

Especificación de Requerimientos de Software

Santiago Videla

24 de septiembre de 2010

Índice

1. Introducción	3
1.1. Propósito	3
1.2. Alcance	3
1.3. Descripción general del documento	3
2. Descripción General	4
2.1. Perspectiva del Producto	4
2.1.1. Interfaces del Sistema	4
2.1.2. Interfaces de Usuario	4
2.1.3. Interfaces de Hardware	4
2.1.4. Interfaces de Software	6
2.1.5. Interfaces de Comunicaciones	6
2.1.6. Restricciones de Memoria	6
2.1.7. Operaciones	6
2.1.8. Requerimientos de Instalación	6
2.2. Funciones del Producto	6
2.2.1. Configuración inicial	6
2.2.2. Configuración de vac-o	6
2.2.3. Generación de secuencias	7
2.2.4. Evaluación de secuencias	7
2.3. Características de Usuarios	7
2.4. Restricciones	7
2.5. Suposiciones y Dependencias	8
2.6. Trabajo Futuro	8
3. Requerimientos	8
3.1. Funciones del Sistema	8
3.1.1. Configuración inicial	8
3.1.2. Configuración de vac-o	8
3.1.3. Generación de secuencias	10
3.1.4. Evaluación de secuencias	10
3.2. Restricciones de Rendimiento	11
3.3. Base de Datos	11
3.4. Restricciones de Diseño	11
3.4.1. Cumplimiento de Estándares	11
3.5. Atributos del Software	11
Apendices	12
A. Definiciones, Acrónimos y Abreviaturas	13
B. Referencias	14

1. Introducción

1.1. Propósito

El propósito de este documento es la especificación de requerimientos de software en el marco de la tesis de grado de la carrera Lic. en Cs. de la Computación de la FaMAF - UNC, “**Diseño de vacunas atenuadas con menor probabilidad de sufrir reversión a la virulencia**”. Los requerimientos del software son provistos por integrantes de FuDePAN en su carácter de autores intelectuales de la solución que se pretende implementar y colaboradores de dicha tesis.

A continuación se enumeran las personas involucradas en el desarrollo de la tesis y que por lo tanto, representan la principal audiencia del presente documento.

- Dra. Laura Alonso Alemany: Directora de tesis, FaMAF
- Daniel Gutson: Colaborador de tesis, FuDePAN
- Santiago Videla: Tesista, FaMAF

1.2. Alcance

El producto que se especifica en este documento se denomina “**vac-o**” y su principal objetivo es encontrar secuencias genómicas de patógenos que mantengan funcionalidad como vacunas con una probabilidad mínima de sufrir reversión a la virulencia. El producto final debe proveer al usuario, la capacidad de hacer uso del software mediante extensiones o *plugins* que implementen las características particulares para cada patógeno.

La principal responsabilidad de vac-o es la de proveer a las extensiones, un motor combinatorio de secuencias genómicas utilizando diferentes estrategias computacionales que conduzcan a la optimización de los cálculos. Por otro lado, las extensiones son responsables de evaluar las secuencias generadas asignando un puntaje a cada una con el fin de que vac-o genere el listado de secuencias resultante.

En su versión inicial, vac-o incluye una extensión para la vacuna Sabin contra la poliomielitis que podría ser tomada como guía para futuras extensiones.

1.3. Descripción general del documento

La estructura de este documento sigue las recomendaciones de la “Guía para la especificación de requerimientos de la IEEE” (IEEE Std 830-1998).

En la sección 2 se presenta una descripción general de vac-o, sus principales funcionalidades, interfaces y perfiles de usuarios.

En la sección 3 se detallan los requerimientos funcionales específicos de vac-o y los principales atributos que debe cumplir el software.

2. Descripción General

2.1. Perspectiva del Producto

Al momento de la confección de este documento, no existen productos de software que brinden las funcionalidades que se pretenden implementar. En este sentido, vac-o representa una innovación en el área de la bioinformática, cuyo principal objetivo es la optimización de vacunas basadas en virus atenuados.

Uno de los problemas de estas vacunas es su potencialidad de revertir a la virulencia. En sucesivas replicaciones, los virus atenuados pueden acumular mutaciones que les devuelvan el carácter patogénico y de esa manera podrían producir enfermedad en el vacunado o sus contactos.

Para lograr su objetivo, vac-o debe ser capaz de encontrar las variantes del virus con menor probabilidad de revertir a la patogenia. Partiendo de la secuencia genómica que se encuentra en la cepa vacunal, se deben buscar las modificaciones de esta secuencia que cumplan las siguientes propiedades:

- Que originen una respuesta inmune protectora contra el virus salvaje.
- Que tengan muy baja probabilidad de mutar hasta convertirse en un virus patógeno.

