# jSSTL - A Tool to Monitor Spatio-Temporal Properties*

Luca Bortolussi
DMG, University of Trieste, Italy. MOSI, Saarland University, Germany. ISTI-CNR, Pisa, Italy
luca@dmi.units.it

Michele Loreti
University of Florence, Italy
michele.loreti@unifi.it

Laura Nenzi
IMT, Lucca, Italy
laura.nenzi@imtlucca.it

## ABSTRACT

Controlling and designing spatio-temporal behaviours requires proper formal tools to describe such properties, and to monitor and verify whether, and how robustly, they are satisfied by a system. In this paper, we introduce jSSTL, a Java tool for the specification of *Signal Spatio-Temporal Logic* (SSTL) properties and their robust monitoring on spatio-temporal trajectories. The tool consists of a jSSTL API and a front-end, integrated in ECLIPSE. We describe in details the framework and the use of the plugin, exploiting a running example of a cholera outbreak.

**Keywords:** Monitoring, Spatio-temporal Logic, Signal Temporal Logic, Java Tool, Eclipse.

## 1. INTRODUCTION

With the advent of Internet-of-Things and the diffusion of many cyber-physical devices, the problem of designing and validating complex systems composed of many heterogeneous interacting entities, distributed in space, is getting more and more central. Examples include smart cities (e.g. bike sharing, intelligent transport), robot swarms, self-driving vehicles.

Model-based development (MBD) will play a central role in the design stage, but the ability to program and analyse efficiently such models has to be complemented with suitable languages to specify requirements, and with algorithmic methods to check them.

Modal logics are an obvious candidate in this respect, owing to their widespread use and the presence of many model checking tools. However, the fundamental role of space in several Cyber-Physical Systems (CPS) requires us to use a spatio-temporal logic, which is more expressive and complex to analyse than standard temporal logic languages.

Linear time spatio-temporal logics can also be monitored in real time, and may then be used to control and guide adaptation of these systems. Monitoring (namely verification of a property on a single system trace) can also be useful in the MBD process for statistical analysis of complex stochastic hybrid models of CPS. To this end, efficient monitoring tools and libraries for expressive spatio-temporal logics need to be developed and deployed.

*Signal Spatio-Temporal Logic* (SSTL) [13, 14] is a linear-time temporal logic, recently designed, suitable to describe behaviours of traces generated from simulations or measured from real systems. It spatially extends *Signal Temporal Logic* (STL) with a number of spatial modality that permit to specify metric and topological properties in a discrete space. In [13, 14], the authors provide a Boolean and quantitative semantics of SSTL, efficient monitoring algorithms for both semantics and interesting case studies, showing the usefulness of the logic.

In order to make this logic usable in practice, there is the need of an efficient and usable implementation. For this reason, we developed jSSTL, a Java-based tool for the specification and the verification of SSTL properties. The tool consists of a Java library (jSSTL API) and a front-end, integrated in ECLIPSE. Both the library and the ECLIPSE plugin can be downloaded from `http://quanticol.sourceforge.net/`. The source code is available at `https://bitbucket.org/LauraNenzi/jsstl`. The ECLIPSE plugin provides a user friendly interface to the tool, whereas the library can be used to integrate jSSTL within other applications and tools, as it has been done in [2]. Furthermore, the modular approach of the implementation allows to develop different front-end for jSSTL.

In this paper, we describe jSSTL and show its usage to analyse spatio-temporal behaviours of complex models. We will use a running example throughout paper. In particular, we will consider a model of a cholera outbreak, a prominent example of a waterborne disease [4, 12]. We presented this case study in [13] to describe some potentialities of the logic. We will use it here to introduce the type of models that we can consider, to give some examples of SSTL properties and to show the tool at work.

**The spread of a cholera infection.** The model describes the spread of a Cholera infection along the communities that live close to a river. The space is represented as a weighted graph, see Figure 1. The nodes $\ell_1$, ..., $\ell_7$ represent the communities; the edges describe the connection between the communities through the river. The graph is equipped with a weight function $w : E \to \mathbb{R}$ that returns the cost of each edge; for this model we interpret the cost $w(\ell_i, \ell_j)$ of an edge as the distance between the locations $\ell_i$ and $\ell_j$.

*Work partially funded by the EU-FET project QUANTICOL (nr. 600708) and FRA-UniTS.


