Drawing UML Sequence Diagram by using pgf-umlsd

Yuan Xu

March 5, 2012 (v0.7)

Abstract

pgf-umlsd is a LaTeX package for drawing UML Sequence Diagrams. As stated by its name, it is based on a very popular graphic package PGF/TikZ. This document presents the usage of pgf-umlsd and collects some UML sequence diagrams as examples. pgf-umlsd can be downloaded from http://code.google.com/p/pgf-umlsd/.

Contents

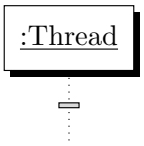
1 The Essentials

1.1 Basic graphics objects

1.1.1 empty diagram

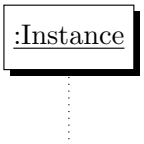
```
\begin{sequencediagram}
\end{sequencediagram}
```

1.1.2 thread



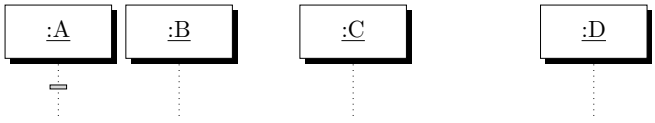

```
\begin{sequencediagram}
  \newthread{name}{:Thread}
\end{sequencediagram}
```

1.1.3 instance




```
\begin{sequencediagram}
  \newinst{name}{:Instance}
\end{sequencediagram}
```

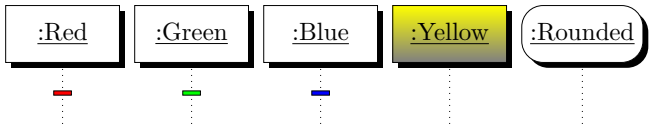
1.1.4 distance between threads and instances




```
\begin{sequencediagram}
  \newthread{a}{:A}
  \newinst{b}{:B}
  \newinst[1]{c}{:C}
  \newinst[2]{d}{:D}
\end{sequencediagram}
```

1.1.5 customization

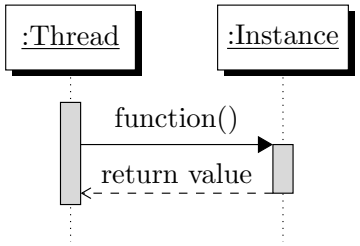
The package has two options for customization: `underline` and `rounded corners`, further customization see the example below:




```
\begin{sequencediagram}
  \newthread[red]{r}{:Red}
  \newthread[green]{g}{:Green}
  \newthread[blue]{b}{:Blue}
  \tikzstyle{inststyle}+=[top color=yellow,
      bottom color=gray]
  \newinst{y}{:Yellow}
  \tikzstyle{inststyle}+=[bottom color=white,
      top color=white, rounded corners=3mm]
  \newinst{o}{:Rounded}
\end{sequencediagram}
```

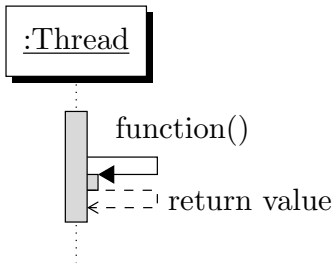
1.2 Call

1.2.1 call



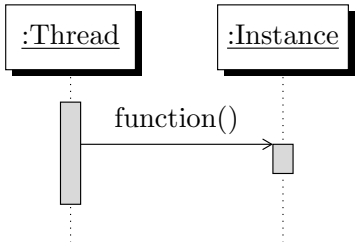

```
\begin{sequencediagram}
  \newthread{t}{:Thread}
  \newinst[1]{i}{:Instance}

  \begin{call}{t}{function()}{i}{return value}
  \end{call}
\end{sequencediagram}
```

1.2.2 call self



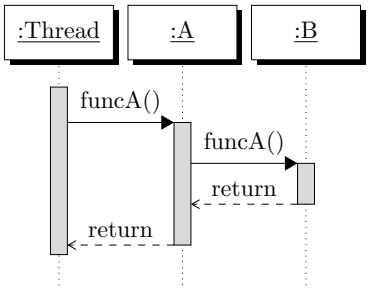

```
\begin{sequencediagram}
  \newthread{t}{:Thread}

  \begin{call}{t}{function()}{t}{return value}
  \end{call}
\end{sequencediagram}
```

1.2.3 message call



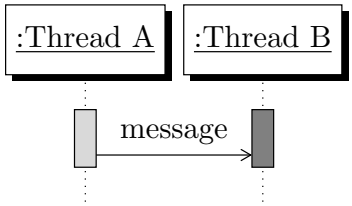

```
\begin{sequencediagram}
  \newthread{t}{:Thread}
  \newinst[1]{i}{:Instance}

  \begin{messcall}{t}{function()}{i}
  \end{messcall}
\end{sequencediagram}
```

