Phases: A Programming Construct for Sensor Applications

A. Arora[1], Y. Choi[2], M. G. Gouda[3], J. Hallstrom[4],
T. Herman[5], W. Leal[6], and N. Sridhar[7]

[1] The Ohio State University, anish@cse.ohio-state.edu
[2] The University of Texas at Austin, yrchoi@cs.utexas.edu
[3] The University of Texas at Austin, gouda@cs.utexas.edu
[4] Clemson University, jasonoh@cs.clemson.edu
[5] The University of Iowa, herman@cs.uiowa.edu
[6] The Ohio State University, leal@cse.ohio-state.edu
[7] Cleveland State University, n.sridhar1@csuohio.edu

Department of Computer Sciences,
The University of Texas at Austin,
Technical Report TR-07-27, April 2007

Abstract. DESAL is an NSF-sponsored project to produce a programming language, also called DESAL, for developing software for sensor network applications. The DESAL programming language is both state-based (rather than event-based) and rule-based (rather than being linearly executed). In this paper, we present a new programming construct called Phases, that can be incorporated into the DESAL programming language. This construct allows a programmer to develop software (for sensor network applications) that can run on many sensor networks, that support phases, without having to worry about the physical characteristics of these sensor networks, such as the number of sensors in the network, and the end-to-end delay of the network.

0 Introduction

DESAL is an NSF-sponsored project to produce a programming language, also called DESAL, for developing software for sensor network applications. The DESAL programming language is both state-based (rather than event-based) and rule-based (rather than being linearly executed). So far, two versions of the DESAL programming languages have been prototyped. The first version is developed in Clemson University by Professor Jason Hallstrom and his students. The second version is developed in University of Iowa by Professor Ted Herman and his students. DESAL-Clemson has more features of the DESAL programming language than DESAL-Iowa, whereas application programs compiled by DESAL-Iowa requires less storage than those compiled by DESAL-Clemson. We refer the reader to [1] for more information about the DESAL programming language.

In this paper, we specify a third version of DESAL, called DESAL-Texas. DESAL-Texas is also state-based and rule-based. However, DESAL-Texas is base station centric, and has a new programming construct called Phases. The phases construct allows a programmer to develop software (for sensor network applications) that can run on many sensor networks, that support phases, without having to worry about the physical characteristics of these sensor networks, such as the number of sensors in the network, and the end-to-end delay of the network. For convenience, we henceforth refer to DESAL-Texas simply as DESAL.

Each DESAL program, say named X, consists of one primary module, also named X, and one or more identical secondary modules, each is also named X. The primary module X resides in the base station and each secondary module X resides in a different sensor in the sensor network. Not every sensor in the sensor network needs to have a secondary module X, and the primary module X does not need to know priori the exact number of secondary modules that exist in the sensor network and where these modules reside.

Associated with the primary module X and each of the secondary modules X is the same list of formal parameters. For example, if only two parameters r and s are associated with each module in program X, then program X can be written in DESAL 1.0 as follows.

```
program X

  primary X(r : <type of r>, s : <type of s>)
```

```
    body of primary X that uses r and s

  end primary


  secondary X(r: <type of r>, s : <type of s>)

    body of the secondary X that uses r and s

  end secondary

end
```

Compiling this program X yields two executable files. One of these files, called primary executable X, is to be loaded into the base station, and a copy of the other file, called secondary executable X, is to be loaded into each sensor where this executable is needed. (Recall that not each sensor needs to host the secondary module X.)

After loading the executable files into the base station and the sensors of the sensor network, one can start executing program X by typing the following command in the base station.

```
    execute X(r:=3, s:=7)
```

Typing this command in the base station will cause two events to occur. First, the primary executable X in the base station starts executing after assigning the two values 3 and 7 to its two parameters r and s, respectively. Second, every secondary executable X in some sensor in the sensor network also starts executing after assigning the two values 3 and 7 to its two parameters r and s, respectively.