Entre todas las secuencias encontradas, vac-o debe construir una lista o *ranking* de secuencias con sus respectivos puntajes. Para lograr esto, se deberá implementar una extensión que brinde la información y las características propias del virus. En particular, cada extensión debe especificar las regiones combinatorias de interés y ser capaz de asignar un puntaje a cada secuencia comparándola con la secuencia original.

En la Figura 1 se presenta el flujo de trabajo y funcionamiento general de vac-o. Notar que este esquema es puramente conceptual y no hace ninguna referencia a los detalles de implementación. La intención es clarificar al lector los diferentes “actores” y sus responsabilidades.

2.1.1. Interfaces del Sistema

No se registran requerimientos.

2.1.2. Interfaces de Usuario

La interfaz con el usuario final consiste de dos formularios y un listado con las secuencias resultantes. Inicialmente, el usuario de vac-o deberá indicar la ubicación o *path* de la extensión que desea usar. El siguiente paso, será completar un formulario con los datos requeridos por la extensión elegida. Finalmente, se da comienzo a la ejecución y se recibe como resultado, la lista de secuencias con sus respectivos puntajes.

2.1.3. Interfaces de Hardware

No se registran requerimientos.

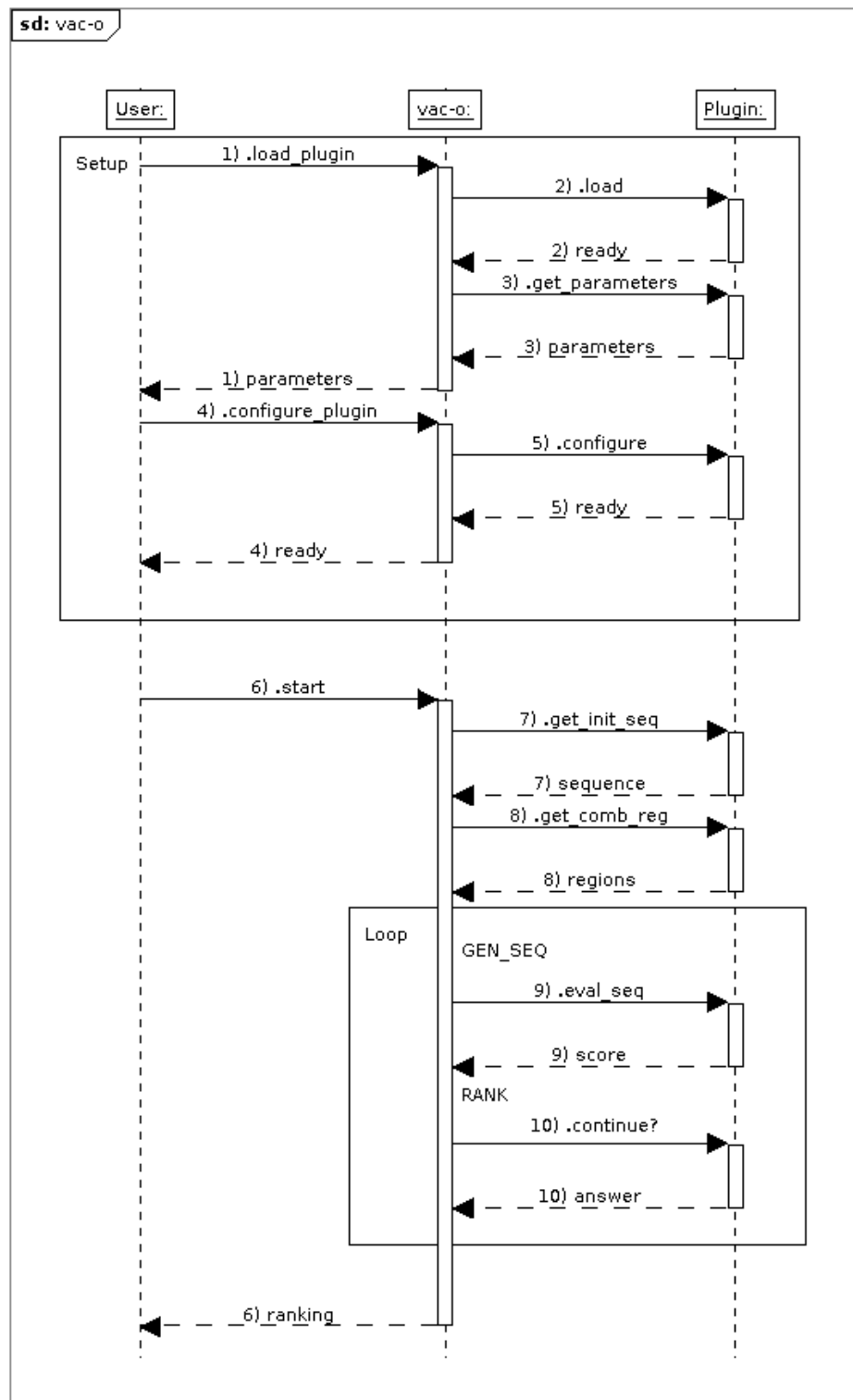


Figura 1: Flujo de trabajo de vac-o

2.1.4. Interfaces de Software

- BioPP: Se debe utilizar la librería BioPP para realizar el *folding* de secuencias genómicas, predicción de la estructura secundaria y cálculo de distancias.
- Boost.Python: Se debe utilizar Boost.Python para brindar a los desarrolladores de extensiones, la posibilidad de hacerlo utilizando el lenguaje de programación Python.

