Software Design Specification
Phyloloc

Nicolás Bombau Carlos Castro Damián Domé
Emmanuel Teisaire

21 de marzo de 2011

Índice

1. Introduction

1.1. Document Purpose

The purpose of the current document is the software design specification for the product "Phyloloc".

1.2. Document Audience

The intended audience for the current document are the people involved in the thesis and future contributors.

1.3. Document Organization Overview

Section 2 mentions the objectives of the current document, the design methodology adopted and enumerates design dependencies.

Section 3 introduces the system's architecture, and its main components and interactions.

Section 4 details the system's high level design, presenting the system's packages and interfaces.

Section 5 specifies the system's low level design, where concrete classes and class diagrams are presented, together with dynamic diagrams, that shall illustrate the system's main processes.

2. Design Considerations

2.1. Objectives

The objective in regard to the software design, is the development of a product that presents a concise, mantainable and extensible object oriented design.

2.2. Design Methodology

The design methodology adopted is "Responsibility Driven Design". This technique focuses on *what* responsibilities must be covered by the system, and which objects shall carry those responsibilities.

2.3. Dependencies

The product shall use Qt for the GUI, and Spirit for text processing in order to work properly. On the other hand, an interface shall be defined, so that the system consumes the methods of the interface, regardless of its implementation, so that the external libraries can be easily changed.

2.4. Tools

- **UML** as modelling language.
- **ARGOUML:** as modelling tool.
- **Enterprise Architect:** as modelling tool.
- **Gcov and LCov:** to calculate code coverage.
- **CCCC:** to calculate code metrics.
- **Doxygen:** to document code.

2.5. Conventions

2.5.1. Interfaces

Interfaces shall be called as their default implementation class name, adding a character "I" at the beginning. For example, the interface of the *Person* class, shall be *IPerson*.

3. System Architecture

In this section is discussed and defined the architecture of the system. In Figure 1 is presented a diagram containing the system's architecture, and the interaction between its components. A brief description of each component is presented below:

3.1. Domain

Responsibility: Define the domain model.

3.2. Phylopp

Responsibility: Provide generic primitives for phylogenetic trees, such as Traversals and Tree Consense.

1. Domain

3.3. Phyloloc

Responsibility: Implement the product-specific algorithms that take part in the propagation process.

Collaborators:

1. Domain
2. Phylopp

3.4. PhyloGUI

Responsibility: Let the user visualize and process phylogenetic trees

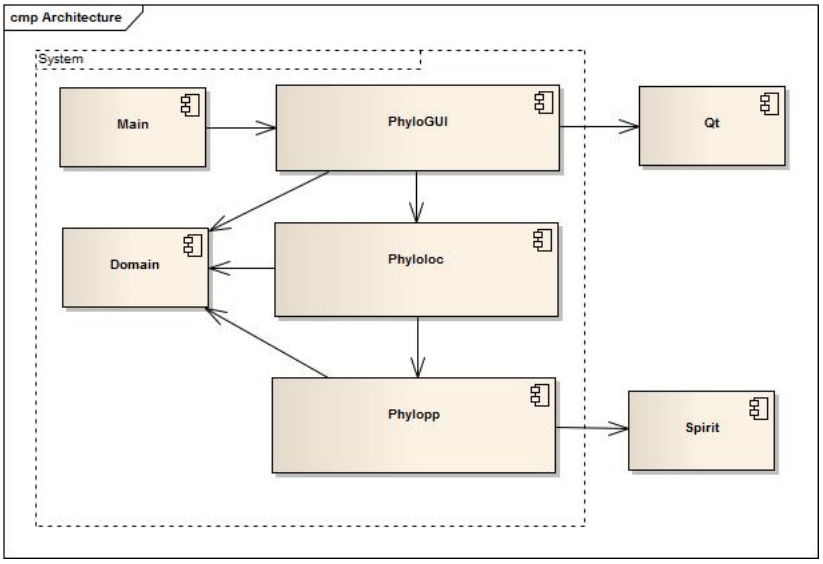

Figura 1: UML - Arquitectura

Collaborators:

1. Domain
2. Phyloloc

3.5. Main

Responsibility: Run initialization and configuration tasks.

Collaborators:

1. PhyloGUI

4. High Level Design

4.1. Packages

The packages detailed below were identified by analyzing the responsibilities of each component, and afterwards grouping those responsibilities in packages.