The first parameter r for each program X is of type positive integer, and so its value is at least 1. This parameter has a special meaning: it is the maximum number of "phases" that will be executed by each secondary module X before it terminates. It is also three less than the total number of "phases" that will be executed by the primary module X before it terminates. This rule indicates that the total number of phases executed by the primary module X is always three more than the maximum number

of phases executed by each secondary module X. (This rule is explained below.)

A phase has a fixed time period, chosen in the range of say 5 seconds to 1 minute, in each sensor network where DESAL programs are to be executed. The time period for a phase in a sensor network should be chosen to satisfy the following condition. The time period for a phase in a sensor network is an upper bound on the propagation delay that is experienced by any message sent from the base station to any sensor in the network or sent from any sensor in the network to the base station. This means that the time period for a phase in a sensor network depends on the number of sensors and how these sensors are distributed in the network. The good news is that the designers of any DESAL program do not need to know the time period for a phase in any sensor network in order to design their program. And yet the resulting program will be executing correctly on different sensor networks, possibly with different time periods for a phase.

Execution of the primary module X consists of a finite number of consecutive phases. Each phase consists of the following three steps. First, the "phase variable" in the primary module X is (re)assigned its initial value. Second, the "enabled actions" in the primary module X are executed one by one until all the actions in the primary module X become disabled. Third, the primary module X enters an idle state and stays in this idle state till the end of the time period for the phase. While in this idle state, the primary module X does not (and in fact can not) execute any of its actions, but its "shared variables" can be updated by the received updates from the secondary modules X that reside in some sensors in the sensor network. At the end of the phase, the primary module X starts to execute the three steps of the next phase, and the cycle repeats. The execution of the primary module X terminates when no actions are enabled for execution at the beginning of the second step of a phase.

Execution of every secondary module X, like that of the primary module X, consists of a finite number of consecutive phases. Each phase consists of the following three steps. First, the "phase variable" in the secondary module X is (re)assigned its initial value. Second, the "enabled actions" in the secondary module X are executed one by one until all the actions in the secondary module X become disabled. Third, the secondary module X enters an idle state and stays in this idle state till the end of the time period for the phase. While in this idle state, the secondary module X does not execute any of its actions, but at a random instant in the middle of the idle state, the secondary module X collects the values

of all the "shared variables", that have been written in the second step of this phase, and send them to the base station, which then uses these received values to update its own "shared variables". At the end of the phase, the secondary module X starts to execute the three steps of the next phase and the cycle repeats. The execution of the secondary module X terminates when no actions are enabled for execution at the beginning of the second step of a phase.

We can now explain the above rule that the total number of phases executed by a primary module X is always three more than the maximum number of phases executed by each secondary module X. Assume that the maximum number of phases executed by a secondary module X is r. Immediately before the base station starts executing the first phase of the primary module X, the base station sends a message to every sensor in the network requesting each sensor that has a secondary module X to start executing its secondary module X. Because the propagation delay of each of these messages is at most the time period for one phase, the execution of each secondary module X starts no later than the instant when the primary module X starts executing its second phase. Thus, execution of r-th phase of each secondary module X starts no later than the instant when the primary module X starts executing its (r+1)-th phase. During the r-th phase of a secondary module X, the current values of some shared variables in the secondary module X can be sent in a message to the primary module X. The propagation delay of this message is no more than the time period for one phase. Therefore, the sent values of the shared variables arrive at the primary module X during its (r+2)-th phase, and so the primary module X can read these values at the beginning of its (r+3)-th phase. This scenario shows the primary module X needs to have a total of (r+3) phases, where r is the maximum number of phases executed by a secondary module X.

Next, we describe seven examples of DESAL programs.