VALUETOOLS 2016, October 25-28, Taormina, Italy

DOI 10.4108/eai.25-10-2016.2266978

There are two agent classes: the bacteria and the individuals. The bacteria have only one state ($B$) but they can be transported to different nodes via the river. An individual, instead, can be in three different states: susceptible ($S$) infected ($I$) and recovered ($R$), but cannot change location. The simplification is justified by the fact that we are not presenting a complete realistic model but just a simple example exploited later to illustrate the tool. We can easily extend the model to more complex scenarios as [12], for which we can specify and verify more complex spatio-temporal behaviours. Beside the inter-node movement of the bacteria, the dynamic of individuals is specified by natality and mortality and by transitions encoding the epidemic behaviour. The model of this system has state variables $\vec{X} = X_S, X_I, X_R, X_B$, counting the number of susceptible, infected, and recovered individuals, and the concentration of the bacteria in each location. The dynamics is specified by a list of transitions, each corresponding to an event and changes the system according to the transition rule (e.g,. $I \to R$ means an infected individual becomes recovered). Each transition has a speed governed by a rate function, that can depend on the concentration of the bacteria and individuals in each state . Below, we will list all the transitions (see [13] for a more detailed description of the model):

⋆ *Infection* : $S + B \to I + B$;
⋆ *Natality*: $\varnothing \to S$;
⋆ *Mortality* of a susceptible: $S \to \varnothing$;
⋆ *Mortality* of an infected: $I \to \varnothing$;
⋆ *Recovery*: $I \to R$
⋆ *Death* of bacteria: $B \to \varnothing$;
⋆ *Bacterial growth*: $\varnothing \to B$.

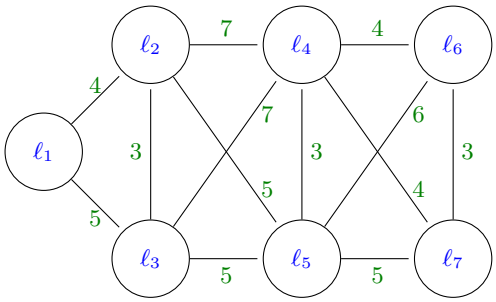


**Figure 1: Representation of the space of the epidemic model. The nodes $\ell_1$, ..., $\ell_7$ represent the different communities; the edges represent the link between the communities through the water basin. The green numbers correspond to the values $w$ of the weight function for each edge.**

For such a description, we can derive both a deterministic model (a set of ODEs) or a stochastic one (a patch CTMCs), that can respectively be integrated with an ODE solver or simulated with standard algorithms as SSA or Gibson-Bruck. The solution of the integration/simulation is a spatio-temporal trace $\vec{x}(t,\ell) = (x_S(t,\ell), x_I(t,\ell), x_R(t,\ell) x_B(t,\ell))$ that describes the concentration of susceptible, recovered, infected individuals and bacterias, at each time, in each location.

Such spatio-temporal trajectories are the input of our jSSTL tool. This means that SSTL specifies properties directly over the spatio-temporal traces of the system. An interesting consideration about this is that we do not necessarily need a mathematical model but just observable traces of some process, including real observations. Hence, the tool can be used also for the analysis of black box systems for which we do not have a model but only informations about their behaviour.

The rest of the paper is organised as follows. In Section 2, we introduce SSTL. The features of the jSSTL Plug-In and how they can be exploited to identify and analyse SSTL properties are described in Section 3. In Section 4, we report the architecture of the tool. Finally, in Section 5, we present some related works and we conclude with final remarks and directions for future works.

## 2. SSTL

*Signal Spatio Temporal Logic* (SSTL) [13, 14] is a spatial extension of STL [7, 11]. SSTL integrates the temporal modalities of STL with two spatial operators. The logic is interpreted on spatio-temporal signals with continuous time and discrete space.

**The space representation.** As we have seen in the example, we consider a discrete representation of space, modelled as a weighted graph, with populations of interacting components evolving in each location (node) and with entities migrating from one location to another one (connected via edges). Formally, a weighted graph is a tuple $G = (L, E, w)$ where $L$ is a set of nodes, $E$ is a set of edges and $w : E \to \mathbb{R}_{\geq 0}$ is the function that returns the cost/weight of each edge. The space is also equipped with a metric, i.e., a function $d : L \times L \to \mathbb{R}_{\geq 0}$ that returns a positive real value for each pair of elements in $L$; in particular, the metric is the *shortest weighted path distance*, i.e., the cost of the shortest path between two different locations. So, for example, in Figure 1, $d(\ell_1, \ell_5) = w(\ell_1, \ell_2) + w(\ell_2, \ell_5) = 9$.

**The signal.** A *spatio-temporal signal*, is a function $\vec{s} : \mathbb{T} \times L \to \mathbb{E}$, where $\mathbb{T}$ is a dense real interval, $L$ is a discrete set of locations and $\mathbb{E}$ is a subset of $\mathbb{R}^* = \mathbb{R} \bigcup \{+\infty, -\infty\}$. Signals with $\mathbb{E} = \mathbb{B} = \{0, 1\}$ are called *Boolean signals*, whereas those where $\mathbb{E} = \mathbb{R}^*$ are called real-valued or *quantitative signals*. A *spatio-temporal trace* is a function $\vec{x} : \mathbb{T} \times L \to \mathbb{D}$, s.t. $\vec{x}(t,\ell) := (x_1(t,\ell), \cdots, x_n(t,\ell)) \in \mathbb{D} \subseteq \mathbb{R}^n$, where each $x_i : \mathbb{T} \times L \to D_i \subseteq \mathbb{R}$, for $i = 1, ..., n$, is the projection on the $i^{th}$ coordinate/variable. Note that these projections have the form of quantitative signals. They are called the *primary signals* of the trace. We can thus see the trace as a set of primary signals. In particular, this means that SSTL can specify property of spatio-temporal traces.

