zusätzliche Code-Module eingebunden. Dabei stehen für eine Vielzahl von Technologien entsprechende Module zur Verfügung (siehe Kapitel 5). Abbildung zeigt die Implementierungsstruktur von SDL-Systemen und die durch das SEnF ermöglichte Anbindung an die Umgebung.

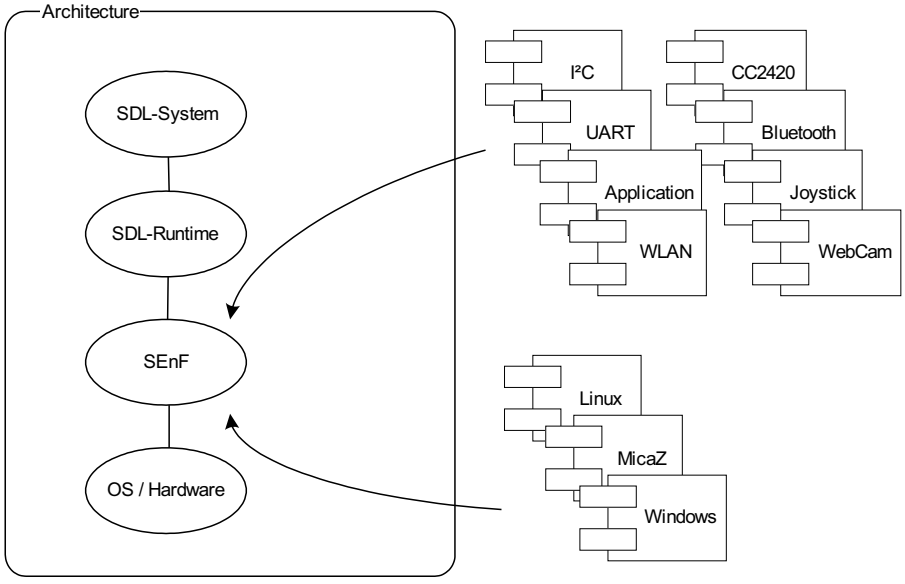

Abbildung 1: Architektur des SEnF

Die Codegeneratoren von Telelogic erzeugen neben dem eigentlichen C-Code die Datei *component.ifc*, welche eine Liste der Signale an die Umgebung in Form von Präprozessoranweisungen enthält. Diese Datei wird verwendet, um mit Hilfe der Präcompiler-Direktive #ifdef zusätzlichen C-Code aus anderen Dateien einzubinden.

So findet man in der Datei *SEnF.c* für das Beispiel WLAN die folgenden Anweisungen:

```
#ifdef WLAN_send
#include "wlan.c"
#endif
```

Durch diese Anweisungen wird automatisch der Quelltext der Datei *wlan.c* in die Datei *SEnF.c* eingebunden, wenn das SDL-Signal WLAN_send im SDL-System definiert wurde. Damit steht Code für das Senden und Empfangen von Paketen über WLAN zur Verfügung. Eine weitere Unterscheidung, ob die Codeabschnitte für das Senden oder das Empfangen eingebunden werden, wird in der eingebundenen Datei (in diesem Beispiel *wlan.c*) getroffen.

Da das Senden und Empfangen von Paketen auf unterschiedlichen Betriebssystemen und Hardwareumgebungen unterschiedlich funktioniert, findet man in der Datei *wlan.c* weitere Direktiven:

```
#ifdef SEnF_WINDOWS
#include "wlan_windows.c"
#endif
#ifdef SEnF_LINUX
...
```

Dadurch wird im Falle von Windows der C-Code für Windows in die Umgebungsdatei eingebunden. Zusätzlich werden je nach Betriebssystem Dateien eingebunden, die eine Abstraktion von aktuellem Betriebssystem und Laufzeitumgebung ermöglich. Es stehen Funktionen zur Prozesserzeugung (Create_Thread), Prozesssynchronisation (Create_Notifier, Notify, Wait-For_Notify, Create_Semaphore, Block usw.), zum Umgang mit Zeit (Sleep, SDL_Time), Speicherverwaltung (Malloc, Free) und Bildschirmausgaben (Message, Debug, Error) zur Verfügung. Durch die Verwendung von Compileroptionen ist es außerdem möglich, gezielt Teile des von Telelogic zur Verfügung gestellten C-Codes zu deaktivieren. Dieses wird zurzeit bei den Funktionen SDL_Time() und main() verwendet:

