本書について

本書の次ページ以降の部分は, 2SECONDS の作者である野村亮氏が, 1994 年 4 月の第 4 回パルテノン研究会において発表された論文を, 氏の承諾を得て書き写したものです.

2SECONDS は SFL 言語の機能シミュレータ SECONDS のメジャー・バージョンアップ版であり, SECONDS との上位互換性を保ちつつ, SECONDS には無かった条件停止や式の扱いなどが加わった, 高機能なシミュレータとなっています.

PARTHENON の最近のリリースでは, この 2SECONDS が同梱されており, すぐに使える状態を提供しています. しかしながら, SECONDS に比べて圧倒的にドキュメントが少なく, どう使ったらよいのか分からないという声をよく耳にします.

そこで, 作者が残された貴重なドキュメントを, ここに再現しました. 学会用の原稿ということもあり, リファレンス・マニュアルというほどの詳細さはありませんが, SECONDS との違いがよく分かる内容となっていますので, SECONDS ユーザが 2SECONDS に移行するための導入としては, 必要十分な内容があると思います.

なお, 書き写しに際しては, 明らかな誤字や脱字を除いて, 可能な限り原文の内容に忠実に再現してあります. したがって, 本書を公式に参照される場合は, 参考文献 (1) のように参照してくださるよう, お願い致します. ただし, 本書あるいは 2SECONDS に関するお問い合わせは, パルテノン研究会までお願い致します. パルテノン研究会の問い合わせ先は, パルテノン・ホームページ内の下記のページを御覧下さい.

http://www.kecl.ntt.co.jp/parthenon/html/kenkyuka.htm

2001 年 12 月
PARTHENON 研究会

参考文献

[1] 野村 亮. *The Design of 2SECONDS*. 第 4 回パルテノン研究会 資料集, pp.33–40, 1994.

The Design of 2SECONDS

Ryo Nomura
N.T.T. Communication Science Laboratories

ABSTRACT

In order to satisfy user's requests and to solve their problems upon SECONDS, I've designed and implemented a new SFL simulator 2SECONDS. It introduces several functions missed in older SECONDS: taking snap-shot of the simulation image, I/O handling to/from processes, simulation monitoring, expression evaluations, user-specific error handlings, and so on;

In this paper, function details and new commands, those are provided to use these functions, are mentioned. 2SECONDS' command set is almost upperword compatible, however, the small incompatibility concering to operation sequence during forward command is involved. The incompatibility is also discussed.

1 はじめに

PARTHENON システムの SFL シミュレータである SECONDS において、ユーザから要望の多かった機能を吸収し、新しいリリース (2SECONDS) としてまとめた。本リリースにおける主な改良点は、

a) 子プロセスから (へ) の入 (出) 力

b) 条件付コマンド実行

c) forward コマンドの条件ブレークを行うモニタ

d) シミュレーションイメージのスナップショット

e) エラーシビアリティの導入

f) シミュレーションプロセスの単純化

などである。

2SECONDS のコマンドセットを表1に示す。表1は、2SECONDS が管理する情報毎にコマンドをまとめてある。アスタリスクの付いているコマンドが、今回追加されたものである。

旧 SECONDS からのコマンドの機能は概ね上位互換である。但し、レポート機能、**print** コマンド、**echo** コマンドによる出力を除いた、コマンドの出力、 及び、 エラーの表示の形式は異なるため、SECONDS の出力をモニタするプログラムを作られている場合には、それらに対する上位互換性はない。ただこのようなパワーユーザであれば、今回の変更にも難無く対処できると考えている。

また、レポート機能を用いた出力も、f) の影響により、若干の非互換性が生じる。以下の各節では、 本リリースで導入された機能とその実現法であるコマンド群の使い方とともに、この非互換性についても述べる。

2 入出力機能の強化

SECONDS では、 シミュレーション結果の出力形式を利用者の都合に合わせられるように、レポート機能や **print** コマンドにおいて、C の printf() に相当する書式付き出力を提供しており、一旦ファイルに出せば、 Unix の多くのユーティリティ (sed, awk, perl 等) が使えるので、それで十分だろうという考え方をしていた。しかし、 X をはじめとするウィンドウの環境ではカレントディレクトリが変わってしまったりしてあまり使い易くないし、 だいいちウィンドウを移すのが面倒である。また、 長いシミュレーションではファイルが大きくなりすぎる。

