Final Project Report on

Implementation of Type Checker in Borealis

By
Anjali Jhingran
M.S., Brown University 2005

A project submitted in partial fulfillment of the requirements for
the Master of Science Degree
in the department of Computer Science at Brown University

Providence, Rhode Island
May 2005

The project by Anjali Jhingran is accepted in the present form by
the Department of Computer Science as satisfying the research project requirement
for the Degree of Master of Science

Date: _____ _____

Professor Ugur Cetintemel, Advisor

Contents

1. Introduction

1.1. Borealis - The Stream Processing Engine

Recently in the database, networking and sensor network communities researchers have developed the notion of a stream management system [1,2,3]. In these systems, there is a continuous flow of data, say from stock tickers, and a collection of relatively static queries that operate on incoming data, e.g. "notify me when MSFT exceeds $200 a share", these queries remain resident for long durations. Researchers have extended these systems, to a distributed setting [4,5], in which there are networks of computers and sensors each node running a portion of a query.

Over the last several years, a great deal of progress has been made in the area of stream processing engines (SPE). Several groups have developed working prototypes (e.g., Aurora [1], STREAM [2], TelegraphCQ [3]) and many papers have been published on detailed aspects of the technology such as stream-oriented languages, resource-constrained one-pass query processing, load shedding, and distributed processing. While this work is an important first step, fundamental mismatches remain between the requirements of many streaming applications [6,7] and the capabilities of first-generation systems.

Borealis is a second-generation distributed stream-processing engine that is being developed at Brandeis University, Brown University, and MIT [8]. Borealis inherits core stream processing functionality from Aurora and distribution functionality from Medusa. Borealis modifies and extends both systems in non-trivial and critical ways to provide advanced capabilities that are commonly required by newly emerging stream-processing applications.

Through sample real-world applications, we motivate the need for. Borealis addresses the challenges of dynamically revising query results and modifying query specifications through an innovative set of features, including revision records, time travel, and control lines. Borealis is based on a highly flexible and scalable QoS-based optimization model that operates across server and sensor networks and a new fault-tolerance model with flexible consistency-availability trade-offs [8].

1.2. Type Checking in Borealis

Before the implementation of the Type Checker, all the data that was coming from a user to the Borealis server used to be stored in a Catalog, which is a data-structure. However, there was no checking performed to see whether the entering data is correct and consistent. Type checking was, therefore, required to make sure that everything stored in Borealis Catalog is correct and does not violate the rules defined by Borealis system. Type checker can be seen as a guard against the incorrect syntax or semantics of incoming data. Type checker in Borealis is integrated with Catalog and is invoked automatically by the system before any information is actually stored in it.

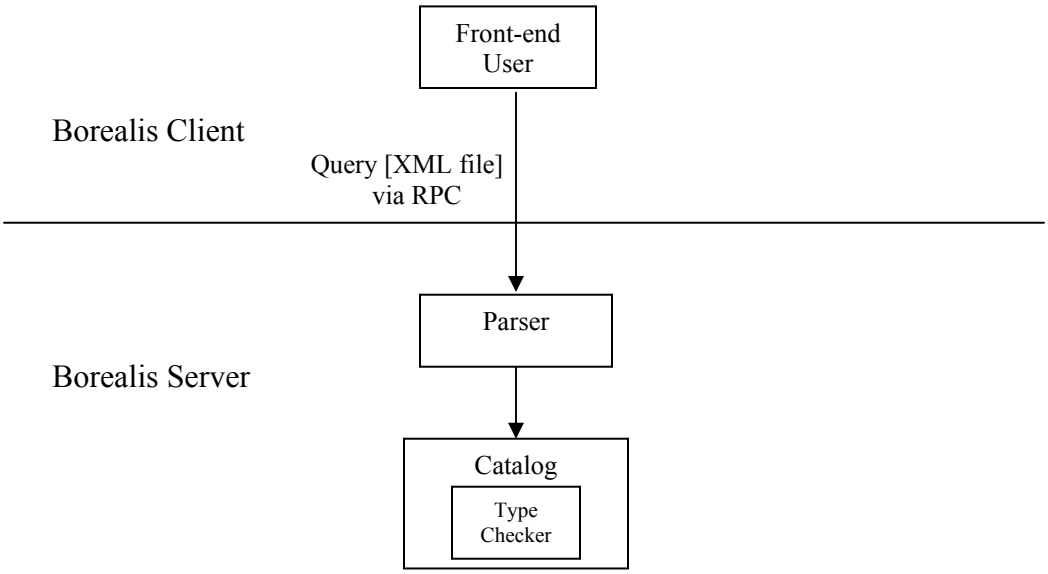
2. Details of implementation of the type checker

2.1. High Level Structure

Borealis database includes two parts – Borealis Client and Borealis Server.

