

# Echtzeit-Linux mit Xenomai

## Das noch junge Projekt Xenomai ist aus der Echtzeit-Erweiterung RTAI hervorgegangen

Mehrere Echtzeit-Erweiterungen für Linux konkurrieren um die Gunst der Anwender. Xenomai zeichnet sich durch seine „Skins“ aus. Das sind Schichten, die die API-Aufrufe verschiedener Echtzeit-Betriebssysteme emulieren. Dadurch wird die Portierung bestehender Anwendungen auf Xenomai besonders einfach.

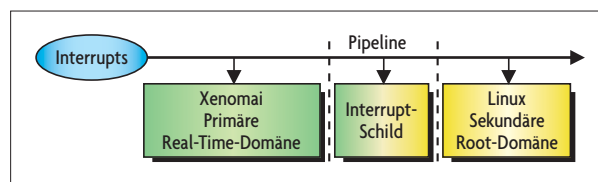
Von Sebastian Smolorz

Bereits seit einiger Zeit ist auf dem Gebiet der Automatisierungstechnik eine wachsende Akzeptanz des frei verfügbaren Betriebssystems Linux zu beobachten. Die Erfolgsgeschichte von Linux auf dem Server-Markt, auf dem sich die Open-Source-Alternative einen festen Platz erobert hat, scheint sich im Automatisierungsumfeld wiederholen zu können. In Umfragen äußern sich befragte Unternehmen zufrieden bezüglich ihrer Erfahrungen mit Embedded-Linux.

Für zahlreiche Steuerungs- und Regelungsaufgaben in der Industrie sind Echtzeit-Systeme eine unabdingbare Voraussetzung. In diesem Bereich herrschen immer noch proprietäre Betriebssysteme vor, die zu harter Echtzeit fähig sind. Linux hat hier einen schweren Stand, weil es traditionell auf Durchsatz und Gesamtleistung getrimmt ist. Weiche Echtzeit-Anforderungen kann Linux im Allgemeinen gut erfüllen. Die erreichbaren Reaktions- und Latenzzeiten hängen dabei stark von der Prozessorarchitektur und der Systemauslastung ab. Für Aufgaben mit deterministischem Zeitverhalten ist Linux allerdings nicht geeignet. Nur ein Echtzeit-Betriebssystem kann die absolute Einhaltung zeitlicher Vorgaben garantieren.

Die Herausforderung, Linux zu harter Echtzeit-Fähigkeit zu verhelfen, ist

somit nicht neu. Seit vielen Jahren beschäftigen sich mehrere Open-Source-Projekte mit diesem Thema. Größere Bekanntheit dürften RTLinux sowie RTAI genießen. Xenomai ist aus dem ehemaligen RTAI/fusion hervorgegangen und war bis Oktober 2005 der Entwicklungszweig von RTAI, welcher ursprünglich in Version 4.0 des RTAI-Projekts münden sollte. Da sich die Hauptentwickler in wichtigen technischen Fragen aber nicht einigen konnten, wurde das Xenomai-Projekt selbstständig. Der auf diese Weise gelöste Konflikt um RTAI/fusion kann als Beispiel dafür gewertet werden, dass Open Source den Wettbewerb um die beste Lösung begünstigt und so zu technischem Fortschritt führt.



**Bild 1.** Im Mittelpunkt der Adeos-Architektur steht die Event-Pipeline, die den Domänen Linux und Xenomai eine gemeinsame Nutzung kritischer Hardware-Ressourcen erlaubt.

Xenomai verfolgt den so genannten Dual-Kernel-Ansatz. Dabei wird der Linux-Kernel nur so weit modifiziert, dass er nahtlos mit dem Xenomai-Mikrokern zusammenarbeitet. Letzterer verfügt über einen eigenen Scheduler, der für die Ausführung der harten Echtzeit-Tasks verantwortlich ist und deshalb jederzeit in der Lage sein muss,

den Linux-Kernel in seiner Ausführung zu unterbrechen. Dagegen dürfen weder der Linux-Scheduler noch die Linux-eigenen Interrupt-Handler von sich aus Xenomai zu einer Unterbrechung zwingen, weil sonst ein deterministisches Verhalten der Echtzeit-Tasks nicht mehr gewährleistet wäre. Linux wird somit zu einem untergeordneten Teil des Gesamtsystems, dem in wichtigen Bereichen wie der Interrupt-Verwaltung die Kontrolle über die Hardware entzogen wird. Um dies zu erreichen, bedient sich Xenomai der Dienste von Adeos/I-pipe.