そこで、2SECONDSでは、入力(コマンド)の形式やその他の情報でも、より柔軟な入出力ができるようにした。といっても、このようなユーティリティの機能を2SECONDS内に実現したわけではなく、2SECONDSでは、このようなUnixユーティリティへ直接出力を渡す機構をつけただけである。

2.1 プロセスへの入出力とスタック

この機構を使うには、単に **listen**, **speak**, **claim** コマンドで、ファイル名を指定していたところに, '|' を前置して、シェル用のコマンドを記述すればよい。'|' が外された残りが、シェル (`$SHELL -c`) へのアーギュメントとして渡され、その標準入力 (出力) が、speaking, claiming, (listening) streamとなる。

但し、プロセスへの出力を使うと、その後の **speak**(**a**), **claim**(**a**) コマンドで、出力を切り替えたとき、プロセスへのパイプが閉じられてしまうという問題が生じる。ファイルだけのときなら、アペンドモードで再オープンすれば良いが、一般にはプロセスを再びオープンすることはできない。そこで、listening stream同様にspeaking, claiming streamにも、スタック機構をつけ、ポップアップされない限り、プロセスへのパイプは閉じないことにした。

また、この機構により、スクリプトが独自のテンポラリファイルを作ってspeaking(claiming)streamを切り替えても、プッシュ (**speakp**, **speakpa**, **claimp**, **claimpa**) ／ポップ (**speakx**, **listenx**) で行えば元に戻せるため、自己完結したスクリプトを記述できる。

尚、テンポラリファイルのユニークな名前を作るために、2SECONDSプロセスのプロセスID (**PID**) とセッション中のコマンドのシーケンシャル番号 (**CNUM**) の設定された環境変数が定義されている (例 1)。

2.2 変数

SECONDSには、シェルにおける‘¡command¿‘とシェル変数への代入のような、コマンドの出力を別のコマンドの入力に利用する一般的な方法が提供されていなかったが (エディタ等で **define** または **setenv** コマンドに書き換える必要があった)、2SECONDSでは、**rmacro** コマンドにより、ファイルの内容を文字列としてローカル変数に設定できるようにした。これは、以下のスクリプトにより、シェルにおける方法と同等のものを提供する。

例 1. スクリプト let

```
#Usage let <var> <cmd>
define var $1; shift
define tmpfile tmp.$PID.$CNUM
speakp $tmpfile
$ARGV
speakx
rmacro $var $tmpfile
setenv $var "$var"
exec "rm $tmpfile"
```

2.3 SECONDSPATH

これらの入出力機能の強化により、2SECONDSの生のコマンドよりもスクリプトが多用されることが予想され、当然スクリプトのライブラリ化の要求がでてくる。このため、2SECONDSでは、**SECONDSPATH** という変数を導入した。これは名前からも想像できるように、シェルの用いる **PATH** 変数に相当し指定の仕方も同じでディレクトリを':'で区切って指定する。但し、シェルが実行許可を要求するのに対して、2SECONDSは読出許可を要求する点が異なる。これにより、カレントディレクトリに、"`ls`"のような名前のスクリプトを作っても、実行許可を消しておけばシェルが誤解することはない。更に、2SECONDSがどのライブラリのスクリプトを優先しているかを確認できるように、**which** コマンドも用意した。

また、オンラインヘルプも、プログラムハードコードから外部ファイルの参照へとそのアーキテクチャが変わっており、ビルトインコマンド以外は、この **SECONDSPATH** にある、ディレクトリの"`.hlp`"サブスクリプトのついたファイルを探す。

従って、スクリプトをライブラリ化し、第三者に利用可能とするためには、スクリプトと"`.hlp`"サブスクリプトのついたヘルプファイルを同じディレクトリに置くのが良い。尚、ビルトインコマンドのヘルプファイルは`$PARTHENON/doc.dir/seconds` [1] である。