In Figure 2 is presented a diagram containing the packages and relations between them.

4.1.1. Domain

Responsibility: Represent the objects of the domain model, such as nodes, trees and groups of phylogenetic trees.

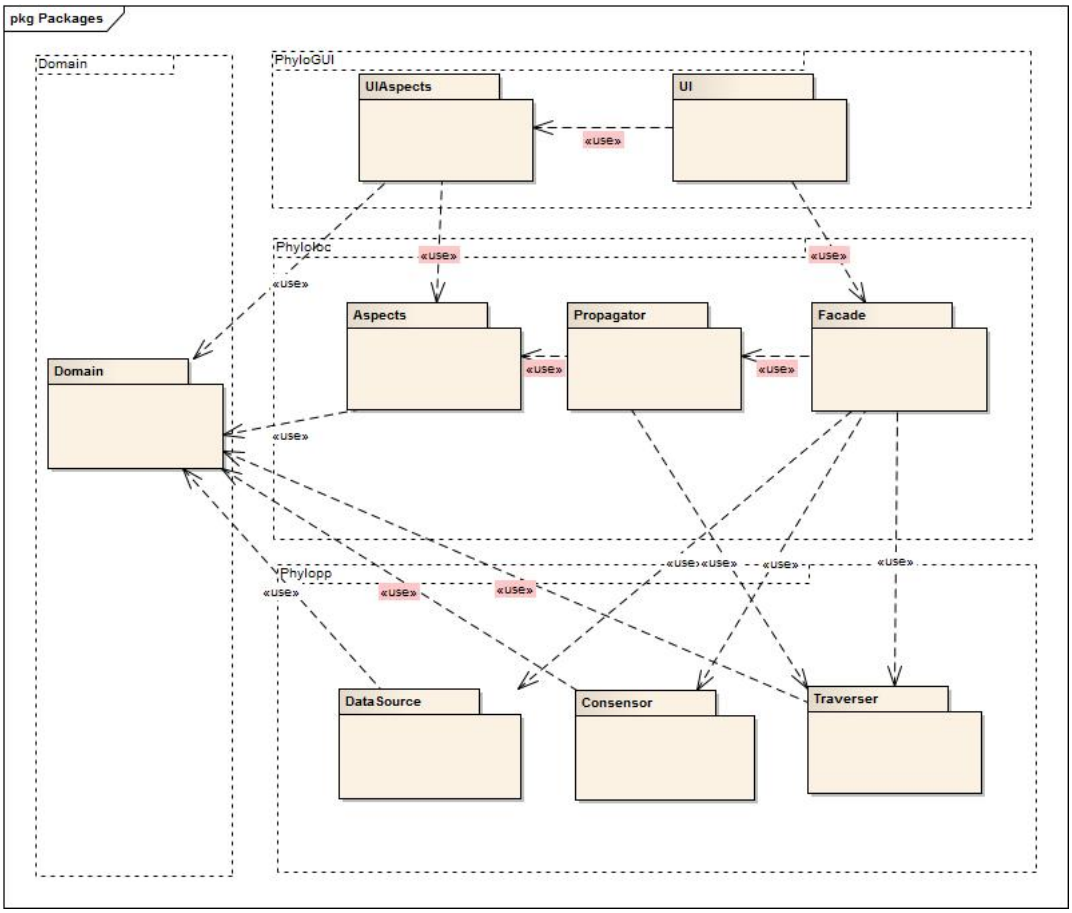

Figura 2: UML - Paquetes

4.1.2. Consensor

Responsibility: Consense a collection of phylogenetic trees, obtaining a single tree. The Consense algorithm to implement must be a Strict Consense algorithm.

Collaborators:

1. Domain

4.1.3. Traverser

Responsibility: Provide simple ways to traverse phylogenetic trees from tips to root and from root to tips.

Collaborators:

1. Domain

4.1.4. DataSource

Responsibility: Import and export phylogenetic trees.

Collaborators:

1. Domain

4.1.5. Propagator

Responsibility: Propagate probabilities through a phylogenetic tree in order to integrate topological and geographical data.

Collaborators:

1. Traverser

Responsibility: Provide application-specific aspects for a node, like propagation and consense information.

Collaborators:

1. Domain

4.1.6. Facade

Responsibility: Provide a common interface or set of interfaces in order to simplify the integration with other systems or user interfaces.