- Die Option XNOMAIN führt zu C-Code ohne main-Funktion. Damit fehlt der Code für das Scheduling der SDL-Prozesse, die Verwaltung der Timer und die Ausführung der Umgebung. Die Option wird beispielsweise bei der Anbindung an den NS-2 verwendet, bei welcher die Ausführung der SDL-Systeme durch den Simulator bestimmt wird. Eine entsprechende main-Funktion wird in diesem Fall durch das SEnF zur Verführung gestellt.
- Die Option XUSER_CLOCK_FUNC blendet die von Telelogic zur Verfügung gestellte Zeit-Funktion aus. Auch in diesem Fall wird durch das SEnF eine eigene Funktion bereitgestellt. Diese Option wird verwendet, wenn eine genauere Zeitbasis erforderlich ist, Telelogic für die Zielplattform keine Zeit ermitteln kann oder wenn der NS2-Simulator dem SDL-System die simulierte Zeit übermitteln soll.

3 Ausführung von SDL-Systemen

3.1 Ausführung der SDL-Systeme

Die Ausführung von SDL-Systemen erfolgt bei Telelogic auf der Basis weniger Funktionen. Es ist möglich abzufragen, ob Timer abgelaufen sind, ob Transitionen feuern können und es ist natürlich möglich, eine Transition zu feuern. Aus diesem Grund ist die main()-Funktion eines SDL-Programms sehr klein. Auch die Laufzeitumgebung, die zum ConTraST erforderlich ist, bietet diese Funktionen an.

Wird ein Signal an die Umgebung geschickt, wird die entsprechende Funktion *XOutEnv* in der C-Umgebungsdatei aufgerufen. Da das Eintreffen eines Signals aus der Umgebung nicht vorhergesagt werden kann, wird *XInEnv* nach jeder Transition aufgerufen, d.h. dem Entwickler wird nach dem Ausführen jeder Transition die Möglichkeit gegeben, dem System ein Signal zu schicken. Bei der Laufzeitumgebung der AG wird das entsprechende Programm des Agenten der Umgebung regelmäßig aufgerufen.

Es ist wichtig, dabei festzustellen, dass in den Umgebungsfunktionen niemals eine Blockierung erfolgen darf, da dies eine Blockierung des kompletten SDL-Systems zur Folge hätte. Aufrufe auf blockierende Betriebssystem-Befehle wie sleep, select oder ähnliches dürfen also nur in separaten Threads erfolgen. Aus diesem Grund ist im SEnF das Warten auf eingehende Ereignisse in eigenen User-Level-Threads realisiert.

Die Funktion *XInEnv* bietet allerdings die einzige synchronisierte Möglichkeit, mit dem SDL Signale auszutauschen. Beim Aufruf von *XInEnv* ist das SDL-System in einem definiertem Zustand, in welchem Signale in das SDL-System versendet werden dürfen. Ein asynchroner Zugriff beim Versenden von Signalen an ein laufendes SDL-System kann schwerwiegende Konsequenzen haben, denn das zeitgleiche Einfügen von Signalen aus der Umgebung in die Warteschlange von Prozessen kann den Verlust von Nachrichten oder einen Absturz des Programms verursachen.

Ein weiteres Problem stellt die hohe Prozessorauslastung dar, da das beschriebene Verhalten von main() in einer Endlos-Schleife ausgeführt wird. Es ist also wünschenswert, die CPU-Auslastung zu senken, aber trotzdem in der Lage zu sein, ein SDL-System mit voller Geschwindigkeit auszuführen und auf Ereignisse von Geräten (z.B. WLAN) zu warten. Aus diesem Grund wurde der folgende Mechanismus in der *XInEnv* entwickelt:

Beim Aufruf der *XInEnv* (nach jeder Transition) wird überprüft, ob es weitere Transitionen zu schalten gibt. Falls ja, wird die Funktion sofort verlassen. Dadurch ist es möglich, ein SDL-System mit voller Geschwindigkeit auszuführen. Natürlich wäre es möglich, dass das System in diesem Fall nie Signale aus der Umgebung erhält, jedoch ist die Frage zu stellen, ob ein SDL-System, welches bereits vollständig ausgelastet ist, mit zusätzlichen Signalen aus der Umgebung beschäftigt werden sollte. Dieses kann nur in Ausnahmefällen unter Beachtung von Prioritäten von Signalen Sinn machen.

Sind keine Transitionen bereit zu feuern, wird überprüft, wann der nächste Timer abläuft. Ist dieser Abstand größer als die Möglichkeit des Betriebssystems, die Ausführung eines Threads zu unterbrechen (Abhängigkeit bezüglich der Zeitauflösung des Betriebssystems), dann wird eine Warte-Funktion aufgerufen, die die Ausführung des Prozesses unterbricht, bis entweder die Zeit abläuft oder ein Ereignis an einem Gerät aufgetreten ist (z.B. WLAN-Paket empfangen). Im Moment liegt diese Zeit bei einer Millisekunde für Windows und Linux (ab Kernel 2.6.0). Wie festgestellt wird, ob Ereignisse an einem Gerät aufgetreten sind, wird im nächsten Abschnitt genauer beschrieben.

