Zeus Cell Library

Zeus Framework

Autor	Benjamin Hadorn
E-Mail	bhadorn@swissinfo.org
Ablage/Website	http://www.xatlantis.ch
Datum	12.03.07
Version	0.4.1

Inhaltsverzeichnis

1 Einleitung

2 Das Klassen-Framework

Zur Entwicklung das Frameworks wurde ein eigenes Klassen-Framework aufgebaut. Es ermöglicht das Entwickeln und Betreiben von Applikationen in Linux und Windows. Das Klassen-Framework umfasst:

- Die Zellenarchitektur nach CCM
  - Zellen
  - Registrierungstellen

2.1 Die Zellenarchitektur

Durch die Zellenarchitektur erhält das System seine Dynamik, während mit X-Objekten die statische Struktur festgelegt wird. Zur Implementation der Zellen sind weitere Konzepte und Verfahren nötig, welche in diesem Kapitel mit den dazugehörigen Klassen beschrieben werden.

2.1.1 Die Zellen

Für das Zeus-Framework wurde eine Zellenumgebung (Klasse `TCellEnvironment`) entwickelt, welche eine beliebige Zellenart beherbergen kann (Abbildung 1). Die Zellenumgebung erbt von der MOM Klasse `TModule` und übernimmt die Kontrolle- und Steuerfunktion eines bestimmten Zellensystems:

- Verwalten der Kommunikationskanäle
- Überwachen der Zellen
- Vermehren der Zellen
- Abbauen von Zellen

Die entsprechenden Zellen werden in Code-Modulen (Bibliotheken) entwickelt. Diese werden zur Laufzeit vom Framework dynamisch geladen damit die Zellenobjekte erzeugt werden können.

API	
`ICellKernel`	Schnittstelle für den Zellkern. Diese muss von einem Applikationsentwickler implementiert werden. `zeusbase/CellModel/Interfaces/ICellKernel.hpp`
`ICellComm`	Schnittstelle des Zellenmantels. Durch diese Schnittstelle können Meldungen aus der Zellenumgebung gelesen oder Meldungen abgesetzt werden. `zeusbase/CellModel/Interfaces/ICellComm.hpp`
`ICell`	Interne Schnittstelle für die Diagnose von Zellen. Sie repräsentiert den äusseren Zellenmantel. `zeusbase/CellModel/Interfaces/ICell.hpp`
`ICellSurvey`	Schnittstelle der Überwachungszelle. `zeusbase/CellModel/Interfaces/ICellSurvey.hpp`
`ILocalPipeRegistry`	Schnittstelle der Registrierungsstelle für lokale und entfernte Kommunikationskanäle. `zeusbase/CellModel/Interfaces/ILocalPipeRegistry.hpp`
`TCellEnvironment`	Umgebung einer oder mehreren Zellen. Die Umgebung erstellt Kommunikationskanäle und registriert die Zellen für lokale oder entfernte Kommunikation. `zeusbase/CellModel/CellEnvironment.h`
`TCell`	Diese Klasse repräsentiert eine Zelleninstanz. Sie stellt Diagnosemöglichkeiten zur Verfügung. Sie besitzt die Fähigkeit des Klonens. `zeusbase/CellModel/Cell.h`
`TCellSurvey`	Diese Zelle ist spezialisiert, um ein Zellensystem zu überwachen und Daten von Zellen auszuwerten. `zeusbase/CellModel/CellSurvey.h`
`TAbstractCell`	Hilfsklasse zur Implementation eines Zellkerns. Dieser Zellkern wird vom Applikationsentwickler geschrieben. `zeusbase/CellModel/AbstractCell.h`
`TRelyCell`	Umleitung von Meldungen. Diese Zelle kann als Gateway verwendet werden, um Meldungen zu filtern. `zeusbase/CellModel/RelayCell.h`
`TLocalPipeRegistry`	Lokale Registrierungsstelle für Kommunikationskanäle. Dadurch können Kanäle lokal von anderen Zellen gefunden werden (lookup). Die Kanäle können aber auch im Namensdienst global registriert werden.