**The syntax** of SSTL is given by

$$\varphi := \mu \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \mathcal{U}_{[t_1,t_2]} \varphi_2 \mid \ \diamondsuit_{[d_1,d_2]}\varphi \mid \varphi_1 \mathcal{S}_{[d_1,d_2]}\varphi_2.$$

where $\mu$ is an atomic predicate (inequalities of the form $\mu_j(x_1, \ldots, x_n) \equiv (f_j(x_1, \ldots, x_n) \geq 0)$, for $f_j : \mathbb{R}^n \to \mathbb{R}$), negation and conjunction are the standard Boolean connectives, and $\mathcal{U}_{[t_1,t_2]}$ is the temporal *until* operator, where $[t_1, t_2]$ is a real positive closed intervals with $t_1 < t_2$. The spatial operators are the *somewhere* operator, $\diamondsuit_{[d_1,d_2]}$, and the *bounded surround* operator $\mathcal{S}_{[d_1,d_2]}$, where $[d_1, d_2]$ is a closed real interval with $d_1 < d_2$. There are also three derivable operators: the *eventually* operator $\mathcal{F}_{[t_1,t_2]}\,\varphi := \mathbf{true}\mathcal{U}_{[t_1,t_2]}\varphi$, the *always* operator $\mathcal{G}_{[t_1,t_2]}\,\varphi := \neg\mathcal{F}_{[t_1,t_2]}\neg\varphi$ and the *everywhere* operator $\boxdot_{[d_1,d_2]}\,\varphi := \neg\ \diamondsuit_{[d_1,d_2]}\ \neg\,\varphi$. Temporal operators are like in standard metric temporal logic, while spatial operators will be described in the next subsection. For more details, we refer the reader to [13, 14]. Below, we give an idea of what the logic can specify giving some examples.

**Examples of SSTL properties.** The idea for the definition of the somewhere and everywhere operators came from the necessity to describe behaviours at a certain distance from a specific point, e.g., "from a bike sharing station, in a radius of 100 meters, there are more than 30 bikes" or "in all the positions around my location, at a

distance less the 1 km, there are no infected individuals". Let us see some more specific examples. Let us consider the epidemic model described in Section 1, with an infection that starts from location $\ell_1$. We want then describe the diffusion of the epidemic along the communities. In particular, we want to check whether the infection has propagated at a certain distance from $\ell_1$ after a certain time. This behaviour can be captured by the SSTL formula:

$$\varphi_1 = F_{[0,T_{inf}]} \diamondsuit_{[d_1,d_2]} (X_I > c_{inf}), \quad (1)$$

verifying it in location $\ell_1$. The exact meaning of the formula is: eventually, in less than $T_{inf}$ unit time, the number of infected individual becomes more than $c_{inf}$ in a location $\ell$ with $d(\ell_1, \ell) \in [d_1, d_2]$, i.e., at a distance from location $\ell_1$ equal or greater than $d_1$ and equal or less than $d_2$.

Let us see also a more complicated property:

$$\varphi_2 = \boxdot_{[0,d_{max}]}(\psi_1 \longrightarrow \psi_2), \quad (2)$$

where:

$$\psi_1 = F_{[0,T_{start}]}(\diamondsuit_{[0,d_{near}]}(X_I > c_{inf})))$$
$$\psi_2 = (F_{[T_{start},T_{start}+DT]} \diamondsuit_{[d_{far},d_{max}]} (X_I > c_{inf}))$$

The property stating that a large infection (with at least $c_{inf}$ individuals) happening at some time $t \in [0, T_{start}]$ and localised within distance $d_{near}$ from a given reference point, will spread further away, at a location at distance between $d_{far}$ and $d_{max}$, at some time $t' \in [T_{start}, T_{start} + DT]$. Furthermore, this is true for every reference point (say at distance at most $d_{max}$ from $\ell_1$).

The surround operator $\varphi_1 \mathcal{S}_{[d_1,d_2]} \varphi_2$, instead, expresses the topological notion of being surrounded by a $\varphi_2$-region, while being in a $\varphi_1$-region, with additional metric constraints. The idea is that one cannot escape from a $\varphi_1$-region without passing from a node that satisfies $\varphi_2$ and, in any case, one has to reach a $\varphi_2$-node at a distance between $d_1$ and $d_2$. For example, this operator permits to describe properties as "there are many infected individuals in my community but all the communities directly connected with me have a very low concentration of infected individuals". For more example of surround properties we refer the reader to [14].