2.1.5. Interfaces de Comunicaciones

No se registran requerimientos.

2.1.6. Restricciones de Memoria

No se registran restricciones de memoria para la ejecución del software. No obstante, se debe tener en cuenta que dada la naturaleza y la complejidad del problema, el tiempo de cálculo estará directamente relacionado con la memoria disponible.

2.1.7. Operaciones

No se registran requerimientos.

2.1.8. Requerimientos de Instalación

No se registran requerimientos.

2.2. Funciones del Producto

2.2.1. Configuración inicial

Inicialmente, vac-o presenta al usuario final una pantalla con un formulario donde el usuario debe indicar la ubicación de la extensión que desea usar. A continuación, vac-o carga la extensión en memoria y presenta al usuario otro formulario con los campos requeridos por la extensión cargada. Una vez que los datos son validados, vac-o configura la extensión con los valores enviados por el usuario y queda listo para comenzar la ejecución cuando el usuario lo indique.

2.2.2. Configuración de vac-o

Antes de dar comienzo a la generación de secuencias genómicas, vac-o debe solicitar a la extensión cargada los siguientes valores:

- Secuencia inicial: La secuencia genómica que se encuentra en la cepa vac-unal y se desea mejorar.
- Regiones combinatorias: Las regiones de la secuencia inicial que resultan de interés, indicando sus posiciones de inicio y fin, y qué tipo de regiones son cada una (restricciones en base a código genético, o en base a estructura secundaria).

- Estrategias de búsqueda: La estrategia que se debe utilizar para la generación o búsqueda de nuevas secuencias.

2.2.3. Generación de secuencias

Luego de las configuraciones básicas, vac-o esta en condiciones de generar nuevas secuencias genómicas que cumplan las restricciones que hayan sido impuestas. Para esto, se deben calcular las posibles “variantes” de cada región combinatoria (teniendo en cuenta el tipo de región y sus posibles intersecciones) y de acuerdo a la estrategia de búsqueda, construir nuevas secuencias que serán evaluadas posteriormente por la extensión. Dado que el espacio de búsqueda podría ser eventualmente muy grande, el criterio de parada para la generación de secuencias también debe ser provisto por la extensión.

Notar que esta funcionalidad es el “corazón” de vac-o y es donde se encuentra la mayor complejidad del problema a resolver.

Supongamos $N \geq 1$ y R_1, \dots, R_N regiones combinatorias. Luego, tendremos:

$$\begin{array}{l} M_{1.1}, \dots, M_{1.j_1} \text{ mutaciones de } R_1 \\ M_{2.1}, \dots, M_{2.j_2} \text{ mutaciones de } R_2 \\ \dots \\ M_{N.1}, \dots, M_{N.j_N} \text{ mutaciones de } R_N \end{array}$$

con $j_1, \dots, j_N \geq 1$. Es decir, que el número de posibles secuencias que vac-o “debería” evaluar sera igual al producto $j_1 \times j_2 \times \dots \times j_N$.

Haciendo uso de diferentes estrategias de búsqueda, combinadas con las funciones de evaluación provistas por la extensión, vac-o intentara optimizar esta generación de secuencias.

2.2.4. Evaluación de secuencias

De acuerdo a los puntajes asignados por la extensión a cada una de las secuencias generadas, vac-o debe construir un listado o *ranking* de secuencias que será dado al usuario como resultado final de la ejecución.

2.3. Características de Usuarios

Se identifican 3 tipos de usuarios de vac-o:

- Final: que solo interactúa a través de la interfaz gráfica.
- Extensionista: que posee los conocimientos de programación suficientes como para implementar extensiones de vac-o. A los fines de ampliar los potenciales usuarios dentro de esta categoría, es que se ofrece la posibilidad de desarrollar extensiones usando el lenguaje Python.
- Contribuidor: que contribuye al código fuente de vac-o realizando mejoras o desarrollando nuevas funcionalidades.

2.4. Restricciones

El producto debe ser desarrollado utilizando el lenguaje de programación C++ y bajo la licencia de software GPLv3.