3.2 Synchronisation mit der Umgebung

Die ersten Versuche zur automatischen Anbindung an eine spezielle Hardwareplattform zeigten schnell, dass ein besonderes wichtiger Aspekt die Synchronisation zwischen der Hardware und dem SDL-System ist. Einerseits ist es notwendig eine enge Kopplung an die Hardware zu haben, um eine möglichst schnelle Reaktionszeit zu haben, und andererseits diese so zu gestalten, dass der Austausch von Signalen mit dem SDL-System keine Fehler verursacht. Das Versenden von Signalen in ein System ist nur zu bestimmten Zeitpunkten möglich. Die Modifikation einer Signalwarteschlange in SDL kann ansonsten zu unvorhersagbarem Verhalten führen oder einen Absturz zur Folge haben.

Aus diesem Grund wurde ein Template entwickelt, welches eine einheitliche Schnittstelle zwischen dem SEnF und den einzelnen Modulen beschreibt. In diesem Template wird auf Probleme mit der Synchronisation, Speicherverwaltung und der Synchronisation mir dem SDL-System eingegangen. Dieses Template wird bei der Entwicklung neuer Module verwendet und definiert Abschnitte, die der Entwickler manuell ergänzen muss:

- Aufruf eines Moduls beim Starten des Systems, bei dem Ressourcen reserviert werden können und Hardwareinitialisierung vorgenommen werden kann.
- Aufruf eines Moduls beim Beenden es Systems, bei dem Ressourcen freigegeben werden und Terminierungen vorgenommen werden können.
- Das Senden eines Signals, das vom SEnF in der *XOutEnv* ausgelöst wird.
- Der Empfang eines Signals, das durch einen Synchonisationsmechanismus mit dem SDL-Systems realisiert wird.

Jedem einzelnen Modul ist es gemäß diesem Template erlaubt einen eigenen, separaten Thread zu starten, um Daten aus der hardwarenahen Umgebung abzuwarten. Dieser beeinflusst jedoch den Ablauf des SDL-Sytems nicht. Ist das Versenden eines Signals an das SDL-System notwendig (z.B. der Empfang von Daten), wird mit Hilfe einer Signalisierung *Notify()* (siehe Abbildung 2) ein entsprechendes Ereignis gesendet. Damit wird dem Prozess des SDL-Systems signalisiert, dass dieser zu einem wohldefinierten Zeitpunkt ein Signal im SEnF generiert und in die Warteschlange eines SDL-Prozesses eingefügt werden muss. Diese Signalisierung wird zu einem Zeitpunkt ausgewertet, in dem das SDL-System keine Transitionen feuert (bzw. kein Programm des Systems ausführt)

und damit kein Konflikt mit den Warteschlangen des SDL-Systems zu erwarten ist.

Die Signalisierung wird unter Linux und Windows mit sogenannten Named Pipes realisiert. Ein entsprechendes Modul (z.B. WLAN) ruft nach dem Empfang von Daten eine vordefinierte Funktion SEnF_Notify() auf und weckt damit bei Bedarf die *XInEnv* des SDL-Systems.