Collaborators:

1. Propagator
2. Consensor
3. Traverser

4.1.7. UI

Responsibility: Control the data to be displayed and perform the requested operations.

Collaborators:

1. Facade

4.1.8. UIAspects

Responsibility: Define aspects for a node, in order to hold information about presentational properties, such as color and selection state.

Collaborators:

1. Aspects

4.1.9. Aspects

Responsibility: Provide application-specific aspects for a node, like propagation and consense information.

Collaborators:

1. Domain

4.2. Interfaces, Responsibilities and Collaborations

In this section shall be presented the system's main interfaces, describing its responsibilities and collaborators.

4.2.1. IIterator

Responsibility: Provide interface to iterate through collections.

4.2.2. INode

Responsibility: Hold topological information and basic data associated to a node of a phylogenetic tree.

Collaborators:

1. IIterator

4.2.3. INodeFactory

Responsibility: Create node with certain definite aspects

Collaborators:

1. INode

4.2.4. ITree

Responsibility: Represent a phylogenetic tree.

Collaborators:

1. INode
2. IIterator

4.2.5. ITreeCollection

Responsibility: Represent a collection of phylogenetic trees.

Collaborators:

1. INode
2. IIterator

4.2.6. IDataSourceInfo

Responsibility: Provide the information needed in order to access a particular data source.

4.2.7. IDataSource

Responsibility: Export and import phylogenetic trees.

Collaborators:

1. IDataSourceInfo
2. ITreeCollection

4.2.8. INodeVisitor

Responsibility: Encapsulate an action to be executed on a node of a phylogenetic tree.

4.2.9. ITraverser

Responsibility: Provide simple way to traverse phylogenetic trees down and up.

Collaborators:

1. INodeVisitor
2. ITree
3. INode

4.2.10. IConsensorObserver

Responsibility: Observe the consense process, and start actions depending on the consense event, for example, when a node is supressed, or consensed.

4.2.11. IConsensor

Responsibility: Consense a collection of phylogenetic trees, resulting in a single tree.

Collaborators:

1. IConsensorObserver
2. ITreeCollection

4.2.12. IPropagator

Responsibility: Propagate probabilities through a phylogenetic tree iteratively up and down. The iteration shall be done a definite number of times, or until some explicit convergence criteria is reached.

Collaborators:

1. ITraverser

5. Low Level Design

5.1. Aspects

There are several sets of data that a node should contain, depending on the context. For example, when a tree exists as a result of a consensus operation, each node holds information specific to the consense process. Like the previous example, there are other groups of information that a node should hold, listed below:

- Color
- Selection
- Collapse
- Consensus
- Propagation

Given this situation, one possible solution regarding class design would be to have a base node class in phylopp, a phyloloc node that should inherit from phylopp's base node, and finally a GUI node that holds presentational information. This approach has the main drawback that, for instance, if in the GUI, if a node with presentational aspects is needed, the GUI programmer is forced to hold information from the lower layers, but perhaps that information is not needed at the moment.

The solution proposed in the current design is to use an AOP approach, so that the different pieces of information that a node must hold are grouped in aspects. This way, there may exist a node that has GUI and consesnsus information, but doesn't have propagation inforrmation.

5.2. Type Restrictions

The design presented in the current document makes use of three different type parameters. The restrictions for these type parameters are listed below:

5.2.1. T

T must implement INode interface. T is intended represent a node with an arbitrary quantity of composed aspects.

5.2.2. K

K should be a class that holds the information necessary to access a particular data source. For example, if the data source is a file, then K should be String containing its path. On the other hand, if the data source is a database, K should be some class that holds the address of the database server, credentials, etc.

5.2.3. V

V must implement INodeVisitor interface. V is the type parameter of the kind of visitor that shall be used.

5.3. Classes

In this section are presented the concrete classes that implement the interfaces identified in section 4.

5.3.1. Domain

In Figure 3 is presented a diagram containing the classes for the package Domain.

5.3.2. DataSource

In Figure 3 is presented a diagram containing the classes for the package DataSource.

5.3.3. Consensor

In Figure 5 is presented a diagram containing the classes for the package Consensor.

5.3.4. Traverser

In Figure 6 is presented a diagram containing the classes for the package Traverser.