API	
	`zeusbase/CellModel/LocalPipeRegistry.h`

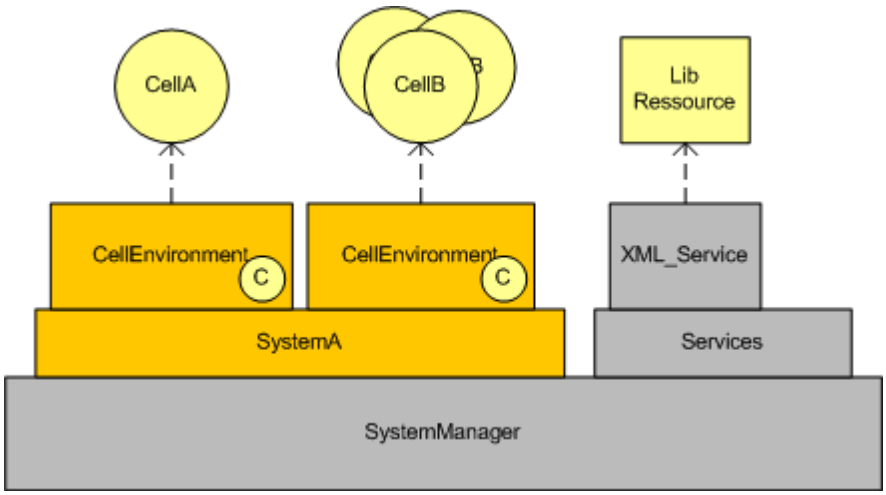

Abbildung 1: Verwendung des MOM. Bilden von hierarchischen Zellensystemen

Das `TCellEnvironment` implementiert die Möglichkeit Zellen zu klonen, und abzubauen. Diese 2 Verfahren sind im Dokument [CCM1] beschrieben.

2.1.2 Kommunikation der Zellen

Will eine Zelle mit einer anderen Zelle kommunizieren, muss zuerst ein Kommunikationskanal erstellt werden. Durch diesen Kanal werden dann Arbeitspakete und Kommandos gesendet.

Das Zellenumgebung verwaltet alle Kommunikationskanäle, durch welche die Zellen zu Informationen gelangen können (Input Pipes). Diese Art von Kanälen werden in einer lokalen und einer globalen Registrierungsstelle (Namensdienst) registriert, damit andere Zellen Meldungen an dieses Zellensystem senden kann.

Die Zellenumgebung holt sich die ausgehenden Kommunikationskanäle (Output Pipes) von den Registrierungsstellen. Aus Gründen der Effizienz können diese ausgehenden Kanäle auch in der Zellenumgebung zwischengespeichert werden. Das Suchen anderer Kommunikationskanäle

wird mehrstufig durchgeführt:

1. Zuerst wird versucht die Zelle lokal zu finden. Das geschieht durch das Suchen einer Zelle im Framework mit Hilfe der lokalen Registrierungsstelle. Wird die Zelle gefunden, wird ein interner Kommunikationskanal erstellt (über den Shared Memory).

2. Wird die Zelle nicht gefunden, wird eine Registrierungsstelle im LAN (Local Area Network) abgefragt. Unter Umständen gibt es mehrere Zellen mit dem gleichen Namen, die aber auf verschiedenen Host-Rechner laufen. Die Registrierungsstelle gibt alle gefunden Zellen zurück, die in Frage kommen könnten. Die Anzahl der Rückgaben kann eingeschränkt werden, wenn viele gleichnamige Zellensysteme eingesetzt werden.

3. Werden entfernte Zellen gefunden, kann mit jeder oder einer gewissen Anzahl der Zellen eine Verbindung hergestellt werden (Netzwerk-Kommunikationskanal).

So kann zum Beispiel die Zelle A mit mehreren Zellen vom Typ B kommunizieren. Ein Arbeitspaket wird aber immer nur an eine bestimmte Zelle des Typs B gesendet. Dabei ist es Zufall, welche Zelle B das Paket erhält.

