Available online at www.sciencedirect.com
ScienceDirect
Procedia Computer Science 00 (2012) 1–10
Procedia Computer Science

International Conference on Computational Science, ICCS 2012

A programming model for spatio-temporal data streaming applications

Shigeru Imai[a,*], Carlos A. Varela[a]

[a] *Computer Science Department, Rensselaer Polytechnic Institute, 110 Eighth Street, Troy, NY 12180, USA*

Abstract

In this paper, we describe a programming model to enable reasoning about spatio-temporal data streams. A spatio-temporal data stream is one where each datum is related to a point in space and time. For example, sensors in a plane record airspeeds (v_a) during a given flight. Similarly, GPS units record an airplane's flight path over the ground including ground speeds (v_g) at different locations. An aircraft's airspeed and ground speed are related by the following mathematical formula: $v_g = \sqrt{v_a^2 + 2v_a \cdot v_w \cdot \cos(\alpha_a - \alpha_w) + v_w^2}$, where v_a and α_a are the aircraft airspeed and heading, and v_w and α_w are the wind speed and direction. Wind speeds and directions are typically forecast in 3,000-foot height intervals over discretely located fix points in 6-12 hour ranges. Modeling the relationship between these spatio-temporal data streams allows us to estimate with high probability the likelihood of sensor failures and consequent erroneous data. Tragic airplane accidents (such as Air France's Flight 447 on June 1st, 2009 killing all 216 passengers and 12 aircrew aboard) could have been avoided by giving pilots better information which can be derived from inferring stochastic knowledge about spatio-temporal data streams. This work is a first step in this direction.

Keywords: programming models, spatio-temporal data, data streaming

1. Introduction

Spatio-temporal data streams, where each datum is related to a point or a range of space and time, are pervasive. We see such data streams in many occasions in our daily lives, for instance, temperatures, prices of gasoline, flight schedules of airplanes, and so on. Massive amounts of spatio-temporal data are continuously generated by sensors, IT systems, or computer programs, and consumed by humans; however, today's most commonly used programming languages (*e.g.*, C/C++, Java, PHP, JavaScript, python, etc.) do not have first-class support for space and time since they are designed to be general-purpose. The downside of the general-purpose approach is the complexity and size of code. Since these programming languages are imperative, meaning that we have to use *for, if*, or *while* to control the flow of the programs and explicitly handle state, the code can get large and complex easily.

In contrast to the general-purpose approach, if we know a specific problem domain very well and want to provide first-class support for key domain concepts (such as space and time), we can take a domain-specific approach. The

*Corresponding author
Email addresses: imais@cs.rpi.edu (Shigeru Imai), cvarela@cs.rpi.edu (Carlos A. Varela)

code written in the domain-specific approach is much simpler and more declarative as in logic programming languages [1] and therefore simpler to write, read, and reason about, but it is generally less expressive, than code written in general-purpose programming languages.

Operating an aircraft is a complicated task since there are a lot of complex correlations between the instruments in a cockpit. If some failure happens during a flight, it is not easy to find the cause of the failure by looking at the available (potentially partially erroneous) data , and also, a misinterpretation of instrument readings could even lead to a tragic accident [2]. We illustrate our programming model with a flight planning system that reports potential sources of data problems such as mechanical failures or extreme weather conditions. An explicit mathematical model of data co-relationships can with high probability signal data errors, potentially providing pilots with better information in emergency scenarios, allowing them to take appropriate actions in a timely manner, and ultimately reaching their destination safely and efficiently.

In this paper, we present our initial effort on designing a programming model for spatio-temporal data streaming applications, aiming to apply the model to a flight planning system. The model provides first-class support for space and time specific operations including data selection and interpolation when no data is available for a certain location and time.

2. Motivation

Auto-pilots cannot take the right action when the data they are receiving is out of date or incorrect. This may have led to the tragic crash of Air France flight 447, killing all 216 passengers and 12 aircrew [2]. The records from the crash have suggested that the pilots lost control of the airplane because they raised the nose of the airplane when it should not have been brought up. Many experts now understand that the airplane went into clouds with thunderstorms and its iced speed sensors provided inaccurate information to the autopilot, causing it to disengage. The pilots then incorrectly reacted to the emergency by raising the nose of the plane when in fact it needed to go down to break the stall.