1.2.4 nested call



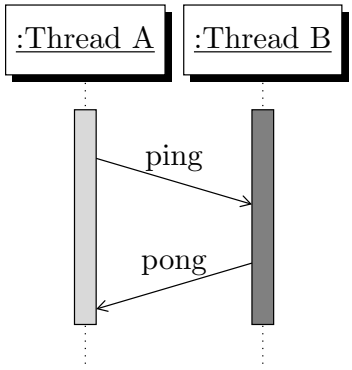

```
\begin{sequencediagram}
  \newthread{t}{:Thread}
  \newinst{a}{:A}
  \newinst{b}{:B}

  \begin{call}{t}{funcA()}{a}{return}
    \begin{call}{a}{funcA()}{b}{return}
    \end{call}
  \end{call}
\end{sequencediagram}
```

1.3 Message



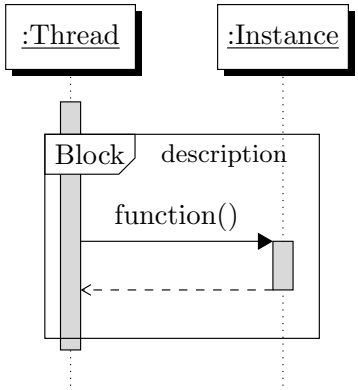

```
\begin{sequencediagram}
  \newthread{a}{:Thread A}
  \newthread[gray]{b}{:Thread B}

  \mess{a}{message}{b}
\end{sequencediagram}
```

Sometimes however, it takes a considerable amount of time to reach the receiver (relatively speaking of course) . For example, a message across a network. Such a non-instantaneous message is drawn as a slanted arrow.




```
\begin{sequencediagram}
  \newthread{a}{:Thread A}
  \newthread[gray]{b}{:Thread B}

  \mess[1]{a}{ping}{b}
  \mess[1]{b}{pong}{a}
\end{sequencediagram}
```

1.4 Block



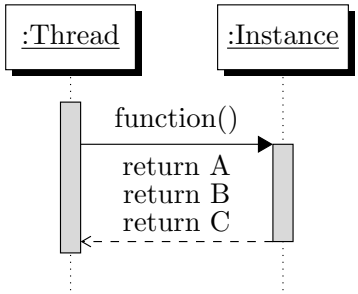

```
\begin{sequencediagram}
  \newthread{t}{:Thread}
  \newinst[1]{i}{:Instance}

  \begin{sdblock}{Block}{description}
    \begin{call}{t}{function()}{i}{}
    \end{call}
  \end{sdblock}
\end{sequencediagram}
```

2 Manually adjustment

The idea of `pgf-umlsd` is users only have to write the logic of diagram, the program generates figure automatically. However, the package can not handle all the use case, it still needs to be adjusted manually.

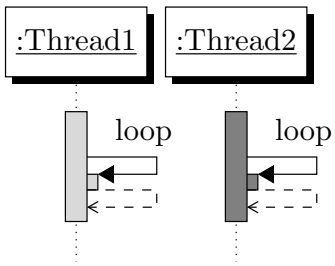
2.1 Level

If the text on the arrows is more than one line (large function name for example) it will overlap other things. `postlevel` can be used to make the time (level) later.




```
\begin{sequencediagram}
  \newthread{t}{:Thread}
  \newinst[1]{i}{:Instance}
    \begin{call}{t}{function()}{i}{\shortstack{
        return A\\ return B\\
          return C}}
      \postlevel
    \end{call}
\end{sequencediagram}
```

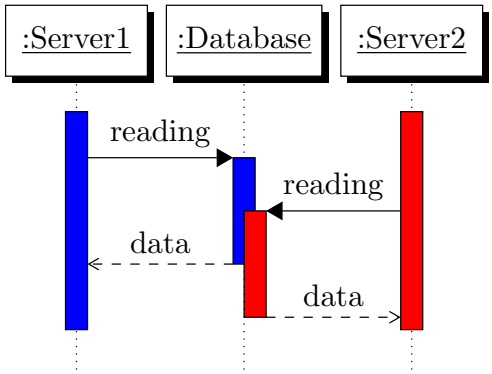
In the situation of multi-threads, some events happen at the same time. `prelevel` can make the call earlier.




```
\begin{sequencediagram}
  \newthread{t1}{:Thread1}
  \newthread[gray]{t2}{:Thread2}

  \begin{callself}{t1}{loop}{}
  \end{callself}

  \prelevel\prelevel

  \begin{callself}{t2}{loop}{}
  \end{callself}
\end{sequencediagram}
```