[1] parthenon version 2.4.1 では`/usr /local /parthenon /doc`

3 条件と式

「レジスタがある値になったら、 **forward** コマンドを停止したい。」という利用者の要求に応えるため、2SECONDS では、 ふたつの条件判定機構が付与されている。ひとつは汎用的な **if** コマンドであり、もう一つは前述の要求に直接答えるモニタ機能である。

3.1 if コマンド

if コマンドのシンタクスには以下の 2 通りがあり、これらは、 条件式の後にトークンが存在するか否かによって区別される。第一形式は、機能的には以下の第二形式の単なる省略形であるが、繰り返しを再帰で実現する方法を採っている SECONDS において、新しいコマンド **listenx**, **evalx** とともにテイルリカージョンを実現するために導入されている。ここで、 **listenx**(**evalx**) は、**listen**(**eval**) と **leave** コマンドを組合わせたものである。

if ⟨*condition*⟩ **;** ⟨*command*⟩ **; endif;**

第二形式は、 **if** コマンドの条件が真 (長さ 1 のビット列 `B1`) であるときは、 その直後のコマンドから **else** コマンド (**else** コマンドがなければ **endif** コマンド) の前までが実行され、偽 (長さ 1 のビット列 `B0`) であるときは、**else** コマンドと **endif** コマンドの間のコマンド群が実行される。

if コマンドはネストして良いが、**else**, **endif** 共に、形式的にはパラメータなしのコマンドであることに注意されたい。

＜第一形式＞

if ⟨*condition*⟩ ⟨*command*⟩

＜第二形式＞

if ⟨*condition*⟩
{⟨*then-command*⟩}
else
{⟨*else-command*⟩}
endif

3.2 モニタ機能

モニタ機能は、 **forward** コマンド実行中に、 指定した条件の集合を監視する機能であり、条件の少なくとも一つが、 真となると **forward** コマンドをブレークする。

以下に、 モニタ機能に関するコマンドセットの形式を示す。

mon_add ⟨*tag*⟩ ⟨*condition*⟩

停止条件 ⟨*condition*⟩ を条件名 ⟨*tag*⟩ として追加。

mon_rmv ⟨*tag*⟩

条件名 ⟨*tag*⟩ の条件を削除。

mon_off ⟨*tag*⟩

条件名 ⟨*tag*⟩ の条件をモニタ集合から外す。

mon_on ⟨*tag*⟩

条件名 ⟨*tag*⟩ の条件をモニタ集合に加える。

mon_stat ⟨*tag*⟩

条件名 ⟨*tag*⟩ の個々の値を表示。

monitor `on`

モニタ機能自身のオン。

monitor `off`

モニタ機能自身のオフ。

monitor `do`

forward コマンド外での条件集合のテスト。

尚、 変数 **MONSTAT** には、 最新のブレーク時または、**monitor** `do` コマンド発行時のモニタの値が入れられている。また、 どの条件が満たされたかは、**mon_stat** コマンドを使えば判別できるし、**forward** コマンドのブレークメッセージでも表示される。

3.3 式

if コマンドやモニタに対する条件として、“式” が定義できる。2SECONDS の “式” は、パーシングの容易さと一般性からポーランド記法を採用した。一般形式は、“(⟨*functor*⟩ {⟨*argument*⟩})” という、Lisp タイプの表記法であり、勿論ネストしてよい。但し、注意していただきたいのは、“式” は、一つのトークンではなく、トークンの列で、マクロ展開等の前処理の後に解釈されるので、’(’ も’)’ も、共にひとつのトークンでなければならず、前後にブランクが必要であるということである。