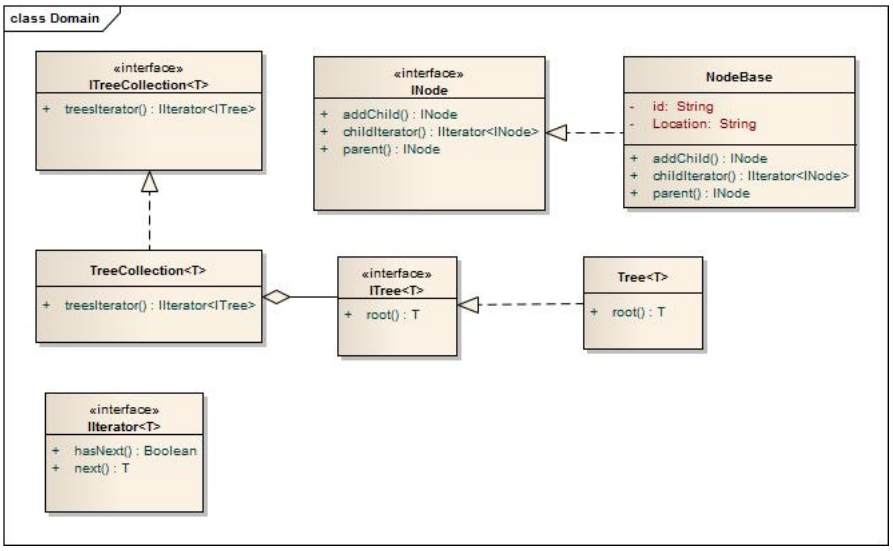

Figura 3: UML - Domain

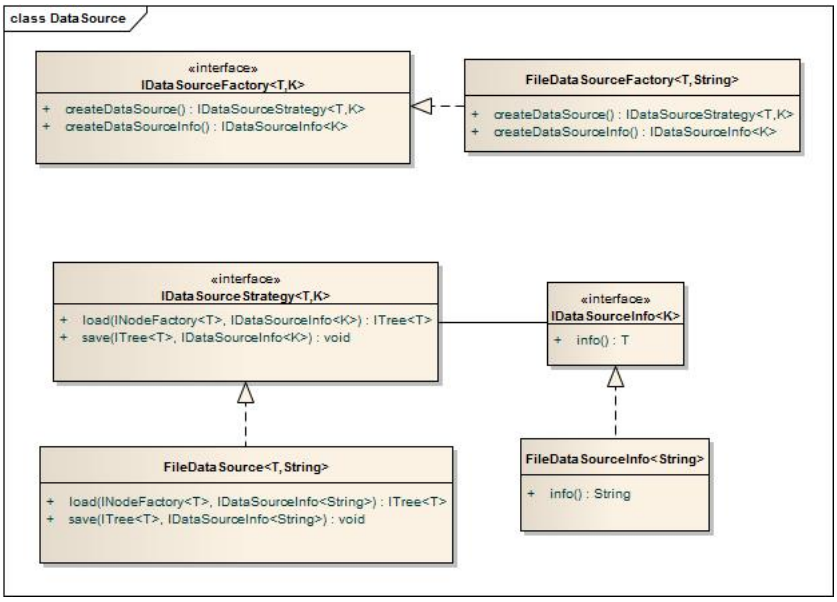

Figura 4: UML - DataSource

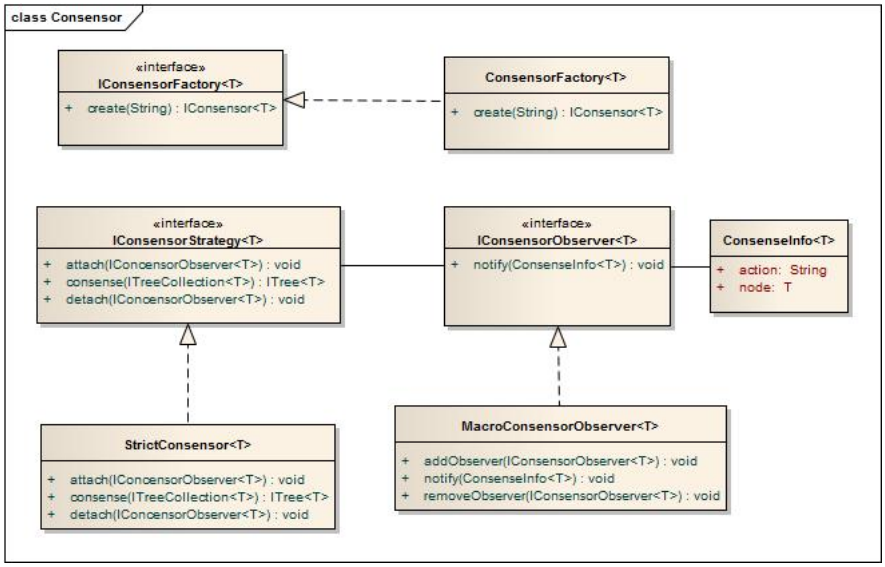

Figura 5: UML - Consensor

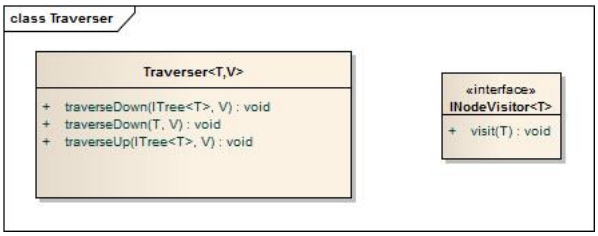

Figura 6: UML - Traverser

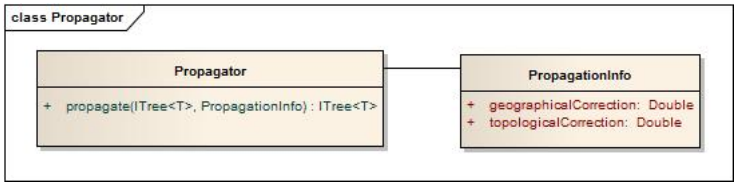

Figura 7: UML - Propagator

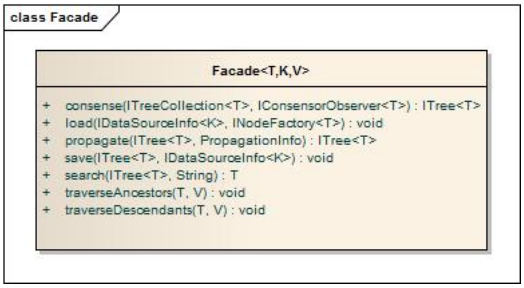

Figura 8: UML - Facade

5.3.5. Propagator

In Figure 7 is presented a diagram containing the classes for the package Propagator.

5.3.6. Facade

In Figure 8 is presented a diagram containing the classes for the package Facade.

5.4. Dynamic Diagrams

5.4.1. Descendants Selection

5.4.2. Tree Consense

5.4.3. Tree Import

5.4.4. Tree Propagation

Referencias

[1] Design Principles and Design Patterns. Robert C. Martin, 2000. `www.objectmentor.com`

[2] Modern C++ Design. Andrei Alexandrescu

[3] The C++ Programming Languaje. Bjarne Stroustrup.

[4] Rebecca Wirfs-Brock and Alan McKean. Object Design: Roles, Responsibilities and Collaborations, Addison-Wesley, 2003