Alle Zellen einer Umgebung teilen sich die gleichen Eingabekanäle. Stirb eine Zelle ab, dann übernehmen andere Zellen dieser Umgebung die Arbeit.

Wenn die ganze Umgebung stirbt, sterben auch alle eingehenden Kommunikationskanäle (Input Pipes) dieser Zellenumgebung ab. Die verwaisten Kommunikationskanäle werden von anderen Zellen als `broken` erkannt und automatisch entfernt. Die ausgehenden Kommunikationskanäle (Output Pipes) werden nicht verändert, da sie nicht dieser Zelle gehören, sondern nur als sendende Schicht verwendet werden.

Durch den zweiten Mechanismus, dem Erkennen von neuen Kommunikationspartner, wird das dynamische Kommunikationsverhalten geschlossen. Stellt eine Zelle A mit Zellen des Typs B einer Verbindung her, kann es sein, dass zu einem späteren Zeitpunkt weitere Zellen des Typs B hinzu kommen. Natürlich ist es wünschenswert, wenn die Zelle A nun auch mit diesen neuen Zellen kommunizieren kann, automatisch versteht sich. Dazu wird nach einer bestimmten Zeit T die Registrierungsstelle erneut nach möglichen Zellen B abgefragt. Die neuen Zellen werden dann als mögliche Kommunikationspartner in der Zelle A registriert.