An active redundant data-driven flight system may help prevent crashes caused by sensor or other data errors. For example, by comparing the airspeed data to the ground speed data, a flight system would be able to fact check a bad airspeed reading, assuming reasonable constraints on the wind speed. If the pilot is only operating by airspeed data alone, they would have no way of knowing that there is an error in the system and they would respond to the incorrect data, upsetting the balance of the plane. The ground speed data would instead provide a fact checking mechanism because if airspeed were swiftly changing, ground speed would be doing the same. If airspeed is changing, but ground speed remains unchanged, the more active flight system would be able to notify the pilot of the discrepancy, allowing for better informed decision making.

Our goal is to develop a data streaming programming model that makes explicit the connections between different spatio-temporal data streams. Flight systems developed using our model would make explicit the redundancies in the data and allow the different data streams to essentially fact check each other greatly reducing the possibility of accidents. The proposed spatio-temporal data streaming programming model can also be applied to other domains generating space and time specific data.

3. Spatio-Temporal Data Streaming Programming Model

3.1. Programming Model

Our proposed programming model is designed for applications that handle spatio-temporal input data streams as shown in Figure 1. In this model, the application gets data $(d'_1, d'_2, \ldots, d'_n)$ from the *data selection* module, which takes incoming data streams $(d_1, d_2, \ldots, d_n)$ as inputs, and then the application indefinitely generates outputs $(o_1, o_2, \ldots, o_m)$ and data errors $(e_1, e_2, \ldots, e_l)$ based on an *application model*. Each input data stream $d_i(x, y, z, t)$ is a function of location and time. The number of arguments of d_i varies depending on dimensions of the location information, that is, $d_i(t)$ for 0-D (*i.e.*, no location support), $d_i(x, t)$ for 1-D, $d_i(x, y, t)$ for 2-D, and $d_i(x, y, z, t)$ for 3-D. Some data streams are coming in real-time whereas some predicted information (*e.g.*, weather forecasts) is associated with future time periods.

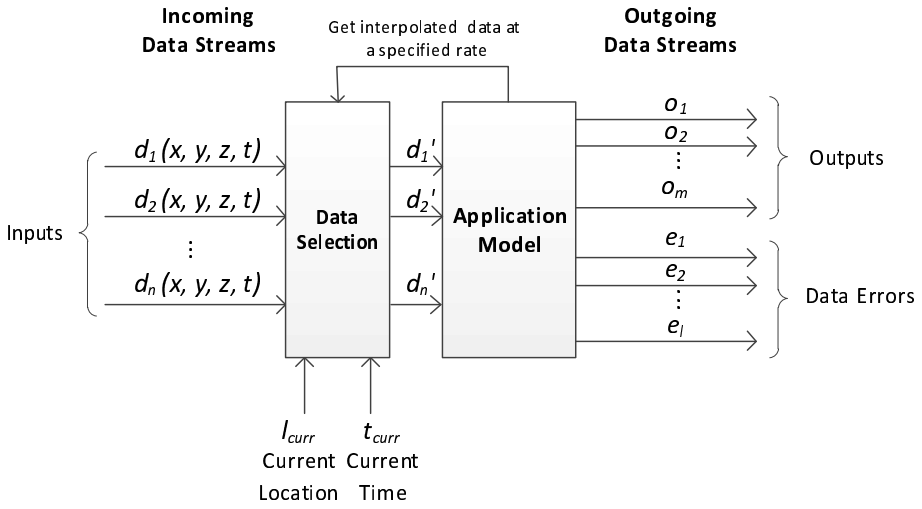

Figure 1: Programming model handling spatio-temporal input data streams

Table 1: An example of a weather forecast input data stream

Location $(lat_1, long_1) - (lat_2, long_2)$	Altitude	Time $(from) - (to)$	Chance of Icing
(42.73, -73.69)-(42.70, -73.66)	8000 ft.	01/30/2012:10:00-12:00 (GMT)	50%
(42.73, -73.69)-(42.70, -73.66)	8000 ft.	01/30/2012:12:00-14:00 (GMT)	60%
(42.73, -73.69)-(42.70, -73.66)	8000 ft.	01/30/2012:14:00-16:00 (GMT)	80%