2.2 Bias of thread line

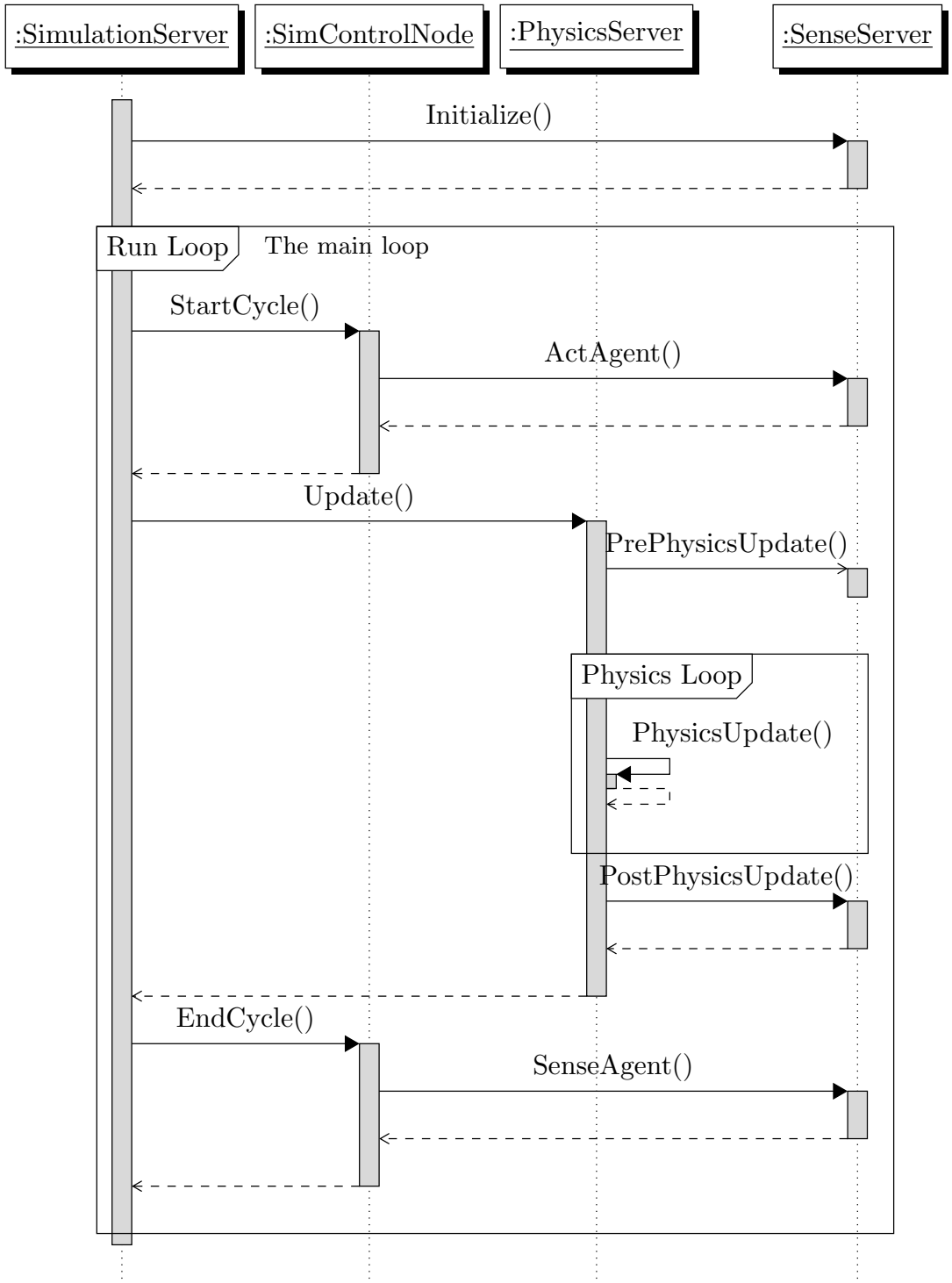
In the situation of multi-threads, the instance cen be accessed at the same time (e.g. two threads reading data at the same time). Currently, we have to adjust the bias of thread line manually for this. Possible parameters for `setthreadbias` are: `center`, `west` and `east`.




```
\begin{sequencediagram}
  \newthread[blue]{s1}{:Server1}
  \newinst{db}{:Database}
  \newthread[red]{s2}{:Server2}

  \begin{call}{s1}{reading}{db}{data}
    \postlevel
  \end{call}

  \prelevel\prelevel

  \setthreadbias{east}

  \begin{call}{s2}{reading}{db}{data}
    \postlevel
  \end{call}
\end{sequencediagram}
```