**The semantics.** Similarly to STL, SSTL has two semantics, the classical *boolean* semantics and a *quantitative* semantics.
The boolean semantics returns true or false depending on whether the trace satisfies the SSTL property, i.e. $(\vec{x}, t, \ell) \models \varphi$ is true if and only if the trace $\vec{x}(t, \ell)$ satisfies $\varphi$. The whole trace satisfies a property in location $\ell$ iff it satisfies the property at time zero, i.e. $(\vec{x}, \ell) \models \varphi \Leftrightarrow (\vec{x}, 0, \ell) \models \varphi$. This because we are working with *future* temporal modalities that talk about true *now* as a function of some true in the future. Furthermore, the trace is verified in each point in space, as we assume no privilege direction or location.

The quantitative semantics, instead, returns a real value $\rho(\varphi, \vec{x}, t, \ell)$ that quantifies the level of satisfaction of the formula by the trajectory $\vec{x}$ at time $t$ in location $\ell$. The absolute value $|\rho(\varphi, \vec{x}, t, \ell)|$ can be interpreted as measure of the robustness of the satisfaction or dissatisfaction. Furthermore, the sign of $\rho(\varphi, \vec{x}, t, \ell)$ is related to the truth of the formula: if $\rho(\varphi, \vec{x}, t, \ell) > 0$, then $(\vec{x}, t, \ell) \models \varphi$, and similarly if $\rho(\varphi, \vec{x}, t, \ell) < 0$, then $(\vec{x}, t, \ell) \not\models \varphi$. In accordance with the boolean semantics, the quantitative value of the whole trace in location $\ell$ is given by its value at time zero, i.e. $\rho(\vec{x}, \ell) = \rho(\vec{x}, 0, \ell)$.

Given a spatio-temporal trace $\vec{x}(t, \ell)$ and a property $\varphi$, the logic has specific algorithms to compute the boolean and the quantitative signal of the formula. Each formula's parse tree is seen as a network of signal transducers, taking as input the primary signals of the trace at the leaves, processing them bottom up, and returning the satisfaction value at the root. For details about monitoring algorithms we refer the reader to [13, 14].

# 3. ECLIPSE PLUGIN

The ECLIPSE plugin provides a simple user interface to specify and verify SSTL properties of spatio-temporal trajectories generated from the simulation of a system or from real observations. We can specify the properties, describe a model of the space (i.e., its graph structure), import the trajectories and then verify if such trajectories satisfy the specified properties. The instructions to download the plugin and to create a jSSTL Project are reported in `http://quanticol.sourceforge.net/`.

In Figure 2, we can see a snapshot of the ECLIPSE plugin. It provides an *editor*, containing the script with the SSTL properties that we want to analyse in our scenario (on the left) and a *view* to visualise the space model, the data and the results of the analyses (on the right).

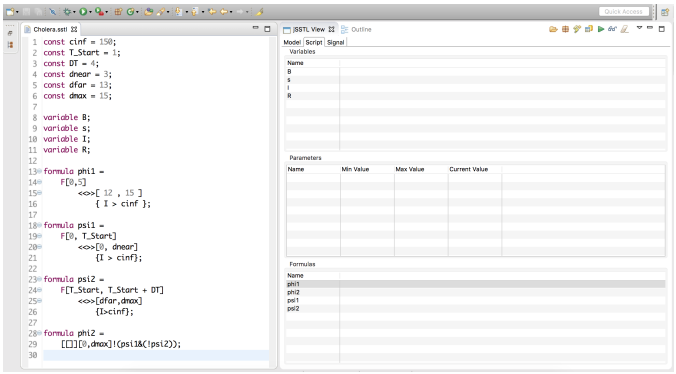


**Figure 2: The jSSTL ECLIPSE plugin**

In the **jSSTL editor**, it is possible to define a script that contains the list of properties that we want to analyse using jSSTL. The editor is based on Xtext[1], a framework for development of programming languages and domain-specific languages. The syntax of the script of our language is reported in Figure 3. Besides the list of formulae, each script contains the list of the variables considered in the model, a set of constants, and a list of parameters that may occur in the formula. The parameters can be declared to take values in an interval. When the monitoring procedure is performed, the user can select a specific value for each parameter in the corresponding interval. Standard expression can be used to define both constant and parameter intervals. In the script, each formula is associated with a name, that is used to select the specific property during the monitoring procedure.

We can see, on the left of Figure 2, the file Cholera.sstl containing the script of properties (1) and (2), described in the previous section. First, we have the declaration of the value of each parameter (const cinf= 150, ...), then the declaration of the variables (variable I,..), and finally the description of each property, e.g., property (1) is defined as: formula phi1 = F[0,5]«»[12,15] { I > cinf }.

