Accessing Databases from Esterel[*]

David White & Gerald Lüttgen

Department of Computer Science, University of York
Heslington, York YO10 5DD, UK

E–mail: {dhw100, luettgen}@cs.york.ac.uk

Technical Report, December 2004

Abstract

A current limitation in reactive systems design is the lack of database support in tools such as Esterel Studio. This report proposes a way of integrating databases and Esterel by providing an Application Programming Interface (API) for relational database use inside Esterel.

As databases and Esterel programs are often executed on different machines, result sets from database queries may be processed either locally and according to the synchrony hypothesis, or remotely along several reactive cycles. These different scenarios are reflected in the development of two APIs detailed in this report. Their utility is demonstrated by means of a case study modelling a warehouse storage system as might be used by a direct order company. The system employs a robot whose task it is to collect items from a customer's order and assemble them in one place. In addition to customer and order data, the underlying database stores spatial data detailing the position of items in the warehouse. Both robot and warehouse are implemented using the Lego Mindstorms robotics system.

1 Introduction

One of the current limitations in reactive systems programming is the lack of *database support* available within *synchronous languages*, such as Esterel [3, 4] and Lustre [8], and their development environments, *Esterel Studio* and *SCADE* [10], respectively. As is, a system designer needs to modify auto–generated code by hand in order to interface to databases, which is both a difficult and error–prone business. This is a problem very much relevant in industry since some reactive systems programmed in synchronous languages would benefit from an easy model of database interaction. For example, synchronous languages are often used to build the flight software for aeroplanes. Adding

[*]This research has been funded by an Undergraduate Research Bursary of the Nuffield Foundation under grant ref. URB/01528/6.

database interaction would enable mapping data to be retrieved and processed directly by the reactive kernel implementing the flight software, so that maps of the currently overflown area could be displayed to pilots in real time.

This report addresses that limitation by providing an *Application Programming Interface* (API) for using relational databases within the Esterel programming language. We choose MySQL [16] as the database and, since reactive kernels are produced as C programs by the academic Esterel compiler [2, 9], the APIs are implemented using the MySQL C interface [17]. MySQL is selected here simply for its convenience, since it is widely used in academia. However, our work can as easily be applied to other relational databases. To the best of our knowledge, no articles on database integration within Esterel have been published before in the academic literature. Indeed, other reactive systems design tools seem to be very limited in this respect as well, including Mathworks' *Simulink/Stateflow* [5] and iLogix' *Statemate* [13].

Because database transactions are relatively complex when compared to responses of reactive kernels, databases and reactive programs must be considered as running asynchronously to each other. This is true regardless of whether they reside on the same machine or on different machines. In the former case, however, *result sets* to database queries may reasonably be assumed to be processed within a single synchronous step of the reactive kernel. In the latter case, result sets are necessarily processed asynchronously to the reactive kernel. For these reasons, an API for each situation is provided: a *Local Result Set API* and a *Remote Result Set API*. The realisation of both relies on Esterel's formidable support for extending the language via external data types and external functions and procedures. While the local result set API makes heavy use of external functions and procedures, the remote result set API only employs valued signals. This is because external functions and procedures are expected to be instantaneous in Esterel.

We demonstrate the utility of our APIs by means of a case study involving a *warehouse storage system*. The idea behind this is that of a direct order company, i.e., orders must be picked from items stored in a warehouse. Accordingly, a *robot* is built and programmed to pick up and drop off items, therefore making it possible for a complete order to be collected and stored. The orders, as well as information about the items, are provided by a database running remotely to the robot on a PC. Part of the information stored on an item is its position in the warehouse, thereby providing mapping data for the robot. The case study is implemented using *Lego Mindstorms* robotics kits [1, 15], which provides a small programmable brick, called *RCX*, that houses a microcomputer capable of running an Esterel reactive kernel. Sensors and actuators connected to the RCX, such as *touch*, *light* and *rotation sensors* and *motors*, respectively, permit interaction with the RCX's environment. The RCX also has a built–in infrared port, which we use to communicate with the warehouse database on the PC.

The remainder of this report is structured as follows. The next section describes both our APIs, emphasising the general model of interaction between Esterel reactive kernels and databases. Our case study is presented in Sec. 3 which provides an example of the usage of the Local Result Set API. Sec. 4

contains our conclusions and suggestions for future work. The appendices give details of the realisation of our APIs and the case study: App. A reproduces the Perl scripts implementing our APIs, while App. B states the Esterel program for each reactive kernel developed for our application. Finally, App. C presents our script for transforming the generated C code for the warehouse robot to one that may be run on the Lego RCX.

2 Database APIs for Esterel

Due to the variety of computing architectures in which an Esterel reactive kernel together with a database may be used, two different APIs for enabling database access within Esterel are devised. The APIs exploit the built–in extensibility of the Esterel language and provide dedicated signals, external data types and external functions and procedures for use inside Esterel.

Both APIs allow local and remote databases to be queried. They consider databases as part of the system environment and as running asynchronously to the reactive kernel. This is because database transactions are typically more complex to process than ordinary reactions. The APIs differ in the storage location considered for the result set returned by a query. Result sets are database tables presented as sets of rows, from which the reactive kernel extracts desired information according to its needs. The *Local Result Set API* views the result set as being local to the reactive kernel, whence operations on the result set can conceptually be considered to take zero time, satisfying the synchrony hypothesis [12]. This API can thus make heavy use of user–defined external functions and procedures, which are required to be instantaneous in the Esterel language [3]. The *remote result set API* allows the result set to be stored remotely to the reactive kernel. As a consequence, this API cannot profit from the elegance and simplicity of employing external functions and procedures, but must solely rely on signals for processing result sets.

2.1 Local Result Set API

The most logical way to view the interaction between a reactive kernel programmed in Esterel and a database is to regard the database simply as an extension of the reactive kernel's environment. For this reason all interactions with the database from within Esterel are modelled using input and output signals, as these are Esterel's facilities for communicating with the environment. Therefore, to perform an operation on the database, a dedicated output signal is emitted, parameterised in a string that formulates a query in SQL syntax. The database's response is awaited via a dedicated input signal whose parameter carries an identifier pointing to the result set. During the time between the emitted query and the results returning, the database is queried and the whole result set is transferred back to the site that also runs the reactive kernel. Note that multiple databases can simply be supported by declaring a dedicated output and input signal for each database.

Once a database has been queried and a result set returned, data can be extracted from the result set using dedicated operations implemented using Esterel's external function and external procedure facility. Note that this is possible since both the result set and the reactive kernel reside at the same site, which implies that accesses of the result set by the kernel may be considered as instantaneous. If the query's SQL command is one that does not return results, such as the command for the deletion of data items, then the only operation provided is one to check the number of affected rows. If the SQL command did return a result set however, the set may be accessed by successively reading the result set row–by–row, extracting the specific data items from each row and coercing them into native Esterel types. Once all rows have been processed, an operation shall be called to free the memory occupied by the result set.

Table 1: Services offered by the Local Result Set API

```
type MYSQL_RES_ptr;
type MYSQL_ROW;

procedure appstr() (string, string);
procedure appint() (string, integer);
procedure appbol() (string, boolean);
procedure appflt() (string, float);
procedure appdou() (string, double);

output <Signal name for emitting query> : string;
input <Signal name for returning results> : MYSQL_RES_ptr;

function check_result(MYSQL_RES_ptr) : boolean;
function get_next_row(MYSQL_RES_ptr) : MYSQL_ROW;
function num_rows(MYSQL_RES_ptr) : integer;

function getint(MYSQL_ROW, integer) : integer;
function getbol(MYSQL_ROW, integer) : boolean;
function getdou(MYSQL_ROW, integer) : double;
function getflt(MYSQL_ROW, integer) : float;
function getstr(MYSQL_ROW, integer) : string;

function num_affected_rows(MYSQL_RES_ptr) : integer;
procedure clear_results() (MYSQL_RES_ptr);
```

Esterel's interaction with a remote database and its local processing of the result sets thus leads to the API displayed in Table 1. In the remainder of this section we explain the API's services in more detail.

We begin with the formation of a query string containing SQL commands. Since Esterel does not provide any facilities for building strings, the string must be generated using a series of append operations. The API offers an append operation for each of Esterel's native data types and implements these operations using Esterel's external procedure call function. For example, the following generates a query which uses an integer variable `order_id`:

```
var query_string : string in
    query_string := "select * from orders where order_id = ";
    call appint() (query_str, order_id);
end var
```

As mentioned before, interaction with the database is via an output query signal and an input result signal. For each database used, they should be declared as such:

```
output item_db_query : string;
input item_db_results : MYSQL_RES_ptr;
```

Each pair of signal names can be chosen by the user. The mapping between chosen signal names and the actual databases is defined elsewhere (cf. Sec. 2.3). The data returned on a result signal is simply an identifier of the external type `MYSQL_RES_ptr` defined in our API's implementation. Note that the identifier value should never be copied since this would not result in a full copy being performed. Our framework effectively allows only one result set per database connection; however, if more result sets are required simultaneously within some Esterel application, additional connections to the same database may be declared.

To ensure that the results are received correctly from the database, the results signal should be *awaited* immediately following the emission of the query:

```
emit item_db_query("select * from item");
await item_db_results;
```

If two queries are issued simultaneously, then the result signals must be awaited in parallel or using *immediate await* statements.

In order to check the success of the SQL command, the boolean function `check_result` should be called and passed the identifier of the result set, i.e., the value of the input result signal. If it returns true, the query succeeded and the operations described below may be used to access the data inside the result set. If it returns false, the data in the result set is not valid.

For working with a result set that contains data — as opposed to an empty one returned by, e.g., an SQL insert statement — rows must be declared inside Esterel. Rows are declared to be of external type `MYSQL_ROW` which is defined in our API's implementation. The lifetime of any data loaded into a row from a result set lasts only as long as the result set itself, i.e., up to the time the `clear_results` operation is called. We recommend that rows are only declared locally and that their scope finishes before the call to `clear_results` occurs.

The functions provided to operate on the result set and rows will now be described. Most of the operations mirror the equivalent MySQL function from the MySQL C API [17]. This interface was chosen for two reasons: the MySQL

C functions are widely and well known and, at a later date, additional functions can easily be included, if desired. Function `get_next_row` is required to load data into a row from a result set. It is passed a result set identifier and, each time it is called, it will return the next row in the result set. Generally, the program will need to know how many rows there are in the result set and, therefore, how many times to call `get_next_row`. This is accomplished by a call to function `num_rows` which, when passed a result set identifier, returns the number of rows in the result set. Once a row has been loaded from a result set, data can be extracted using a `get<type>` function which is provided for each of the native Esterel data types. In addition to a row, an integer is also passed indicating the index of the column from where the data is to be retrieved. The following is a simple example of data extraction using our API:

```
var row_holder : MYSQL_ROW,
    item_name : string,
    item_location : integer in
  row_holder := get_next_row(?item_db_results);
  item_name := getstr(row_holder, 1);
  item_location := getint(row_holder, 2);
end var;
```

In this case, the results are identified with the valued signal `item_db_results`, and the item's name and location are in the second and third column of the row, respectively. Note that indexing is as in the C programming language and thus starts with index 0.

One function that remains to be described for accessing the result set is `num_affected_rows`. This is used when the result set contains no data but a user wants to know how many rows were affected by the SQL command. As such, `num_affected_rows` can be employed to test the success of an SQL query, e.g., to check whether a delete query has had the desired effect.

The final operation provided by the API, which has already been referred to above, clears the memory occupied by the result set:

```
procedure clear_results()(MYSQL_RES_ptr);
call clear_results()(?item_db_results);
```

It is essential here that there are no rows loaded from the result set after it is cleared, since the data within these rows is cleared with the result set as well.

2.2 Remote Result Set API

The Remote Result Set API should be used in situations where it is not feasible to transfer the entire result set to the system running the reactive kernel, i.e., when both the database *and* the result set must be viewed as part of the environment. Since remote communication must be taken into account, the API is quite different to that of the Local Result Set API. This is because external functions and procedures can only be used in Esterel if their operations may be considered as instantaneous [3]. Consequently, one must either employ Esterel's task concept or must solely rely on signals. In both cases and as a consequence of operations on the result set not being instantaneous, the Esterel kernel must be informed of when an operation is complete. This is accomplished by awaiting an "acknowledge" signal after every operation. In the remainder we focus on the solution via signals rather than tasks, as the authors were unfamiliar with Esterel's intricate task concept at the time the research was carried out.

An important aim of the Remote Result Set API is to minimise the amount of data that must be transfered. To achieve this, rows are not handled by the reactive kernel but are moved database–side instead. As a consequence, rows are part of the environment, like the result set. To prevent the complexity of handling multiple rows in the kernel, each database is limited to only one row. This is a reasonable restriction since systems that use this API are unlikely to be performing complex database manipulations that require multiple rows. If multiple row access should indeed be necessary, additional connections can be specified to achieve that. This work–around can also be employed to support multiple databases, similar to what is suggested for the Local Result Set API.

Table 2: Services offered by the Remote Result Set API

```
input <db_id>_ackstr : string;
input <db_id>_ackint : integer;
input <db_id>_ackbol : boolean;
input <db_id>_ackflt : float;
input <db_id>_ackdou : double;

procedure appstr() (string, string);
procedure appint() (string, integer);
procedure appbol() (string, boolean);
procedure appflt() (string, float);
procedure appdou() (string, double);

output <db_id>_query_out : string;   after emission, await <db_id>_ackbol
output <db_id>_fetch_next_row;       after emission, await <db_id>_ackbol
output <db_id>_num_rows;             after emission, await <db_id>_ackint
output <db_id>_num_affected_rows;    after emission, await <db_id>_ackint
output <db_id>_clear_results;        after emission, await <db_id>_ackbol

output <db_id>_getint : integer;     after emission, await <db_id>_ackint
output <db_id>_getbol : integer;     after emission, await <db_id>_ackbol
output <db_id>_getflt : integer;     after emission, await <db_id>_ackflt
output <db_id>_getdou : integer;     after emission, await <db_id>_ackdou
output <db_id>_getstr : integer;     after emission, await <db_id>_ackstr
```

Our API for remote result set access is displayed in Table 2. The remainder of this section explains the API's services in more detail. Similar to the naming of the query and result signals in the Local Result Set API, each signal `signal_name` is prefixed with a textual database identifier `db_id`, which we denote by `<db_id>_signal_name`. Moreover, the offered string generation functions are identical to those in the Local Result Set API.

The main difference to the Local Result Set API is the way in which the results to a query are accessed. There is now one result set and one row per database defined, and since each signal is prefixed by a unique string, there is no need for a result set identifier to be returned. The only data returned after issuing a query is the success of that query. Therefore, after a query has been issued, the boolean acknowledge input signal for that database must be immediately awaited:

```
emit item_db_query("select * from item");
await <db_id>_ackbol;
```

Its carried value is the same as the one returned by the `check_result` operation in the Local Result Set API.