[5] Unified Modeling Language: `http://www.uml.org/`

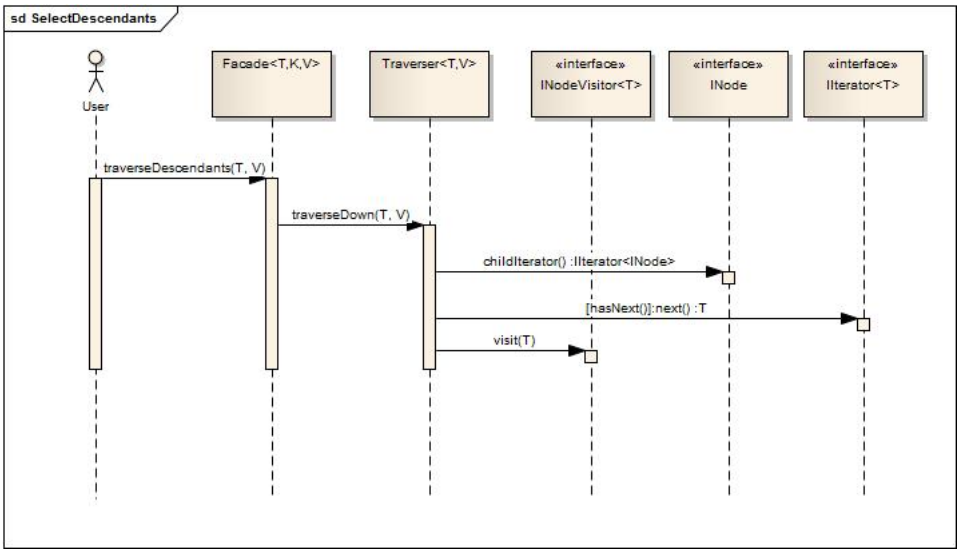

Figura 9: UML - Descendants Selection

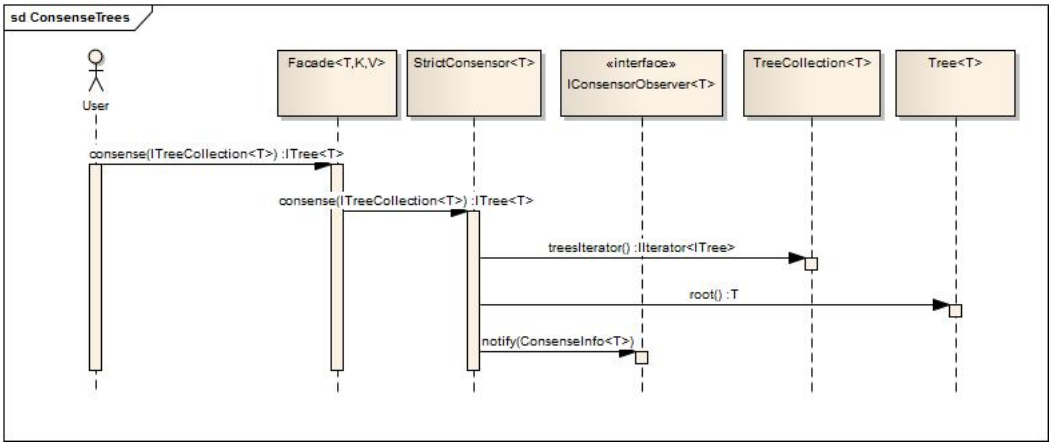

Figura 10: UML - Tree Consense

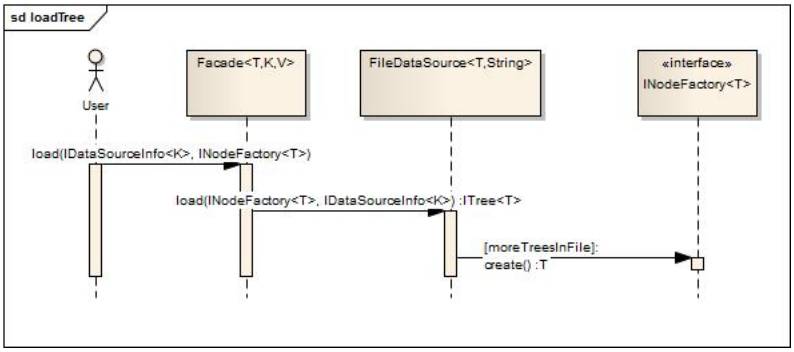

Figura 11: UML - Tree Import

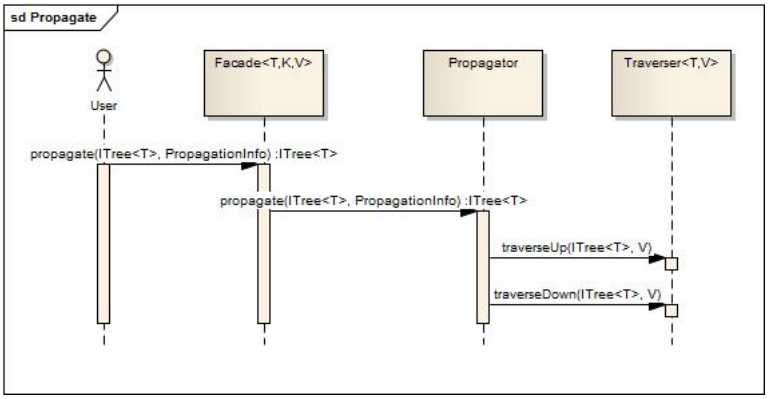

Figura 12: UML - Tree Propagation

[6] ArgoUML: `http://argouml.tigris.org/`