### ■ Adeos „entmündigt“ Linux

Adeos (Adaptive Domain Environment for Operating Systems) ist streng genommen kein Bestandteil von Xenomai, aber die Grundlage für seine Echtzeit-Fähigkeit. Als ein Patch für den Linux-Kernel übernimmt Adeos den direkten Zugriff auf einige elementare hardware-nahe Funktionen und virtualisiert sie für den Rest des Systems, das in so genannte Domänen eingeteilt wird. Linux stellt grundsätzlich die Root-Domäne dar, während Xenomai eine eigene Domäne für sich beansprucht.

Im Mittelpunkt der Adeos-Architektur steht die „Event-Pipeline“, die den Domänen eine gemeinsame Nutzung kritischer Hardware-Ressourcen erlaubt. Dabei handelt es sich vor allem um Hard- und Software-Interrupts – daher der Namenszusatz „I-pipe“ für „Interrupt Pipeline“. Die Interrupts werden den Domänen über die Pipeline in einer festen Reihenfolge zugeführt, die sich aus den Prioritäten der Domänen ergibt.

Aus **Bild 1** wird ersichtlich, dass Xenomai in der Rolle der primären Domäne von der Event-Pipeline als erstes berücksichtigt wird. Hat sich ein Treiber oder eine Task in der Echtzeit-Domäne mit einem Interrupt-Handler für einen Interrupt registriert, wird die-

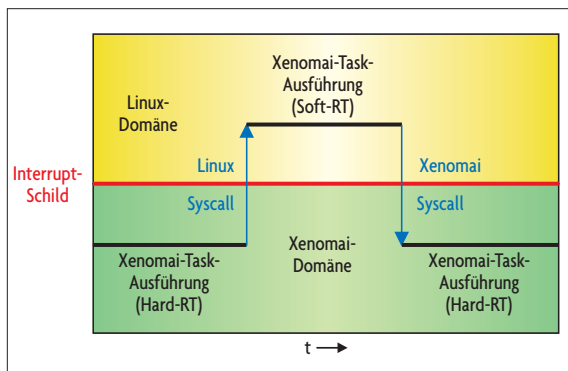
ser nach seinem Auftreten von dem Handler verarbeitet. Am Ende seiner Ausführung entscheidet der Interrupt-Handler, ob der Interrupt von der Pipeline an die nächste Domäne weitergegeben wird. Bestimmte Interrupts können also vor Linux verborgen werden, wenn sie exklusiv in der primären Domäne behandelt werden sollen. Interrupts, die keinen Handler in der Echtzeit-Domäne vorfinden, werden von der Pipeline weitergereicht.

Der Linux-Kernel nutzt an vielen Stellen die Möglichkeit, Interrupts auf

die exklusive Kontrolle über RT-Tasks im Userspace, die sich analog zu den Bezeichnungen der Domänen in Bild 1 entweder im *primären* Modus – dann sind sie hart echtzeitfähig – oder im nicht deterministischen *sekundären* Modus befinden.

Eine Task bleibt so lange im sekundären Modus, bis sie einen API-Aufruf an den Xenomai-Kernel startet, der einen Übergang in den primären Modus erfordert. Dem Linux-Scheduler wird die Kontrolle über eine solchermaßen migrierende Task wieder entzogen

(Bild 2). Da die Wechsel in beide Richtungen völlig transparent von Xenomai durchgeführt werden und der Programmierer einer RT-Applikation keine expliziten Befehle dafür bemühen muss, sollte man ein besonderes Augenmerk auf diejenigen Code-Abschnitte legen, die harte Echtzeit benötigen. Linux-Systemaufrufe



**Bild 2.** Wenn Xenomai-Tasks Systemaufrufe starten müssen, die den Echtzeit-Ablauf blockieren könnten, werden diese in einen sekundären Modus mit geringerer Priorität überführt.

Hardware-Ebene abzuschalten. Um das ganze Konzept der Interrupt-Pipeline dadurch nicht nutzlos werden zu lassen, modifiziert Adeos diese Funktionen des Linux-Kernels so, dass sich das Abschalten und Reaktivieren von Interrupts nur noch auf die Linux-eigene Domäne auswirkt.