Table 1 shows an example of a weather forecast input data stream coming to the Data Selection module. As noted from the table, the location information is given by regions where each region is represented by two horizontal locations (latitude, longitude) and an altitude, and the time information is given by time periods where each period is represented by two time points in GMT. Since not all the data is on the table, for example, chance of icing is not defined when the time is 01/30/2012:09:00 (GMT); however, we can define data by selecting or interpolating the existing data when no data is defined for a given location and time. In our programming model, the Data Selection module stores some amount of incoming data stream until it becomes out of date. The application acquires the selected or interpolated data $(d'_1, d'_2, \ldots, d'_n)$ from the Data Interpolation module at a certain rate specified in the application and computes both outputs and data errors. The application continues this computing process in an infinite loop until the user requests to stop the computation. The Data Selection module essentially allows an application to view a set of heterogeneous data streams as a homogeneous data stream, and therefore enables a separation of concerns: application programmers can focus on their application model.

3.2. Support for Spatio-Temporal Data Selection

We define two types of data selection and one data interpolation method for the location and time as shown below. These operations are applicable to either single variables (*i.e.* t, x, y, or z) or multiple variables (*i.e.* combinations of t, x, y, and z. By using these operations, application programmers can use locally related data even in the case when the given data is sparse.

- *closest*
  This method takes a 1-D argument (*i.e.,* t, x, y, or z) to find the data closest to a given location or time. Figure 2 shows examples of selecting closest data to the current time and location respectively. In Figure 2(a), when selecting the closest time to the current time t_{curr}, $d_i(t_{curr})$ is not defined, but $d_i(t)$ is defined for $\{t \mid t_1 \leq t \leq$

$t_2, t_3 \leq t \leq t_4, t_5 \leq t \leq t_6\}$. Since t_4 is closest to t_{curr}, we define $d'_i(t_{curr}) \triangleq d_i(t_4)$. Similarly, we define $d'_i(x_{curr}) \triangleq d_i(x_3)$ for the example shown in Figure 2(b).

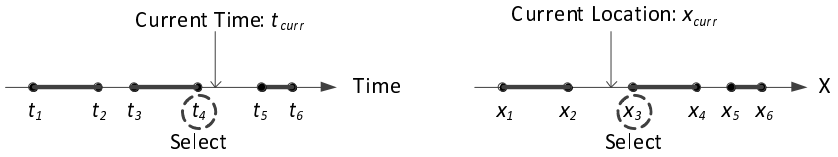

Figure 2: (a) Selecting the closest time; (b) Selecting the closest x value

- *euclidean*
  This method takes 2-D or 3-D arguments to find the data closest to a given location. Figure 3 shows an example for the 2-D case, where data is not defined for the current location $l_{curr} = (x_{curr}, y_{curr})$, but are defined for l_0, and l_1. Since l_{curr} is closest to $l_0 = (x_0, y_0)$ in Euclidean distance, we define $d'_i(x_{curr}, y_{curr}) \triangleq d_i(x_0, y_0)$.

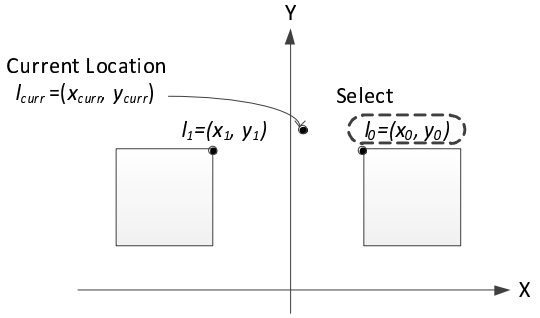

Figure 3: Selecting the closest 2D region in Euclidean distance

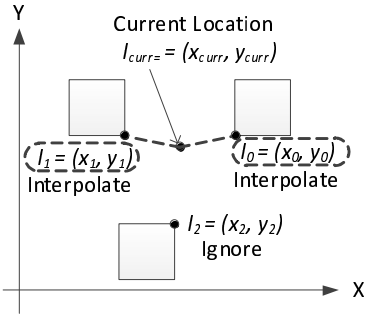

Figure 4: Linear interpolation

- *linear interpolation*
  This method takes 1-D, 2-D or 3-D arguments to interpolate the defined data. It also takes another argument n_{interp} to select closest n_{interp} data from a given location to interpolate. Suppose we have a situation shown in Figure 4, where data is not defined for the current location $l_{curr} = (x_{curr}, y_{curr})$, but are defined for l_0, l_1, and l_2. Also, suppose that $n_{interp} = 2$, we select l_0 and l_1 since they are closer to l_{curr} than l_2. In such a case, we linearly interpolate the data defined for l_0 and l_1 by taking a weighted sum based on the Euclidean distance as follows:

$$d'_i(x_{curr}, y_{curr}) \triangleq (1 - \frac{\|l_0 - l_{curr}\|}{\sum_{i=0}^{1} \|l_i - l_{curr}\|}) \cdot d_i(x_0, y_0) + (1 - \frac{\|l_1 - l_{curr}\|}{\sum_{i=0}^{1} \|l_i - l_{curr}\|}) \cdot d_i(x_1, y_1) \quad (1)$$

  Note that the equation (1) can be easily extended to n data points.

Multiple methods can be specified in the application program and they apply to the input data in order. If multiple data get selected by one method (*e.g.*, more than one closest point), a subsequent method takes that multiple data as the input and further select data. If there still remains more than one data after applying all the methods, then we implicitly apply linear interpolation to output the final value.

4. Spatio-Temporal Data Streaming Programming Language

In this section, we describe a spatio-temporal programming language by defining its grammar and showing two example programs.

4.1. Grammar Definition

A grammar definition for a declarative programming language following the proposed programming model is shown in Figure 5. A program (*Program*) consists of four parts: a program name, inputs, outputs, and errors. The program name is defined by a variable name (*Var*). The inputs can have multiple entries of *Input*, which is defined by one or more input variables (*Vars*), a dimension of the inputs (*Dim*), and a data selection method described in the previous section (*Method*). The outputs and errors are defined separately, but have the same output format (*Output*). Output is defined by one or more output variables (*Vars*), mathematical expressions (*Exps*), and a time interval to specify the frequency of the output (*Time*).

```
Program   ::=  program Var;
                      inputs Input*
                      outputs Output*
                      errors Output*
Input     ::=  Vars:  Dim using Methods;
Output    ::=  Vars:  Exps at every Time;
Dim       ::=  '(t)' | '(x,t)' | '(x,y,t)' | '(x,y,z,t)'
Methods   ::=  Method | Method, Methods
Method    ::=  (closest | euclidean | interpolate) '(' Exps ')'
Time      ::=  Number (nsec | usec | msec | sec | min | hour | day)
Exps      ::=  Exp | Exp, Exps
Exp       ::=  Func(Exps) | Exp Func Exp | '(' Exp ')' | Value
Func      =    { +, -, *, /, sqrt, sin, cos, tan, abs, ...}
Value     ::=  Number | Var
Number    ::=  Sign Digits | Sign Digits'.'Digits
Sign      ::=  '+' | '-' | ''
Digits    ::=  Digit | Digits Digit
Digit     =    { 0, 1, 2, ...,9 }
Vars      ::=  Var | Var, Vars
Var       =    { a, b, c, ...}
```

Figure 5: Spatio-temporal data streaming programming language grammar

4.2. Example Programs

4.2.1. Simple Example

Here we show one of the simplest programs implemented by the proposed programming model. It takes two input streams, $a(t)$ and $b(t)$, and outputs an error defined by $e = (b' - 2a')$ every 1 second as shown in Figure 6. Note that these two input streams are not associated to any location information.

An example specification of the above simple program is shown in Figure 7. In this example, there are two input data streams in which each stream is a function of time. The data selection method used in the program is specified by `closest(t)`, which means that the program instructs the Data selection module to select the closest t to the current time t_{curr}.

Errors behave differently depending on the input data streams, thus they tell us valuable information about the information sources. Figure 8 shows three different types of errors generated by simulations of the simple example program specification described in Figure 7. The assumption here is that every $1 \pm \epsilon$ seconds, the variable `a`'s input comes as $1\pm\epsilon, 2\pm\epsilon, 3\pm\epsilon, \ldots$ whereas the variable `b`'s input comes as $2\pm\epsilon, 4\pm\epsilon, 6\pm\epsilon, \ldots$, *i.e.*, they hold the mathematical relationship: `b = 2 * a`. In Figure 8(a), most of the time the error stays at zero, but there are several spikes due to transient fluctuation of the data input timing. It happens occasionally since the use of `closest(t)` causes the Data Selection module to select data at one second earlier or one second later than it is supposed to select. This type of