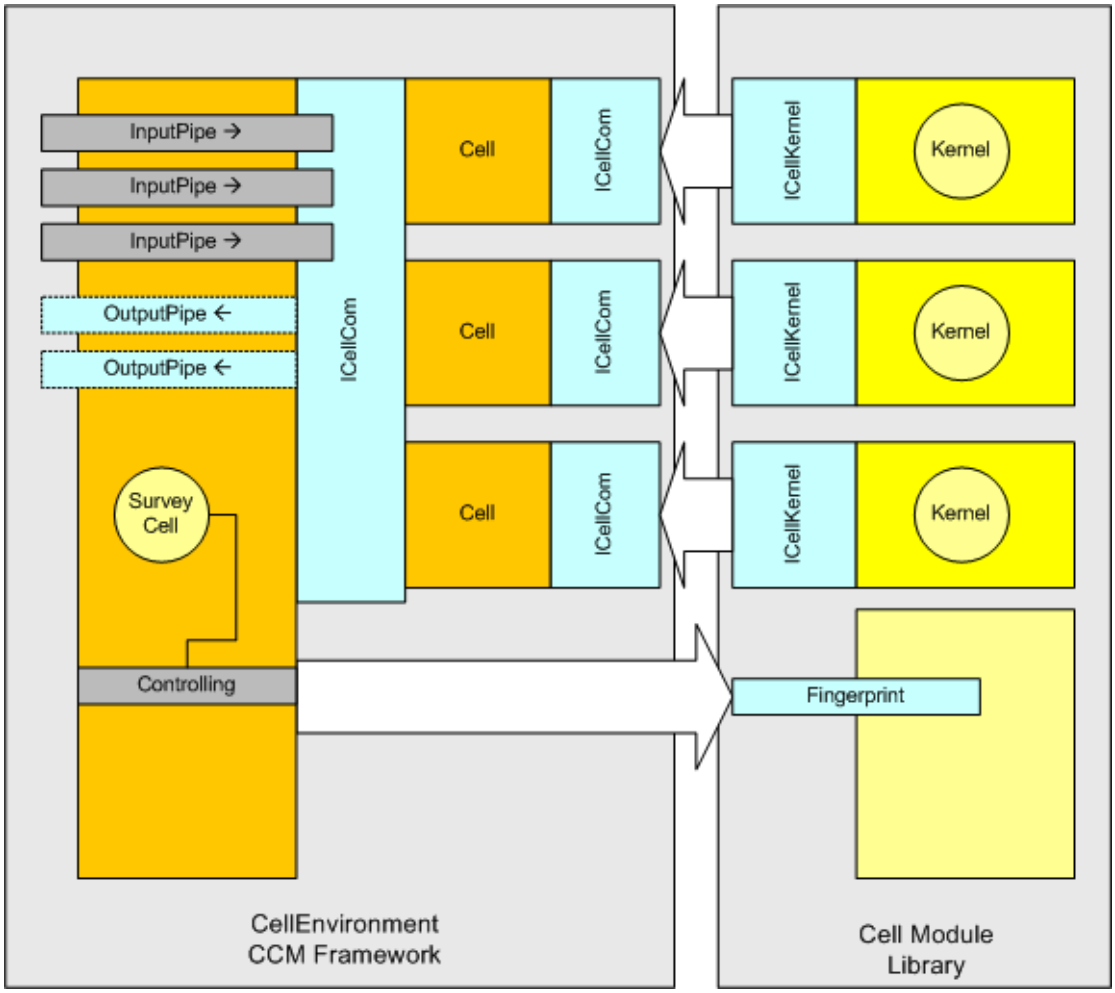

Abbildung 2: Die Bibliothek (Shared Library rechts) wird durch das Framework (links) geladen und verwaltet. Die Zellkerne befinden sich in der Bibliothek. Die Verwaltung der Kommunikationskanäle und der Zellen wird durch das Framework betätigt.

2.1.2.1 Lokale Kommunikationskanäle

Lokale Kommunikationskanäle sind intern im Framework implementiert. Es gibt unterschiedliche Arten der Kommunikation:

- Queuing: Die Arbeitspakete werden in eine Queue abgelegt. Für Queues gibt es keine Grenze der Anzahl Einträge. Der Klon-Algorithmus regelt die Auslastung der Queues eines Zellensystems (Siehe Kapitel Klonen von Zellen).

- Queuing with Replace: Werden viele gleiche Pakete übertragen, interessieren aber nur die aktuellsten Pakete, kann das Queuing with Replace angewandt werden. Jedes Arbeitspaket besitzt eine ID. Wird die gleiche ID nochmals verwendet, und ist das vorhergehende

Arbeitspaket noch in der Queue, wird dieses durch das aktuelle Arbeitspaket ersetzt. Der Vorteil ist, dass die Grösse der Queue klein gehalten werden kann. Dieses Verfahren wird vor allem bei der Signal-Visualisierung verwendet, um schnell auf Änderungen reagieren zu können und das Schleppen von Werten zu umgehen (Stau in der Queue). Dieses Verfahren steht in der aktuellen Version noch nicht zur Verfügung.

Ein lokaler Kommunikationskanal wird durch folgende Kenngrössen beschrieben:

- Name der Zelle
- Name des Kommunikationskanals (Pipe)

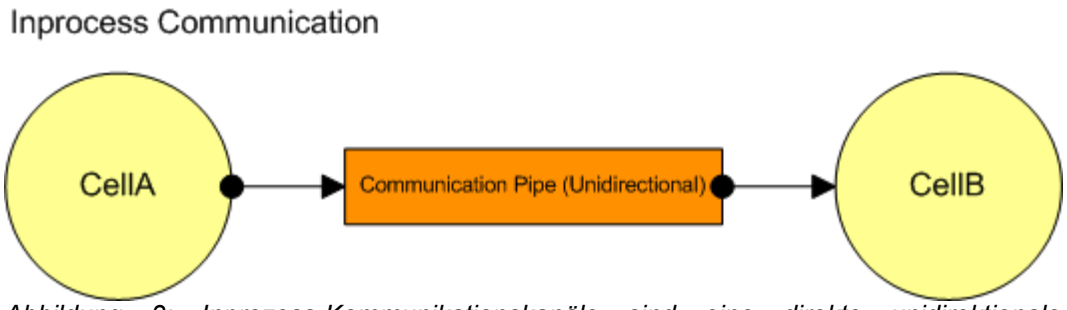

Abbildung 3: Inprozess-Kommunikationskanäle sind eine direkte unidirektionale Verbindung zwischen 2 Zellen