1 Example 1

In this example, we describe a DESAL program X that prints out at the base station the measured temperature of some sensor, say sensor 1150, in a sensor network. Program X consists of a primary module X that resides in the base station and one secondary module X that resides in sensor 1150. The primary module X has one shared variable tmp that stores the received temperature of sensor 1150, one local variable k that stores the number of phases that has so far been executed, and one phase variable

i whose value is reassigned the initial value 0 at the beginning of each phase. The total number of phases that the primary module X executes is r+3, and the total number of phases that the secondary module X executes is r, where r can be chosen 1, 2, 3, and so on.

The primary module X can be written in DESAL as follows.

```
primary X(r : integer)

shared tmp : integer   /initially 0
phase  i   : 0..1      /initially 0
local  k   : 0..r+3    /initially 0

begin
    i = 0  ^  k < r+2        -->  i := 1;
                                  k := k+1

[]  i = 0  ^  k = r+2        -->  if tmp <> 0 --> print tmp
                                  [] tmp =  0 --> print 'no tmp is rcvd'
                                  fi;
                                  i := 1;
                                  k := r+3
end
```

Execution of the primary module X consists of r+3 phases. In each of the first r+2 phases, the primary module X simply executes the first action once. Execution of these first r+2 phases basically does nothing except that it makes the primary module X wait for r+2 phase periods until its shared variable tmp is written (possibly more than once) by the secondary module X. In the last (r+3)-th phase, the primary module X executes its second action which causes X to print the current value of its shared variable tmp in the base station, provided that this current value is different from the initial value of 0.

The secondary module X can be written in DESAL as follows.

```
secondary X(r : integer)

shared tmp : integer   /initially 0
phase  i   : 0..1      /initially 0
local  k   : 0..r      /initially 0

begin
    i = 0  ^  k < r     -->  tmp := Gettemperature;
```

```
                              i := 1;
                              k := k+1
end
```

Execution of the secondary module X consists of r phases. In each phase, the secondary module X writes its current temperature into the shared variable tmp, and the written value of tmp is sent to the base station to be assigned to the shared variable tmp in the primary module X. Note that the secondary module X sends r consecutive temperature values to the primary module X, which gets to print only the last temperature value. This feature provides fault tolerance to the program X. It is possible that a value of the shared variable tmp sent by the secondary module X is lost, and does not reach the base station. But, a chance that a value of tmp sent by the secondary module X reaches the base station will increase as the secondary module X sends its value of tmp multiple times. The larger value r has, the more chance the primary module X has to receive at least one value of tmp sent by the secondary module X.

2 Example 2

In this example, we describe a DESAL program X that prints out at the base station the measured temperature of a certain sensor, specified by a parameter named "id". Program X consists of a primary module X that resides in the base station, and a secondary module X that resides in every sensor in the network.

The primary module X can be written in DESAL as follows.

```
primary X(r : integer, id: integer)

shared tmp  : integer  /initially 0
phase  i    : 0..1     /initially 0
local  k    : 0..r+3   /initially 0

begin
   i = 0  ^  k < r+2         -->  i := 1;
                                  k := k+1

[] i = 0  ^  k = r+2         -->  if tmp <> 0 --> print tmp, id
                                  [] tmp =  0 --> print 'no tmp from' id
                                  fi;
                                  i := 1;
```

```
                                    k := r+3
end
```

Execution of the primary module X in this example is similar to that of Example 1. In the last (r+3)-th phase, the primary module X prints not only the current value of its shared variable tmp, but also the identifier id of the sensor that sent this value of tmp.

The secondary module X can be written in DESAL as follows.

```
secondary X(r : integer, id: integer)

shared tmp : integer  /initially 0
phase  i   : 0..1     /initially 0
local  k   : 0..r     /initially 0

begin
   id=Getid ^ i = 0  ^  k < r    -->  tmp := Gettemperature;
                                      i := 1;
                                      k := k+1
end
```

Execution of the secondary module X in this example is also similar to that of Example 1. However, when a sensor receives a message that requests to start executing the secondary module X, the sensor needs to check whether this request is intended to it or not. In the second step of the first phase of the secondary module X, the secondary module X compares the received id to its own id, which can be obtained by calling the function Getid. If the received id is equal to its own id, the secondary module X executes r phases as it does in Example 1. Otherwise, the execution of the secondary module X terminates, since no actions are enabled for execution. Thus, execution of the secondary module X in the intended sensor consists of r phases, while execution of the secondary module X in every other sensor consists of only one phase.