error is unavoidable without a special synchronization mechanism between multiple data streams. Figure 8(b) shows an example of the out-of-sync error. As shown in the graph, the error becomes consistently large at around 30 seconds of the simulation time. This is because the variable `a`'s input data stream becomes consistently one second behind the variable `b`'s input data stream. Figure 8(c) suggests more critical failure of the variable `a`'s input data source. At around 40 seconds of the simulation time, the error starts growing linearly. This linear increase of the error explains that the input data stream of the variable `a` stops coming after 40 seconds of the simulation time, which potentially means that a critical failure occurred at the source of the variable `a`.

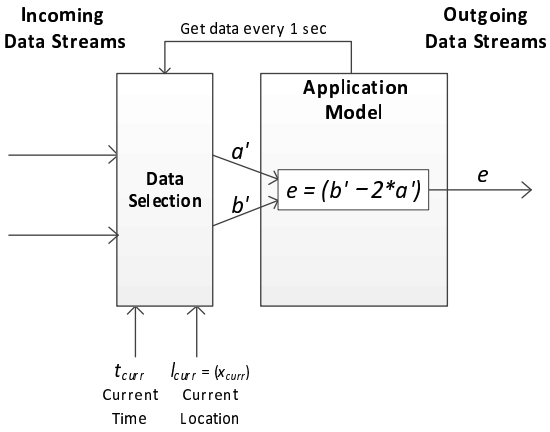

Figure 6: A simple application with temporal data streams

```
program twice;
    inputs
        a: (t) using closest(t);
        b: (t) using closest(t);
    outputs;
    errors
        e: (b - 2 * a)
            at every 1 sec;
end;
```

Figure 7: A simplest program specification

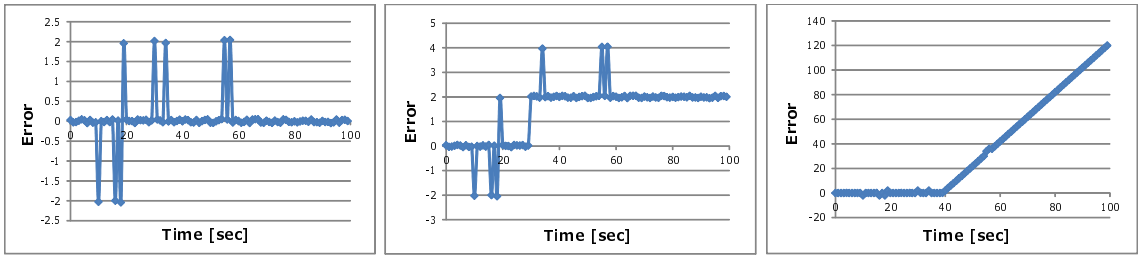

Figure 8: Examples of errors generated by a simple example: (a) Timing transient error; (b) Out-of-sync error; (c) Data input failure

4.2.2. *Flight Planning*

This is an example of a simplified flight planning system. Suppose that sensors in a plane record airspeeds v_a during a given flight and GPS units record the airplane's flight path over the ground including ground speeds v_g at different locations. An aircraft's airspeed and ground speed are related by the following mathematical formula: $v_g = \sqrt{v_a^2 + 2v_a \cdot v_w \cdot \cos(\alpha_a - \alpha_w) + v_w^2}$, where v_a and α_a are the aircraft airspeed and heading, and v_w and α_w are the wind speed and direction. Also, we can compute crosswind velocity: $v_x = v_w \cdot \sin(\alpha_a - \alpha_w)$. Therefore, given the aircraft desired course α_d, it is possible to compute the crab angle δ by using the formula (2) so that the aircraft can use $\alpha_a = \alpha_d + \delta$ as the heading to maintain the desired direction under varying wind conditions. The above mentioned relationship can be modeled as an application shown in Figure 9, which outputs the crab angle δ and error e that is the difference between the monitored ground speed v_g from GPS and the calculated one from v_a, v_w, α_a, and α_w.

$$\delta = \arcsin(v_x/v_g) = \arcsin\left(\frac{v_w \cdot \sin(\alpha_a - \alpha_w)}{\sqrt{v_a^2 + 2v_a \cdot v_w \cdot \cos(\alpha_a - \alpha_w) + v_w^2}}\right) \tag{2}$$