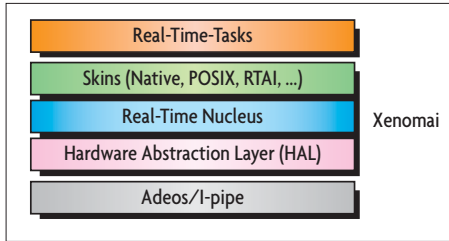
### Interrupt-Schild verbindet Linux- und Xenomai-Domäne

Der so genannte Interrupt-Schild befindet sich als Domäne zwischen Xenomai und Linux und ist in der Standardkonfiguration abgeschaltet. Seine Aufgabe beruht auf der nahtlosen Integration von Xenomai-Tasks, die im nicht privilegierten Userspace laufen, in die Linux-Umgebung. Solche Tasks können nicht nur RT-Funktionen (Abk. für Real-Time) unter harten Echtzeit-Bedingungen nutzen, sondern auch herkömmliche Systemaufrufe an den Linux-Kernel absetzen. Erreicht wird dies durch die automatische Migration der RT-Tasks in die Linux-Domäne. Xenomai und Linux teilen sich also

sind hier nicht erlaubt. Xenomai unterstützt das Aufspüren von ungewollten Domänenwechseln einer RT-Task mit zwei Instrumenten: Erstens kann eine Task so eingerichtet werden, dass sie bei jedem Übertritt vom primären in den sekundären Modus mittels des Signals *SIGXCPU* benachrichtigt wird, und zweitens gibt ein Blick in die Datei */proc/xenomai/stat* in der Spalte MSW Aufschluss über die Anzahl der erfolgten „Mode Switches“ laufender RT-Tasks.

Auf der anderen Seite erleichtert die automatische Umschaltung zwischen hartem und weichem Echtzeit-Modus die Entwicklung einer Gesamtapplikation, die immer auch NRT-Anteile (Abk. für non-Real-Time) enthält. In folgenden Fällen ist eine Nutzung von Linux-Funktionen, die im sekundären Modus ausgeführt werden, sinnvoll:

► **Initialisierung:** In der Anfangsphase eines Programms können alle von den RT-Threads benötigten Ressourcen vorbereitet werden, zu deren Bereitstellung Linux-Systemaufrufe notwen-



**Bild 3.** Xenomai ist in Schichten aufgeteilt. Da der HAL die hardware-abhängigen Details enthält, zeichnet sich der Echtzeit-Kern Nucleus durch Hardware-Neutralität aus.

dig sind, die entweder nicht streng deterministisch oder zeitaufwendig sind.

- ▶ Datenweiterverarbeitung und -visualisierung: Eine typische gemischte RT-NRT-Applikation besteht aus einem RT-Thread, der Messwerte aufnimmt, und einem weiteren Thread, der diese Werte weiterverarbeitet und keine zeitkritischen Aufgaben zu erledigen hat. Xenomai bietet dem Programmierer für die Datenübergabe mehrere IPC-Mechanismen (Interprozesskommunikation) an, die zu keiner Migration des Mess-Thread in den sekundären Modus führen, den zweiten Thread aber zeitweilig in den primären Modus versetzen.

- ▶ Debugging und Fehlermeldungen: In einem RT-Thread kann es zu Testzwecken sinnvoll sein, vorübergehend Funktionsaufrufe wie printf(), welcher zum Verlassen des primären Modus

führt, einzusteuern, um sich einzelne Werte anzeigen zu lassen. In gleicher Weise können Fehlersituationen, die den Abbruch eines Programms oder andere nicht deterministische Ausnahmebehandlungen einleiten, komfortabel an den Benutzer gemeldet werden.

Die weichen Echtzeit-Bedingungen, denen eine migrierte RT-Task in der sekundären Domäne ausgesetzt ist, können durch den Einsatz des Interrupt-Schildes verbessert werden. Als Domäne vor dem Linux-Kernel angeordnet, verhindert der Schild die Weiterleitung von Interrupts, solange eine Xenomai-Task im sekundären Modus ausgeführt wird. So wird sichergestellt, dass der Linux-Kernel solche Tasks nicht durch asynchrone Aktivitäten unterbricht.