Borealis Client - It is a user-friendly graphical interface that enables users to define queries using a Borealis tool built in Java. It allows user to define a query diagram that includes boxes, streams etc., without worrying about the syntax. At the same time, an XML representation of the query diagram is generated which can be modified by the user manually as well.

Borealis Server – Borealis server is the place where all the data is processed and system related activities, such as load distribution, HA, partitioning etc., are handled as well. An important aspect of Borealis Server is to maintain all the data in a Catalog, which is an in-memory data structure defined to store query data. However, to make sure that entered data is valid, type checking is performed on all the data that comes in the Catalog for storage. Type Checker is integrated within the Catalog and is automatically called from Catalog.



2.2. Flow of the Program

User (front-end) sends XML files Borealis server using an RPC (Remote Procedure Call) over the network. XML files are first parsed using a Parser program (Diagram.cc). The program passes all the values to Catalog by making calls to appropriate functions of Catalog (Catalog.cc). The Catalog expects query in a standard format. To ensure the correctness of input data it in turn calls relevant functions of Type Checker (CatalogValidate.cc) before storing values in it. A Type Checker is a program that acts as a filtering layer for catalog and facilitate catalog by making sure that the query schema and all the input parameters passed to the Catalog are valid. If it finds

any parameter that violates the required semantic or syntax, it generates an error message. This message is returned to the user through an RPC call that contains the XML file name, parameter related information and the reason of error. If there is no error in the XML file, all the data is stored in the Catalog.

The type checking process does not wait for the whole XML file to parse and produce errors collectively. Instead, as soon as a violation in the syntax or semantic is encountered, an error message is generated that is immediately sent back to the user. Thus, type checking is performed at each step as the data is stored in the catalog.

Note: Please refer to Appendix 1 to see the list of XML parameter on which the type checking has been implemented.

2.3. Type Checker Implementation

2.3.1. Added Program to Borealis

A new program has been added to the existing Borealis system to implement type checking on the server side. The program is called "CatalogValidate.cc" and is located under the directory "borealis/src/module/catalog/distributed/." The program defines all the functions that are declared in Catalog.h program and are called from Catalog.cc program. Functions of CatalogValidate.cc program return "success" if there is no error, otherwise return an error message to the calling program.

2.3.2. Modified Programs in Borealis

Catalog.h and Catalog.cc have been modified to implement type checking. Catalog.h program declares all the functions that are defined in CatalogValidate.cc and Catalog.cc program calls the relevant function of CatalogValidate.cc program with appropriate parameters.

Diagram.cc has also been modified, since in some cases there were not enough functions defined in Catalog.cc program to handle the kind of type checking needed. Hence, those functions have first been declared in Catalog.h and then, have been defined and called from Catalog.cc program.

2.3.3. Test Program

A test program has been created to check whether type checker is compiling and running fine; and is producing the desired result. The test program is called "Diagramtest.cc" and is located under directory "borealis/utility/test/unit/." A Makefile.am file is also been created with appropriate sources and dependencies to run to the program, which is present in the same directory as "Diagramtest.cc."

3. User's guide to Borealis Type Checker

Since Borealis Type Checker is integrated into the Catalog, it is automatically called from the Catalog before catalog tries to store any data into it. To test Type Checker manually, write a test program that explicitly passes an XML file to the Parser (Diagram.cc) and then all the subsequent calls will be made automatically to the actual type checking function to perform type checking. (More explicitly subsequent calls are made to Catalog.cc and then CatalogValidate.cc)

Note: Create a make file to execute the code with appropriate files as sources and dependencies.

4. Programmer's guide to Borealis Type Checker

The existing Type Checker could be modified when there is a need to

- Add a new function or
- Change the number of parameters passed to a function or
- Change the type of parameters passed to a function (when the data structure changes).

In all the above cases, three to four programs could be affected and hence need modification. Those programs are namely the parser (Diagram.cc), Catalog (Catalog.h and Catalog.cc) and Type Checker (CatalogValidate.cc).

To enhance or modify the existing functionality of Type Checker, first make changes in the Catalog.h file (either declare a new function or modify the existing type checking function). Then, call that function from an appropriate function of Catalog.cc program. Finally, define the function in CatalogValidate.cc program.

However, there might be a possibility that the new function to be added for type checking does not fit any functions defined in Catalog.cc program. In this situation, call this function from the parser (Diagram.cc) program, which is the first program in the sequence that is called from the Borealis server after it receives an XML file. This also requires creating a new function in Catalog (in Catalog.h and Catalog.cc) that would eventually call the type checking function (in CatalogValidate.cc). Finally, declare and define validation function in Catalog.h and CatalogValidate.cc respectively.