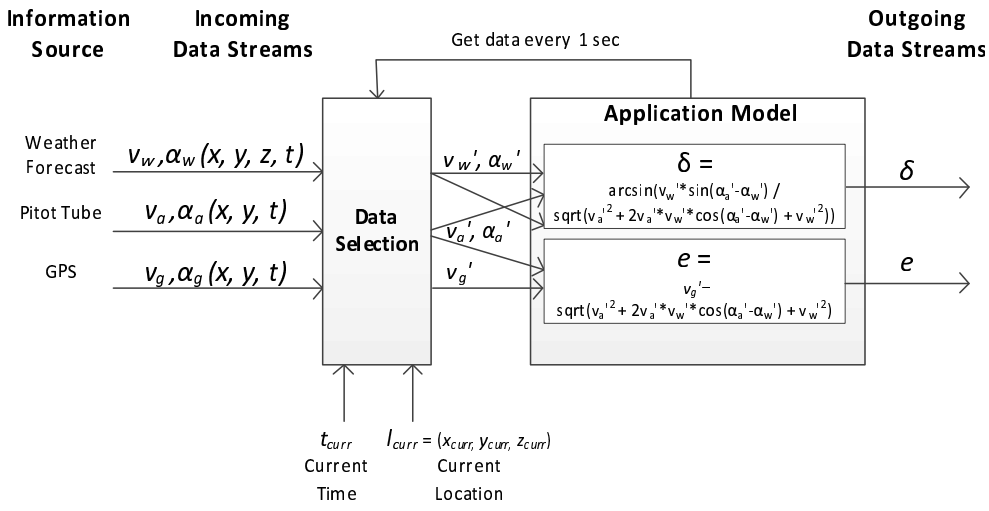

Figure 9: A flight planning application model using the spatio-temporal data streaming programming model

A code example of the flight planning application is shown in Figure 10. In this example, there are three input data streams in which each stream has two functions. Since each of these two functions has the same source of information and arguments, they are declared as a single input data stream. In the case of the first input data stream, there are two functions, `wind_speed(x,y,z,t)` and `wind_angle(x,y,z,t)`, that share the same arguments and information source (weather forecast). Data interpolation methods used for and `wind_speed(x,y,z,t)` and `wind_angle(x,y,z,t)` are specified by `euclidean(x,y)`, `closest(t)`, and `interpolate(z, 3)`. These methods apply in order: first, the closest x and y to x_{curr} and y_{curr} in Euclidean distance are selected; second, the closest t to the current time t_{curr} is selected; and finally, the final value is linearly interpolated on the z-axis using up to three closest data points to z_{curr} as specified in the argument.

```
program flightplan;
    inputs
        wind_speed, wind_angle: (x, y, z, t)
            using euclidean(x, y), closest(t), interpolate(z, 3);

        air_speed, air_angle: (x, y, z, t)
            using euclidean(x, y), closest(t);

        ground_speed, ground_angle: (x, y, z, t)
            using euclidean(x, y), closest(t);

    outputs
        crab_angle: arcsin(wind_speed * sin(wind_angle - air_angle) /
                           sqrt(air_speed^2 + 2 * air_speed * wind_speed *
                                cos(wind_angle - air_angle) + wind_speed^2))
            at every 1 sec;

    errors
        e: ground_speed - sqrt(air_speed^2 + 2 * air_speed * wind_speed *
                               cos(wind_angle - air_angle) + wind_speed^2))
            at every 1 sec;
end;
```

Figure 10: A declarative specification of the flight planning application

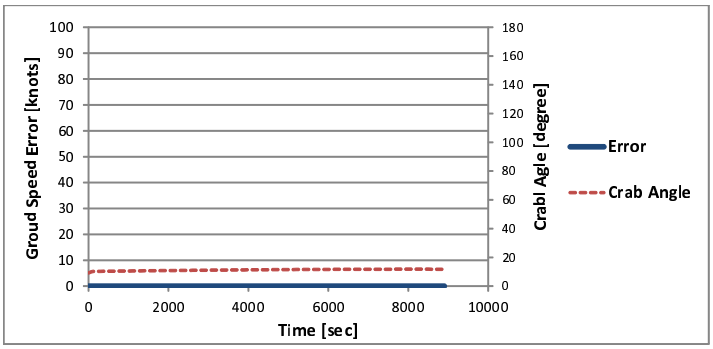

Figure 11: Normal conditions signature observed from a flight planning simulation