2.1.2.2 Entfernte Kommunikationskanäle

Damit Zellen über ein Netzwerk miteinander kommunizieren können, braucht es eine Registrierung. Folgende Daten der Zellen werden dort verwaltet:

- Name der Zelle
- Name des Kommunikationskanals (Pipe)
- Adresse des Host (Namen des Computers oder IP Adresse) (TCP/IP)
- Port des Socket (TCP/IP)

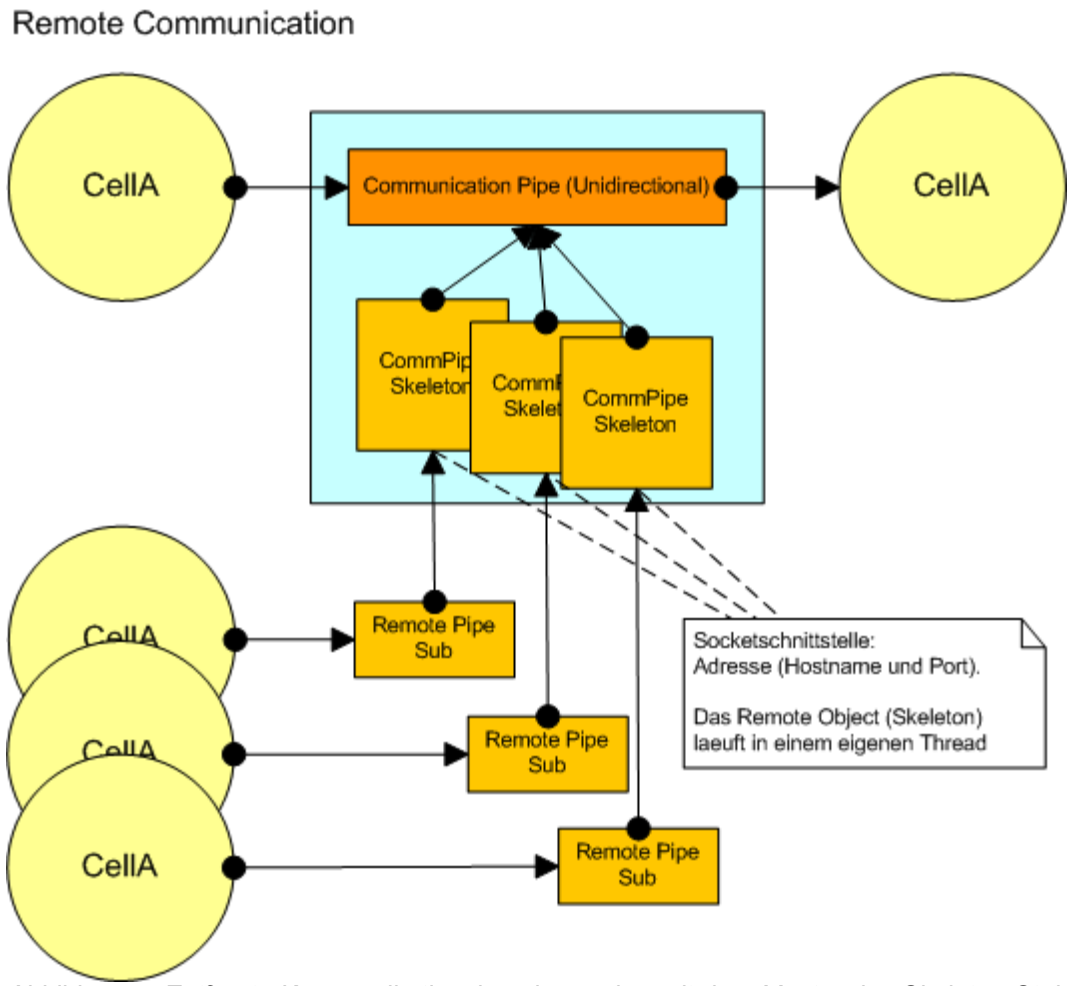

Abbildung 4: Entfernte Kommunikationskanäle werden mit dem Muster des Skeleton-Stub implementiert.