表 2は、現時点での ⟨*functor*⟩ の一覧である。

現在は、“式” は、**if** コマンドとモニタ、及び “式” の値を計算し表示する **expr**, **exprn** コマンド (**exprn** は表示後改行しない) だけで使用可能であるが、正式のリリースでは、整数やビット列アーギュメントの殆んどで使用可能とする予定である。この際、従来のコマンドとの互換性とコマンド入力の簡素化のために、ファシリティ名は “(**ref** ⟨ファシリティ名⟩)” の省略形として、そのまま “式” として使えるし、メモリ名@アドレスは “(**mref** ⟨メモリ名⟩ ⟨アドレス⟩)” と解釈される (但し、表 2で大文字で示した箇所ではファシリティそのものと解釈される)。また、ビット列や SFL の状態名も “式” である。

4 スナップショット

SECONDS に対する要求のうち、筆者にとり、**forward** の条件ブレークとともに、最もリーズナブルなものは、「リセットの機能が欲しい。」「初期化の終わったところからシミュレーションを再開したい。」というものである。

これを実現するためには、シミュレーションイメージの状態を再現する機能があれば良い訳で、もっとも簡単な方法は、その時点でプロセスをフォークしてしまうことである。しかし、SECONDS には、いくつかの制御用変数やデータベースがあり、すべてを再現する必要があるのか (してよいのか) は状況に依存する。そこで、各変数や各データベース毎に、それらの状態を保存する機能を用意することにし、どれが必要であるかを利用者が指定できるようにした。

その方法としては、いくつか考えられるが 2SECONDS では、個々の変数やデータベース毎に状態を復帰するコマンド列を出力することにした。これは、必要な修正を施すためには、テキスト形式であるべきであり、かつ、SECONDS では、コマンド以外の入力テキストを解釈しないためである。

以下のコマンドをパラメータなしで指定すると、それぞれの状態を回復するコマンド列を生成する。

time	シミュレーション時刻
stsave	レジスタ系ファシリティの値
set	入力端子など外部設定
hold	ホールド情報
sch_add	スケジュール DB
req_add	リクエスト DB
rpt_add	レポート DB
mon_add	モニタ DB
move, **marker**	現在位置、マーカ位置
dir	ワーキングディレクトリ
define, **setenv**	変数
mkrand	乱数発生器

これに伴い SECONDS では表示機能だけであった **time** コマンドは、シミュレーション時刻の設定ができるように拡張されている。ただこの方法では、再設定時に前後関係が影響する場合があるので注意されたい。

尚、非常に大きなデータ量になりうるレジスタやメモリについては、専用 (バイナリ形式) のファイル (マシンポータブルである) を用いることができ、**stload** コマンドを用いて回復する。言うまでもないと思うが、**set** コマンド (リクエスト含む) で設定したもの以外の非記憶系のファシリティの値保存がこの体系に入っていないのは、状態回復後自動的に再計算されるためである。

5 2SECONDS におけるエラーの扱い

SECONDS の利用者の最も多いクレームは、「エラーメッセージがうるさい。」「重大なエラーなのか無視してよいエラーなのかを解析するのが大変。」等であった。

これは、Apollo の DM 以外の利用者 (殆んど全部) には、かなり深刻な問題であろう。筆者はいまでも DM を愛用しており、正直いうとザマミロと思ってい

るし、エラーの設計は、システムの設計にとって最も難しい問題なので、できれば避けたかったのだが、プログラムからの利用や、**if** コマンドを使ったユーザ定義のエラーの導入を考え、2SECONDS で、 エラー処理のアーキテクチャを一新した。

これが SECONDS から 2SECONDS に名前を変えた最も大きな理由である。

5.1 エラーシビアリティ

2SECONDS では、 各コマンドはその実行によるエラーシビアリティを返す (これは環境変数 **STATUS** に設定されている)。エラーシビアリティは、 非負整数で表され、 0 が「文句なし」、値が大きくなるほどエラーは重大なものと考える。

予め定義されたシビアリティとしては、Informational(1), Warning(100), Errorneous(200), Fatal(300) がある。コマンドスクリプトは最後に実行された (プリミティブ) コマンドのエラーシビアリティがそのスクリプトのエラーシビアリティとなる。