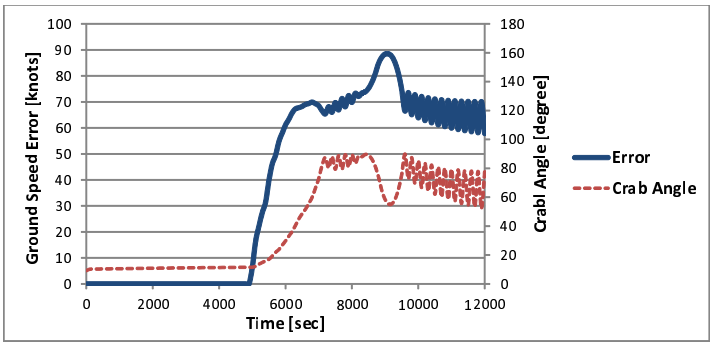

Figure 12: Pitot tube failure error signature observed from a flight planning simulation (the monitored airspeed starts decreasing at time = 5000 due to the pitot tube failure)

Example simulation results of the flight planning application generated from the program specification described in Figure 10 are shown in Figure 11 and Figure 12. In this simulation, a simple autopilot system navigates an airplane by using the crab angle received from the application. The airplane flies at 100 knots from Washington D.C. to Albany, which is 281 nautical miles (323 miles) away. The simulation also takes into account the effect of winds. In a normal condition, the autopilot successfully navigates the airplane to the destination with an ideal path as shown in Figure 11. The error of ground speed stays zero all through the simulation and the crab angle is almost constant (it actually slightly increases to adapt the wind speed changes). Figure 12 shows an error signature caused by a pitot tube failure. At time = 5000, a pitot tube starts icing and that causes the monitored airspeed to decrease gradually and gets almost zero eventually, while the airplane keeps flying at 100 knots. As seen from the graph, the autopilot's navigation does not work successfully this time. We can see a clear signature of the error here: the ground speed error grows quickly as soon as the airspeed start decreasing at time = 5000 and remains around 70 knots. The crab angle also grows up as the airspeed decreases.

As we can see from the simulation, a pilot can benefit from this application by following the crab angle to control the direction of the airplane, and also monitoring the error output to see if there is a mechanical failure of the instruments or the forecast information is wrong.

5. System Interaction

Figure 13 shows how a spatio-temporal application interacts with the system. The interactions are based on a client-server model using Internet sockets in which the application works as a server and takes inputs from the clients on a single port. It outputs some values to the specified output ports as well as error values to the specified error ports.

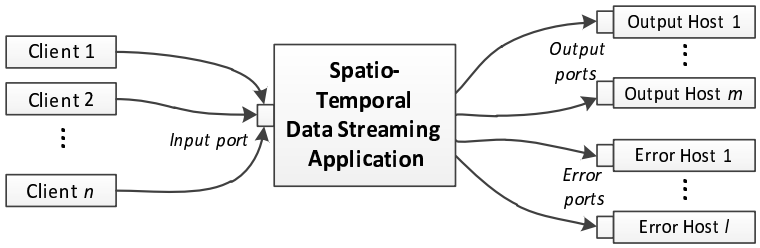

Figure 13: System interaction of a spatio-temporal data streaming application

When executing a binary object generated from the program specification, we specify input, output, and error ports as follows. In this example, the `flightplan` application illustrated in Section 4.2.2 takes an input data stream on the port `10001` and sends output and error streams to the hosts specified by `10.0.0.1:20001` and `10.0.0.2:30001` respectively.

```
$./flightplan -input 10001 -outputs 10.0.0.1:20001 -errors 10.0.0.2:30001
```

The input, output, and error data streams share the same format shown in Table 2. The first line is used to declare one or more variables (`var0, var1,...`) in a single data stream. The values of the declared variables start from the second line. The data stream can have multiple values (`value0, value1,...`) with various spatial and temporal combinations: ex1 is defined for a 3-D region and a time interval; ex2 is defined for a 2-D point and a time interval; ex3 is defined for a 1-D interval and a particular time; ex4 is defined for no location and a particular time. All lines have to end with an end-of-line marker (`\r\n`). Especially, the last line have to have only one end-of-line marker.