3 Examples

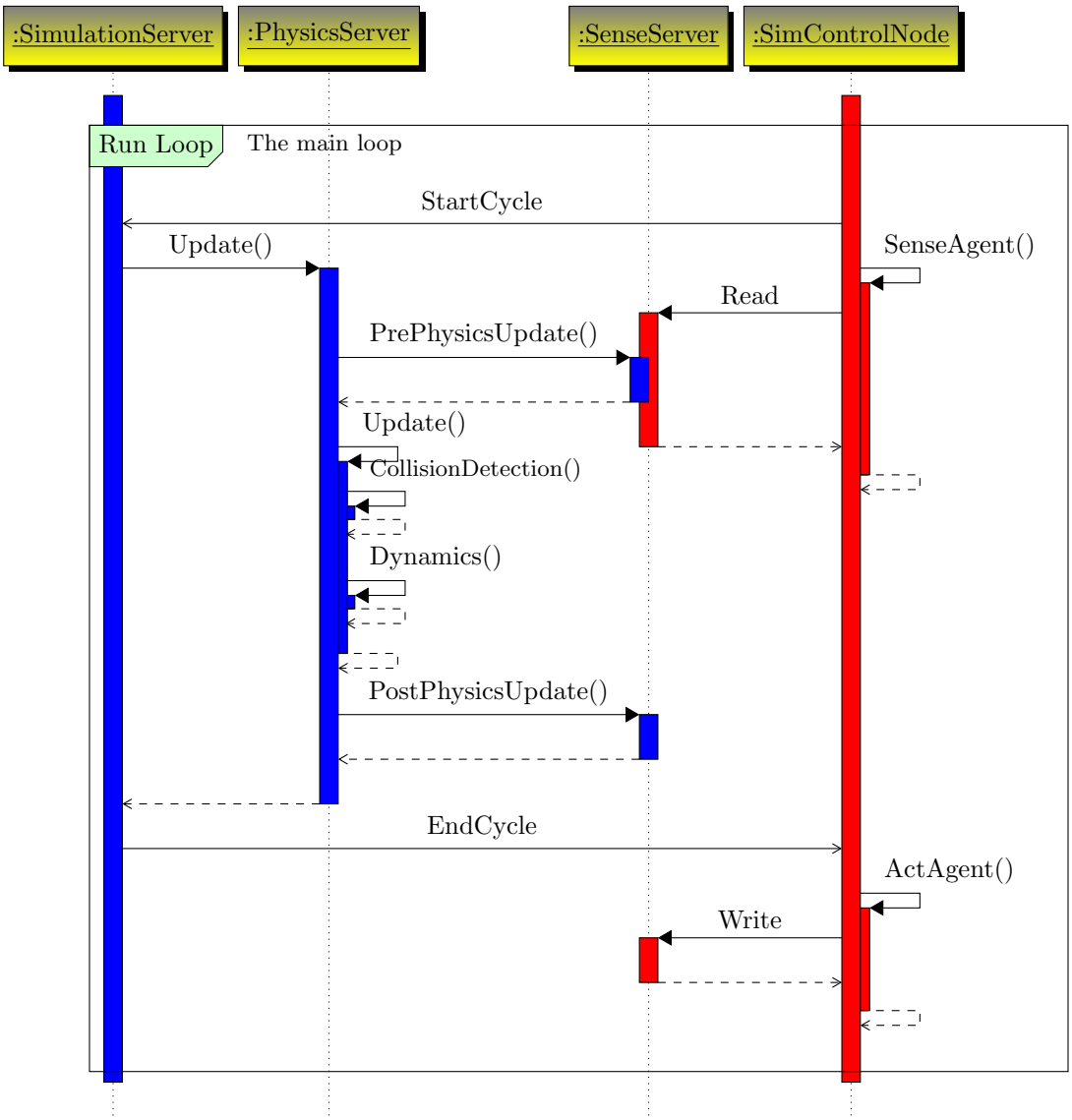
3.1 Single thread




```
\begin{sequencediagram}
  \newthread{ss}{:SimulationServer}
  \newinst{ctr}{:SimControlNode}
  \newinst{ps}{:PhysicsServer}
  \newinst[1]{sense}{:SenseServer}

  \begin{call}{ss}{Initialize()}{sense}{}
  \end{call}
  \begin{sdblock}{Run Loop}{The main loop}
    \begin{call}{ss}{StartCycle()}{ctr}{}
      \begin{call}{ctr}{ActAgent()}{sense}{}
      \end{call}
```

```
    \end{call}
    \begin{call}{ss}{Update()}{ps}{}
      \begin{messcall}{ps}{PrePhysicsUpdate()}{sense}{state}
      \end{messcall}
      \begin{sdblock}{Physics Loop}{}
        \begin{call}{ps}{PhysicsUpdate()}{ps}{}
        \end{call}
      \end{sdblock}
      \begin{call}{ps}{PostPhysicsUpdate()}{sense}{}
      \end{call}
    \end{call}
    \begin{call}{ss}{EndCycle()}{ctr}{}
      \begin{call}{ctr}{SenseAgent()}{sense}{}
      \end{call}
    \end{call}
  \end{sdblock}
\end{sequencediagram}
```