5.2 エラーのハンドリング

2SECONDS は、 エラーを検出すると以下の 3 つのユーザ設定のシビアリティレベルとシステムバンドルの一つのシビアリティにより振る舞いをかえる。

メッセージレベル (**message** コマンド; 初期値 1) : このレベル以上のシビアリティを検出すると、 エラーメッセージを出力する。

ブレークレベル (**break** コマンド; 初期値 200) : **forward** コマンド実行中に、 このレベル以上のシビアリティを検出すると、実行中の **forward** コマンドを打ち切る。

アボートレベル (**abort** コマンド; 初期値 200) : このレベル以上のシビアリティを検出すると、 実行中のスクリプトをアボートする。

エラーレベル (200 に固定) : 複数アーギュメントを対象にしたコマンドにおいて、これ以上のエラーシビアリティを検出すると以下のアーギュメントについてのコマンドの適用をやめる。

5.3 エラーメッセージ

2SECONDS では、 エラーメッセージが次のように標準化してあり、エラーメッセージをユーザプログラムなどで解析しやすくなっている。

シビアリティ : エラー名 : コマンド : 時刻
: メッセージ <tab> { 継続メッセージ }

5.4 シビアリティの設定

2SECONDS は、 エラーのタイプとそのエラーを起こしたコマンド名のペアで、個々のエラーを区別するが、 ユーザはこの単位で、 エラーのシビアリティを設定することができる (**setsvrt**)。この機能をつけた最大の理由は、SFL の解釈によって、 エラーとすべきか否かの判断が難しいケースがあるからである。

この代表的なものは、 タスクの起動に関するものである。例えば、 まだタスクが終了していないのに、他のステージから起動されたり、異なるステージから同時に起動されたりした場合、常識的な SFL の解釈としてはエラーであろうが、タスクのジョインと考えれば、 エラーとは断定しきれない。シビアリティの設定機能は、 判断をユーザに委ねるためのものである。ちなみに、 SECONDS では「疑わしきは罰せず」で上記の事態をエラーとはしていないが、2SECONDS では、 デフォルトではシビアリティ100 のエラーとなる。

ただ、 本来ならば、 発生時刻やエラーに関与した SFL のファシリティも含めて、区別すべきであろうが、残念ながら本リリースではそこまでには至っていない。3SECONDS までご辛抱願いたい。

5.5 error コマンド

2SECONDS は、エラーとして検出しないが、利用者としては適切なメッセージでエラー報告して欲しい場合がある (典型的なものとしては、 予め与えられているシミュレーション結果と 2SECONDS でのシミュレーション結果とが異なった場合が挙げられる)。このため、 2SECONDS では、 利用者からエラー状態を励起できるように、**error** コマンドを用意した。**error** コマンドの形式は、

error ⟨*errname*⟩ ⟨*dfltsvrt*⟩ ⟨*format*⟩ {⟨*facility*⟩}

である。

ここで、⟨*format*⟩, {⟨*facility*⟩} は、**print** コマンドのそれと同じであるが、上述のヘッダが付与されることと、出力先が、claiming stream であることが異なる。⟨*errname*⟩ は、ヘッダメッセージのエラー名として用いられる。

また、**error** コマンドによるエラーも、シビアリティ設定の対象となり、シビアリティ設定がされていないときは、⟨*dfltsvrt*⟩ が、エラーシビアリティとして用いられる。**setsvrt** コマンドでのシビアリティの設定には、エラー名として ⟨*errname*⟩ を、コマンド名として **error** を用いる。

6 処理順序の変更と非互換性

2SECONDS では、SECONDS の内部処理の順序を変えた。SECONDS と 2SECONDS の内部処理の順序を図 1に示す。

利用者から観た両者の大きな違いは、レポートの出力されるタイミングとエラーが報告されるタイミングにある。

これらのタイミングの変更は、SECONDS の会話型というもともとの設計 (ブレッドボードによる回路のテストを考えていただければ良い) に沿うもので、外部端子の設定 (**set** コマンド) を当該クロック内で、エラーレポートの出力前に行うことができるようにするためのものである (SECONDS では、このためには **sch_add** コマンドの使用を強要していたが、このコマンドは会話型の使用には適合しない)。