Note that the data format for input is compatible with output and error, that is, we can connect either an output or error port to an input port of another application.

Table 2: The data stream format of input, output, and error

first line	`#var0, var1,...\r\n`
after second line	ex1) `x0,y0,z0-x1,y1,z1:t0-t1:value0,value1,...\r\n`
	ex2) `x,y:t0-t1:value0,value1,...\r\n`
	ex3) `x0-x1:t:value0,value1,...\r\n`
	ex4) `:t:value0,value1,...\r\n`
last line	`\r\n`

6. Related Work

Spatio-temporal constraint logic programming has been proposed. STACLP [3] offers first-class support for representing and reasoning about spatial and temporal data. A similar logic language to STACLP, MuTACLP [4][5], is used to analyze geographical data especially for GIS (Geographical Information Systems). Both STACLP and MuTACLP are implemented based on a Prolog system. Programming languages that support probabilistic reasoning have also been proposed. PRISM[6] is a logic-based language that integrates logic programming and stochastic reasoning including parameter learning. PRISM is capable of parameter learning from a given set of data and estimates the probability to best explain the data. PRISM is also built on top of a Prolog system. Our proposed programming language is also highly declarative and it is to generate code that takes as inputs, spatio-temporal data streams.

There are programming languages for time-critical systems such as automatic control and monitoring systems. LUSTRE [7] [8], Giotto [9], and Esterel[10] are included in this category. These programming languages are declarative and designed to respond input events synchronously. Although their target systems are similar to ours, the main focus of these languages is real time behavior and there is no special support for spatial information nor reasoning capabilities.

7. Conclusions and Future Work

We present a programming model for spatio-temporal data streaming applications. In particular, it has first-class support for data selection and interpolation when no data is available for a given location and time. Towards our primary goal of applying this programming model to flight planning applications, we have several future research directions: 1) development of a compiler of the proposed language and a fully-functional application using real spatio-temporal data to demonstrate the advantage of the proposed programming model; 2) learning error signatures for common failures in aviation system, 3) adding the probability of data accuracy as inversely proportional to the spatio-temporal distance between the current location and time and the defined data points; 4) studying stochastic reasoning techniques and investigating the applicability of such techniques to spatio-temporal data streaming applications.

Acknowledgments

This research is partially supported by the Air Force Office of Scientific Research.

References

[1] J. W. Lloyd, Foundations of logic programming. Symbolic computation, Springer-Verlag, Berlin, Germany, 1984.
[2] Wikipedia, Air france flight 447, `http://en.wikipedia.org/wiki/Air_France_Flight_447`.
[3] A. Raffaet, T. W. Frhwirth, Spatio-temporal annotated constraint logic programming., in: PADL'01, 2001, pp. 259–273.
[4] P. Mancarella, G. Nerbini, A. Raffaet, F. Turini, MuTACLP: A language for declarative GIS analysis., in: Computational Logic'00, 2000, pp. 1002–1016.
[5] P. Baldan, P. Mancarella, A. Raffaet, F. Turini, MuTACLP: A language for temporal reasoning with multiple theories., in: Computational Logic: Logic Programming and Beyond'02, 2002, pp. 1–40.
[6] T. Sato, Y. Kameya, Parameter learning of logic programs for symbolic-statistical modeling, in: Journal of Artificial Intelligence Research(JAIR), Vol. 15, 2001, pp. 391–454.
[7] N. Halbwachs, P. Caspi, P. Raymond, D. Pilaud, The synchronous dataflow programming language LUSTRE, in: Proceedings of the IEEE, 1991, pp. 1305–1320.
[8] P. Caspi, D. Pilaud, N. Halbwachs, J. Plaice, Lustre: A declarative language for programming synchronous systems., in: POPL'87, 1987, pp. 178–188.
[9] T. A. Henzinger, B. Horowitz, C. M. Kirsch, Giotto: A time-triggered language for embedded programming., in: EMSOFT'01, 2001, pp. 166–184.
[10] G. Berry, The foundations of Esterel., in: Proof, Language, and Interaction'00, 2000, pp. 425–454.