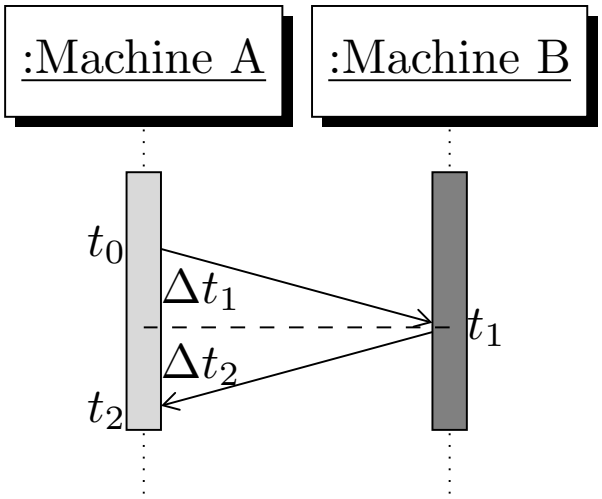
3.2 Multi-threads




```
\begin{sequencediagram}
  \tikzstyle{inststyle}+=[bottom color=yellow] % custom the style
  \newthread[blue]{ss}{:SimulationServer}
  \newinst{ps}{:PhysicsServer}
  \newinst[2]{sense}{:SenseServer}
  \newthread[red]{ctr}{:SimControlNode}

  \begin{sdblock}[green!20]{Run Loop}{The main loop}
```

```
    \mess{ctr}{StartCycle}{ss}
    \begin{call}{ss}{Update()}{ps}{}
      \prelevel
      \begin{call}{ctr}{SenseAgent()}{ctr}{}
        \begin{call}[3]{ctr}{Read}{sense}{}
        \end{call}
      \end{call}
      \prelevel\prelevel\prelevel\prelevel
      \setthreadbias{west}
      \begin{call}{ps}{PrePhysicsUpdate()}{sense}{}
      \end{call}
      \setthreadbias{center}
      \begin{call}{ps}{Update()}{ps}{}
        \begin{call}{ps}{\small CollisionDetection()}{ps}{}
        \end{call}
        \begin{call}{ps}{Dynamics()}{ps}{}
        \end{call}
      \end{call}
      \begin{call}{ps}{PostPhysicsUpdate()}{sense}{}
      \end{call}
    \end{call}
    \mess{ss}{EndCycle}{ctr}
    \begin{call}{ctr}{ActAgent()}{ctr}{}
      \begin{call}{ctr}{Write}{sense}{}
      \end{call}
    \end{call}
  \end{sdblock}

\end{sequencediagram}
```

3.3 Annotation




```
\begin{sequencediagram}
  \newthread{a}{:Machine A}
  \newthread[gray]{b}{:Machine B}

  \mess[1]{a}{}{b}
  \node[anchor=east] (t0) at (mess from) {$t_0$};
  \node[anchor=west] (t1) at (mess to) {$t_1$};
  \prelevel
  \mess[1]{b}{}{a}
  \node[anchor=east] (t2) at (mess to) {$t_2$};

  \path (t0.east) |- coordinate(t12) (t1);
  \draw[dashed] (t1) -- (t12);
  \node[anchor=south west] at (t12) {$\Delta{}t_1$};
  \node[anchor=north west] at (t12) {$\Delta{}t_2$};
\end{sequencediagram}
```

3.4 Known Issue

`pgf-umlsd` confilts with tikz `backgrounds` library.

4 Acknowledgements

Many people contributed to `pgf-umlsd` by reporting problems, suggesting various improvements or submitting code. Here is a list of these people: Nobel Huang, Dr. Ludger Humbert, MathStuf, Vlado Handziski, Frank Morgner, and Dirk Petrautzki.