6.1 レポートの出力

SECONDS では、レポートの出力は各サイクルにおける、ユーザの介入前の値を出力しており、介入後が必要な場合は、**report** `do` コマンドを発行することになっていたが、2SECONDS では、レポート出力は各サイクルの (必要に応じたユーザの介入の後の) 最終結果として作成される。

これにより、SECONDS で非常に良く使われているであろうスクリプトが、2SECONDS では異なる出力を生じることになる。例えば、図1(a)のスクリプトを与えると、2SECONDS では、時刻 0 のレポートが 2 度表示され、時刻 3 のレポートが表示されない (これは、時刻 0 だけの特殊な場合だけではない)。これを解決するには、図1(b)のスクリプトのように、**forward** コマンドの前の **report** `do` コマンドを、**forward** コマンドの後ろに移せば良く、これと同等の処理を行えば、この非互換性を回避できる。

6.2 エラーの報告

2SECONDS ではエラー報告の機会が増えており、利用者にとっては SECONDS より更にうるさいかもしれないが、筆者は実際の計算がいつ行われているかが分かって良い考えている (SECONDS ではこれを利用者から隠そうとしたが返って混乱を招くことになった)。うるさいと思えば、前節の機能を使ってエラーサプレスしていただきたい。

7 その他

2SECONDS におけるその他の変更点は以下のとおりである。

7.1 スタートアップ

2SECONDS のたち上がりにおいて、ユーザのホームディレクトリ配下の “⟨コマンド名⟩`.rc`” をコマンドスクリプトとして読み込み実行する。ここで ⟨コマンド名⟩ は、2SECONDS を起動したときの名前であり、通常は `2seconds` であるが、ソフトリンクを使用すれば設計毎にスタートアップスクリプトを変えることができる。

7.2 SFL バイナリの形式の変更

sflsave コマンドによる SFL バイナリの形式を変更し、以下の形式により、複数のモジュールクラスをひとつのファイルに格納できるようにした。尚、SECONDS でセーブした SFL バイナリは、2SECONDS でも読むことができるが逆はできない。

sflsave {⟨*module-class*⟩} ⟨*file-spec*⟩

7.3 その他のコマンド

最後に、これまでの記述で説明されていない新コマンドを簡単に説明しておく。

tree：シミュレーションイメージのファシリテイ木のノード名とそのタイプを全て表示する。

li：モジュールクラス毎に、そのインスタンスモジュールを表示する。

lot：最外部の端子とサブモジュールの端子と結合されていない端子を表示する。

echon：**echo** コマンドと同じだが、最後の改行を行わない。

echoc：**echo** コマンドと同じだが、出力先が claiming stream である。

echocn：**echoc** コマンドと同じだが、最後の改行を行わない。

mkrand：再現可能な乱数系列を作り出す、乱数発生器を作る。

undef：ローカル変数を未定義にする。

8 まとめと課題

これまで、2SECONDS におけるいろいろな機能の拡張について述べてきたが、まとめると、SECONDS のコマンドを、プログラム言語にしたことに尽きる。すなわち, **if** コマンドの導入が全てといってもよい。後は、2SECONDS のもっている情報をこのプログラム言語からどうアクセスするかだけの問題で、「必要なアクセス方法は提供しました。後は、利用者の方で好きなようにカスタマイズしてください。」というのが、SECONDS 利用者からの様々な要求に対する筆者の答えである。