The value of the `<db_id>_ackbol` input signal should then be tested to determine whether the query has succeeded or not. If the query has succeeded and the result set is not empty, then the first row can be loaded by emitting the `<db id>_fetch_next_row` signal. It is again necessary to await `<db_id>_ackbol` after this to determine when the operation has completed. True is returned if there exists a valid row to load, and false if there are no more rows available.

Now that a row has been loaded, the signals for accessing it may be used. Similar to the Local Result Set API, there is a `<db_id>_get<type>` signal for each of the Esterel native types. In the following example, it is shown how to extract an integer from the first column of the row:

```
emit <db_id>_getint(0);
await <db_id>_ackint;
order_id := ?<db_id>_ackint;
```

All the operations contained in the Local Result Set API are also available in the Remote Result Set API, but implemented as signals instead of external functions. However, in order to cope with the additional remote nature of result sets and thus rows, it is necessary to await an acknowledge signal after each operation.

2.3 Implementation of the APIs

Both APIs are implemented in a combination of Perl and C. The implementations heavily rely on the MySQL C API [17] and are, for most parts, rather straightforward. The complete source code for both APIs is included in App. A.

The involvement of the Perl scripting language [20] in the realisation of the APIs may be surprising at first. The reason is our desire to support *multiple* databases with *user–declared* signal names for emitting SQL queries and awaiting result sets. As reactive systems typically interact with an arbitrary but fixed

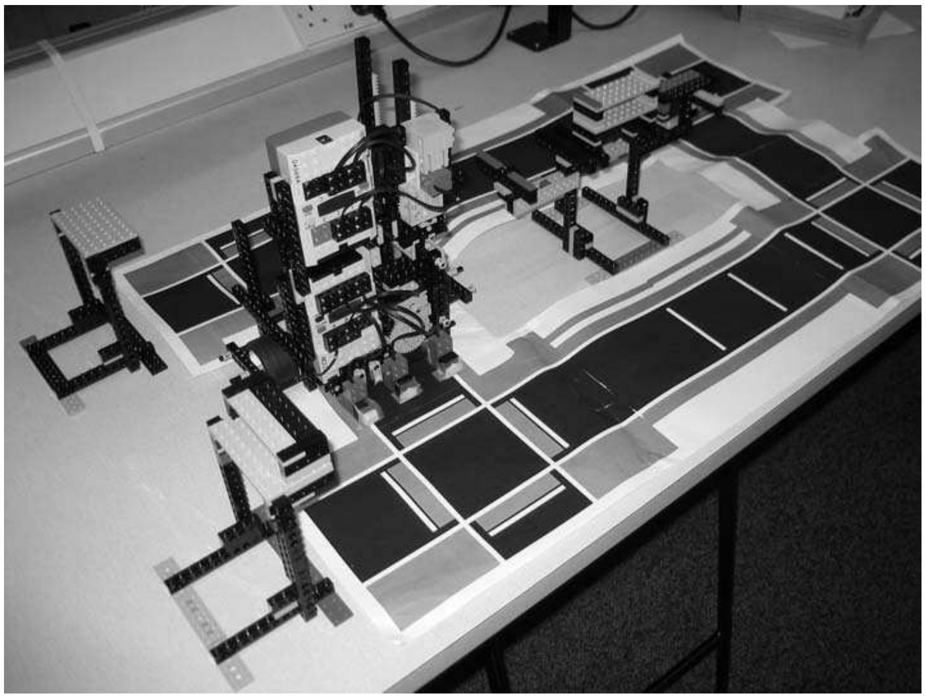

Figure 1: Aerial photograph of our warehouse system.

number of databases, it is unnecessary to provide API services for dynamically binding signal names to databases; even a static binding should not be defined within an Esterel program at all, as it is not part of reactive behaviour. Instead, we opt to provide such a binding as a parameter to our Perl script for each API, which appropriately combines the C–code generated by the academic Esterel compiler [9] for some reactive program with C–code implementing the APIs services used in the program.

Since database connections are generally permanent throughout the time a reactive system is running, our APIs provide no explicit facilities for connecting and disconnecting from a database. Instead, connection and disconnection is handled implicitly by the APIs. However, if explicit connect and disconnect services would be required from within an Esterel program, the APIs could be extended by defining and implementing according dedicated signals.

3 Case Study

In this section we present a case study demonstrating the utility of our APIs: an automated warehouse storage system modelling a direct order company, where items from orders are picked, stored and finally removed from the warehouse. This requires producing a warehouse containing various items and a robot capable of moving the items within the warehouse, for which we use *Lego*

Mindstorms' robotics kits [15] (cf. Fig. 1). Lego Mindstorms provides both a construction tool with sensors and actuators and a microcontroller, called the RCX, which is capable of running a reactive kernel programmed in Esterel. The database behind our warehouse model is that of a standard order system but which also includes mapping data about the location of the items. As this case study is meant to exemplify the use of our database APIs, only the part of the solution employing the APIs is focused on below.

3.1 Lego Mindstorms, the RCX and BrickOS

Lego Mindstorms is a platform for building computer–controlled robots within the *Lego system* [15]. At the heart of Lego Mindstorms is the RCX. This "brick" is a small battery–powered computer system capable of controlling up to three *actuators* and reading up to three *sensors*. In Lego, actuators are normally motors, and sensors can be light, rotation or touch sensors. Each RCX also provides an *infrared transmitter and receiver* used for both downloading programs from a PC and for inter–RCX communication. The infrared download device used on the PC can also participate in communications with RCXs.

The RCX provides great flexibility though its re–programmable firmware. BrickOS [6], formerly known as LegOS, is an open source replacement firmware for the RCX. It boasts a number of features that make it considerably more complex than the standard Lego firmware. Foremost, it allows programs written in the C programming language to be executed on the RCX. Obviously this is especially important for this project since the Esterel compiler [9] generates C code as a target language.

BrickOS also provides infrared communication through the *Lego Network Protocol* (LNP) [6]. This protocol has two layers, an integrity layer and an addressing layer. The integrity layer guarantees that, if a message is received, it will be the same message that was sent — similar to the Internet protocol UDP. However, it differs from UDP in that it implements a broadcast mode, i.e., any RCXs in the receiving area will pick up the message. To provide *unicast* messages, an addressing layer is placed on top of the network protocol stack. In our Lego Mindstorms' setting, each RCX has a unique identifier which serves as its address and is specified when the BrickOS firmware is downloaded.

3.2 Hardware

The hardware requirements of our warehouse storage system are high in Lego Mindstorms' terms, requiring more sensors and actuators than one RCX can control. Therefore, it is necessary to use two RCXs, one to control the movement of the robot and the second to control the forklift, hereafter referred to as the *Movement RCX* and *Forklift RCX*, respectively. Again, communication between the two RCXs is handled using the infrared link provided on each RCX. Because the Forklift RCX does not need to communicate with the PC, the infrared download tower is set up to allow the following communication to take place:

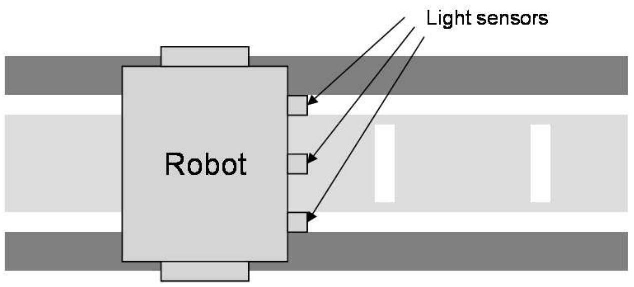

Figure 2: Track section.

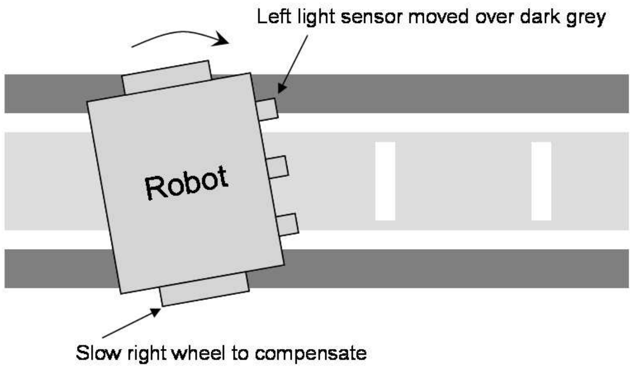
Left light sensor moved over dark grey
Robot
Slow right wheel to compensate

Figure 3: Track enabling straight robot movement.

host computer to Movement RCX and Movement RCX to Forklift RCX, as shown in Fig. 6 below.

3.2.1 Movement Subsystem and Track Description

Since the robot is required to pick up and place items in the warehouse with high precision and since the movement of the robot greatly influences its precision, it is necessary to make the movement subsystem as accurate as possible. Our solution to this challenge involves a combination of a drive system that guarantees straight line motion and facilities for the robot to detect an error in its direction and correct it. The underlying drive that moves the robot is implemented as a *dual differential drive*. This provides mechanically guaranteed straight motion and precise control of each wheel's speed. Each section of track that the robot moves on has two widely separated thin white lines over which two light sensors rest (cf. Fig. 2, outer light sensors). By using *thin* widely spaced tracks, the number of positions the robot can be in and that the light sensors still register as straight are limited, and therefore accuracy is enhanced. Any error in the starting direction will be detected by the light sensors moving over a non–white

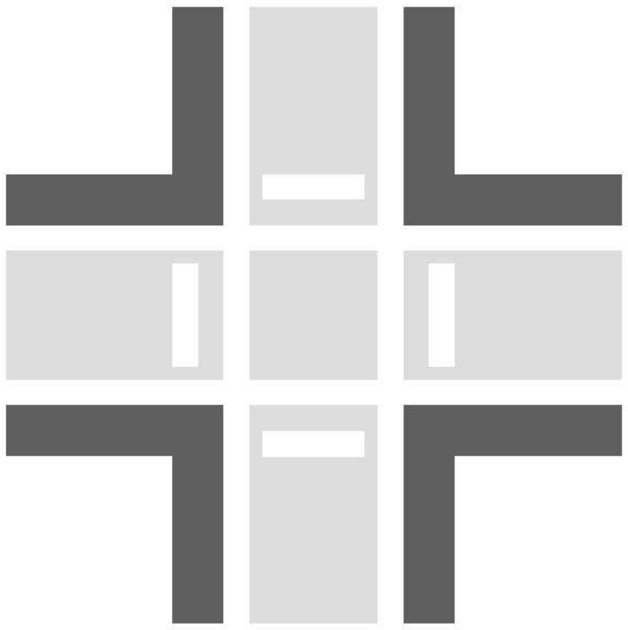

Figure 4: Track intersection.

coloured part of the track. The areas inside the track and outside the track are different colours, so the error's direction can be inferred and corrected for by slightly slowing one wheel (cf. Fig. 3). This is the kind of precise wheel control that the dual differential drive excels at.

Distance travelled by the robot is determined using a third light sensor which rests over the middle of the track. Thin white lines in the middle of the track, perpendicular to the direction of travel, are counted by this sensor, therefore enabling the robot to determine what distance it has travelled. Rather than placing the white lines a regular distance apart, they are placed beside objects that the robot might need to perform an operation at, such as to turn or to pick up an item. This enables the robot to stop at precisely the correct place and prevents error in the measured distance from accumulating.

At corners where sections of track intersect, the white lines used for distance measuring are replaced by the movement lines of the perpendicular track (cf. Fig. 4). The movement lines of the perpendicular track are designed to be identical in width to the distance measuring lines, thus providing a system that works over a track intersection.

At intersections, the same movement lines help the robot to perform precise 90 degree turns. After beginning the turn, the condition of both light sensors over a white part of the track is awaited. This indicates that the robot has turned 90 degrees. Again, because the lines are widely spaced, at the end of a turn the robot will be very close to straight for the next section of track.

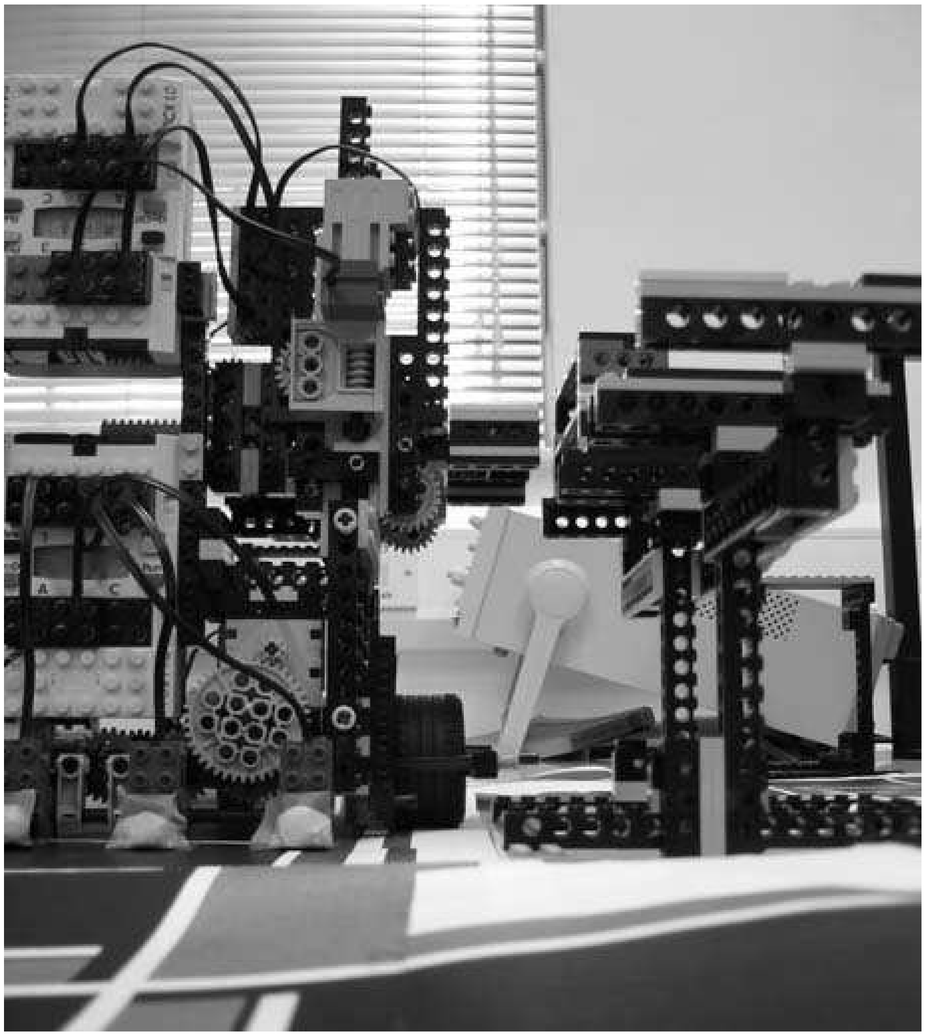

Figure 5: Forklift and palates.