Die Registrierungsstelle ist eine eigene Applikation, welche für die laufenden Zeus-Applikationen zugänglich ist. Beim Starten einer Zelle werden ihre Kommunikationskanäle automatisch bei der Registrierungsstelle registriert.

Als Kommunikationsprotokoll wird das Cell Communication Transfer Protocol CCTP verwendet. Es handelt sich um ein binäres Protokoll, welches einfach über Sockets angewendet werden kann.

A Messungen vom Klonverhalten

A.1 Klonen von Zellen

Das Zeus-Framework verwendet den stabilen Klon-Algorithmus, welcher auch bei instabilen Arbeitslast der Zellen ein Optimum zwischen Anzahl Zellen und Auftragseingang (Work load) findet. Siehe [CCM1].

Das Klonen ist der elementarer Bestandteil zur Wachstumskontrolle der Zeus-Zellen. Die Kontrolle wird durch eine Zellenüberwachung in der Zellenumgebung betätigt. Der Klon-Algorithmus misst die gesamte Auslastung einer Zellenumgebung.

Der Algorithmus benötigt folgende Eingabeparameter:

- Anzahl Meldungen eines Zellensystems zu einem beliebigen Zeitpunkt t: $d(t)$
- Maximale Grenze des Meldungseingangs (Grösse der Meldungsqueue) N_{MAX}
- Reaktionsgrenze N_{react}
- Obere Schranke der Messzeit T_{MAX}
- Untere Schranke der Messzeit T_{MIN}

A.2 Abbauen von Zellen

Das zweite Verfahren steuert den Abbau von Zellen. Dabei wird jeder neu erstellten Zelle die Lebenskraft 100 zugewiesen (bedeutet 100%).

Der Algorithmus verwendet folgende Eingabeparameter:

- Lebenskraft der Zelle in Abhängigkeit der Zeit t: $g(t)$
- Erhöhung der Lebenskraft L_{add}

- Verringerung der Lebenskraft L_{sub}
- Faktor für Wahrscheinlichkeit des Abbaus ω

A.3 Messungen

In diversen Versuchen wurden empirisch folgende Werte für Zellen mit variabler Eingangslast ermittelt[1]:

Parameter	*Wert*
T_{MAX}	1.0 [sec]
T_{MIN}	0.1 [sec]
N_{MAX}	20 [jobs]
N_{react}	10 [jobs]
L_{add}	1.4
L_{sub}	1
ω	7

1 Die Messung erfolgte auf einem Laptop Sony PCG-FX290K mit 1GHz Prozessor und 512MB RAM. Als Systemsoftware wurde SUSE Linux 9.1 mit Kernel 2.6.4 verwendet (GCC 3.3.3)