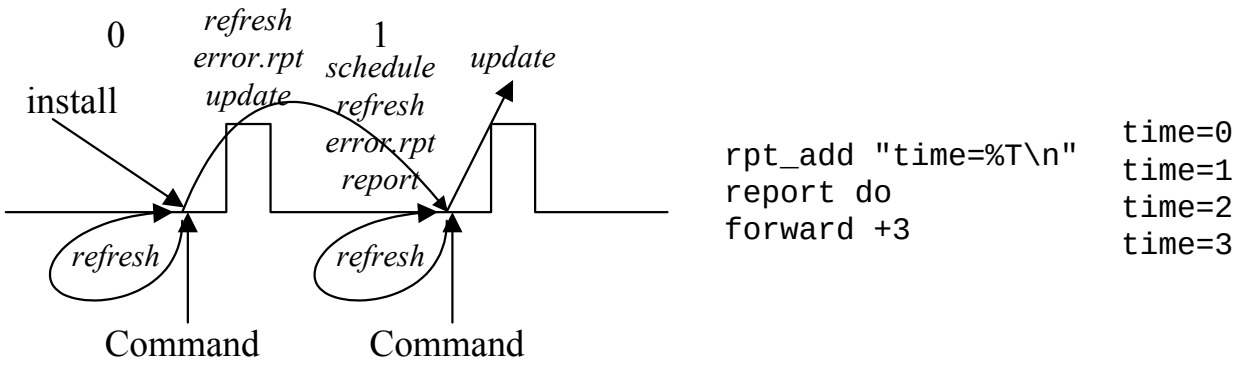
if コマンドは、もともと SECONDS に入れたくて仕方のなかったものであるが、利用者からの、SECONDS に対する要求や不満がなければ入らなかったであろうし、これらは、**if** コマンドをどう作るか、2SECONDS のもっている情報をどうアクセスするかを考える上で大変参考にもなった。ここに感謝の意を表するとともに、今後も有益なコメントをお願いしたい。

尚、今回のリリースで対応しきれなかった要求もいくつか存在する。そのうちの主なものは、高速化、多層クロック及び GUI である。高速化、多層クロックについては、SECONDS の次のメジャーリリース (3SECONDS) で実現する予定である。GUI については NTT データからの供給が予定されている。

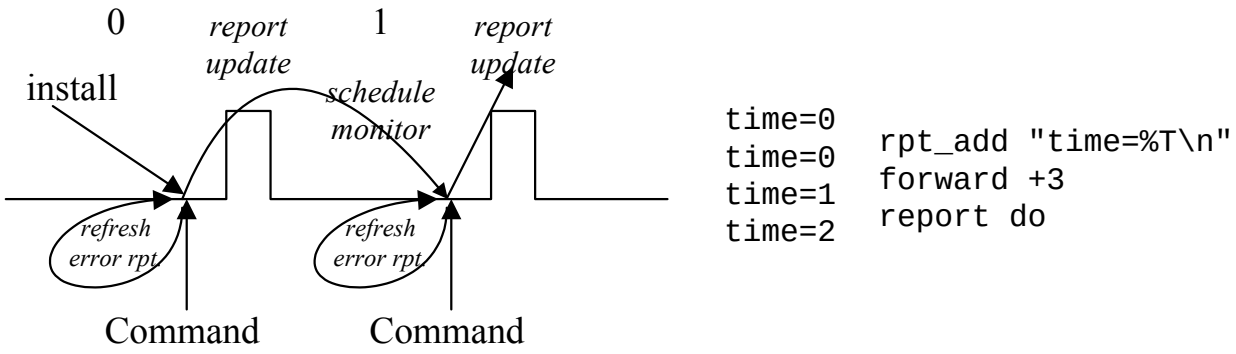
2SEDONDS のユーザインタフェースでは、コマンドと式が異なる概念として乖離しており、関数の定義ができないし、式の中にコマンドを入れることができないなどの問題がある。3SECONDS では、両者を統一することも課題である。

Categories	Items	Individual Commands
Input and Output	Listening	**listen listenx* eval evalx* which* leave**
	Speaking	**speak speaka speakp* speakpa* speakx* flush**
	Claiming	**claim claima claimp* claimpa* claimx***
Simulation image	image structure	**autoinstall install describe lc lcn tree* lot* rxfer wxfer move marker**
	image status	**set memset memclr hold release stload* forward print task stsave***
Databases	Module classes	**sflread sflload sflsave lmc li* rename**
	Scheduling	**schedule sch_add sch_rmv sch_mv**
	Reporting	**report rpt_add rpt_rmv rpt_mv rpt_on rpt_off**
	Monitoring*	**monitor* mon_add* mon_rmv* mon_on* mon_off* mon_stat***
	Requesting	**request req_add req_rmv**
	Local variables	**define rmacro* undef* shift**
	Global variables	**setenv**
	Random generators*	**mkrand***
	Error Severities	**setsvrt* getsvrt* error***
	Environments	**setbase* message* break* abort* stop time dir**
Miscellaneous		**expr* exprn* if* else* endif* ldir exec help sleep bye version echo echon* echoc* echocn* verbose**