4.1. Example

The example shows how to add a new function that does not fit to be called from any existing Catalog (Catalog.cc) functions. The function to be added here is **numberOfParam(…)**. This function checks whether there is at least one parameter defined for each box except box-type “Union.” (A union may or may not have a parameter defined for it)

Step 1: Declare two functions, one for Catalog and one for Type Checker.

```
Catalog.h
class Catalog
{ public:
…..
void  numberOfParam( /// Box operation,  Number of Parameters passed to a box
                                        string  type,       int  number_of_params )
                    throw( AuroraBadEntityException );

Status  validateNumberOfParam( // Operation on a box, Number of Parameters passed to one box
                                          string  type,             int  number_of_params );
…..
}
```

Step 2: Define body of the calling function. This makes call to Type Checker function.

```
Catalog.cc
void  Catalog::numberOfParam( // Box operation,      Number of parameters
                                        string  type,          int  number_of_params )
        throw( AuroraBadEntityException )
{
            Status  status = validateNumberOfParam( type,  number_of_params );
            if(!status)
            {  Throw( AuroraBadEntityException, status.as_string() );
            }
return;
}
```

Step 3: Define body of the Type Checker Function.

```
CatalogValidate.cc
Status  Catalog::validateNumberOfParam(// box type,     number of parameters
                                                                    string  type,  int  number_of_params )
{  // Except box-type “Union,” all boxes require at One parameter to be defined inside them.
   if( type != "union" &&  number_of_params == 0)
    {   return( " Box ( " + type + " ) expects at least one predicate " );
     }
}
return( true );
}
```

Step 4: Finally calling the function from Parser.

```
Diagram.cc
void  Diagram::parse1Box( const DOMElement  *box )  throw( AuroraBadEntityException )
{
…..
xmlChildElements( parameter, box, "parameter" );
// only union allowed to have no parameters.
numberOfParam( type,  parameter.size() );
……
}
```

Note: If there is a change in the data structure then relevant programs such as CatalogStream.h, CatalogBox.h may affect type checker. In this case, necessary changes need to be made in those programs as well as in the type checker with the consent of others (Borealis group).

5. Bugs and Limitations

Presently, all the type checking that has been implemented is working fine and there are no Bugs reported. However, there are certain considerations taken into account while implementing and testing the type checker. First of all, the Catalog is an in-memory process meaning that data stays in Catalog as long as the Borealis server is running and it is not stored on any persistent storage media, such as hard disk or tape drive. Thus, every time a Borealis server is started or restarted, the Catalog is initialized and first type checking is done against an empty set of values.

6. Conclusions and Future Work

Type checking is a crucial part Borealis system. It provides two major benefits – stability and consistency. It ensures more stability of Borealis during a run and enables the Catalog to maintain a consistent state of data. So far, there was no dedicated program to type check the input data that was stored in the Catalog. There were a few programs that were doing very elementary checking, but the checking was limited to their functionality. This is the first program that serves the purpose in detail and considers all possible kinds of type checking that could be required on Borealis data.

There are some areas left to type check the incoming XML file for accuracy. For example, type checking for some of the boxes is not implemented yet (refer Appendix 1). Additionally, type checking may also be required if semantics of Borealis system change or if there are any enhancement made into the Borealis data structure.

There is also scope for future work in type checking. An interesting future work in Borealis could be to implement type inference. It can be seen as an extension to type checking. Currently, there is no type inferencing implemented in the Borealis system and given the present description of the Borealis system, there is a need to implement it for boxes. Type inferencing is required to make sure that all the intermediate boxes, that are derive from other boxes, have correct schema and streams connected to them. This will be a completion of type checking aspect of Borealis.

7. References

[1] Daniel Abadi, Don Carney, Ugur Cetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Michael Stonebraker, Nesime Tatbul, and Stan Zdonik, Aurora: A New Model and Architecture for Data Stream Management, VLDB Journal, Vol. 12, No. 2, August, 2003

[2] The STREAM Group. STREAM: The Stanford Stream Data Manager IEEE Data Engineering Bulletin, Vol. 26 No. 1, March 2003

[3] Sirish Chandrasekaran, Owen Cooper, Amol Deshpande, Michael J. Franklin, Joseph M. Hellerstein, Wei Hong, Sailesh Krishnamurthy, Samuel R. Madden, Vijayshankar Raman, Fred Reiss, and Mehul A. Shah. TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. To appear, CIDR 2003.