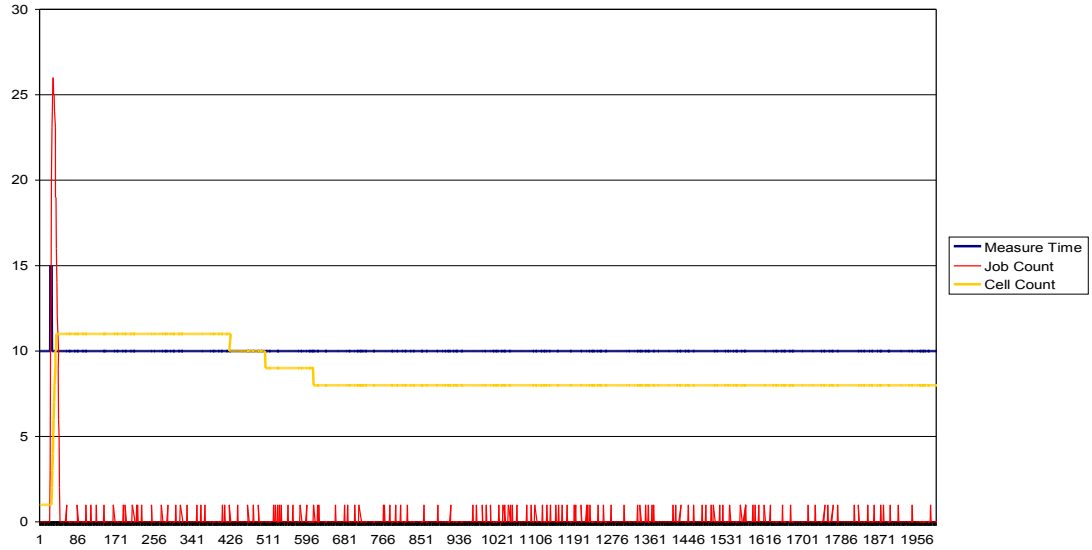

Abbildung 5: Messung eines stabilen Systems, mit variabler Eingangslast. Measure Time zeigt die Messdauer (Dargestellt mit Faktor 100), Job Count ist die Grösse des Meldungseingangs und Cell Count zeigt auf, wie viele Zellen aktiv sind.

Das System wird mit folgenden Parameter instabil:

Parameter	***Wert***
T_{MAX}	1.0 [sec]
T_{MIN}	0.1 [sec]
N_{MAX}	20 [jobs]
N_{react}	15 [jobs]
L_{add}	1
L_{sub}	1.2
ω	7

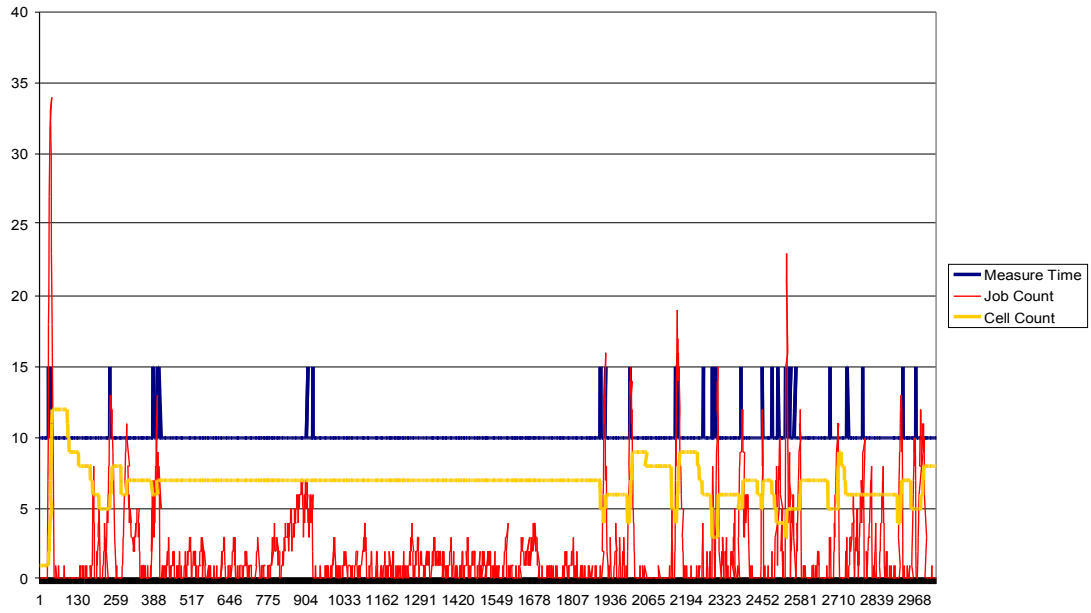

Abbildung 6: Messung eines instabilen Systems. Klar ersichtlich ist der Unterschied zur vorangehenden Abbildung. Weil sehr viele Zellen erstellt und wieder abgebaut werden, ist die Performance beim Verarbeiten der Arbeitspakete schlechter (Measure Time-blaue Kurve).

B Versionsgeschichte

Änderungen am Dokument sind hier festgehalten

Version	*Beschreibung*	*Datum*
0.1	Erstellen der Version 0.4	12.03.07

C Literaturverzeichnis

[CCM1]: Hadorn, Benjamin, Cell Computing Model, 2003

D Index

4, 10, 14