As we can see in Figure 2 (where the panel with the Script is selected), the **jSSTL view** provides three different panels: to visu-

[1] `https://eclipse.org/Xtext/`

| | | |
|---|---|---|
| script | ::= | element* |
| element | ::= | variableDec \| constDec \| parameterDec \| formulaDec |
| variableDec | ::= | **variable** *name* |
| constDec | ::= | **const** *name* = expr |
| parameterDec | ::= | **parameter** *name* **in** interval |
| interval | ::= | [expr, expr] |
| expr | ::= | baseExpr \| expr + expr \| expr × expr \| ⋯ |
| baseExpr | ::= | int \| float \| literalExpr |
| formulaDec | ::= | **formula** *name* = formula |
| formula | ::= | formula**&**formula \| formula\|formula |
| | \| | formula**U**interval formula \| **G**interval formula |
| | \| | **F**interval formula \| formula**S**interval formula |
| | \| | **<<>>** interval formula \| **[[]]**interval formula |
| | \| | **!**formula \| relExpr |
| relExpr | ::= | expr<expr \| expr≤expr \| expr>expr \| expr≥expr |
| | \| | expr==expr \| ! =expr |

**Figure 3: jSSTL formula syntax.**

alise the spatial models (*Model* panel), to summarise the relevant data declared in a script (*Script* panel), and to plot the system's trajectories and the Boolean and quantitative satisfaction signals (*Signal* panel). The icons on the right top of the jSSTL view serve to: open a graph model, create a grid model, open an SSTL script, open a signal data (the trajectory), execute the monitor, display the signal and clear the data.

The *Model* panel can be used to see a graph-based representation of the spatial model. The spatial model can be can be imported as a .tra file or, in case of regular grid, can be directly created, selecting the number of rows and columns. In Figure 4, we can see, in the model panel, the representation of the space for the Cholera model that corresponds to the graph reported in Figure 1.

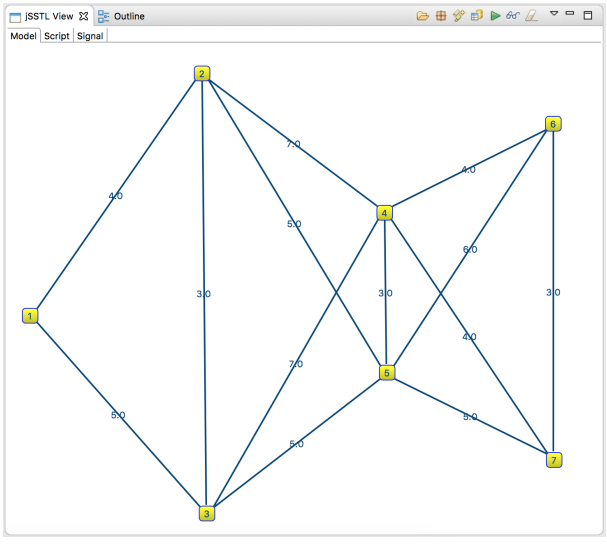


**Figure 4: Visualisation of the space in the jSSTL ECLIPSE plugin.**

The *Script* panel outlines the main informations of our scenario. We can see here the list of variables, parameters and formulae names. Currently, CSV and tabular based ASCII files are supported for both input and output of signals. The traces have to be imported with a single file for each variable and location, e.g., for the infected individual $I$ we will have the files values_$i$_$I$.dat, with $i \in \{1, .., 7\}$. Having selected the space model, the trajectory and the property, we can compute the related Boolean and quantitative satisfaction signals. To do that, we have to select from the Script panel which property we want to verify, and to chose the specific values for the parameters for which we considered a range.

In the *Signal* panel, the spatio-temporal trajectories and the Boolean and quantitative signals can be visualised. We can choose which variables and which locations to plot. In Figure 5, we can see the trace $x_I$, the variation in time of the number of infected individuals in each location. We can observe as, at time 0, we have infected individuals only in $\ell_1$. The temporal trajectories of each location are indicated adding $@i$ to each variable, e.g., $x_I(t, \ell_1)$ is indicated in the plot as I@1.