[4] Mitch Cherniack, Hari Balakrishnan, Don Carney, Ugur Cetintemel, Ying Xing and Stan Zdonik, Scalable Distributed Stream Processing, Proceedings of the Conference for Innovative Database Research (CIDR), January, 2003, Asilomar, CA.

[5] Y. Ahmad, U. Cetintemel. Network-Aware Query Processing for Stream-Based Applications. (Proceedings of the 30th International Conference on Very Large Data Bases (VLDB '04).)

[6] STREAM Query Repository http://www-db.stanford.edu/stream/sqr/

[7] S. Babu, L. Subramanian, and J. Widom. A Data Stream Management System for Network Traffic Management In Proc. of the Workshop on Network-Related Data Management (NRDM 2001), May 2001

[8] Daniel Abadi, Yanif Ahmad, Hari Balakrishnan, Magdalena Balazinska, Ugur Cetintemel, Mitch Cherniack, Jeong-Hyon Hwang, John Jannotti, Wolfgang Lindner, Samuel Madden, Alexander Rasin, Michael Stonebraker, Nesime Tatbul, Ying Xing, Stan Zdonik. The Design of the Borealis Stream Processing Engine. Technical Report. CS-04-08. Brown University, July 2004.

Appendix 1

Type Checking Rules In Borealis

Anjali Jhingran (anjali@cs.brown.edu)
****
For more details on box specifications : "SQuAl" by Cherniack

A generic XML file

```
<borealis owner="sand://delay.cs.brown.edu/">
  <schema name="pl_site_cpu">
    <field name="ip"    type="string"     size="16"/>
    ...
  </schema>
  <input  stream="harv_cpu_load"       schema="pl_site_cpu" />
  ...
  <output stream="union_heavily_used"  schema="pl_resource_usage" />
  ...
  <query name="group1" >
    <table name="testTable"  schema="TableTuple" >
       <index     field="f4"   value="a" />
       <parameter  name="key"  value="f1,f2,f3" />
            ...
    </table>
    ...
    <box name="group2_join_cpu_mem_box" type="join">
            <in      stream="group2_filter_mem"/>
            ...
            <out     stream="group2_heavily_used"/>
            ...
            <parameter name="right-order-by"    value="TUPLES"/>
            ...
    </box>
    ...
  </query>
  ...
</borealis>
```

```
*** ********************************************************************** ***
***                          TYPE CHECKING ON XML PARAMETERS                ***
*** ********************************************************************** ***
```

1. SCHEMA

- To be added must be new (No Duplicate).

2. SCHEMA FIELD

- At least one field must be defined.
- Can be added after adding the schema (name).
- Datatype must be defined for the field.
- Size of the field must be defined for String datatype.

3. STREAM

- Schema must be declared before adding a stream.
- To be added must be new (No Duplicate).

4. INPUTOUPUT STREAM

- Cannot add Input/Output to an undeclared stream.

5. MULTIPLE INPUT STREAMS

- If more than one input streams are added to a Box then,
  All the input streams must have the SAME Schema.

6. BOXES

- "Box name" can be anything
- "Box type" has to match what is declared in the box
- "Input and Output streams" can be anything
- Parameters follow by * are optional
- Box (name) to be added must be new.
- Box must be defined inside the query.
- Box may have the following types.
  Filter, Map, Union, BSort, Aggregate, Join, Resample, Select, Insert, Update,
  Lock, Unlock, Waitfor, RandomDrop, RevisionAggregate, RevisionFilter, SControl,
  SJoin, SOutput, SUnion, WindowDrop.
- Parameters with * are optional.

6.1 FilterQBox (filter)

Number of input	: 1
Number of output	: >= 1
predicate	: >=1
parameter	: expression.i (i=integer)
	: pass-on-false-port

- Allowed to have more than one output streams.
- Output schema should match with input schema.

Comment -
expression.i : predicate (eg. "price > 5")
pass-on-false-port* : 1 (to output false tuples), 0 (otherwise)

6.2. MapQBox (map)

Number of input	: 1
Number of output	: 1
predicate	: >=1
parameter	: expression.i (i=integer)
	: output-field-name.i (i=integer)

Comment -
expression.i : expression (eg. "price + 5")
output-field-name.i : field name for expression.i (eg. "increased_price")

6.3. UnionQBox (union)

Number of input	: >=1
Number of output	: 1

- No parameter is allowed.
- All input streams must have the same schema.
- Output schema should match with input schema.
- Allowed to have MORE than ONE input stream.

6.4. BSortQBox (bsort)

Number of input	: 1
Number of output	: 1
predicate	: >=1
slack	: int
order-on	: field required

- Output schema should match with input schema.