表 1: The List of 2SECONDS Commands



(a) Process Sequence in SECONDS



(b) Process Sequence in 2SECONDS

図 1: Difference between SECONDS and 2SECONDS

Functor	Returns
(**now**)	current time.
(**null**)	'not-driven'
(**isnull** e)	`B1` if e is not 'not-driven'
(**has0** e)	`B1` if e has bit-value '0'.
(**has1** e)	`B1` if e has bit-value '1'.
(**hasu** e)	`B1` if e has bit-value 'u'.
(**hasdc** e)	`B1` if e has bit-value '?'.
(**parity** e)	exclusive-or of all bit-values in e.
(**size** e)	bit-length of e.
(**comp** e)	bitwise complement of e.
(**and** e_1 e_2)	bitwise and of e_1 and e_2.
(**or** e_1 e_2)	bitwise or of e_1 and e_2.
(**xor** e_1 e_2)	bitwise xor of e_1 and e_2.
(**exp** e_1 e_2)	enlarge the e_1 to e_2 bits prepending MSB values.
(**substr** e_1 e_2 e_3)	substring of the e_1 from e_2 to e_3.
(**concat** e_1 e_2)	concatenates e_2 to e_1 on the right.
(**merge** e_1 e_2)	replace e_1's bits by e_2's except '?' bits.
(**match** e_1 e_2)	`B1` if merged value of e_1 and e_2 is same value e_1.
(**eq** e_1 e_2)	`B1` if $e_1 = e_2$.
(**ne** e_1 e_2)	`B1` if $e_1 \neq e_2$.
(**gt** e_1 e_2)	`B1` if $e_1 > e_2$.
(**ge** e_1 e_2)	`B1` if $e_1 \geq e_2$.
(**lt** e_1 e_2)	`B1` if $e_1 < e_2$.
(**le** e_1 e_2)	`B1` if $e_1 \leq e_2$.
(**add** e_1 e_2)	e_1 `+` e_2.
(**sub** e_1 e_2)	e_1 `-` e_2.
(**mul** e_1 e_2)	e_1 `*` e_2.
(**div** e_1 e_2)	e_1 `/` e_2.
(**mod** e_1 e_2)	e_1 `%` e_2.
(**tasks** S)	number of active tasks in the stage S.
(**ntasks** S)	number of active tasks on the next cycle in the stage S.
(**width** F)	bit-length of the facility F, for state-valued facilities it returns 0.
(**length** M)	words of the memory M, for scolor facilities it returns 1.
(**writing** F)	`B1` if the facility F is updated (register-type) or driven (terminal-type).
(**staying** S)	`B1` if the current state of the stage S is out of any segments.
(**staying** G)	`B1` if the current state of a stage is in the segment G.
(**nstaying** S)	`B1` if the next state of the stage S is out of any segments.
(**nstaying** G)	`B1` if the next state of a stage is in the segment G.
(**starting** T)	`B1` if the task T is being invoked.
(**starting** S)	`B1` if one of the tasks in stage S is being invoked.
(**ending** T)	`B1` if the task T is terminating.
(**ending** S)	`B1` if one of the tasks in stage S is terminating.
(**ref** F)	the current value of the scolar facility F.
(**next** F)	the next value of the scolar facility F.
(**mref** M i)	the memory content of address i of memory M.
(**mnext** M i)	the next value for address i of memory M.
(**mwriting** M i)	`B1` if address i of memory M is being updated.
(**cond** $e1$ $e2$ $e3$)	if e_1 become `B1` then e_2 ,othewise e_3.
(**index** $e1$ $e2$)	the left-most position of substring matched to regular expression e_2.
(**rindex** $e1$ $e2$)	the right-most position of substring matched to regular expression e_2.
(**string** e)	string literal e

表 2: The List of Functors available in 2SECONDS Expressions