### ■ Xenomai kann APIs anderer Echtzeit-Betriebssysteme emulieren

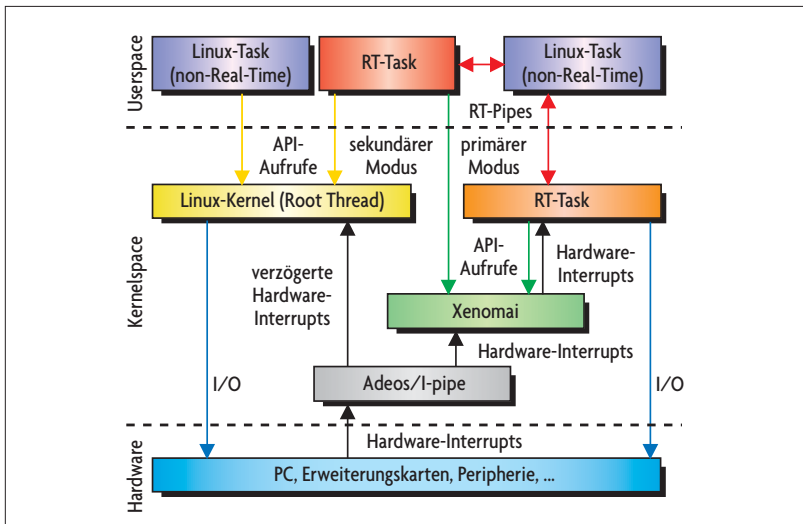
Eine besondere Eigenschaft von Xenomai stellt das Angebot verschiedener Real-Time-APIs dar. Es umfasst sowohl eine eigenes, natives API als auch Emulatoren von APIs traditioneller Echtzeit-Betriebssysteme wie VRTX, VxWorks und pSOS+. Mög-

lich macht dies ein abstrakter RTOS-Kern, über dem die einzelnen APIs als so genannte Skins angesiedelt sind. Der „Nucleus“ benannte Echtzeit-Kern exportiert einen Satz generischer Echtzeit-Dienste und -Funktionen, auf die die Skins zugreifen und in ihren API-Funktionen spezialisieren. Die Portierung bestehender Echtzeit-Applikationen für ein Betriebssystem, dessen API Xenomai emuliert, wird so wesentlich erleichtert.

Wie **Bild 3** zeigt, ist Xenomai in drei Schichten aufgeteilt. Jede davon nimmt die Dienste der unter ihr liegenden Schicht über definierte Schnittstellenfunktionen in Anspruch. Die unterste Xenomai-Schicht, der Hardware Abstraction Layer (HAL), kommuniziert direkt mit Adeos. Im HAL sind alle hardware-abhängigen Details verborgen, so dass sich insbesondere der Nucleus durch Hardware-Neutralität auszeichnet. Bei Portierungen von Xenomai auf andere Rechnerplattformen

Architektur	Subarchitektur
x86	i386, ia64
PowerPC	ppc, ppc64
ARM	PXA, SA1100, Integrator, S3C24xx, iMX21, AT91
Blackfin	BF53x

**Tabelle 1.** Von Adeos und Xenomai unterstützte Hardware



**Bild 4.** Xenomai/Linux-System. Der Linux-Kernel wird von Xenomai als Echtzeit-Task niedrigster Priorität behandelt. Wenn keine andere Echtzeit-Task ansteht, kann er die non-Real-Time-Task ausführen.

men gewinnt dieser Aspekt zusätzlich an Bedeutung. Welche Architekturen Xenomai und damit implizit Adeos/I-pipe aktuell unterstützen, kann **Tabelle 1** entnommen werden.

Echtzeit-Applikationen unter Xenomai verwenden grundsätzlich die APIs der Skins und nicht direkt die Nucleus-Funktionen. Im Allgemeinen sollten RT-Tasks als Userspace-Programme programmiert werden, weil nur hier der Speicherschutz greift und auch das Debugging bequemer ist. Andererseits ist im Kernspace der direkte Hardware-Zugriff möglich. Echtzeit-Treiber werden deshalb im Kernspace und mit Hilfe des RTDM-Skins (Real-Time Driver Model) umgesetzt, der ebenfalls Bestandteil von Xenomai ist. Da im Kernspace auch geringere Latenzzeiten als im Userspace erreicht werden können, ist eine Implementierung von RT-Tasks als Kernelmodule in seltenen Fällen gerechtfertigt. Die Xenomai-Skins bieten dazu in beiden Adressräumen identische APIs an und gewährleisten so eine konsistente und vereinfachte Programmierung.