6.5. AggregateQBox (aggregate)

Number of input	: 1
Number of output	: 1
predicate	: >=1
order-by	: "FIELD" or "TUPLENUM"
window-size-by	: "VALUES" or "TUPLES"
window-size	: int
advance	: int
slack	: int
timeout*	: int
order-by	: required
window-size-by	: required

- order-on-field needed when window-size-by=VALUES or order-by=FIELD

6.6. JoinQBox (join)

Number of input	: 2
Number of output	: 1
predicate	: >=1
left-order-by	: "VALUES" or "TUPLES"
right-order-by	: "VALUES" or "TUPLES"
left-buffer-size	: int
right-buffer-size	: int
timeout*	: int
predicate	: required
left-order-on-field	: required
right-order-on-field	: required

6.7. ResampleQBox (resample)

Number of input	: 2
Number of output	: 1
left-order-by	: "FIELD" or "TUPLENUM"
right-order-by	: "FIELD" or "TUPLENUM"
window-size	: int
left-slack	: int
right-slack	: int

Comment -
- This box has not been run by many networks and users
- This box may not be perfect, may crash, etc.

6.8. InsertQBox (insert)

Number of input	: 1
Number of output	: 1
predicate	: >=1
table	: required
sql	: required
value	: of sql parameter requires a SQL statement

- A table must be present inside the box.

Comment -
pass-input* : 1 (to pass input after insertion), 0 (otherwise)

6.9. SelectQBox (select)

Number of input	: 1
Number of output	: >=1
predicate	: >=1
table	: required
sql	: required
value	: of sql parameter requires a SQL statement

- Number of input stream must be exactly TWO when "pass-on-no-results" is "true"
- A table must be present inside the box.

Comment -
pass-on-no-results* : 1 (to pass input on no results), 0 (otherwise)

6.10. UpdateQBox (update)

Number of input	: 1
Number of output	: 1
predicate	: >=1
table	: required
sql	: required
value	: of sql parameter requires a SQL statement
parameter	: output-field-name.i (i=integer)

- A table must be present inside the box.

Comment -
pass-input* : 1 (to pass input after insertion), 0 (otherwise)

6.11. LockQBox (lock)

Number of input	: 1
Number of output	: 1
predicate	: >=1
key	: int
lockset	: required

6.12. UnlockQBox (unlock)

Number of input	: 1
Number of output	: 1
predicate	: >=1
key	: int
lockset	: required

6.13. WaitForQBox (waitfor)

Number of input	: 2
Number of output	: 1
predicate	: >=1
timeout	: int
predicate	: required

7. VIEW

- To be added must be new (No Duplicate).

8. TABLE

- Must be added after declaring the schema.
- To be added must be new (No Duplicate).

9. QUERY

- To be added must be new (No Duplicate).
- Cannot add another query when one is being defined.

*** ********** ***

Boxes not implemented yet –
RandomDrop, RevisionAggregate, RevisionFilter, SControl, SJoin, SOutput,
SUnion, WindowDrop.