3 Example 3

In this example, we describe a DESAL program X that prints out at the base station r consecutive measured temperatures of a sensor, whose identifier is id, in the sensor network. Program X consists of a primary module X that resides in the base station, and a secondary module X that

resides in every sensor in the network. The primary module X has one shared variable tmp, whose type is an array, that stores four consecutively received temperature values. Note that regardless of the value of r, we only need an array whose size is four to store and later process all the received temperature values.

The primary module X can be written in DESAL as follows.

```
primary X(r : integer, id : integer)

shared tmp  : array[0..3] of integer  /initially 0
phase  i    : 0..1                    /initially 0
local  k    : 0..r+3                  /initially 0

begin
    i = 0  ^  k < 3            -->  i := 1;
                                    k := k+1
[]  i = 0  ^  3<= k <= r+2     -->
        if tmp[k-3 mod 4] <> 0 -->  print tmp[k-3 mod 4], k-3, id;
                                    tmp[k-3 mod 4] := 0
        [] tmp[k-3 mod 4] =  0 -->  print 'no tmp from', k-3, id
        fi;
        i := 1;
        k := k+1
end
```

In each of the first 3 phases, the primary module X simply executes the first action once. Execution of these first 3 phases basically does nothing except that it makes the primary module X wait for 3 phase periods until the first element of its shared variable tmp is guaranteed to be written by the secondary module X in sensor id. In each of the remaining phases, the primary module X executes its second action once. The execution of the second action at phase k of the remaining phases causes X to print the current value of tmp[k-3 mod 4], the phase number (i.e. k-3) in which this temperature value is measured by the secondary module X, and the identifier id of the sensor that sent the temperature values.

The secondary module X can be written in DESAL as follows.

```
secondary X(r : integer, id: integer)

shared tmp : array[0..3] of integer  /initially 0
phase  i   : 0..1                    /initially 0
```

```
local  k   : 0..r                    /initially 0

begin
    id=Getid ^ i = 0  ^  k < r     -->  tmp[k mod 4] := Gettemperature;
                                        i := 1;
                                        k := k+1
end
```

For the sensor whose identifier is "id", each phase of the secondary module X writes the current temperature into tmp[k mod 4], and the written value of tmp[k mod 4] is sent to the base station to be assigned to the shared variable tmp[k mod 4] in the primary module X.

4 Example 4

In this example, we describe a DESAL program X that prints out at the base station the average value of r consecutive measured temperatures of a sensor id in the sensor network. Program X consists of a primary module X that resides in the base station, and a secondary module X that resides in every sensor in the network. The primary module X has two shared variables, a shared variable tmp that stores the received average temperature, and a shared variable nmb that stores the number of temperature values considered for the average temperature in tmp.

The primary module X can be written in DESAL as follows.

```
primary X(r : integer, id : integer)

shared tmp  : integer  /initially 0
shared nmb  : 0..r     /initially 0
phase  i    : 0..1     /initially 0
local  k    : 0..r+3   /initially 0

begin
   i = 0  ^  k < r+2         -->  i := 1;
                                  k := k+1

[] i = 0  ^  k = r+2         -->  print tmp, nmb, id;
                                  i := 1;
                                  k := r+3
end
```

In the last (r+3)-th phase, the primary module X prints the current values of its shared variables tmp and nmb, and the identifier of the sensor that sent these values.