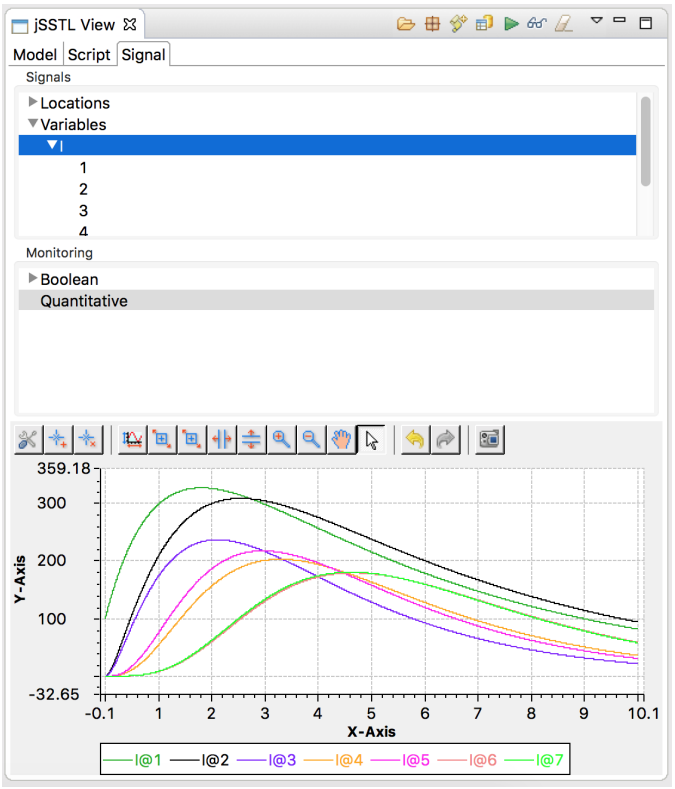


**Figure 5: Plot of $x_I$, number of infected individual in each location.**

Figure 6 shows Boolean satisfaction of the property psi1 (in each location). Instead, in Figure 7, we can see the quantitative satisfaction of the only location $\ell_1$ of the properties phi1 and psi1. We can see from the plot that the trajectories satisfy both properties, but that property psi1 is much robust than property phi1 . The different length of the signals is due to the different temporal intervals in the two formulas. Indeed, in the monitoring procedure, if the signal has duration $T$, and a formula has temporal horizon $h$ (i.e. its truth depends on what happens in $h$ units of time in the future), then the duration of the Boolean/ satisfaction signal will be $T - h$.

In any case, we remind that the satisfaction of a formula for the whole trace is determined at time zero and for a specific location; hence, we can say that the $\bar{x}$ satisfies $\varphi_1$ in location $\ell_1$ iff phi1@1 at time 0 is greater than 0.

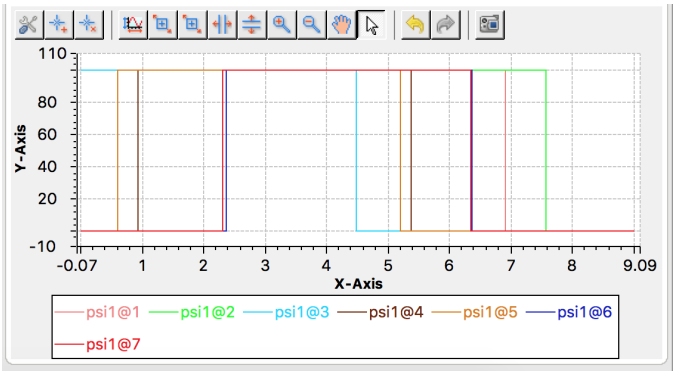


**Figure 6: Plot of the Boolean semantics of the properties psi1.**

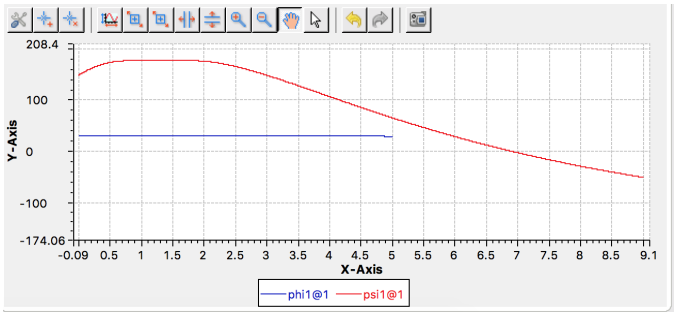


**Figure 7: Plot of the quantitative semantics of the properties phi1 and psi1 for location 1.**

Both the plots of the trajectories of Boolean and quantitative signals can be easily exported. Furthermore, the tool permits also to import a set of trajectories, instead of a single one, to compute the Boolean and quantitative satisfaction signals of such trajectories for a specific property and then to evaluate their mean (the average robustness) and their variance. This can be very useful when we want to analyse stochastic systems.

We stress that the plug in and the library work with much more complex models, like the one in [14], than the one presented here, which has been kept simple for the sake of clarity.

# 4. JSSTL JAVA LIBRARY

Our library, available at `https://bitbucket.org/LauraNenzi/jsstl`, has been designed in a modular way, which allows us to change/improve specific classes without changing the overall behaviour. It consists of three main packages: util, core, and io (Figure 8).