3.2.2 Forklift Subsystem and Item Storage Design

To manipulate the warehouse items, a *forklift* design was chosen. To be compatible with the forklift design, all items must be placed on *palates* that are the right size for the forklift. Placing the items on palates also allows items to be stacked. We assume that all items of the same type, i.e., the same product, will be stored in one stack (cf. Fig. 5).

The forklift subsystem differs slightly from a normal forklift: instead of moving the whole robot forward to get the lift under a palate, the forklift arm is simply extended. This means that the forklift is situated on one side of the

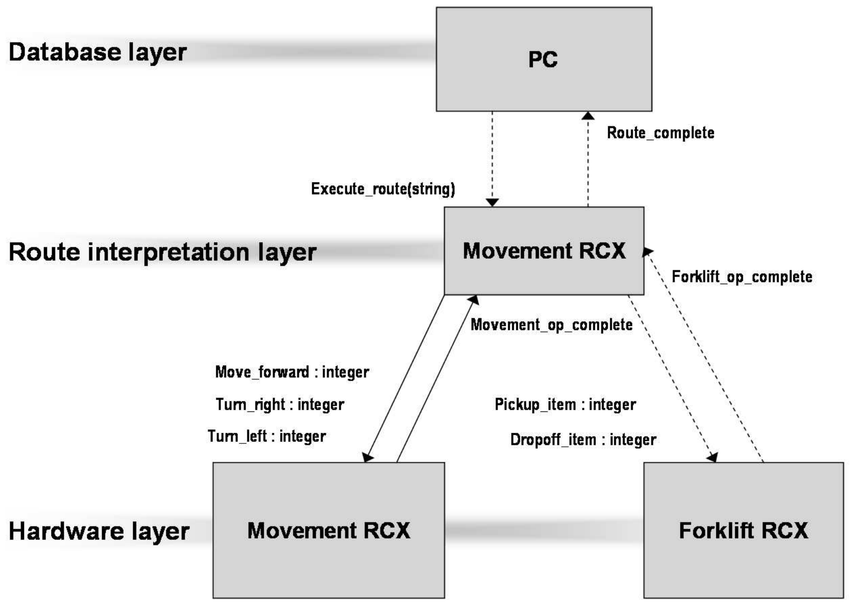

Figure 6: Diagram showing the inter–layer signals. (A normal line indicates internal communication and a dashed line communication over infrared link.)

robot rather than the front. The raising of the forklift and the extending of the arm are both controlled by motors and feedback is provided by two rotation sensors. Also, because the forklift subsystem is physically built on top of the movement subsystem, it is not possible to retrieve items from ground height. Therefore, all items are raised on top of a shelf system.

3.3 Software

The software for the warehouse system is structured in three layers: *database access layer*, *route interpretation layer* and *hardware layer*. Each device used in the system, the two RCXs and the PC, executes a reactive kernel programmed in Esterel. The database layer runs on the PC and is responsible for accessing the warehouse's database and for generating routes for the robot to execute. The second layer runs on the Movement RCX and interprets the route sent from the PC. The third layer is responsible for interacting with the Lego hardware and performs operations such as *move the robot* and *pick up the item*. This layer is present on the Movement RCX and the Forklift RCX. The signals used for communicating between the various layers are shown in Fig. 6.

3.3.1 Database

Our warehouse's database models a simple ordering system. Since the main emphasis within this case study is on retrieving spatial data, the aspects of the database concerning ordering information are kept as simple as possible: a customer may make many orders, each order is identified by an order identifier and must contain one or more order lines, and each order line must contain exactly one item. An SQL schema describing the database is given in App. B.1. Stored separately is a table describing the drop–off bins in the warehouse. A drop–off bin is used by the robot to place parts of an order before it is complete. When the order is complete, the items the bin contains are removed from it, and the bin is then ready for another order. For each bin, its location is stored, as is the order identifier of the order in the bin when it is in use.

3.3.2 Database Access Layer

The database access layer uses our Local Result Set API. It maintains two connections to the same database since, at one point in the program, it is necessary to manipulate the database while retaining the result set of an earlier operation. The input and output signals for the main connection to the database are called `orders_query_out` and `orders_results`, respectively, and for the additional connection `stock_results` and `stock_query_out`, respectively, since those are only used to update stock levels:

```
output orders_query_out : string;
input orders_results : MYSQL_RES_ptr;

output stock_query_out : string;
input stock_results : MYSQL_RES_ptr;
```

Since the database access layer's only function is to wait for an order that needs to be picked and then to instruct the robot how to pick it, the main module is constructed as a loop. Inside the loop is a *trap* statement which handles all the ways in which the database and the system can fail. Therefore, all failure modes are dealt with in one place, and the error handling process is simplified.

The operation of the database access layer is roughly as follows (cf. Fig. 7). First, an order is retrieved from the database. One of the lines of this order is then extracted and the item details are stored locally. The robot is then sent to retrieve all items, one by one, and to deliver them in an available drop off bin using a pre–generated route. After each item has been collected, it is necessary to update the stock level of the item's type. However, at this point the database result set still contains uncollected order lines which are required for later in the program's execution. Therefore, the second database connection is used to perform the update without overwriting the result set containing the order's details.

Since the operation of the database access layer is quite simple and most database operations are effectively the same, only the database operation that

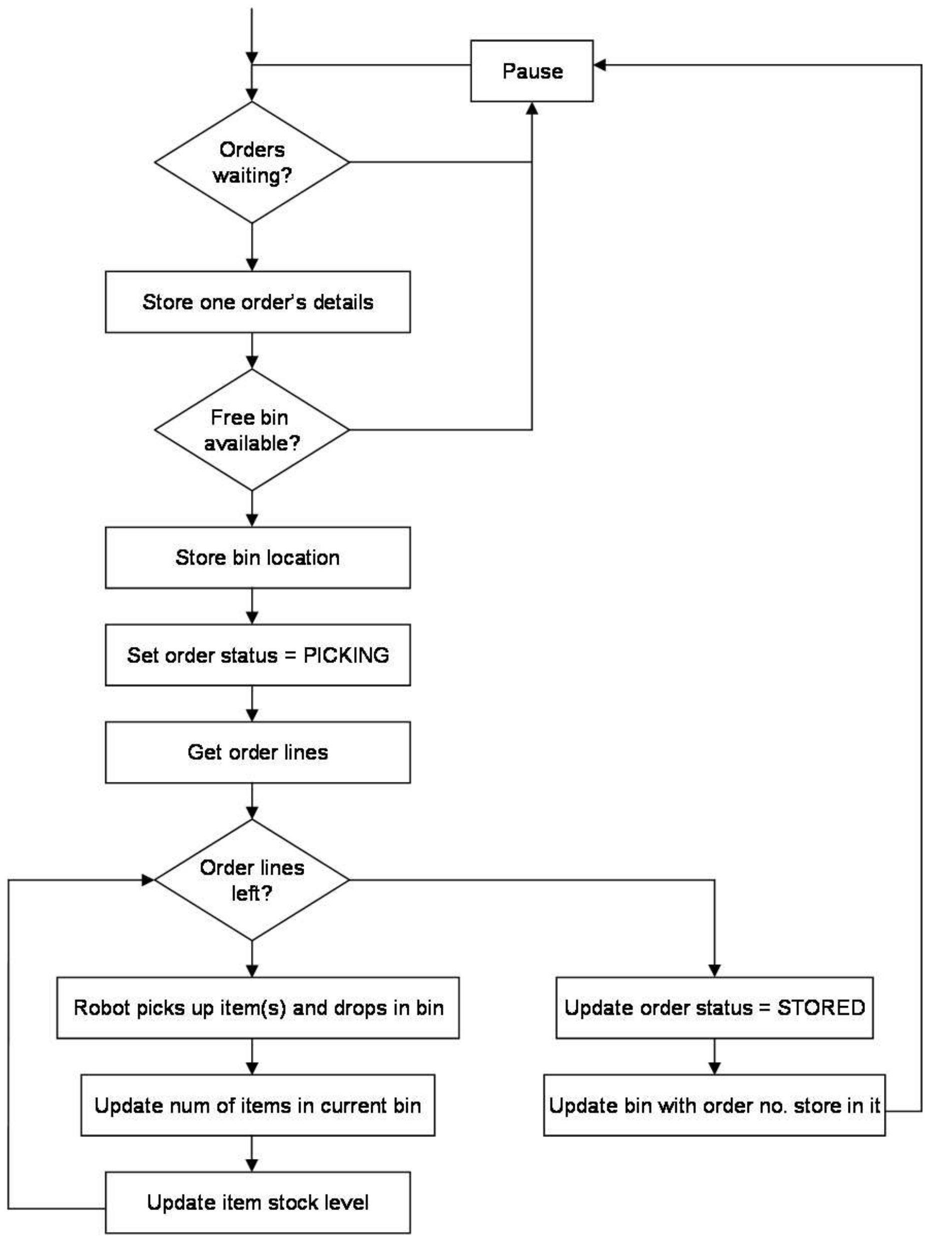
Pause
Orders waiting?
Store one order's details
Free bin available?
Store bin location
Set order status = PICKING
Get order lines
Order lines left?
Robot picks up item(s) and drops in bin
Update num of items in current bin
Update item stock level
Update order status = STORED
Update bin with order no. store in it

Figure 7: Flowchart describing the database access layer's behaviour.

retrieves waiting orders will be given here. The complete Esterel program can be found in App. B. The operation starts by emitting the query on output signal `orders_query_out` and then awaits the results:

```
emit orders_query_out("select order_id, customer.customer_id,
    name, address from customer, orders where
    customer.customer_id = orders.customer_id and
    orders.status = 'AWAITING_PICKING' order by order_id");
await orders_results;
```

The returned results are then checked for validity using our API function `check_result`. If the query has succeeded, then `num_rows` is called on the result set to see if any rows were returned. If rows have been returned, then there are orders waiting and, consequently, the first row is loaded into the local variable `row`. The details of the row are then retrieved using the `get<type>` functions and emitted on the corresponding local signals: `order_id`, `customer_id`, `customer_name` and `customer_address`. Now the result set is no longer needed, and the `clear_results` procedure is called. The following Esterel code captures this algorithm:

```
if (check_result(?orders_results)) then
  if (num_rows(?orders_results) > 0) then
    var row : MYSQL_ROW in
      row := get_next_row(?orders_results);
      emit order_id(getint(row,0));
      emit customer_id(getint(row,1));
      emit customer_name(getstr(row,2));
      emit customer_address(getstr(row,3));
    end var;
    call clear_results()(?orders_results);
  else ...
```

There are two ways in which this operation for retrieving orders can fail: firstly, if the query fails and, secondly, if there are no waiting orders. Each is catered for by an *exit* statement which corresponds to the trap mentioned above:

```
if (check_result(?orders_results)) then
  if (num_rows(?orders_results) > 0) then
    %Code snipped
  else
    call clear_results()(?orders_results);
    exit no_waiting_orders;
  end if;
else
  exit bad_query;
end if;
```

In each case, the program flow jumps to the end of the loop and emits an appropriate error message before pausing and then repeating the main loop. Note that function `clear_results` must be called after it has been determined that there are no waiting orders, freeing the memory occupied by the result set.

The generation of the pick–up and drop–off routes is the only non–database related function of the database access layer. To generate the route, first the item's height is computed and passed to a function called `generate_route_to_pickup_item()` along with the item's x and y coordinates. The function generates a string consisting of *op codes* that represent the operations the robot must perform to pick up that item and to return to the communication point in the warehouse. The string is sent to the robot via a signal, and then another signal indicating the completion of executing the route is awaited. After the pick–up route is complete, a drop–off route is computed by calling function `generate_route_to_drop_item()` and passing it the order's drop–off bin location. The route is emitted and the completion awaited on the same signals used by the pick–up operation. Again, details can be found in App. B.

By using external functions to generate the route, the warehouse can be redesigned in any way consistent with the item locations stored in the database, and only the two route generating functions will have to be rewritten.

3.3.3 Route Interpretation Layer

To interpret a route string of op codes, a number of external C functions are provided that extract parts of the string. First, function `get_num_ops()` is called to determine how many separate operations the route string contains. A loop is then repeated this number of times. On each iteration, a function `get_op()` is invoked to extract the type of operation and `get_param()` to extract the parameters to the operation. Once the operation type has been determined, an emission is made on the appropriate signal with the value obtained from `get_param()`. When the robot has completed the operation, either the `movement_op_complete` or `forklift_op_complete` signal is emitted, informing the route interpretation layer that the robot is ready for the next operation. When all operations have been performed in this manner, the signal `route_complete` is emitted which lets the database access layer know that the robot has finished its route.

3.3.4 Hardware Layer

The Movement RCX is required to run both the route interpretation layer and part of the hardware layer. To accomplish this, both layers are run in parallel and local signals are used to communicate between them. To communicate with the hardware layer running on the Forklift RCX, the output signals `pickup_item` and `drop_item` and the input signals `forklift_op_complete` are used (cf. Fig. 6). These combined with the signals of the hardware layer running on the Movement RCX, i.e., signals `move_forward`, `turn_left`, `turn_right` and `movement_op_complete`, give the complete range of commands provided by the hardware layer. A full listing of all the layers' implementations in Esterel is contained in App. B.

4 Conclusions

This report presented two APIs for interfacing the synchronous programming language Esterel to the relational database MySQL. The *Local Result Set API* assumes the result set to a database query to be stored locally to a reactive Esterel kernel and largely relies on the external function concept of Esterel. The *Remote Result Set API* considers the result set to be stored remotely and is realised via signals.

Both database APIs worked well in testing. In particular, the Local Result Set API is heavily used in our case study and, although none of the database operations are particularly complex, they are representative of the kind of database operations performed by reactive systems. In contrast to the elegance exhibited by the Local Result Set API, the Remote Result Set API appears to be slightly convoluted. This is due to modelling all database operations as signals, which became necessary since remoteness implies that one cannot expect instantaneous responses and, hence, cannot use external functions. It remains to be explored whether an implementation based on Esterel's task concept would be more elegant.

It should be emphasised that the introduction of a database using either API in an Esterel reactive system does not undermine Esterel's synchrony hypothesis. However, since the response times for returning query results or for accessing remote result sets cannot be guaranteed, the system can end up waiting for a signal that may never arrive. If the database is one that can provide guaranteed response times, such as a real–time database [14], the problem is elevated. Otherwise, the problem must be solved via timeouts in system design. In our case study, all database operations are performed at non–time–critical points, whence any unexpected delay from the database simply results in the system pausing, not malfunctioning.

Future Work

Future work is proposed to proceed along three directions. Firstly, some restrictions on the usage of our Local Result Set API may be removed, thereby making the API safer for use. In particular, our implementation implies that row variables become undefined once the result set is cleared. In a similar vein, checks could be implemented in our APIs to detect whether they are used in any way other than that intended.