The secondary module X can be written in DESAL as follows.

```
secondary X(r : integer, id: integer)

shared tmp : integer  /initially 0
shared nmb : 0..r     /initially 0
phase  i   : 0..1     /initially 0
local  k   : 0..r     /initially 0
local  sum : integer  /initially 0

begin
   id=Getid ^ i = 0  ^  k < r     -->  sum := sum + Gettemperature;
                                       tmp := sum/(k+1);
                                       nmb := k+1;
                                       i := 1;
                                       k := k+1
end
```

The secondary module X has one more local variable named "sum", which stores the sum of measured temperatures. For a sensor whose identifier is "id", in each phase k, the secondary module X computes the sum of measured temperates until phase k, and writes the average temperature and the number of measured temperature values into its shared variables tmp and nmb, respectively. These written values of tmp and nmb are sent to the base station to be assigned to the shared variables tmp and nmb in the primary module X, respectively.

5 Example 5

In this example, we describe a DESAL program X that prints out at the base station the average values of r consecutive measured temperatures of n sensors in the sensor network. Program X consists of a primary module X that resides in the base station, and a secondary module X that resides in every sensor in the network. The primary module X has two shared variables, tmp and nmb. Each of these shared variable is of type "structure" that can store n values. Shared variable tmp stores the received average temperature values of any n sensors in the network, and shared variable nmb stores the received number of temperature values,

of the same n sensors, considered for the average temperature values in tmp.

When the base station receives the value of a shared variable from a sensor, the base station checks if there exists an element in each of its structures, tmp and nmb, for this sensor or not. If there is an element for this sensor, then the base station assigns the received value to the element in the corresponding structure. Otherwise, the base station checks if there is an empty element in each of its structures. If yes, the base station reserves an empty element in each of its structures for this sensor, and assigns the received value to the element in the corresponding structure. If no, the base station drops the received value. Therefore, if the total number of sensors in the network is larger than n, then all sensors in the network will execute their secondary modules, but the primary module X only keeps track of the values of shared variables of n sensors, and drops the received values of shared variables of the remaining sensors. If the total number of sensors is smaller than or equal to n, then the primary module X can keep track of the values of shared variables of all the sensors in the network.

The primary module X can be written in DESAL as follows.

```
primary X(r : integer)

constant n = <fixed positive integer>
shared tmp  : structure [0..n-1] of integer /initially 0
shared nmb  : structure [0..n-1] of 0..r    /initially 0
phase  i    : 0..1                          /initially 0
local  k    : 0..r+3                        /initially 0
local  v    : 0..n                          /initially 0

begin
   i = 0  ^  k < r+2    -->  i := 1;
                             k := k+1

[] i = 0  ^  k = r+2    -->
        v := 0;
        do v<n ^ ID(v)<>0 --> print tmp.v, nmb.v, ID(v);
                              v := v+1
        od;
        i := 1;
        k := r+3
end
```

In the last (r+3)-th phase, the primary module X first checks if each element in its structure has been reserved for a certain sensor. This can be done by calling the function ID(v). This function ID(v) returns 0 if each element whose index is v in its structure has not been reserved for any sensor. Otherwise, this function returns an associated sensor id for the element whose index is v. If ID(v) is not zero, the primary module X prints the current values of tmp.v, nmb.v, and ID(v).

The secondary module X can be written in DESAL as follows.

```
secondary X(r : integer)

shared tmp : integer  /initially 0
shared nmb : 0..r     /initially 0
phase  i   : 0..1     /initially 0
local  k   : 0..r     /initially 0
local  sum : integer  /initially 0

begin
   i = 0  ^  k < r     -->  sum := sum + Gettemperature;
                            tmp := sum/(k+1);
                            nmb := k+1;
                            i := 1;
                            k := k+1
end
```

6 Example 6

In this example, we describe a DESAL program X that prints out at the base station the values of measured temperatures and/or pressures of n sensors in the sensor network. Program X consists of a primary module X that resides in the base station, and a secondary module X that resides in every sensor in the network. The primary module X has two shared variables, tmp and prs, that are of type "structures". Shared variable tmp stores the received temperature values of any n sensors in the network, and shared variable prs stores the received pressure values of the same n sensors.