**Package util** handles the temporal signals of the logic. Its class BooleanSignal defines the temporal Boolean signals of the logic. They are represented as a sequence of time intervals with value true or false. The class BooleanSignalTransducer implements a method to compute the interval covering of a set of Boolean signals, a method to convert piecewise constant signals into Boolean signals and the methods to perform operations between temporal signals:

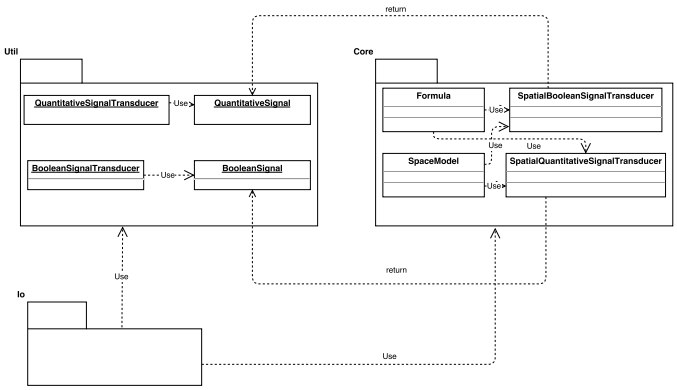


**Figure 8: jSSTL: UML Package Diagram.**

and, or, not, until, eventually, always. The class QuantitativeSignal defines the temporal quantitative signals of the logic, represented as piecewise constant signals. The class QuantitativeSignalTransducer implements the same methods as the Boolean case, for the operations between quantitative signals.

**Package core** is divided in three subpackages: space, monitor and formula.
In the space subpackage, the class SpaceModel is an interface that represents a generic space model. It is parametrised with respect to the type of data that represents the nodes and the edges. It provides three methods to obtain informations about nodes, edges and the external border of the model. The GraphModel class implements the interface as a space model based on a graph, and contains the methods to design, work, and have informations about the graph. It exploits the JGraphT[2] package to compute the matrix distance of the graph.

The monitor package handles the spatial Boolean and quantitative signals of the logic. They are treated as an HashMap that associates a temporal (Boolean or quantitative) signal to each location of the graph. The classes SpatialBooleanSignalTransducer and SpatialQuantitativeSignalTransducer contains all the methods to make operations between spatio-temporal signals: surround, somewhere, everywhere, and, or, not , until, always, eventually.

The third subpackage, formula, provides the classes used to represent SSTL formulae. These classes mimic the *abstract syntax tree* of formulas. The base interface is the class formula, it contains the methods booleanCheck and quantitativeCheck that compute the spatial Boolean and quantitative signals, taking as input the GraphModel and the Signal. The Signal class contains the method to translate the data, i.e., the primary signals, in a spatial Boolean or quantitative signal. Each operator has a specific class formula parametrised over one or two sub-formulae (depending on whether the operator is unary or binary) and an interval for some operators. The methods in each subformula refer to the methods in the SignalTransducer class of the specific operator. For example, the SurroundFormula class has the method booleanCheck that returns the monitoring evaluation SpatialBooleanSignalTransducer.surround of the signals of the two subformulae for the given (spatial) interval. Hence, the Monitoring algorithm is implemented following the *visitor pattern*. It is performed via a visit of a formula

[2] `http://jgrapht.org`

that implements a bottom-up evaluation. It is important to remark that the use of this pattern simplifies the integration of possible alternative monitoring algorithms.

**Package io** provides a set of classes that can be used to read graph models and input signals from an input stream and to write monitoring results to an output stream. Currently, CSV and tabular based ascii files are supported for both input and output of signals. Specific interfaces are also provided to simplify the integration of new specific input/output data formats.

The library can be easily integrated in other Java project, as we have done in [2], or used inside Matlab, exploiting Xtext for the specification of the property.

# 5. DISCUSSION

**Related Works.** There already exist useful tools to specify and monitor behaviours of systems with a temporal dynamics, as Breach [6] for STL properties or S-TaLiRo [1] for MTL specifications.The situation is very different, instead, when spatio-temporal dynamics are considering.

In [5], a *Spatial Logic for Closure Spaces* (STLCS) is proposed for a discrete and topological notion of space, based on closure spaces [9]. It is a spatial extension of the branching logic CTL, where the temporal aspects are considering as "snapshot" models. The logic cannot be used to monitor properties of spatio-temporal trajectories and, furthermore, it does not have a quantitative semantics. The related model-checker can be found at `https://github.com/vincenzoml/topochecker`.

Mudi [15], available at `http://mudi.modelchecking.org/`, is a spatio-temporal model checker employed for the automatic validation of complex biological systems. Their consider only regular discrete spaces, in particular a pseudo-3D representation of space (2D space plus a density measure). The properties are specified using the *Probabilistic Bounded Linear Spatial Temporal Logic* (PBLSTL) that presents only a boolean semantics. The essential difference between their approach and ours is that their logic considers space properties only as atomic predicates, i.e., the logic restricts the spatial specification to only the number of different predicates that have been designed, it does not have spatial operators that can be nested between them and with temporal ones. This allows us to express more complex spatio-temporal properties.