Secondly, although MySQL is a popular database, our APIs would be applicable to others if they supported the *Open Database Connectivity* (ODBC) API [19]. This would allow any database to be used with no increase in complexity to our present approach.

Thirdly, database APIs for Lustre/SCADE are envisioned along similar lines than ours for Esterel/Esterel Studio.

Acknowledgements

We would like to thank the participants of the SYNCHRON '04 workshop for their constructive comments, especially for pointing out the appropriateness of Esterel's task concept for the Remote Result Set API.

References

[1] D. Baum, M. Gasperi, R. Hempel, and L. Villa. *Extreme Mindstorms – An advanced guide to Lego Mindstorms.* Apress, 2000.

[2] G. Berry. The constructive semantics of pure Esterel. CMA, Ecole des Mines, INRIA, 1999. Draft version 3.0.

[3] G. Berry. The Esterel v5 language primer. CMA, Ecole des Mines, INRIA, 2000.

[4] G. Berry. The foundations of Esterel. In *Essays in Honour of Robin Milner.* MIT Press, 2000.

[5] R.H. Bishop. *Modern Control Systems: Analysis and Design Using MATLAB and SIMULINK.* Addison Wesley, 1997.

[6] BrickOS. *brickos.sourceforge.net*, last visited in 2004.

[7] D.R. Butenhof. *Programming with POSIX Threads.* Addison Wesley, 1997.

[8] P. Caspi, D. Pilaud, N. Halbwachs, and J.A. Plaice. LUSTRE : A declarative language for programming synchronous systems. In *14th ACM Symposium on Principles of Programming Languages.* ACM, 1987.

[9] Esterel compiler. *www-sop.inria.fr/meije/esterel/esterel-eng.html*, last visited in 2004.

[10] Esterel Technologies. Esterel Studio and SCADE Suite & Drive design tools. *www.esterel-technologies.com*, last visited in 2004.

[11] X. Fornari. Lego C code generator. *www.emn.fr/x-info/lego/scLego.html*, last visited in 2004.

[12] N. Halbwachs. *Synchronous Programming of Reactive Systems.* Kluwer Academic Publishers, 1993.

[13] D. Harel and M. Politi. *Modeling Reactive Systems with Statecharts: The STATEMATE Approach.* McGraw Hill, 1998.

[14] K.-Y. Lam and T.-W. Kuo. *Real-time Database Systems: Architecture and Techniques.* Kluwer Academic Publishers, 2000.

[15] Lego Mindstorms robotics kit. *mindstorms.lego.com*, last visited in 2004.

[16] The MySQL open source database. *www.mysql.com*, last visited in 2004.

[17] The MySQL C API. *dev.mysql.com/doc/mysql/en/C.html*, last visited in 2004.

[18] Reactive languages and Lego Mindstorms. *www.emn.fr/x-info/lego*, last visited in 2004.

[19] R. Signore, J. Creamer, and M.O. Stegman. *The ODBC Solution: Open Database Connectivity in Distributed Environments.* McGraw Hill, 1995.

[20] L. Wall, T. Christiansen, and J. Orwant. *Programming Perl.* O'Reilly, 2000.

A Scripts supporting the APIs

There is one script for each of the two APIs. The local result set API script is used as follows:

```
gendb.pl <Main Module Name>
         <Max length of strings and queries>
         <DB Name>
         <Host>
         <User>
         <Password>
         <Signal name for emitting query>
         <Signal name for returning results>
```

The parameters carry the following meaning:

- *Main Module Name:*

  The name of the module compiled with Esterel; it is assumed that the auto–generated C code is in `<Main Module Name>.c`.

- *Max length of strings and queries:*

  Since all strings must have a fixed length representation in Esterel, this specifies what the maximum length is. It must be ensured that the program never exceeds it. This includes getting strings from the database using `getstr()`. The type of data that `getstr()` is called on should be guaranteed not to exceed the maximum length. For example, if the maximum length was 200, the database field could be of type `VARCHAR{150}`, the string could be appended by up to 50 characters.

- *DB Name:* The name of the database on the host.
- *Host:* The location of the MySQL DB; if hosted locally, then `localhost` is to be used.
- *User:* The username to be used to connect to the database.
- *Pass:* The associated password.
- *Signal name for emitting query:* The name of the signal that is used to send a query to this database.
- *Signal name for returning results:* The name of the signal that will be awaited for results.

The last six parameters may be repeated any number of times for additional databases, whence a single database can be accessed via multiple connections.

A note on the implementation of both APIs. In Sec. 2 it is mentioned that databases are running asynchronously to reactive kernels, since database transactions are relatively complex and thus cannot be assumed to respect the synchrony hypothesis. The current implementations of our local result set API does not implement this intention explicitly. This is however not a problem in the API since, immediately following a query emission, the result input signal should be awaited. Because there can be no statement between these two operations, it does not matter that the reactive cycle has temporally paused.

However, if an implementation of our APIs should explicitly support the asynchronous view between synchronous reactive kernels and databases, then it would not be difficult to do so by introducing threading using POSIX [7]. The query output function would need to be changed to inform a separate thread that would perform the database operation at hand. At the start of each iteration of the reactive system, the thread would then be queried to determine if any database operations were finished. If so, the location of the result set would be passed to the reactive kernel thread and the database result input signal would be emitted in that cycle.

A.1 Local Result Set API

```
#!/usr/bin/perl -w

$main_module = shift(@ARGV);
$max_strlen = shift(@ARGV);
$num_dbs = scalar(@ARGV)/7;

# Load database parameters into array of arrays
for (my $x=0; $x<$num_dbs; $x++) {
  for (my $y=1; $y<7; $y++) {
    #start at 1 so dont read in db_id, this will be internal
    $dbs[$x][$y] = shift(@ARGV);
  }
}

# Assign each db a unique id
for (my $x=0; $x<$num_dbs; $x++) {
  $dbs[$x][0] = $x;
}

#Print out the user entered info
print "Main module: ${main_module}\n";
for (my $x=0; $x<$num_dbs; $x++) {
  print "DB: ${dbs[$x][0]}\n";
  print "\tDB Name:\t${dbs[$x][1]}\n";
  print "\tHost:\t\t${dbs[$x][2]}\n";
  print "\tUser:\t\t${dbs[$x][3]}\n";
  print "\tPass:\t\t${dbs[$x][4]}\n";
  print "\tQuery signal:\t${dbs[$x][5]}\n";
  print "\tResults signal:\t${dbs[$x][6]}\n";
}

if (!open (OUT, ">${main_module}_out.c")) {
    print "Cannot open outfile.c for writing \n";
    exit 1;
}


if (!open (EST_FILE, "${main_module}.c")) {
    print "Cannot open ${main_module}.c for reading \n";
    exit 1;
}

while (($line = <EST_FILE>) ne "#include \"${main_module}.h\"\n") {
  if ($line =~ /#define STRLEN/) {
    print OUT ("#define STRLEN ${max_strlen}\n");
  }
  else {
    print OUT $line;
  }
}
while (($line = <EST_FILE>) ne "/* INITIALIZED CONSTANTS */\n") {
}
print OUT $line;
while (($line = <EST_FILE>) ne "/* MEMORY ALLOCATION */\n") {
  print OUT $line;
}

### Print MISC
##############

print OUT <<"PRINT_MISC";

/*************************/
/** DB code starts here **/
/*************************/

#include <stdio.h>
#include <stdlib.h>
#include "mysql.h"
typedef MYSQL_RES* MYSQL_RES_ptr;

MYSQL_ROW NULL_ROW = NULL;

int done;

PRINT_MISC

### Print global variables for each db
######################################

for (my $x=0; $x<$num_dbs; $x++) {
  print OUT ("\n//Global variables for ${dbs[$x][0]}\n");
  print OUT ("MYSQL init_${dbs[$x][0]}, *sock_${dbs[$x][0]};\n");
  print OUT ("MYSQL_RES *result_${dbs[$x][0]};\n");
  print OUT ("int query_pending_${dbs[$x][0]},
              query_suceeded_${dbs[$x][0]};\n");
}

### Generate DB Functions
#########################
```

```
print OUT <<"PRINT_CHECK_RESULT1";

int check_result(MYSQL_RES_ptr res) {
  if (res == result_${dbs[0][0]}) {
    return query_suceeded_${dbs[0][0]};
  }
PRINT_CHECK_RESULT1
for (my $x=1; $x<$num_dbs; $x++) { #carefull: loop starts at 1
  print OUT <<"PRINT_CHECK_RESULT2";
  else if (res == result_${dbs[$x][0]}) {
    return query_suceeded_${dbs[$x][0]};
  }
PRINT_CHECK_RESULT2
}
print OUT ("}\n");

print OUT <<"PRINT_AFFECTED_ROWS1";

int num_affected_rows(MYSQL_RES_ptr res) {
  if (res == result_${dbs[0][0]}) {
    return mysql_affected_rows(sock_${dbs[0][0]});
  }
PRINT_AFFECTED_ROWS1
for (my $x=1; $x<$num_dbs; $x++) { #careful: loop starts at 1
  print OUT <<"PRINT_AFFECTED_ROWS2";
  else if (res == result_${dbs[$x][0]}) {
    return mysql_affected_rows(sock_${dbs[$x][0]});
  }
PRINT_AFFECTED_ROWS2
}
print OUT ("}\n");


print OUT <<"PRINT_DB_FUN";

MYSQL_ROW get_next_row(MYSQL_RES_ptr res) {
  return mysql_fetch_row(res);
}

int num_rows(MYSQL_RES_ptr res) {
  return mysql_num_rows(res);
}

void clear_results(MYSQL_RES_ptr res) {
  mysql_free_result(res);
}

int getint(MYSQL_ROW row, integer pos) {
  return atoi(row[pos]);
}

char* getstr(MYSQL_ROW row, int pos, char* str) {
  strcpy(str, row[pos]);
  return str;
}

boolean getbol(MYSQL_ROW row, int pos) {
  return atoi(row[pos]);
}

float getflt(MYSQL_ROW row, int pos) {
  return atof(row[pos]);
}

float getdou(MYSQL_ROW row, int pos) {
  return atof(row[pos]);
}

PRINT_DB_FUN

### Print string manipulation functions
#######################################

print OUT <<"PRINT_STR_APPEND_FUN";
//SQL creation functions

void appstr(string query ,string toappend)
{
  strcat(query, toappend);
}

void appint(string query ,int toappend)
{
  char buf[12];
  sprintf(buf, "%d", toappend);
  strcat(query, buf);
}

void appbol(string query ,boolean toappend)
{
  if (toappend) {
    strcat(query, "1");
  }
  else {
    strcat(query, "0");
  }
}

void appdou(string query ,double toappend)
{
  char buf[30];
  sprintf(buf, "%E", toappend);
  strcat(query, buf);
}
PRINT_STR_APPEND_FUN

### Print user-def types' copy functions

print OUT <<"PRINT_COPY_FUN";

//Assignment copy functions

void _MYSQL_RES_ptr(MYSQL_RES_ptr* new, MYSQL_RES_ptr old) {
  *new = old;
```

```
}

void _MYSQL_ROW(MYSQL_ROW* new, MYSQL_ROW old) {
  *new = old;
}

PRINT_COPY_FUN

### Print output functions
##########################

for (my $x=0; $x<$num_dbs; $x++) {
  print OUT <<"PRINT_SQL_OUTPUT_PROC";

void ${main_module}_O_${dbs[$x][5]}(string query_string)
{
  query_pending_${dbs[$x][0]} = 1;
  query_suceeded_${dbs[$x][0]} = 0;  //query_suceeded status now remains
                                     // until next sql is issued

  if (mysql_query(sock_${dbs[$x][0]},query_string))
  {
    query_suceeded_${dbs[$x][0]} = 0;
    fprintf(stderr,"Query failed (%s)\\n",
      mysql_error(sock_${dbs[$x][0]})); // error
  }
  else // query succeeded, process any data returned by it
  {
    printf("SQL Sent: %s\\n", query_string);
    result_${dbs[$x][0]} = mysql_store_result(sock_${dbs[$x][0]});

    if (result_${dbs[$x][0]})  // there are rows
    {
      query_suceeded_${dbs[$x][0]} = 1;
      printf("\\tRows exist - %d rows, %d fields\\n",
              mysql_num_rows(result_${dbs[$x][0]}),
              mysql_num_fields(result_${dbs[$x][0]}));
    }
    else  // mysql_store_result() returned nothing;
          // should it have?
    {
      if(mysql_field_count(sock_${dbs[$x][0]}) == 0)
      {
        query_suceeded_${dbs[$x][0]} = 1;
        printf("\\tNo rows returned\\n");
      }
      else // mysql_store_result() should have returned data
      {
        query_suceeded_${dbs[$x][0]} = 0;
        fprintf(stderr, "\\t
                Error: %s\\n", mysql_error(sock_${dbs[$x][0]}));
      }
    }
  }
}
PRINT_SQL_OUTPUT_PROC
}
```

```
print OUT <<"PRINT_MORE_OUTPUT_PROCS";
void ${main_module}_O_done()
{
  done = 1;
}

void ${main_module}_O_intout(int value)
{
  printf("intout: %d\\n", value);
}

void ${main_module}_O_bolout(int value)
{
  printf("boolout: %d\\n", value);
}

void ${main_module}_O_strout(string value)
{
  printf("strout: %s\\n", value);
}

void ${main_module}_O_fltout(float value)
{
  printf("floatout: %E\\n", value);
}

void ${main_module}_O_douout(double value)
{
  printf("doubleout: %E\\n", value);
}


PRINT_MORE_OUTPUT_PROCS

### Copy Esterel autogen C code
###############################

print OUT <<"COMMENTS1";
/************************/
/** DB code stops here **/
/************************/

COMMENTS1

print OUT $line;
while ($line = <EST_FILE>) {
    print OUT $line;
}

print OUT <<"COMMENTS2";

/*************************/
/** DB code starts here **/
/*************************/

COMMENTS2
```

```
### Print main cyclic loop - exits using done output
####################################################

print OUT <<"PRINT_MAIN1";
int main(int argc, char **argv)
{
  done = 0;
PRINT_MAIN1

for (my $x=0; $x<$num_dbs; $x++) {
  print OUT <<"PRINT_DB_CONNECTS";
  mysql_init(&init_${dbs[$x][0]});
  if (!(sock_${dbs[$x][0]} = mysql_real_connect(&init_${dbs[$x][0]},
      "${dbs[$x][2]}", "${dbs[$x][3]}","${dbs[$x][4]}","${dbs[$x][1]}",
      0,NULL,0)))
  {
    fprintf(stderr,"Couldn’t connect to DB, ID:
        ${dbs[$x][0]}\\n%s\\n\\n",mysql_error(&init_${dbs[$x][0]}));
    perror("");
    exit(1);
  }
PRINT_DB_CONNECTS
}

print OUT <<"PRINT_MAIN2";
  ${main_module}_reset();

  while (!done)
  {
PRINT_MAIN2

for (my $x=0; $x<$num_dbs; $x++) {
  print OUT <<"PRINT_INPUT_CALLS";
    if (query_pending_${dbs[$x][0]}) {
      ${main_module}_I_${dbs[$x][6]}(result_${dbs[$x][0]});
      query_pending_${dbs[$x][0]} = 0;
    }
PRINT_INPUT_CALLS
}

print OUT "\t\t${main_module}();\n\t}\n\n";

for (my $x=0; $x<$num_dbs; $x++) {
  print OUT  "\tmysql_close(sock_${dbs[$x][0]});\n";
}

print  OUT "\texit(0);\n}\n";

print OUT <<"COMMENTS3";
/***************************/
/** DB code finishes here **/
/***************************/
COMMENTS3

### Script done
exit(0);
```

A.2 Remote Result Set API

The script given below supports the Remote Result Set API. It is implemented in terms of the Lego Mindstorms architecture. For example, RCX1 runs the reactive kernel that accesses/updates the database. The program generated for the PC does not run Esterel code, it simply listens for database requests from RCX1, the Movement RCX, and performs them on the database. It then replies to RCX1 with one of the six acknowledge messages. The code for RCX2, the Forklift RCX, is only included for completeness.