The primary module X can be written in DESAL as follows.

```
primary X(r : integer)
```

```
constant n = <fixed positive integer>
shared tmp  : structure [0..n-1] of integer /initially 0
shared prs  : structure [0..n-1] of integer /initially 0
phase  i    : 0..1                          /initially 0
local  k    : 0..r+3                        /initially 0
local  v    : 0..n                          /initially 0

begin
   i = 0  ^  k < r+2    -->  i := 1;
                             k := k+1

[] i = 0  ^  k = r+2    -->
        v := 0;
        do v<n ^ ID(v)<>0 --> print tmp.v, prs.v, ID(v);
                              v := v+1
        od;
        i := 1;
        k := r+3
end
```

The secondary module X can be written in DESAL as follows.

```
secondary X(r : integer)

shared tmp : integer  /initially 0
shared prs : integer  /initially 0
phase  i   : 0..1     /initially 0
local  k   : 0..r     /initially 0

begin
   i = 0  ^  k < r   -->  if Gett   -->  tmp := Gettemperature;
                          [] ~Gett  -->  skip
                          fi;
                          if Getp   -->  prs := Getpressure;
                          [] ~Getp  -->  skip
                          fi;
                          i := 1;
                          k := k+1
end
```

In each phase, the secondary module X first checks if it can get temperature value by calling the function Gett, which returns a boolean value. If function Gett returns true, then it writes its current temperature value into its shared variable tmp. Then it checks if it can get pressure value

by calling the function Getp, which returns a boolean value. If function Getp returns true, then it writes its current pressure value into its shared variable prs. Notice that the secondary module X can send a temperature value only, a pressure value only, or both values to the primary module X, depending on its abilities to measure temperature and pressure.

7 Example 7

In this example, we describe a DESAL program X that prints out at the base station one measured temperature of some sensor in the sensor network. Program X consists of a primary module X that resides in the base station and a secondary module X that resides in every sensor in the network. The primary module X has one shared variable tmp that is of type structure and whose size is 1. Thus, the primary module X keeps track of the received temperatures of a sensor that sent its temperature value to the base station for the first time, while the primary module X drops all received temperatures from the other sensors in the network.

The primary module X can be written in DESAL as follows.

```
primary X(r : integer)

shared tmp  : structure [0..0] of integer  /initially 0
phase  i    : 0..1                         /initially 0
local  k    : 0..r+3                       /initially 0

begin
    i = 0  ^  k < r+2   -->  i := 1;
                             k := k+1

[]  i = 0  ^  k = r+2   -->  if ID(0) <> 0 --> print tmp.0, ID(0)
                             [] ID(0) =  0 --> print 'no tmp is rcvd'
                             fi;
                             i := 1;
                             k := r+3
end
```

The primary module X in this example is very similar to that in Example 1. In the last (r+3)-th phase, both modules prints out one temperature value from a sensor. The primary module X in Example 1 has a shared variable tmp, that is of type integer, to share a temperature value with a particular sensor, but the primary module X in this example has

a shared variable tmp, that is of type structure whose size is 1, to share a temperature value with any one sensor in the network. As a shorthand, we can replace the type "structure [0..0] of integer" in this example by the type "integer" in Example 1.

The secondary module X can be written in DESAL as follows.

```
secondary X(r : integer)

shared tmp : integer   /initially 0
phase  i   : 0..1      /initially 0
local  k   : 0..r      /initially 0

begin
    i = 0  ^  k < r     -->  tmp := Gettemperature;
                             i := 1;
                             k := k+1
end
```

Acknowledgment

This work was supported in part by the US National Science Foundation under Grant No. 0520250.

References

1. A. Arora, M. Gouda, J. Hallstrom, T. Herman, W. Leal, and N. Sridhar. A state-based language for sensor-actuator networks. In *Proceedings of the International Workshop on Wireless Sensor Network Architecutre (WWSNA-07)*, 2007.