*** ********** ***

Appendix 2

1. A Sample XML File

```
<borealis owner="sg_">
   <!--  stream schemas    -->
   <schema name="cube_init">
        <field name="clientnum" type="int" />
        <field name="source_ip" type="string" size="16" />
        <field name="player_name" type="string" size="32" />
        <field name="team_name" type="string" size="32" />
        <field name="lifesequence" type="int" />
        <field name="to_server_map" type="string" size="128" />
        <field name="next_mode" type="int" />
   </schema>
   <schema name="cube_disconnect">
        <field name="clientnum" type="int" />
        <field name="isbot" type="int" />
   </schema>
   <schema name="cube_position">
        <field name="clientnum" type="int" />
        <field name="position_x" type="single" />
        <field name="position_y" type="single" />
        <field name="position_z" type="single" />
        <field name="yaw" type="single" />
        <field name="pitch" type="single" />
        <field name="roll" type="single" />
        <field name="velocity_x" type="single" />
        <field name="velocity_y" type="single" />
        <field name="velocity_z" type="single" />
        <field name="rest_info" type="int" />
        <field name="isbot" type="int" />
   </schema>
   <schema name="cube_inventory_item">
        <field name="clientnum" type="int" />
        <field name="item_index" type="int" />
        <field name="total_items" type="int" />
   </schema>
   <schema name="cube_item_pickup">
        <field name="clientnum" type="int" />
        <field name="item_index" type="int" />
        <field name="spawn_secs" type="double" />
        <field name="client_pickup" type="int" />
   </schema>
   <schema name="cube_item_acc">
        <field name="clientnum" type="int" />
        <field name="item_index" type="int" />
   </schema>
   <schema name="cube_item_spawn">
        <field name="clientnum" type="int" />
        <field name="item_index" type="int" />
    </schema>
```

```
 <schema name="cube_map_change">
      <field name="clientnum" type="int" />
      <field name="to_server_map" type="string" size="128" />
      <field name="next_mode" type="int" />
</schema>
<schema name="cube_map_reload">
      <field name="clientnum" type="int" />
</schema>
<schema name="cube_mode">
      <field name="clientnum" type="int" />
      <field name="next_node" type="int" />
</schema>
<schema name="cube_timeup">
      <field name="time_remain" type="int" />
</schema>
<schema name="cube_shot">
      <field name="clientnum" type="int" />
      <field name="gun_number" type="int" />
      <field name="from_x" type="single" />
      <field name="from_y" type="single" />
      <field name="from_z" type="single" />
      <field name="to_x" type="single" />
      <field name="to_y" type="single" />
      <field name="to_z" type="single" />
</schema>
<schema name="cube_damage">
      <field name="clientnum" type="int" />
      <field name="target" type="int" />
      <field name="damage" type="int" />
      <field name="lifesequence" type="int" />
      <field name="client_isbot" type="int" />
      <field name="target_isbot" type="int" />
</schema>
<schema name="cube_frags">
      <field name="clientnum" type="int" />
      <field name="frags" type="int" />
      <field name="isbot" type="int" />
</schema>
<schema name="cube_death">
      <field name="clientnum" type="int" />
      <field name="client_isbot" type="int" />
      <field name="actor" type="int" />
      <field name="actor_isbot" type="int" />
</schema>
<schema name="cube_sound">
      <field name="clientnum" type="int" />
      <field name="playsound_n" type="int" />
      <field name="isbot" type="int" />
</schema>
<schema name="cube_chat">
      <field name="clientnum" type="int" />
      <field name="chat_text" type="string" size="1024" />
</schema>
```

```
<schema name="cube_server_msg">
	<field name="server_msg" type="string" size="1024" />
</schema>
<schema name="cube_ping">
	<field name="clientnum" type="int" />
	<field name="ping_type" type="int" />
	<field name="last_millis" type="int" />
</schema>
<schema name="cube_edit">
	<field name="edit_type" type="int" />
	<field name="amount" type="int" />
	<field name="lasttex" type="int" />
	<field name="selection_x" type="int" />
	<field name="selection_y" type="int" />
	<field name="selection_xs" type="int" />
	<field name="selection_ys" type="int" />
	<field name="selection_type" type="int" />
</schema>
<schema name="cube_edit_ent">
	<field name="entity_length" type="int" />
	<field name="entity_type" type="int" />
	<field name="entity_x" type="int" />
	<field name="entity_y" type="int" />
	<field name="entity_z" type="int" />
	<field name="entity_attribute1" type="int" />
	<field name="entity_attribute2" type="int" />
	<field name="entity_attribute3" type="int" />
	<field name="message" type="int" />
</schema>
<schema name="cube_add_bot">
	<field name="host_clientnum" type="int" />
	<field name="clientnum" type="int" />
	<field name="player_name" type="string" size="32" />
	<field name="team_name" type="string" size="32" />
	<field name="lifesequence" type="int" />
</schema>
<!-- table schemas -->
<schema name="cube_map_table">
	<field name="to_server_map" type="string" size="128" />
	<field name="game_mode" type="int" />
</schema>
<schema name="cube_items_table">
	<field name="item_index" type="int" />
	<field name="spawned" type="int" />
	<field name="spawn_secs" type="double" />
</schema>
<!-- input streams -->
  <input stream="player_inits" schema="cube_init" />