The script is called as follows:

```
gendbremote.pl <Main Module Name>
               <Max length of strings and queries>
               <DB Signal prefix name>
               <Host>
               <User>
               <Password>
```

The last four parameters may be repeated for specifying additional database connections.

```
#!/usr/bin/perl -w

################################################################
## Store arguments
################################################################

$main_module = shift(@ARGV);
$max_strlen = shift(@ARGV);
$num_dbs = scalar(@ARGV)/4;

# Load database parameters into array of arrays
for (my $x=0; $x<$num_dbs; $x++) {
  $dbs[$x][5] = shift(@ARGV);
  for (my $y=1; $y<5; $y++) { #start at 1 so dont read in db_id,
                              # this will be internal
    if (!defined($dbs[$x][$y] = shift(@ARGV))) {
      die "Database specified with out enough arguments\n";
    }
  }
}

# Assign each db a unique id
for (my $x=0; $x<$num_dbs; $x++) {
  $dbs[$x][0] = $x;
}

#Print out the user entered info
print "Main module: ${main_module}\n";
for (my $x=0; $x<$num_dbs; $x++) {
  print "DB: ${dbs[$x][0]}, ${dbs[$x][5]}\n";
  print "\tDB Name:\t${dbs[$x][1]}\n";
  print "\tHost:\t\t${dbs[$x][2]}\n";
  print "\tUser:\t\t${dbs[$x][3]}\n";
  print "\tPass:\t\t${dbs[$x][4]}\n";
}


print "Invoked Esterel on ${main_module}.strl\n";
if (system "esterel ${main_module}.strl") {
  die "Esterel reports an error\n";
}

###############################################################
## Build code for RCX (Running the database processing kernel)
###############################################################

print "Code build for RCX started\n";

if (!open (RCX, ">${main_module}_rcx.c")) {
    print "Cannot open ${main_module}_rcx.c for writing \n";
    exit 1;
}


if (!open (EST_FILE, "${main_module}.c")) {
    print "Cannot open ${main_module}.c for reading \n";
    exit 1;
}

while (($line = <EST_FILE>) ne "#include \"${main_module}.h\"\n") {
  if ($line =~ /#define STRLEN/) {
    print RCX ("#define STRLEN ${max_strlen}\n");
  }
  else {
    print RCX $line;
  }
}
while (($line = <EST_FILE>) ne "/* INITIALIZED CONSTANTS */\n") {
}
print RCX $line;
while (($line = <EST_FILE>) ne "/* MEMORY ALLOCATION */\n") {
  print RCX $line;
}

### Print MISC
##############

print RCX <<"PRINT_MISC";

#include <stdlib.h>
#include <time.h>
#include <unistd.h>
#include <lnp.h>
#include <conio.h>
#include <string.h>
#include <lnp-logical.h>
```

```
#include <dkey.h>

#define MY_PORT 2

#define COMP_HOST 0x8
#define COMP_PORT 0x7
#define COMP_ADDR ( COMP_HOST << 4 | COMP_PORT )

#define RCX2_HOST 0x2
#define RCX2_PORT 0x3
#define RCX2_ADDR ( RCX2_HOST << 4 | RCX2_PORT )

#define TIMEOUT_MS 500
#define NO_ACK 0
#define NO_PAYLOAD 1
#define INT_PAYLOAD 2
#define BOOL_PAYLOAD 3
#define STRING_PAYLOAD 4

#define SLOW_DOWN 1

int done;

time_t send_time;

int send_msg_to_comp;
int send_msg_to_rcx2;
int ack;
int message_sending_thread_seen_ack;
char outgoing_buffer[255];
int outgoing_length;
int int_payload;
int bool_payload;
char string_payload[254];
/*
char * strcat(char *str1, char *str2) {
  char *ptr1;
  char *ptr2;
  ptr1 = str1;
  while (!(*ptr1 == '\\0')) {
    ptr1++;
  }
  ptr2 = str2;
  while (!(*ptr2 == '\\0')) {
    *ptr1 = *ptr2;
    ptr1++;
    ptr2++;
  }
  *ptr1 = '\\0';
  return str1;
}
*/
char *strcat(char *s1, char *s2) {
  char *s = s1;

  while (*s1) {
    s1++;
  }
  while ((*s1++ = *s2++)) {
  }
  return s;
}

char *itoa(char *dst, int value) {
  char *digit_ptr = dst;
  int div = 10000;
  while (div>value) {
    div = div/10;
  }
  while (div>0) {
    *digit_ptr = (value / div)+'0';
    value = value % div;
    div = div/10;
    digit_ptr++;
  }
  *digit_ptr = '\0';
  return dst;
}


void addr_handler(const unsigned char* data,unsigned char length,
unsigned char src)
{
  switch(data[0]) {
    case 'a' :
      ack = NO_PAYLOAD;
      break;
    case 'i' :
      ack = INT_PAYLOAD;
      int_payload = data[2]*256+data[1];
      break;
    case 'z' :
      ack = INT_PAYLOAD;
      int_payload = data[1]*256+data[2];
      break;
    case 'b' :
      ack = BOOL_PAYLOAD;
      if (data[1] == 't') {
        bool_payload = 1;
      }
      else {
        bool_payload = 0;
      }
      break;
    case 's' :
      ack = STRING_PAYLOAD;
      strcpy(string_payload, &data[1]);
      break;
  }
}

wakeup_t msg_to_send(wakeup_t ignore) {
  return send_msg_to_comp || send_msg_to_rcx2;
}
```

```
wakeup_t timeout_or_ack(wakeup_t ignore) {
  return ((get_system_up_time() - send_time) > TIMEOUT_MS) || ack;
}

int message_sending_thread( int argc, char **argv )
{
  int dest;

  while (1) {
    wait_event(msg_to_send,0);
    if (send_msg_to_comp) {
      send_msg_to_comp = 0;
      dest = COMP_ADDR;
    }
    else {
      send_msg_to_rcx2 = 0;
      dest = RCX2_ADDR;
    }
    while (1) {
      send_time = get_system_up_time();
      lnp_addressing_write( outgoing_buffer, outgoing_length,
                            dest, MY_PORT);
      wait_event(timeout_or_ack, 0);
      if (ack) {
        message_sending_thread_seen_ack = 1;
        break;
      }
      cputs("Tout");
    }
  }
}

PRINT_MISC

### Generate DB Functions
########################

for (my $x=0; $x<$num_dbs; $x++) {
  print RCX <<"DB_OUTPUT_FUNS";

void ${main_module}_O_${dbs[$x][5]}_send_query(char * string) {
  outgoing_buffer[0] = 'q';
  outgoing_buffer[1] = ${dbs[$x][0]};
  strcpy(&outgoing_buffer[2], string);
  outgoing_length = strlen(string)+3;
    //+1 for null, 1 for 'q', 1 for db id
  send_msg_to_comp = 1;
}

void ${main_module}_O_${dbs[$x][5]}_fetch_next_row(void) {
  outgoing_buffer[0] = 'f';
  outgoing_buffer[1] = ${dbs[$x][0]};
  outgoing_length = 2;
  send_msg_to_comp = 1;
}
void ${main_module}_O_${dbs[$x][5]}_num_rows(void) {
  outgoing_buffer[0] = 'n';
  outgoing_buffer[1] = ${dbs[$x][0]};
  outgoing_length = 2;
  send_msg_to_comp = 1;
}

void ${main_module}_O_${dbs[$x][5]}_affected_rows(void) {
  outgoing_buffer[0] = 'a';
  outgoing_buffer[1] = ${dbs[$x][0]};
  outgoing_length = 2;
  send_msg_to_comp = 1;
}

void ${main_module}_O_${dbs[$x][5]}_clear_results(void) {
  outgoing_buffer[0] = 'c';
  outgoing_buffer[1] = ${dbs[$x][0]};
  outgoing_length = 2;
  send_msg_to_comp = 1;
}

void ${main_module}_O_${dbs[$x][5]}_getint(int pos) {
  outgoing_buffer[0] = 'g';
  outgoing_buffer[1] = 'i';
  outgoing_buffer[2] = ${dbs[$x][0]};
  memcpy(&outgoing_buffer[3], &pos, 2);
  outgoing_length = 5;
  send_msg_to_comp = 1;
}

void ${main_module}_O_${dbs[$x][5]}_getbol(int pos) {
  outgoing_buffer[0] = 'g';
  outgoing_buffer[1] = 'b';
  outgoing_buffer[2] = ${dbs[$x][0]};
  memcpy(&outgoing_buffer[3], &pos, 2);
  outgoing_length = 5;
  send_msg_to_comp = 1;
}

void ${main_module}_O_${dbs[$x][5]}_getstr(int pos) {
  outgoing_buffer[0] = 'g';
  outgoing_buffer[1] = 's';
  outgoing_buffer[2] = ${dbs[$x][0]};
  memcpy(&outgoing_buffer[3], &pos, 2);
  outgoing_length = 5;
  send_msg_to_comp = 1;
}


DB_OUTPUT_FUNS
}

### Print string manipulation functions
#######################################

print RCX <<"PRINT_STR_APPEND_FUN";
//SQL creation functions
```

```
void appstr(string query ,string toappend)
{
  strcat(query, toappend);
}

void appbol(string query ,boolean toappend)
{
  if (toappend) {
    strcat(query, "1");
  }
  else {
    strcat(query, "0");
  }
}

void appint(string query ,int toappend)
{
  char buf[12];
  itoa(buf, toappend);
  strcat(query, buf);
}
/*
void appdou(string query ,double toappend)
{
  char buf[30];
  sprintf(buf, "%E", toappend);
  strcat(query, buf);
}
*/
PRINT_STR_APPEND_FUN


print RCX <<"PRINT_MORE_OUTPUT_PROCS";
void ${main_module}_O_done()
{
  done = 1;
}

void ${main_module}_O_LCD_INT(int value)
{
  lcd_int(value);
}

void ${main_module}_O_LCD_STR(string value)
{
  cputs(value);
}

void ${main_module}_O_LCD_BOL(int value)
{
  if (value) {
    cputs("true");
  }
  else {
    cputs("false");
  }
```

```
}

void ${main_module}_O_msg_rcx2(string msg_string) {
  strcpy(outgoing_buffer, msg_string);
  outgoing_length = strlen(msg_string)+1;
  send_msg_to_rcx2 = 1;
}

PRINT_MORE_OUTPUT_PROCS

### Copy Esterel autogen C code
###############################

print RCX <<"COMMENTS1";
/************************/
/** DB code stops here **/
/************************/

COMMENTS1

print RCX $line;
while ($line = <EST_FILE>) {
    print RCX $line;
}

print RCX <<"COMMENTS2";

/*************************/
/** DB code starts here **/
/*************************/

COMMENTS2

### Print main cyclic loop - exits using done output
####################################################

print RCX <<"PRINT_MAIN1";
int main(int argc, char **argv)
{
  tid_t msg_handler;

  send_msg_to_comp = 0;
  send_msg_to_rcx2 = 0;
  ack = NO_ACK;
  message_sending_thread_seen_ack = 0;

  lnp_logical_range ( 0 );

  lnp_addressing_set_handler (MY_PORT, addr_handler );

  msg_handler = execi(&message_sending_thread,0,0,
                      PRIO_NORMAL,DEFAULT_STACK_SIZE);

  ${main_module}_reset();

  while (!done) {
    if (ack && message_sending_thread_seen_ack) {
      switch (ack) {
        case NO_ACK :
          //No incomming msg this cycle - do nothing
          break;
        case NO_PAYLOAD :
          //Ack received, no payload
          ${main_module}_I_ack();
          ack = 0;
          message_sending_thread_seen_ack = 0;
          break;
        case INT_PAYLOAD :
          //Ack received, integer payload
          ${main_module}_I_ackint(int_payload);
          ack = 0;
          message_sending_thread_seen_ack = 0;
          break;
        case BOOL_PAYLOAD :
          ${main_module}_I_ackbol(bool_payload);
          ack = 0;
          message_sending_thread_seen_ack = 0;
          break;
        case STRING_PAYLOAD :
          ${main_module}_I_ackstr(string_payload);
          ack = 0;
          message_sending_thread_seen_ack = 0;
          break;
      }
    }
    ${main_module}();
    msleep(10);
  }

  exit(0);
}

PRINT_MAIN1

close(RCX);
close(EST_FILE);

########################################################
## Build code for PC deamon (No reactive kernel, just
## listens for RCX DB requests and processes them.
## Then sends one of the 6 acks back.
########################################################

print "Code build for PC started\n";

if (!open (PC, ">${main_module}_pc.c")) {
    print "Cannot open ${main_module}_pc.c for writing \n";
    exit 1;
}

print PC <<"PRINT_MISC1";
#include "liblnp.h"
#include <stdio.h>
#include <signal.h>
```

```
#include <stdlib.h>
#include <unistd.h>
#include <sys/time.h>
#include <sys/types.h>
#include <string.h>

#define MY_PORT 7
#define RCX1_HOST 0x0
#define RCX1_PORT 0x2
#define RCX1_ADDR ( RCX1_HOST << 4 | RCX1_PORT )

#define debug_msg

#include <stdio.h>
#include <stdlib.h>
#include "mysql.h"

PRINT_MISC1

for (my $x=0; $x<$num_dbs; $x++) {
  print PC <<"DB_VARS";
MYSQL init_${x}, *sock_${x};
MYSQL_RES *result_${x};
MYSQL_ROW row_${x};

DB_VARS
}

print PC <<"PRINT_MISC2";
unsigned char outgoing_msg[255];
int done;

int ack_pending;
int int_ack_pending;
int int_ack_payload;
int bool_ack_pending;
int bool_ack_payload;
int string_ack_pending;
char string_ack_payload[255];

void addr_handler(const unsigned char* data,unsigned char length,
unsigned char src)
{
int i;

#ifdef debug_msg
  printf("msg (%d): ", length);
  for (i=0; i<length; i++) {
    printf("%x ", data[i]);
  }
  printf("\\n");
#endif
  switch (data[0]) {
  //Output messages
    case 's' :
      printf("RCX1: %s", &data[1]);
      ack_pending = 1;
      break;
  //Database messages
    case 'q' :
      printf("RCX1: Requests query - %s (db id: 0)\\n", &data[2]);
      switch (data[1]) {

PRINT_MISC2
for (my $x=0; $x<$num_dbs; $x++) {
  print PC <<"DB_QUERY_SWITCH";
        case ${x} :
          bool_ack_payload = 0;
          if (mysql_query(sock_${x},&data[2]))
          {
            // error
            bool_ack_payload = 0;
            fprintf(stderr,"\\t
                    Query failed (%s)\\n",mysql_error(sock_${x}));
          }
          else // query succeeded, process any data returned by it
          {
            result_${x} = mysql_store_result(sock_${x});

            if (result_${x})  // there are rows
            {
              bool_ack_payload = 1;
              printf("\\tRows exist - %d rows, %d fields\\n",
                     mysql_num_rows(result_${x}),
                     mysql_num_fields(result_${x}));
            }
            else  // mysql_store_result() returned nothing;
                  // should it have?
            {
              if(mysql_field_count(sock_${x}) == 0)
              {
                bool_ack_payload = 1;
                printf("\\tNo rows returned\\n");
              }
              else // mysql_store_result() should have returned data
              {
                bool_ack_payload = 0;
                fprintf(stderr, "\\t
                        Error: %s\\n", mysql_error(sock_${x}));
              }
            }
          }
          bool_ack_pending = 1;
          break;
DB_QUERY_SWITCH
}
print PC <<"PRINT_MISC3";
      }
      break;
    case 'f' :
      switch (data[1]) {
PRINT_MISC3
for (my $x=0; $x<$num_dbs; $x++) {
  print PC <<"DB_FETCH_SWITCH";
```

```
        case ${x} :
          printf("RCX1: Requests fetch row (DB: ${dbs[$x][5]})\\n");
          row_${x} = mysql_fetch_row(result_${x});
          bool_ack_payload = (int)row_${x};
          bool_ack_pending = 1;
          break;
DB_FETCH_SWITCH
}
print PC <<"PRINT_MISC4";
      }
      break;
    case 'g' :
      switch (data[1]) {
        case 'i' :
          switch (data[2]) {
PRINT_MISC4
for ($x=0; $x<$num_dbs; $x++) {
  print PC <<"PRINT_GETINT_SWITCH";
            case ${x} :
              printf("RCX1: Requested integer from position %d
                     (DB: ${dbs[$x][5]})\\n", 256*data[3]+data[4]);
              int_ack_payload = atoi(row_${x}\[256*data[3]+data[4]]);
              int_ack_pending = 1;
              break;
PRINT_GETINT_SWITCH
}
print PC <<"PRINT_CASE_GB";
          }
          break;
        case 'b' :
          switch (data[2]) {
PRINT_CASE_GB
for (my $x=0; $x<$num_dbs; $x++) {
  print PC <<"PRINT_GETBOL_SWITCH";
            case ${x} :
              printf("RCX1: Requested boolean from position %d
                     (DB: ${dbs[$x][5]})\\n", 256*data[3]+data[4]);
              bool_ack_payload = atoi(row_${x}\[256*data[3]+data[4]]);
              bool_ack_pending = 1;
              break;
PRINT_GETBOL_SWITCH
}
print PC <<"PRINT_CASE_GS";
          }
          break;
        case 's' :
          switch (data[2]) {
PRINT_CASE_GS
for (my $x=0; $x<$num_dbs; $x++) {
  print PC <<"PRINT_GETSTR_SWITCH";
            case ${x} :
              printf("RCX1: Requested string from position %d
                     (DB: ${dbs[$x][5]})\\n", 256*data[3]+data[4]);
              strcpy(string_ack_payload, row_${x}\[256*data[3]+data[4]]);
              string_ack_pending = 1;
              break;
PRINT_GETSTR_SWITCH
}
print PC <<"PRINT_CASE_GC";
          }
          break;
      }
      break;
    case 'c' :
      switch (data[1]) {
PRINT_CASE_GC
for (my $x=0; $x<$num_dbs; $x++) {
  print PC <<"PRINT_RESCLEAR_SWITCH";
        case ${x} :
          printf("RCX1: Requests results clear for DB:
                  ${dbs[$x][5]}\\n");
          mysql_free_result(result_${x});
          ack_pending = 1;
          break;
PRINT_RESCLEAR_SWITCH
}
print PC <<"PRINT_CASE_N";
      }
      break;
    case 'n' :
      switch (data[1]) {
PRINT_CASE_N
for (my $x=0; $x<$num_dbs; $x++) {
  print PC <<"PRINT_NUMROW_SWITCH";
        case ${x} :
          printf("RCX1: Requests num of rows in result for db
                  ${dbs[$x][5]}\\n");
          int_ack_payload = mysql_num_rows(result_${x});
          int_ack_pending = 1;
          break;
PRINT_NUMROW_SWITCH
}
print PC <<"PRINT_CASE_A";
      }
      break;
    case 'a' :
      switch (data[1]) {
PRINT_CASE_A
for (my $x=0; $x<$num_dbs; $x++) {
  print PC <<"PRINT_AFFROW_SWITCH";
        case ${x} :
          printf("RCX1: Requests num of affected rows for db
                  ${dbs[$x][5]}\\n");
          int_ack_payload = mysql_affected_rows(sock_${x});
          int_ack_pending = 1;
          break;
PRINT_AFFROW_SWITCH
}
print PC <<"PRINT_MISC6";
      }
      break;
  }
}
```

```
void signal_handler( int signal ) {
  printf("\\nCaught signal - exiting...\\n");
  done = 1;
}

int main ( int argc, char **argv ) {

  //LNP Setup
  ///////////

  if ( lnp_init ( 0, 0, 0, 0, 0 ) ) {
    perror ( "lnp_init\\n" );
    exit(1);
  }
  else {
    printf ( "init OK\\n" );
  }

  lnp_addressing_set_handler (MY_PORT, addr_handler );

  //DB Setup
  //////////
PRINT_MISC6
for (my $x=0; $x<$num_dbs; $x++) {
  print PC <<"PRINT_DB_CONNECTS";
  mysql_init(&init_${dbs[$x][0]});
  if (!(sock_${dbs[$x][0]} =
      mysql_real_connect(&init_${dbs[$x][0]},"${dbs[$x][2]}",
        "${dbs[$x][3]}","${dbs[$x][4]}","${dbs[$x][1]}",0,NULL,0)))
  {
    fprintf(stderr,"Couldn't connect to DB, ID:
            ${dbs[$x][5]}\\n%s\\n\\n",
            mysql_error(&init_${dbs[$x][0]}));
    perror("");
    exit(1);
  }
  else {
    printf("Connected to DB ${dbs[$x][5]}\\n");
  }
PRINT_DB_CONNECTS
}
print PC <<"PRINT_MISC7";
  //Signals

  signal( SIGINT, signal_handler );
  done = 0;

  //Main loop

  while (!done) {
    if (ack_pending) {
      outgoing_msg[0] = 'a';
      lnp_addressing_write( outgoing_msg, 1, RCX1_ADDR, MY_PORT);
      printf("Comp: Sent ack\\n");
      ack_pending = 0;
    }
    else if (int_ack_pending) {
      outgoing_msg[0] = 'i';
      memcpy(outgoing_msg + 1, &int_ack_payload, 2);
      lnp_addressing_write( outgoing_msg, 3, RCX1_ADDR, MY_PORT);
      printf("Comp: Sent ack with int: %d\\n", int_ack_payload);
      int_ack_pending = 0;
    }
    else if (bool_ack_pending) {
      outgoing_msg[0] = 'b';
      if (bool_ack_payload) {
        outgoing_msg[1] = 't';
      }
      else {
        outgoing_msg[1] = 'f';
      }
      lnp_addressing_write( outgoing_msg, 2, RCX1_ADDR, MY_PORT);
      printf("Comp: Sent ack with bool: %c\\n", outgoing_msg[1]);
      bool_ack_pending = 0;
    }
    else if (string_ack_pending) {
      outgoing_msg[0] = 's';
      strcpy(&outgoing_msg[1], string_ack_payload);
      lnp_addressing_write(outgoing_msg, strlen(outgoing_msg)+2,
                           RCX1_ADDR, MY_PORT);
      printf("Comp: Sent ack with string: %s\\n", string_ack_payload);
      string_ack_pending = 0;
    }
  }

  //Cleanup
PRINT_MISC7
for (my $x=0; $x<$num_dbs; $x++) {
  print PC <<"PRINT_DB_CLOSE";
  mysql_close(sock_${x});
PRINT_DB_CLOSE
}
print PC <<"PRINT_MISC8";

  return 0;

}
PRINT_MISC8

###################################################################
## Build code for RCX2 ( Just a slave to RCX1, responds to requests
## for manipulating the forklift.
###################################################################

print "Code build for RCX2 started\n";

if (!open (PC, ">${main_module}_rcx2.c")) {
    print "Cannot open ${main_module}_rcx2.c for writing \n";
    exit 1;
}

print PC <<"END_RCX2_PROG";
#include <stdlib.h>
#include <time.h>
```

```
#include <unistd.h>
#include <lnp.h>
#include <conio.h>
#include <string.h>
#include <dsensor.h>
#include <lnp-logical.h>
#include <dkey.h>

#define MY_PORT 3

#define RCX1_HOST 0x0
#define RCX1_PORT 0x2
#define RCX1_ADDR ( RCX1_HOST << 4 | RCX1_PORT )

char outgoing_msg[255];
unsigned int reading;
int msg_arrived;
char short_int;

void addr_handler(const unsigned char* data,unsigned char length,
unsigned char src)
{
  switch(data[0]) {
    case 'p' :
      msg_arrived = 1;
      break;
  }
}

wakeup_t wait_msg_arrived(wakeup_t ignore) {
  return msg_arrived;
}


int main(int argc, char **argv)
{
  msg_arrived = 0;
  reading = 0;

  ds_active(&SENSOR_1);

  lnp_logical_range ( 0 );

  lnp_addressing_set_handler (MY_PORT, addr_handler );

  while (1) {
    wait_event(wait_msg_arrived,0);
    msg_arrived = 0;
    outgoing_msg[0] = 'z';
    reading = LIGHT_1;
    memcpy(outgoing_msg + 1, &reading, 2);
    lnp_addressing_write(outgoing_msg, 3, RCX1_ADDR, MY_PORT);
    msleep(100);
  }

  exit(0);
}
```

```
END_RCX2_PROG

###
### Script done
###

exit(0);
```