**Bild 4** zeigt den schematischen Aufbau eines integrierten Xenomai/Linux-Systems. Der Linux-Kernel erfährt vom Xenomai-Scheduler eine besondere Behandlung. Als spezielle Echtzeit-Task („Root-Thread“) der niedrigsten Priorität kommt er immer dann zur Ausführung, wenn keine andere Echtzeit-Task lauffähig ist. In diesem

Fall findet ein Wechsel in die sekundäre Domäne statt. Der Linux-Kernel kann nun alle aufgelaufenen, von der Event-Pipeline an ihn weitergeleiteten Interrupts bearbeiten und die NRT-Tasks ausführen.

#### ■ RT-Pipes transportieren Informationen zwischen Xenomai und Linux

Interprozesskommunikationsmechanismen (IPC) des Linux-Kernels können von RT-Tasks, die mit normalen Linux-Prozessen Informationen austauschen wollen, nur im sekundären Modus genutzt werden. Xenomai bietet für die Kommunikation zwischen RT-Tasks im primären Modus auf der einen und Linux-Prozessen auf der anderen Seite aber auch die RT-Pipes an. Sie stellen einen nach dem FIFO-Prinzip funktionierenden Informationskanal dar, auf den von beiden Domänen lesend und schreibend zugegriffen werden kann. Die Echtzeit-Seite ruft dazu API-Funktionen eines Skins auf, der RT-Pipes unterstützt, z.B. des Native Skins. Ein Linux-Prozess kann hingegen eine der Gerätedateien `/dev/rtp0-31` öffnen und die üblichen Lese- und Schreiboperationen auf sie anwenden.

Xenomai bietet Applikationen Ersatz und Ergänzungen von Diensten an, die in ihrer Linux-Variante nicht deterministisch und damit für Echtzeit-Programme unbrauchbar sind. Da-

Testmodus	keine NRT-Last			mit NRT-Last (dd if=/dev/zero of=/dev/null und ping -f)		
	min	avg	max	min	avg	max
Usermode-Task	15,294 µs	17,294 µs	98,352 µs	64,588 µs	155,058 µs	231,176 µs
Kernel-basierte Task	7,294 µs	9,049 µs	96,941 µs	7,764 µs	50,730 µs	149,764 µs
In-Kernel Timer Handler	2,353 µs	2,975 µs	70,235 µs	2,235 µs	17,200 µs	120,471 µs

**I** Tabelle 2. Xenomais Latency-Test lief jeweils 15 Minuten mit einer Messfrequenz von 1 ms auf einem Samsung S3C2440 mit 306 MHz. Dargestellt sind die minimale, die durchschnittliche und die maximale Latenz jedes Testlaufs.

bei handelt es sich vor allem um IPC-Mechanismen wie Semaphoren und Mutexe, aber auch die RT-Task-Verwaltung oder das dynamische Speichermanagement. Einzelheiten können der umfangreichen API-Dokumentation von Xenomai entnommen werden, die alle Konzepte und Funktionen sehr genau beschreibt.

Generell sind bei der Erstellung eines Echtzeit-Programms folgende Grundsätze zu beachten:

- ▶ Gleich zu Beginn sollte mit dem Befehl `mlockall(MCL_CURRENT | MCL_FUTURE)`; sichergestellt werden, dass keine Speicherseiten des gesamten Prozesses ausgelagert werden dürfen.

- ▶ Dynamisches Anfordern von Speicherbereichen sollte innerhalb von RT-Threads nur mit Hilfe der entsprechenden Xenomai-Funktionen stattfinden. Die Größe des so genannten Memory Heap, von dem Xenomai dynamisch angeforderten Speicher entnimmt, ist standardmäßig auf 128 Kbyte eingestellt, kann aber während der Kernelkonfiguration geändert werden. Wenn nicht unbedingt die Notwendigkeit besteht, während einer harten Echtzeit-Phase Speicher anzufordern, ist eine

vorsorgliche Speicherreservierung in der Initialisierungsphase eines Programms mittels herkömmlicher Linux-Funktionen die bessere Alternative.