  <input stream="player_positions" schema="cube_position" />
  <input stream="player_items" schema="cube_inventory_item" />
  <input stream="item_pickups" schema="cube_item_pickup" />
  <input stream="player_shots" schema="cube_shot" />
  <input stream="player_damage" schema="cube_damage" />
```

```
    <input stream="player_frags" schema="cube_frags" />
    <input stream="player_deaths" schema="cube_death" />
    <input stream="map_change" schema="cube_map_change" />
    <input stream="mode_change" schema="cube_mode" />
    <input stream="sound" schema="cube_sound" />
    <input stream="chat" schema="cube_chat" />
    <input stream="player_ping" schema="cube_ping" />
    <input stream="world_edit" schema="cube_edit" />
    <input stream="entity_edit" schema="cube_edit_ent" />
<!-- bot streams -->
    <input stream="new_bots" schema="cube_add_bot" />
<!-- server streams -->
<!-- TODO: these should probably be outputs -->
    <input stream="client_disconnect" schema="cube_disconnect" />
    <input stream="reload_map" schema="cube_map_reload" />
    <input stream="map_timeup" schema="cube_timeup" />
    <input stream="server_msgs" schema="cube_server_msg" />
<!-- server generated output streams -->
    <output stream="item_spawns" schema="cube_item_spawn" />
<!-- main game query -->
<query name="game_query" node="128.148.38.64:15000">
    <table name="server_map_table" schema="cube_map_table">
        <key field="to_server_map" />
        <parameter name="truncate" value="0" />
        <parameter name="create" value="1" />
    </table>
    <table name="server_items_table" schema="cube_items_table">
        <key field="item_index" />
        <parameter name="truncate" value="0" />
        <parameter name="create" value="1" />
    </table>
    <!-- Game initialization logic -->
        <!-- handle sign on, get current valid map, output map to new player, notify other
        players of new player. -->
    <box name="check_clientnum" type="filter">
        <in stream="player_inits" />
        <out stream="invalid_clientnum" />
        <out stream="valid_clientnum" />
        <parameter name="expression.0" value="clientnum = -1" />
        <parameter name="pass-on-false-port" value="1" />
    </box>
    <box name="generate_clientnum" type="map">
        <in stream="invalid_clientnum" />
        <out stream="new_players" />
        <parameter name="output-field-name.0" value="clientnum" />
        <parameter name="expression.0" value="sequence()" />
        <parameter name="output-field-name.1" value="source_ip" />
        <parameter name="expression.1" value="source_ip" />
        <parameter name="output-field-name.2" value="player_name" />
        <parameter name="expression.2" value="player_name" />
        <parameter name="output-field-name.3" value="team_name" />
        <parameter name="expression.3" value="team_name" />
        <parameter name="output-field-name.4" value="lifesequence" />
```

```
        <parameter name="expression.4" value="lifesequence" />
        <parameter name="output-field-name.5" value="to_server_map" />
        <parameter name="expression.5" value="to_server_map" />
        <parameter name="output-field-name.6" value="next_mode" />
        <parameter name="expression.6" value="next_mode" />
</box>
<box name="new_player_map_select" type="select">
        <in stream="new_players" />
        <out stream="player_and_map" />
        <out stream="player_and_empty_map" />
        <table name="server_map_table" />
        <parameter name="pass-on-no-results" value="true" />
        <parameter name="sql" value="select clientnum, source_ip, player_name, team_name,
            lifesequence, sg_server_map_table.to_server_map, sg_server_map_table.game_mode from
            sg_server_map_table where sg_server_map_table.to_server_map != 'DUMMYSTRING'" />
</box>
<box name="set_map_from_player" type="insert">
        <in stream="player_and_empty_map" />
        <out stream="player_and_valid_map" />
        <table name="server_map_table" />
        <parameter name="pass-input" value="1" />
        <parameter name="sql" value="insert into sg_server_map_table
            values(to_server_map, next_mode)" />
  </box>
<box name="generate_valid_player" type="union">
        <in stream="player_and_map" />
        <in stream="player_and_valid_map" />
        <in stream="valid_clientnum" />
        <out stream="valid_player" />
</box>
<!--  Game map logic  -->
<!--  TODO: implement voting    -->
        <!--  handle map update for now only update if no valid map exists -->
<box name="change_map" type="select">
        <in stream="map_change" />
        <out stream="valid_map" />
        <out stream="empty_map" />
       <table name="server_map_table" />
        <parameter name="pass-on-no-results" value="true" />
        <parameter name="sql" value="select clientnum, sg_server_map_table.to_server_map,
            sg_server_map_table.game_mode from sg_server_map_table where
            sg_server_map_table.to_server_map != 'DUMMYSTRING'" />
</box>
<box name="set_valid_map" type="insert">
        <in stream="empty_map" />
        <out stream="inserted_map" />
        <table name="server_map_table" />
        <parameter name="pass-input" value="1" />
        <parameter name="sql" value="insert into sg_server_map_table
            values(to_server_map, next_mode)" />
</box>
<box name="union_valid_maps" type="union">
        <in stream="valid_map" />
```

```
        <in stream="inserted_map" />
        <out stream="new_map" />
    </box>
    <!-- Game entities logic -->