B Case Study - Continued Description

This appendix contains the MySQL database schema used for our case study, as well as the Esterel code for all reactive kernels, i.e., for the host computer PC, the Movement RCX and the Forklift RCX.

B.1 MySQL Database Schema

```
CREATE TABLE ‘customer‘ (
  ‘customer_id‘ int(11) NOT NULL auto_increment,
  ‘name‘ varchar(50) NOT NULL default '',
  ‘address‘ varchar(200) NOT NULL default '',
  PRIMARY KEY  (‘customer_id‘)
) TYPE=MyISAM AUTO_INCREMENT=3 ;

CREATE TABLE ‘orders‘ (
  ‘order_id‘ int(11) NOT NULL default '0',
  ‘customer_id‘ int(11) NOT NULL default '0',
  ‘status‘ enum('AWAITING_PICKING','PICKING','STORED','LEFT_WAREHOUSE')
    NOT NULL default 'AWAITING_PICKING',
  ‘dropoff_point‘ int(11) default NULL,
  PRIMARY KEY  (‘order_id‘,‘customer_id‘)
) TYPE=MyISAM;

CREATE TABLE ‘order_lines‘ (
  ‘order_id‘ int(11) NOT NULL default '0',
  ‘item_id‘ int(11) NOT NULL default '0',
  ‘quantity‘ int(11) NOT NULL default '0',
  PRIMARY KEY  (‘order_id‘,‘item_id‘)
) TYPE=MyISAM;

CREATE TABLE ‘item‘ (
  ‘item_id‘ int(11) NOT NULL default '0',
  ‘name‘ varchar(200) NOT NULL default '',
  ‘stock_level‘ int(11) NOT NULL default '0',
  ‘shelf_height‘ int(11) NOT NULL default '0',
  ‘pos_x‘ int(11) NOT NULL default '0',
  ‘pos_y‘ int(11) NOT NULL default '0',
  PRIMARY KEY  (‘item_id‘)
) TYPE=MyISAM;

CREATE TABLE ‘dropoff_points‘ (
  ‘bin_id‘ int(11) NOT NULL default ’0’,
  ‘order_id‘ int(11) default NULL,
  ‘pos_x‘ int(11) NOT NULL default ’0’,
  ‘pos_y‘ int(11) NOT NULL default ’0’,
  PRIMARY KEY  (‘bin_id‘)
) TYPE=MyISAM;
```