The only linear-time spatio-temporal logic with a robust monitoring procedure for checking properties directly over spatio-temporal trajectories that we are aware of is *Spatial-Temporal Logic* (SpaTeL) [10], in which spatial properties are expressed using ideas from image processing, namely *quad trees* [8]. This allows one to capture very complex spatial structures, but at the price of a complex formulation of spatial properties, which are in practice only learned from some template image. To our knowledge, a proper tool for the logic is not yet available.

**Conclusions and Future Works.** In this paper, we presented a Java-based tool to specify spatio-temporal properties in the linear time spatio-temporal logic SSTL, and to monitor trajectories coming from simulations or from measurements of real systems. The tool can be used stand-alone, via an ECLIPSE plug-in interface, or as a Java library. We stress here few features of the tool.

⋆ It can handle traces coming from different sources, hence it can be used to monitor simulations but also real systems.

⋆ The parametrisation of formulae enables to integrate the tool to perform parameter synthesis and estimation, see for instance [2].

⋆ The possibility to directly check a set of trajectories can simplify the statistical analysis of stochastic systems.

⋆ It has a quantitative semantics that can be used to evaluate the robustness of the satisfaction and to drive the behaviour of the system, see for instance [3].

The plugin is still in development and new features will be integrated. For instance, we are working on new kinds of formats and sources for input and output of data, and on a better integration with `Matlab`. We are also developing better monitoring algorithms and extending the logic with new operators, implementing them in the tool. In particular, we are developing an online monitoring procedure where trajectories are collected from external data sources, and a semantics to incorporate uncertainty in measurements.

# 6. REFERENCES


[1] Y. Annpureddy, C. Liu, G. Fainekos, and S. Sankaranarayanan. S-TaLiRo: A Tool for Temporal Logic Falsification for Hybrid Systems. *In Proc. of TACAS*, 2011.

[2] E. Bartocci, L. Bortolussi, D. Milios, L. Nenzi, and G. Sanguinetti. Studying Emergent Behaviours in Morphogenesis Using Signal Spatio-Temporal Logic. In *Proc. of HSB*, 2015.

[3] E. Bartocci, L. Bortolussi, L. Nenzi, and G. Sanguinetti. System design of stochastic models using robustness of temporal properties. *Theoretical Computer Science*, 2015.

[4] E. Bertuzzo, S. Azaele, A. Maritan, M. Gatto, I. Rodriguez-Iturbe, and A. Rinaldo. On the space-time evolution of a cholera epidemic. *Water Resources Research*, 2008.

[5] V. Ciancia, S. Gilmore, G. Grilletti, D. Latella, M. Loreti, and M. Massink. An Experimental Spatio-Temporal Model Checker. In *SEFM*, 2015.

[6] A. Donzé. Breach, A Toolbox for Verification and Parameter Synthesis of Hybrid Systems. In *Proc. of CAV*, 2010.

[7] A. Donzé and O. Maler. Robust satisfaction of temporal logic over real-valued signals. In *Proc. of FORMATS*, 2010.

[8] R. A. Finkel and J. L. Bentley. Quad trees a data structure for retrieval on composite keys. *Acta Informatica*, 1974.

[9] A. Galton. The mereotopology of discrete space. In *Spatial Information Theory. Cognitive and Computational Foundations of Geographic Information Science*, 1999.

[10] I. Haghighi, A. Jones, J. Z. Kong, E. Bartocci, Grosu R., and C. Belta. SpaTeL: A Novel Spatial-Temporal Logic and Its Applications to Networked Systems. In *Proc. of HSCC*, 2015.

[11] O. Maler and D. Nickovic. Monitoring temporal properties of continuous signals. In *Proc. FORMATS*, 2004.

[12] L. Mari, E. Bertuzzo, L. Righetto, R. Casagrandi, M. Gatto, I. Rodriguez-Iturbe, and A. Rinaldo. Modelling cholera epidemics: the role of waterways, human mobility and sanitation. *Journal of The Royal Society Interface*, 2012.

[13] L. Nenzi and L. Bortolussi. Specifying and monitoring properties of stochastic spatio-temporal systems in signal temporal logic. In *Proc. of VALUETOOLS*, 2014.

[14] L. Nenzi, L. Bortolussi, V. Ciancia, M. Loreti, and M. Massink. Qualitative and Quantitative Monitoring of Spatio-Temporal Properties. In *Proc. of RV*, 2015.

[15] O. Pârvu and D. Gilbert. Automatic validation of computational models using pseudo-3d spatio-temporal model checking. *BMC Systems Biology*, 2014.