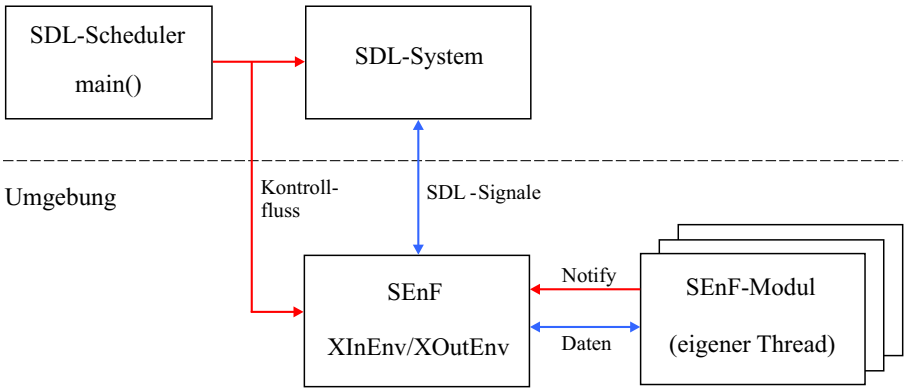

Abbildung 2: Interaktion von SDL-System und Umgebung

Nach dem Feuern der SDL-Transitionen wird der Kontrollfluss, durch den Aufruf von *XInEnv*, an die SDL-Umgebung abgegeben. In der Umgebungsfunktion wird überprüft, ob ein SEnF-Modul ein SEnF_Notify() geschickt hat. In diesem Fall stehen neue Daten zur Verfügung, die in das SDL-System geschickt werden sollen. Zunächst wird ein neues SDL-Signal generiert und anschließend, durch einen Funktionsaufruf im entsprechenden SEnF-Modul, die Daten als Parameter des Signals eingetragen. Das nun vollständige SDL-Signal wird in die Warteschlange des SDL-Systems eingetragen und kann zu einem späteren Zeitpunkt einem SDL-Prozess zugestellt werden. Der Kontrollfluss wird nach diesem Vorgang an das SDL-System zurückgegeben.

4 Anbindung an den NS-2

Zur Anbindung eines SDL-Systems an den NS-2 Simulator [5], sind folgende Schritte notwendig. Durch die Optionen XNOMAIN und XUSER_CLOCK_FUNC wird in die Ausführung bzw. Scheduling der SDL-Prozesse eingegriffen, und der Simulator ist in der Lage, dem SDL-System eine eigene Zeitbasis *now()* zur Verfügung zu stellen. Zusätzlich wird für die Signale WLAN_send und WLAN_recv ein Code eingebunden, welcher die auszutauschenden Pakete an den NS-2 Simulator übermittelt.

Beim Start einer Simulation wird der NS-2 incl. Konfigurationsskript gestartet, welcher wiederum für jeden einzelnen Knoten (SDL-System) einen eigenen Betriebssystemprozess startet. Der Simulator und die einzelnen SDL-Systeme werden über Named Pipes verbunden, so dass sie interagieren und Signale austauschen können.

5 Unterstützte Umgebung

Zurzeit wird durch das SEnF die Anbindung an die folgenden Plattformen unterstützt:

- Windows (ab Windows 2000)
- Linux ab Kernel 2.4
- NS-2 Simulator
- MICAz
- Windows Mobile [PDA] (in Arbeit)

Dabei können die folgenden Technologien verwendet werden:

- WLAN (Windows, Linux, Simulator, Windows Mobile)
- Application-Interface (Windows, Linux)
- Serielle Schnittstelle RS232 (Windows, Linux)
- Bluetooth (Windows, Linux)
- I^2C-Bus (Linux)
- Joystick (Windows, Linux)
- Videokamera (Windows, Linux)
- CC2420 (MICAz)

Außerdem werden durch das SDL Environment Framework C-Funktionen mit dazugehörigen SDL-Packages bereitgestellt, die die Erzeugung von Zufallszahlen und das Generieren von Log-Dateien ermöglichen.

6 Zusammenfassung

In diesem Bericht wurde das in der AG Vernetzte Systeme entwickelte SDL Environment Framework (SEnF) vorgestellt. Es ermöglicht, ohne manuelle Eingriffe des Entwicklers, die Anbindung von SDL-Systemen an vordefinierte Betriebssysteme und Hardwareumgebungen. Das SDL Environment Framework wird seit über einem Jahr in der Lehre und verschiedenen Projekten (Demonstratoren) erfolgreich eingesetzt. Dabei wird die Entwicklung von Kommunikationssystemen deutlich erleichtert, da die zuvor schwierige und fehlerbehaftete Anbindung an die Umgebung automatisch erfolgt.

Das SDL Environment Framework wird kontinuierlich weiterentwickelt, um neue Plattformen (WindowsCE), Hardware (Mikrokontroller) und weitere Codegeneratoren zu unterstützen. Desweiteren werden verschiedene Möglichkeiten untersucht, wie das Debuggen zur Laufzeit und im Simulator verbessert werden kann.

Literatur

[1] Telelogic: Tau 4.5 SDL Suite, 2003

[2] Christian Weber: Entwurf und Implementierung eines konfigurierbaren SDL Transpilers für eine C++ Laufzeitumgebung, Diplomarbeit TU Kaiserslautern, September 2005

[3] ITU-T Recommendation Z.100 (11/99): Specification and Description Language (SDL), International Telecommunication Union (ITU), 1999

[4] ITU-T Recommendation Z.100 F3 (11/00): Specification and Description Language (SDL), International Telecommunication Union (ITU), 2000

[5] Thomas Kuhn, Alexander Geraldy, Reinhard Gotzhein, Florian Rothländer: ns+SDL - The Network Simulator for SDL Systems. SDL Forum 2005: 103-116