B.2 Code Listings for all Reactive Kernels

B.2.1 Host computer – PC

```
module pc:

%%% API declarations
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  type MYSQL_RES_ptr;
  type MYSQL_ROW;

  procedure appstr() (string, string);
  procedure appint() (string, integer);

  function check_result(MYSQL_RES_ptr) : boolean;
  function get_next_row(MYSQL_RES_ptr) : MYSQL_ROW;
  function num_affected_rows(MYSQL_RES_ptr) : integer;
  function num_rows(MYSQL_RES_ptr) : integer;
  procedure clear_results() (MYSQL_RES_ptr);

  function getint(MYSQL_ROW, integer) : integer;
  function getstr(MYSQL_ROW, integer, string) : string;

%%% Application specific declarations
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  input orders_results : MYSQL_RES_ptr;
  output orders_query_out : string;

  input stock_results : MYSQL_RES_ptr;
  output stock_query_out : string;


  output print : string;
  output intout : integer;

  constant SPEED = 500 : integer;

  function generate_route_to_pickup_item(integer, integer, integer) : string;
  function generate_route_to_drop_item(integer, integer) : string;
  output execute_route(string);
  input route_complete;

signal bin : integer,
       bin_x : integer,
       bin_y : integer,
       order_id : integer,
       customer_id : integer,
       customer_name : string,
       customer_address : string,
       item_id : integer,
       item_quantity : integer,
       item_name : string,
       item_stock_level : integer,
       shelf_height : integer,
       item_x : integer,
       item_y : integer,
       num_items_in_bin : integer,

       move_and_get_item,
       move_and_drop_item,
       robot_high_level_op_complete in
  [
    loop
      trap bad_query,
           no_free_bins,
           less_lines_than_num_rows_reports,
           no_waiting_orders,
           order_status_update_failed,
           stock_update_failed,
           bin_update_failed,
           item_quantity_exceeds_stock in

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %%% Get order details (or no more orders to process currently
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        emit orders_query_out("select order_id, customer.customer_id,
            name, address from customer, orders where
            customer.customer_id = orders.customer_id and
            orders.status = 'AWAITING_PICKING' order by order_id");
        await orders_results;

        if (check_result(?orders_results)) then
          if (num_rows(?orders_results) > 0) then
            var row : MYSQL_ROW,
                buf : string in
              row := get_next_row(?orders_results);
              emit order_id(getint(row,0));
              emit customer_id(getint(row,1));
              emit customer_name(getstr(row,2,buf));
              emit customer_address(getstr(row,3,buf));
            end var;

            var comp_str : string in
              comp_str := "Processing order ";
              call appint() (comp_str, ?order_id);
              call appstr() (comp_str, " for customer ");
              call appstr() (comp_str, ?customer_name);
              call appstr() (comp_str, " (");
              call appint() (comp_str, ?customer_id);
              call appstr() (comp_str, "), ");
              call appstr() (comp_str, ?customer_address);
              emit print(comp_str);
            end var;

          else
            call clear_results()(?orders_results);
            exit no_waiting_orders;
          end if;
        else
          exit bad_query;
        end if;

        call clear_results()(?orders_results);

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %%% Get location of a free bin
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        emit orders_query_out("select bin_id, pos_x, pos_y from
            dropoff_points where order_id is NULL");
        await orders_results;

        if (check_result(?orders_results)) then
          if (num_rows(?orders_results) > 0) then
            var row : MYSQL_ROW in
              row := get_next_row(?orders_results);
              emit bin(getint(row,0));
              emit bin_x(getint(row,1));
              emit bin_y(getint(row,2));
            end var;

            var comp_str : string in
              comp_str := "Using bin ";
              call appint() (comp_str, ?bin);
              call appstr() (comp_str, " at pos (");
              call appint() (comp_str, ?bin_x);
              call appstr() (comp_str, ", ");
              call appint() (comp_str, ?bin_y);
              call appstr() (comp_str, ")");
              emit print(comp_str);
            end var;
          else
            exit no_free_bins;
          end if;
        else
          exit bad_query;
        end if;

        call clear_results()(?orders_results);

        emit num_items_in_bin(0);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Update order status to PICKING
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

var query_str : string in
  query_str := "update orders set status = 'PICKING' where
    order_id = ";
  call appint() (query_str, ?order_id);
  emit orders_query_out(query_str);
  await orders_results;
end var;

if (check_result(?orders_results)) then
  if (num_affected_rows(?orders_results) <> 1) then
    exit order_status_update_failed;
  end if;
else
  exit bad_query;
end if;

call clear_results()(?orders_results);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Get order lines & get goods
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

var query_str : string in
  query_str := "SELECT item.item_id, quantity, name, stock_level,
      shelf_height, pos_x, pos_y FROM order_lines, item WHERE
      order_lines.item_id = item.item_id
      AND order_lines.order_id = ";
  call appint() (query_str, ?order_id);
  emit orders_query_out(query_str);
  await orders_results;
end var;

if (check_result(?orders_results)) then
  var num_order_lines : integer in
    num_order_lines := num_rows(?orders_results);

    %iterate through each order line
    repeat num_order_lines times

      var row : MYSQL_ROW,
          buf : string in
        row := get_next_row(?orders_results);
        emit item_id(getint(row,0));
        emit item_quantity(getint(row,1));
        emit item_name(getstr(row,2,buf));
        emit item_stock_level(getint(row,3));
        emit shelf_height(getint(row,4));
        emit item_x(getint(row,5));
        emit item_y(getint(row,6));

        var comp_str : string in
          comp_str := "Getting ";
          call appint() (comp_str, ?item_quantity);
          call appstr() (comp_str, " ");
          call appstr() (comp_str, ?item_name);
          call appstr() (comp_str, " (ID: ");
          call appint() (comp_str, ?item_id);
          call appstr() (comp_str, ") from stock of ");
          call appint() (comp_str, ?item_stock_level);
          call appstr() (comp_str, " at pos (");
          call appint() (comp_str, ?item_x);
          call appstr() (comp_str, ",");
          call appint() (comp_str, ?item_y);
          call appstr() (comp_str, "), shelf height: ");
          call appint() (comp_str, ?shelf_height);

          emit print(comp_str);
        end var;

        var route_string : string,
            item_height : integer in

          item_height := ?item_stock_level - ?item_quantity +
            1 + ?shelf_height;
          route_string := generate_route_to_pickup_item(?item_x,
            ?item_y, item_height);

          emit execute_route(route_string);
          await route_complete;

          item_height := ?num_items_in_bin+1;
          route_string := generate_route_to_drop_item(?bin_y,
            item_height);

          emit execute_route(route_string);
          await route_complete;
        end var;

        emit num_items_in_bin(pre(?num_items_in_bin)+?item_quantity);

        %% Update stock level
        %%%%%%%%%%%%%%%%%%%%%%%

        var query_str : string in
          query_str := "update item set stock_level = ";
          call appint()
           (query_str, ?item_stock_level - ?item_quantity);
          call appstr() (query_str, " where item_id = ");
          call appint() (query_str, ?item_id);
          emit stock_query_out(query_str);
          await stock_results;
        end var;

        if (check_result(?stock_results)) then
          if (num_affected_rows(?stock_results) <> 1) then
            exit stock_update_failed;
          end if;
        else
          exit bad_query;
        end if;
```

```
        call clear_results()(?stock_results);

      end var;
    end repeat;
  end var;
else
  exit bad_query;
end if;

call clear_results()(?orders_results);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Update order status to STORED
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

var query_str : string in
  query_str := "update orders set status = 'STORED' where
    order_id = ";
  call appint() (query_str, ?order_id);
  emit orders_query_out(query_str);
  await orders_results;
end var;

if (check_result(?orders_results)) then
  if (num_affected_rows(?orders_results) <> 1) then
    exit order_status_update_failed;
  end if;
else
  exit bad_query;
end if;

call clear_results()(?orders_results);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Update order location (update drop off bin)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

var query_str : string in
  query_str := "update dropoff_points set order_id = ";
  call appint() (query_str, ?order_id);
  call appstr() (query_str, " where bin_id = ");
  call appint() (query_str, ?bin);
  emit orders_query_out(query_str);
  await orders_results;
end var;

if (check_result(?orders_results)) then
  if (num_affected_rows(?orders_results) <> 1) then
    exit bin_update_failed;
  end if;
else
  exit bad_query;
end if;

call clear_results()(?orders_results);
```

```
        %%%% Done - repeat for next order
        %%%%%%%%%%%%%%%

      handle bad_query do
        emit print("Bad query");
        await SPEED tick;

      handle no_free_bins do
        emit print("no free bins");
        await SPEED tick;

      handle no_waiting_orders do
        emit print("no waiting orders");
        await SPEED tick;

      handle item_quantity_exceeds_stock do
        %Because of loacl vars the msg is already emitted here
        await SPEED tick;

      handle order_status_update_failed do
        emit print("status update failed");
        await SPEED tick;

      handle stock_update_failed do
        emit print("stock update failed");
        await SPEED tick;

      handle bin_update_failed do
        emit print("bin update failed");
        await SPEED tick;

      handle less_lines_than_num_rows_reports do
        emit print("less_lines_than_num_rows_reports");
        await SPEED tick;

      end trap
    end loop
  ||
    loop
      await
        case route_complete do
          emit print("route complete");
      end await;
      pause;
    end loop;
  ]
end signal;
end module
```

B.2.2 Movement RCX

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%  ROBOT CONTROL
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
module rcx1:

  constant MAX_SPEED : integer;
  constant OFF = 0 : integer;
  constant FWD = 1 : integer;
  constant REV = 2 : integer;
  constant BRAKE = 3 : integer;

  output MOTOR_A_DIR := OFF : integer;
  output MOTOR_A_SPEED := 0 : integer;

  output MOTOR_B_DIR := OFF : integer;
  output MOTOR_B_SPEED := 0 : integer;

  output MOTOR_C_DIR := OFF : integer;
  output MOTOR_C_SPEED := 0 : integer;

  output LCD_INT : integer;
  output CPUTS : string;

  sensor LIGHT_1 : integer;
  sensor LIGHT_2 : integer;
  sensor LIGHT_3 : integer;

  function get_op(string, integer) : integer;
  function get_param(string, integer) : integer;
  function get_num_ops(string) : integer;

  constant MOVE_FORWARD = 102 : integer;
  constant TURN_LEFT = 108 : integer;
  constant TURN_RIGHT = 114 : integer;
  constant PICKUP_ITEM = 112 : integer;
  constant DROP_ITEM = 100 : integer;

  input execute_route : string;
  output route_complete;

  output pickup_item : integer;
  output drop_item : integer;
  input forklift_op_complete;

  signal move_forward : integer,
         turn_left : integer,
         turn_right : integer,
         movement_op_complete in
    [
      loop
        await execute_route;
        var op_num : integer,
            op_type : integer,
            op_param : integer in
          op_num := 0;

          repeat get_num_ops(?execute_route) times

            op_type := get_op(?execute_route, op_num);
            op_param := get_param(?execute_route, op_num);
            if (op_type = MOVE_FORWARD) then
              emit CPUTS("for");
              emit LCD_INT(op_param);
              emit move_forward(op_param);
              await movement_op_complete;
            elsif (op_type = TURN_LEFT) then
              emit CPUTS("left");
              emit LCD_INT(op_param);
              emit turn_left(op_param);
              await movement_op_complete;
            elsif (op_type = TURN_RIGHT) then
              emit CPUTS("right");
              emit LCD_INT(op_param);
              emit turn_right(op_param);
              await movement_op_complete;
            elsif (op_type = PICKUP_ITEM) then
              emit CPUTS("pick");
              emit LCD_INT(op_param);
              emit pickup_item(op_param);
              await forklift_op_complete;
            elsif (op_type = DROP_ITEM) then
              emit CPUTS("drop");
              emit LCD_INT(op_param);
              emit drop_item(op_param);
              await forklift_op_complete;
            end if;

            op_num := op_num+1;
            pause;
          end repeat;
          pause;
          emit route_complete;
        end var;
      end loop;
    ||
      loop
        await

          case immediate move_forward do
            run move_forward;

          case immediate turn_left do
            signal num_turns : integer, direction : integer in
              emit num_turns(?turn_left/90);
              emit direction(1);
                %left turn - primary sensor: light 1
              run turn_robot [signal LIGHT_1 / light_sensor;
                 signal LIGHT_3 / secondary_light_sensor];
            end signal;

          case immediate turn_right do
            signal num_turns : integer, direction : integer in
              emit num_turns(?turn_right/90);
              emit direction(2);
                %right turn - primary sensor: light 3
              run turn_robot [signal LIGHT_3 / light_sensor;
```

```
                signal LIGHT_1 / secondary_light_sensor];
            end signal;

        end await;
        pause;
        emit movement_op_complete;

      end loop;
    ];
  end signal;
end module

%%% MOVE ROBOT FORWARD
%%%%%%%%%%%%%%%%%%%%%%%%%
module move_forward:

  constant MAX_SPEED : integer;
  constant OFF = 0 : integer;
  constant FWD = 1 : integer;
  constant REV = 2 : integer;
  constant BRAKE = 3 : integer;

  output MOTOR_A_DIR : integer;
  output MOTOR_A_SPEED : integer;

  output MOTOR_B_DIR : integer;
  output MOTOR_B_SPEED : integer;

  output LCD_INT : integer;

  sensor LIGHT_1 : integer;
  sensor LIGHT_2 : integer;
  sensor LIGHT_3 : integer;

  constant grey = 43 : integer;
  constant black = 33 : integer;

  input move_forward : integer;

  trap dest_reached in
    signal dest_reached in
      emit MOTOR_A_SPEED(120);
      emit MOTOR_B_SPEED(30);
      emit MOTOR_A_DIR(FWD);
      emit MOTOR_B_DIR(OFF);
      [
        loop
          if (?LIGHT_1 < grey or ?LIGHT_3 < black) then
          %turning too far left
            emit MOTOR_B_DIR(FWD);
            trap on_line in
              loop
                if (?LIGHT_1 > grey or ?LIGHT_3 > black) then
                  exit on_line;
                end if;
                pause;
              end loop
            end trap;
            emit MOTOR_B_DIR(OFF);
          end if;

          if (?LIGHT_3 < grey or ?LIGHT_1 < black) then
          %turning too far right
            emit MOTOR_B_DIR(REV);
            trap on_line in
              loop
                if (?LIGHT_3 > grey or ?LIGHT_1 > black) then
                  exit on_line;
                end if;
                pause;
              end loop
            end trap;
            emit MOTOR_B_DIR(OFF);
          end if;
          pause;
        end loop;
      ||
        var count : integer in

          count := 0;

          await 40 tick;

          trap off_white in
            loop
              if (?LIGHT_2 < 49) then
                %so light2 is over a black square...
                exit off_white;
              end if;
              pause;
            end loop
          end trap;

          loop
            trap on_white in
              loop
                if (?LIGHT_2 > 49) then
                  %so light2 is over a white square...
                  exit on_white;
                end if;
                pause;
              end loop
            end trap;

            trap on_dark_colour in
              loop
                if (?LIGHT_2 < 49) then
                  %so light2 is over a black square...
                  exit on_dark_colour;
                end if;
                pause;
              end loop
            end trap;
```

```
            count:= count+1;
            if (count = ?move_forward) then
              exit dest_reached;
            end if;
            pause;
          end loop;
        end var;
      ]
    end signal
  handle dest_reached do
    pause;
    emit MOTOR_A_DIR(BRAKE);
    emit MOTOR_B_DIR(BRAKE);
  end trap;

end module

%%% TURN ROBOT
%%%%%%%%%%%%%%%%%%%%%%%%

module turn_robot:

  constant MAX_SPEED : integer;
  constant OFF = 0 : integer;
  constant FWD = 1 : integer;
  constant REV = 2 : integer;
  constant BRAKE = 3 : integer;

  output MOTOR_A_DIR : integer;
  output MOTOR_A_SPEED : integer;

  output MOTOR_B_DIR : integer;
  output MOTOR_B_SPEED : integer;

  output LCD_INT : integer;

  constant grey = 43 : integer;
  constant black = 33 : integer;

  input direction : integer;
  input num_turns : integer;
  sensor light_sensor : integer;
  sensor secondary_light_sensor : integer;


  emit MOTOR_B_DIR(?direction);

  repeat ?num_turns times

    emit MOTOR_B_SPEED(200);