- ▶ Um unter harten Echtzeit-Bedingungen mit anderen Threads zu kommunizieren, stehen die verschiedenen Xenomai-Synchronisationsmethoden und IPC-Mechanismen bereit.

### ■ Linux-Tasks beeinflussen Latenz von RT-Tasks

Die Testsuite von Xenomai enthält unter anderen den Latency-Test zum Messen der Scheduling-Latenz einer periodisch ausgeführten harten RT-Routine. Wählbar sind drei Messmodi: Im ersten Fall handelt es sich um eine Mess-Task im Userspace, im zweiten um eine Kernelspace-Task und im dritten Fall um einen Timer-Handler. In allen drei Modi wird mit einer zu Beginn des Programms anzugebenden Frequenz die Task bzw. der Handler periodisch aufgeweckt und die Differenz zwischen aktueller Zeit und Sollzeit gemessen. Da für RT-Anwendungen naturgemäß der Worst-Case ausschlaggebend ist und ein Echtzeit-Betriebssystem unabhängig von jeglicher NRT-

Last die RT-Tasks rechtzeitig zum Zuge kommen lassen muss, hat Xenomais Latency-Test bei einem ausgelasteten Linux-System die größte Aussagekraft. Wie aus **Tabelle 2** zu entnehmen ist, haben NRT-Aufgaben auf dem S3C2440-Prozessor (ARM920T-Kern) durchaus merkbaren Einfluss auf die Latenz von RT-Tasks, die zudem deutlich größer und damit schlechter ausfällt als auf x86-Maschinen ähnlicher Taktfrequenz. Die Verantwortung hierfür trägt der ARM-Cache. Im Gegensatz zu einem x86-Prozessor werden bei der ARM9-Architektur die virtuellen Adressen im Cache zwischengespeichert. Deswegen tritt nach einem Kontext-Switch zunächst eine Reihe von Cache-Misses auf, weil die virtuellen Adressen im neuen Kontext ungültig sind. Das erhöht die Latenz nach Task-Wechseln massiv.

Es gibt zwei vielversprechende Ansätze, um die in der ARM-Hardware begründeten Unzulänglichkeiten software-seitig zu umgehen und so die ausufernden Latenzzeiten in den Griff zu bekommen. Dafür sind jedoch tiefgreifende und hardware-nahe Modifikationen am Kernel notwendig, die Xenomai noch nicht unterstützt. Sollte es dazu kommen, sind Verbesserungen der Latenzzeiten um den Faktor 4 bis 9 zu erwarten. *jk*

### Xenomai und Echtzeit im Standard-Kernel

Neben Xenomai oder RTAI gibt es Bestrebungen, den Linux-Kernel nativ echtzeitfähig zu machen. Dazu sind bereits Teile des Realtime Preemption Patch (RT-Preempt) in den offiziellen Kernel eingeflossen, andere harren noch ihrer Integration (siehe Seite 58ff. in diesem Heft). Über kurz oder lang wird der Linux-Kernel aber von sich aus harte Echtzeit bieten können. Das Xenomai-Projekt hat sich auf diese Entwicklung eingestellt und plant,

einen Kernel mit RT-Preempt-Fähigkeiten alternativ zu einem I-pipe-erweiterten zu unterstützen. Xenomai wird durch RT-Preempt nicht überflüssig, da die hervorstechendste Eigenschaft von Xenomai die Skin-Palette mit ihren unterschiedlichen RT-APIs ist. Eine Echtzeit-Applikation, die heute für Xenomai geschrieben wird, wird später auch mit einem RT-Preempt-Unterbau anstelle des I-pipe-Patches funktionieren.



**Sebastian Smolorz**

studiert an der Leibniz Universität Hannover Elektrotechnik/Technische Informatik und beschäftigt sich seit knapp zwei Jahren mit RTAI/fusion bzw. Xenomai. Er entwickelte zu großen Teilen den Xenomai-CAN-Stack RT-Socket-CAN. Bei der emlix GmbH in Göttingen betreut er Embedded-Linux-Projekte mit Echtzeit-Anforderungen und zeichnet für die Portierung von Adeos auf Samsungs Mikrocontroller S3C2440 sowie die Vervollständigung des I-pipe Tracers für ARM verantwortlich. [ssm@emlix.com](mailto:ssm@emlix.com)