    <!-- add items from players to map -->
    <box name="check_items" type="select">
        <in stream="player_items" />
        <out stream="existing_items" />
        <out stream="new_items" />
        <table name="server_items_table" />
        <parameter name="pass-on-no-results" value="1" />
        <parameter name="sql" value="select sg_server_items_table.item_index,
            sg_server_items_table.spawned, sg_server_items_table.spawn_secs from
            sg_server_items_table where sg_server_items_table.item_index =
            input.item_index" />
    </box>
    <box name="add_item" type="insert">
        <in stream="new_items" />
        <table name="server_items_table" />
        <parameter name="sql" value="insert into sg_server_items_table values(item_index,
            1, 0.0)" />
    </box>
        <!-- handle item pickups + update items table + acknowledge pickup to one player +
        notify pickup to remaining players -->
    <!-- TODO: what about bot pickups? -->
    <box name="record_pickup" type="update">
        <in stream="item_pickups" />
        <out stream="successful_pickup" />
        <table name="server_items_table" />
        <parameter name="sql" value="update sg_server_items_table set sg_server_items_table.spawned =
            0, sg_server_items_table.spawn_secs = input.spawn_secs where
            sg_server_items_table.item_index = input.item_index and sg_server_items_table.spawned > 0"
            />
        <parameter name="output-field-name.0" value="clientnum" />
        <parameter name="output-expression.0" value="input.clientnum" />
        <parameter name="output-field-name.1" value="item_index" />
        <parameter name="output-expression.1" value="new.item_index" />
        <parameter name="output-field-name.2" value="spawn_secs" />
        <parameter name="output-expression.2" value="new.spawn_secs" />
      </box>
        <!-- TODO: redundant box, simply change item_acc tuple type to be the same as pickup,
        and match clientnums at the game client. -->
     <box name="acknowledge_pickup" type="map">
        <in stream="successful_pickup" />
        <out stream="ack_pickup" />
        <parameter name="output-field-name.0" value="clientnum" />
        <parameter name="expression.0" value="clientnum" />
        <parameter name="output-field-name.1" value="item_index" />
        <parameter name="expression.1" value="item_index" />
    </box>
        <!-- handle item spawns + update items table + notify players of item -->
        <!-- TODO: need a box to emit a tuple at a given time instance, ie after spawn secs.
```

```
        Timers are tricky because they need scheduler interaction to ensure they can after xxx period of
        time.Note: could syphon timestamps off other (high frequency) streams...This is essentially a special kind
        of join ... and implemented by a WaitForQBox  -->
<box name="respawn_control_gen" type="map">
      <in stream="player_positions" />
      <out stream="item_spawn_control" />
      <parameter name="output-field-name.0" value="current" />
      <parameter name="expression.0" value="now()" />
</box>
<box name="respawn_buffer_gen" type="map">
      <in stream="successful_pickup" />
      <out stream="new_pickup" />
      <parameter name="output-field-name.0" value="clientnum" />
      <parameter name="expression.0" value="clientnum" />
      <parameter name="output-field-name.1" value="item_index" />
      <parameter name="expression.1" value="item_index" />
      <parameter name="output-field-name.2" value="delay_until" />
      <parameter name="expression.2" value="now(spawn_secs)" />
</box>
<box name="detect_respawn" type="waitfor">
      <in stream="new_pickup" />
      <in stream="item_spawn_control" />
      <out stream="trigger_respawn" />
      <parameter name="predicate" value="buffered.delay_until < enabler.current" />
      <parameter name="timeout" value="300" />
  </box>
<box name="respawn_item" type="update">
      <in stream="trigger_respawn" />
      <out stream="item_spawns" />
      <table name="server_items_table" />
      <parameter name="sql" value="update sg_server_items_table set
          sg_server_items_table.spawned = 1, sg_server_items_table.spawn_secs = 0.0
          where sg_server_items_table.item_index = input.item_index" />
      <parameter name="output-field-name.0" value="clientnum" />
      <parameter name="output-expression.0" value="input.clientnum" />
      <parameter name="output-field-name.1" value="item_index" />
      <parameter name="output-expression.1" value="new.item_index" />
</box>
<!--  Introspection logic  -->
      <!-- handle client ping requests + pings generate pongs.  + clientpings are simply
      forwarded.  -->
<box name="split_pings" type="filter">
      <in stream="player_ping" />
      <out stream="ping" />
      <out stream="player_clientping" />
      <parameter name="expression.0" value="ping_type = 19" />
      <parameter name="expression.1" value="ping_type = 21" />
</box>
<box name="ping_pong" type="map">
      <in stream="ping" />
      <out stream="player_pong" />
      <parameter name="output-field-name.0" value="clientnum" />
      <parameter name="expression.0" value="clientnum" />
```

```
            <parameter name="output-field-name.1" value="ping_type" />
            <parameter name="expression.1" value="20" />
            <parameter name="output-field-name.2" value="last_millis" />
            <parameter name="expression.2" value="last_millis" />
        </box>
    </query>
</borealis>
```