    %on black > test, move to white execute the rest
    %on white > the rest (wait for grey first)
    %on grey  > pass through first trap instantly,
    %           continue as normal
    if (?light_sensor < black) then
      trap on_white in
```

```
        loop
          if (?light_sensor > grey) then
            exit on_white;
          end if;
          pause;
        end loop
      end trap;
    end if;

    trap moved_off_line in
      loop
        if (?light_sensor < grey) then
          exit moved_off_line;
        end if;
        pause;
      end loop
    end trap;

    trap hit_white_line in
      loop
        if (?light_sensor > grey) then
          exit hit_white_line;
        end if;
        pause;
      end loop
    end trap;

    trap moved_off_line in
      loop
        if (?light_sensor < grey) then
          exit moved_off_line;
        end if;
        pause;
      end loop
    end trap;

    emit MOTOR_B_SPEED(50);

    trap hit_both_white_lines in
      loop
        if (?light_sensor > grey and
            ?secondary_light_sensor > grey) then
          exit hit_both_white_lines;
        end if;
        pause;
      end loop
    end trap;

    pause;
  end repeat;
  emit MOTOR_B_DIR(BRAKE);

end module
```

B.2.3 Forklift RCX

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%  FORKLIFT CONTROL
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Assumes forklift is calibrated so a reading of 0 on
% rotation sensor 1 lines the forklift up with the hole
% under the lowest item.

module rcx2:

  constant MAX_SPEED : integer;
  constant OFF = 0 : integer;
  constant FWD = 1 : integer;
  constant REV = 2 : integer;
  constant BRAKE = 3 : integer;

  constant ITEM_HEIGHT = 118 : integer;
  constant PICKUP_HEIGHT = 59 : integer;
  constant MAX_ARM_EXTENSION = 230 : integer;

  output MOTOR_A_DIR := OFF : integer;
  output MOTOR_A_SPEED := 0 : integer;

  output MOTOR_B_DIR := OFF : integer;
  output MOTOR_B_SPEED := 0 : integer;

  output MOTOR_C_DIR := OFF : integer;
  output MOTOR_C_SPEED := 0 : integer;

  output LCD_INT : integer;

  sensor ROTATION_1 : integer;
  sensor ROTATION_2 : integer;

  input pickup_item : integer;
  input drop_item : integer;
  output forklift_op_complete;

  loop
    await
      case immediate pickup_item do
        emit MOTOR_A_SPEED(MAX_SPEED);
        emit MOTOR_B_SPEED(MAX_SPEED);

        %move forklift to item height (to get in under it)
        emit MOTOR_A_DIR(FWD);
        trap reached_height in
          loop
            if (-?ROTATION_1 >= ((?pickup_item-1) * ITEM_HEIGHT))
            then exit reached_height;
            end if;
            pause;
          end loop
        end trap;
        emit MOTOR_A_DIR(BRAKE);
        %extend arm
        emit MOTOR_B_DIR(REV);
        trap arm_extended in
          loop
            if (-?ROTATION_2 >= MAX_ARM_EXTENSION) then
              exit arm_extended;
            end if;
            pause;
          end loop
        end trap;
        emit MOTOR_B_DIR(BRAKE);

        %raise forklift to lift the item
        emit MOTOR_A_DIR(FWD);
        trap picked_up_item in
          loop
            if (-?ROTATION_1 >=
               (((?pickup_item-1) * ITEM_HEIGHT)+PICKUP_HEIGHT))
            then exit picked_up_item;
            end if;
            pause;
          end loop
        end trap;
        emit MOTOR_A_DIR(BRAKE);

        %retract arm
        emit MOTOR_B_DIR(FWD);
        trap arm_retracted in
          loop
            if (-?ROTATION_2 <= 0) then
              exit arm_retracted;
            end if;
            pause;
          end loop
        end trap;
        emit MOTOR_B_DIR(BRAKE);

        %lower forklift
        emit MOTOR_A_DIR(REV);
        trap reached_bottom in
          loop
            if (-?ROTATION_1 <= 0) then
              exit reached_bottom;
            end if;
            pause;
          end loop
        end trap;
        emit MOTOR_A_DIR(BRAKE);

      case immediate drop_item do
        emit MOTOR_A_SPEED(MAX_SPEED);
        emit MOTOR_B_SPEED(MAX_SPEED);

        %move forklift to eventual item height +
        %a bit more (higher than all items)
        emit MOTOR_A_DIR(FWD);
        trap reached_height in
          loop
            if (-?ROTATION_1 >=
                ((?drop_item-1) * ITEM_HEIGHT + PICKUP_HEIGHT))
            then exit reached_height;
            end if;
            pause;
          end loop
        end trap;
        emit MOTOR_A_DIR(BRAKE);

        %extend arm
        emit MOTOR_B_DIR(REV);
        trap arm_extended in
          loop
            if (-?ROTATION_2 >= MAX_ARM_EXTENSION) then
              exit arm_extended;
            end if;
            pause;
          end loop
        end trap;
        emit MOTOR_B_DIR(BRAKE);

        %lower forklift to set down the item
        emit MOTOR_A_DIR(REV);
        trap set_down_item in
          loop
            if (-?ROTATION_1 <= ((?drop_item-1) * ITEM_HEIGHT)) then
              exit set_down_item;
            end if;
            pause;
          end loop
        end trap;
        emit MOTOR_A_DIR(BRAKE);

        %retract arm
        emit MOTOR_B_DIR(FWD);
        trap arm_retracted in
          loop
            if (-?ROTATION_2 <= 0) then
              exit arm_retracted;
            end if;
            pause;
          end loop
        end trap;
        emit MOTOR_B_DIR(BRAKE);

        %lower forklift
        emit MOTOR_A_DIR(REV);
        trap reached_bottom in
          loop
            if (-?ROTATION_1 <= 0) then
              exit reached_bottom;
            end if;
            pause;
          end loop
        end trap;
        emit MOTOR_A_DIR(BRAKE);
```

```
    end await;
    pause;
    emit forklift_op_complete;
  end loop
end module
```

C Lego Wrapping Script

The Lego Mindstorms wrapping script allows the use of Lego Mindstorms' devices from an Esterel program. It is written in Perl [20] and loosely based on the *legolus* script of Christophe Mauras [18]. The API specification is the same as that used in *scLego* [11], which is included with the Esterel compiler. Motors are controlled by two output signals, one for direction and one for speed. Rotation and light sensors are specified as sensor signals and touch sensors are represented by input signals.

The script begins by performing some preprocessing on the sensor list that it is passed as a parameter. This is just a checking phase to ensure the user has specified the sensors correctly. It then opens the output file and starts inserting code. Firstly, it writes out a series of #includes required by BrickOS. The motor output functions corresponding to the motor output signals are inserted next. Then, the sensor functions for rotation and light sensors are written out. Lastly, the main function is generated, which is responsible for initialising the system and performing the "reactive loop", i.e., setting up the inputs, calling the step function, and pausing to ensure the cycle time for each step is fixed. The script can also be instructed not to generate a main function and instead to insert the Lego specific code into a pre–generated main function. This is needed when another script that generates a main function has already been run on the output file.

To display its usage information, the script is run with the `-help` option:

```
#!/usr/bin/perl -w

$first_arg = $ARGV[0];

if (!defined($first_arg) || ($first_arg =~ /help/)) {
  print <<"PRINT_USAGE";

Esterl2Lego
#########
Creates wrapping code given an Esterel generated reactive system
for the Lego Mindstorms architecture.

Usage:
esterel2lego <main module name> <input file> <output file>
  <generate main procedure> <sensor list>

Main module name:
  Name of main (top level) module.
```

```
Input file:
  The file containing the Esterel generated C code.

Output file:
  The file where the wrapped code will be placed.

Generate main procedure:
  Specify whether to generate main procedure or not. Use "1"
  to generate and "0" not to. If "0" is used then the input
  file must specify where the sensor initialization and input
  calls must be placed. This is done by placing the comment
  "/* extra init start here */" on a line by itself for
  initializations and "/* extra input calls start here */ on
  a line by itself for calling inputs.

Sensor list:
  Specify which sensors the program uses on which ports by
  listing them in order of ports (use none if no sensor is
  attached). If a light sensor is specified then immediately
  follow it with an 'a' for active mode or 'p' for passive mode.

  eg. for a program that uses a passive light sensor on port 1,
      nothing on port 2 and a touch sensor on port 3 the script
      call would be like this:

      light p none touch

Full example:
  Main node called robot_control, C code in file robot_control.c,
  we want a main procedure to be generated and the robot uses an
  active light sensor on ports 1 and 2.  This would be specified
   as such:

  esterel2lego robot_control robot_control.c robot_control_out.c
  1 light a light a none

  The output from the script would be placed in the file
  robot_control_out.c

Lego API:

  constant MAX_SPEED : integer;
  constant OFF = 0 : integer;
  constant FWD = 1 : integer;
  constant REV = 2 : integer;
  constant BRAKE = 3 : integer;

  output MOTOR_A_DIR := OFF : integer;
  output MOTOR_A_SPEED := 0 : integer;

  output MOTOR_B_DIR := OFF : integer;
  output MOTOR_B_SPEED := 0 : integer;

  output MOTOR_C_DIR := OFF : integer;
  output MOTOR_C_SPEED := 0 : integer;

  output LCD_INT : integer;
  sensor ROTATION_1 : integer;
  sensor ROTATION_2 : integer;
  sensor ROTATION_3 : integer;

  sensor LIGHT_1 : integer;
  sensor LIGHT_2 : integer;
  sensor LIGHT_3 : integer;

  input TOUCH_1;
  input TOUCH_2;
  input TOUCH_3;

  Only one sensor on each port can be used. e.g., DO NOT declare
  both ROTATION_1 and LIGHT_1.


PRINT_USAGE
exit;
}
```

The following gives the implementation of the Perl script:

```
$main_module = shift(@ARGV);
$code_file = shift(@ARGV);
$out_file = shift(@ARGV);
$generate_main = shift(@ARGV);

for (my $i=1; $i<4; $i++) {
  if (defined($input = shift(@ARGV))) {
    if ($input eq "light") {
      $light_mode[$i] = shift(@ARGV);
      if ($light_mode[$i] ne "a" && $light_mode[$i] ne "p") {
        die("light sensor $i mode not specified or invalid mode
             (use a or p)\n");
      }
      $sensors[$i] = "LIGHT";
    }
    elsif ($input eq "touch") {
      $sensors[$i] = "TOUCH";
    }
    elsif ($input eq "rotation") {
      $sensors[$i] = "ROTATION";
    }
    elsif ($input eq "none") {
      $sensors[$i] = "NONE";
    }
    else {
      die("Invalid sensor name: $input (Valid names:
           light, touch, rotation, none)");
    }
  }
  else {
    die("Not enough sensors specified (use none to signal
        that no sensor is connected to that port)\n");
  }
}

if (!open (IN, "${code_file}")) {
    print "Cannot open ${code_file} for reading\n";
    exit 1;
}

if (!open (OUT, ">${out_file}")) {
    print "Cannot open ${out_file} for writing\n";
    exit 1;
}

print OUT <<"INCLUDES";

#include <unistd.h>
#include <dmotor.h>
#include <conio.h>
#include <dsensor.h>
#include <dlcd.h>
#include <stdlib.h>
INCLUDES

print OUT <<"OUTPUT_FUNS";

void ${main_module}_O_MOTOR_A_DIR(int dir) {
  motor_a_dir(dir);
}

void ${main_module}_O_MOTOR_B_DIR(int dir) {
  motor_b_dir(dir);
}

void ${main_module}_O_MOTOR_C_DIR(int dir) {
  motor_c_dir(dir);
}

void ${main_module}_O_MOTOR_A_SPEED(int speed) {
  motor_a_speed(speed);
}

void ${main_module}_O_MOTOR_B_SPEED(int speed) {
  motor_b_speed(speed);
}

void ${main_module}_O_MOTOR_C_SPEED(int speed) {
  motor_c_speed(speed);
}

void ${main_module}_O_CPUTS(char* string) {
  cputs(string);
}

void ${main_module}_O_LCD_INT(int value) {
  lcd_int(value);
}

OUTPUT_FUNS
```

```
print OUT <<"SENSOR_FUNS";

int ${main_module}_S_LIGHT_1() {
  return LIGHT_1;
}

int ${main_module}_S_LIGHT_2() {
  return LIGHT_2;
}

int ${main_module}_S_LIGHT_3() {
  return LIGHT_3;
}

int ${main_module}_S_ROTATION_1() {
  return ROTATION_1;
}

int ${main_module}_S_ROTATION_2() {
  return ROTATION_2;
}

int ${main_module}_S_ROTATION_3() {
  return ROTATION_3;
}
SENSOR_FUNS

if ($generate_main eq "1") {

  while (<IN>) {
    if ($_ =~ /${main_module}.h/) {
    }
    else {
      print OUT $_;
    }
  }

  print OUT "int main(int ARGC, char **ARGV) {\n";
  print_sensor_init();
  print OUT "\t${main_module}_reset();\n";
  print OUT "\twhile(1) {\n";
  print_touch_input_calls();
  print OUT "\t\t${main_module}();\n";
  print OUT "\t\tmsleep(10);\n";
  print OUT "\t}\n";
  print OUT "}\n";
}
else {

  while ($line = <IN>) {
    if ($line =~ /${main_module}.h/) {
    }
    elsif ($line =~ /extra init start here/) {
      print OUT $line;
      last;
    }
    else {
      print OUT $line;
    }
  }

  print_sensor_init();

  while ($line = <IN>) {
    if ($line =~ /extra input calls start here/) {
      print OUT $line;
      last;
    }
    else {
      print OUT $line;
    }
  }

  print_touch_input_calls();

  while (<IN>) {
    print OUT;
  }
}

sub print_touch_input_calls {
  for (my $i=1; $i<4; $i++) {
    if ($sensors[$i] eq "TOUCH") {
      print OUT "\t\tif (TOUCH_${i}) {\n";
      print OUT "\t\t\t${main_module}_I_TOUCH_${i}();\n";
      print OUT "\t\t}\n";
    }
  }
}

sub print_sensor_init {
  for (my $i=1; $i<4; $i++) {
    if ($sensors[$i] eq "LIGHT") {
      if ($light_mode[$i] eq 'a') {
        print OUT "\tds_active(&SENSOR_${i});\n";
      }
      else {
        print OUT "\tds_passive(&SENSOR_${i});\n";
      }
    }
    elsif ($sensors[$i] eq "ROTATION") {
      print OUT "\tds_active(&SENSOR_${i});\n";
      print OUT "\tds_rotation_on(&SENSOR_${i});\n";
      print OUT "\tds_rotation_set(&SENSOR_${i},0);\n";
      print OUT "\tmsleep(100);\n";
    }
  }
}
```