2.5. Suposiciones y Dependencias

- Sistema operativo: GNU/Linux

2.6. Trabajo Futuro

Probablemente una de las principales mejoras a la primer versión de vac-o, será realizar las modificaciones necesarias para permitir la ejecución del software en paralelo y de esta manera reducir los tiempos de cálculo. Por otro lado, se deberá profundizar en el desarrollo de extensiones para diferentes tipos de vacunas y virus.

3. Requerimientos

3.1. Funciones del Sistema

3.1.1. Configuración inicial

El objetivo de esta función es la carga en memoria de la extensión que se desea utilizar para ejecutar vac-o.

1. **Carga de la extensión:** El sistema recibe la ubicación del archivo *.so* y lo carga en memoria. Si la carga es exitosa, se devuelven los parámetros requeridos por la extensión (*[(nombre, tipo, validaciones, defecto)]*) para su posterior configuración. En caso de que la carga de la extensión fracase, se devuelve un mensaje de error.
2. **Validación de valores:** La interfaz de usuario es responsable de la validación de los datos ingresados por el usuario para la configuración de la extensión. Para esto, se deberán usar los diferentes *criterios de validación* para cada parámetro y solo cuando la validación sea exitosa, los valores son enviados al sistema. Caso contrario, se debe presentar un mensaje de error al usuario.
3. **Configuración de la extensión:** El sistema recibe de la interfaz de usuario, la lista de parámetros requeridos por la extensión (*[(nombre, valor)]*) y se asignan los valores en la extensión. Finalmente, se devuelve un mensaje de éxito indicando que vac-o está listo para comenzar la ejecución.
4. **Tamaño del ranking:** El sistema recibe de la interfaz de usuario, el tamaño del ranking de secuencias que dará como resultado de la ejecución.

3.1.2. Configuración de vac-o

El objetivo de esta función es solicitar a la extensión cargada en memoria, la información necesaria para comenzar la ejecución del motor combinatorio.

1. **Secuencia inicial:** El sistema solicita a la extensión cargada en memoria, la secuencia de ARN que se desea optimizar. La extensión debe devolver la secuencia y el sistema la almacena en memoria.

2. **Regiones combinatorias:** El sistema solicita a la extensión cargada en memoria, las regiones combinatorias de interés para la optimización. La extensión debe devolver las regiones $([inicio, fin, tipo, evaluador])$ y el sistema inicializa cada región en memoria. Para cada región, el parámetro *tipo* debe ser uno de los siguientes:
- Estructura secundaria (SS): Las posibles variantes de la región deben conservar la estructura secundaria.
 - Código genético (GC): Las posibles variantes de la región deben resultar silentes en términos de expresión aminocídica.
 - Personalizada (CU): La extensión debe proporcionar la implementación de la región combinatoria.

Además, se debe considerar la posibilidad de que en el futuro se necesiten nuevos tipos de regiones combinatorias.

El parámetro *evaluador* debe ser una función $f : Secuencia \rightarrow (0, 1)$ que será utilizada por el sistema para determinar localmente la “bondad” de las diferentes variantes de cada región.

3. **Estrategia de búsqueda:** El sistema solicita a la extensión un *umbral* de “bondad” y debe implementar diferentes estrategias de optimización, asegurando que se cumpla lo siguiente:

Sea n el número de regiones combinatorias, s_i la secuencia seleccionada de la región i y $f_i : Secuencia \rightarrow (0, 1)$ la función de evaluación de la región i , con $i = 1, \dots, n$.

$$\prod_{i=1}^n f_i(s_i) \geq umbral$$

4. **Regiones de validación:** El sistema solicita a la extensión cargada en memoria, las regiones sobre las que se debe hacer el “control de calidad”. La extensión debe devolver las regiones $([inicio, fin, prueba, criterio])$ y el sistema inicializa cada región en memoria. Para cada región, el parámetro *prueba*, debe ser uno de los siguientes:
- Mutaciones sistemáticas (ALL): Se calculan todas las posibles mutaciones.
 - Mutaciones al azar (RAND): Se calculan N mutaciones al azar.
 - Mutaciones personalizadas (CU): La extensión debe proporcionar la implementación.

El parámetro *criterio* debe ser uno de los siguientes:

- Similitud estructural (SS): La estructura secundaria de las mutaciones debe tener un porcentaje de similitud mayor o igual que un valor dado.
- Disimilitud estructural (DS): La estructura secundaria de las mutaciones debe tener un porcentaje de similitud menor o igual que un valor dado.

Además la extensión debe especificar la profundidad M con la que se desea realizar el “control de calidad”.

3.1.3. Generación de secuencias

El objetivo de esta función es la construcción de nuevas secuencias genómicas que cumplan con las restricciones impuestas.

1. **Generar variantes de región combinatoria:** Para cada región combinatoria, el sistema debe ser capaz de generar las posibles variantes que cumplan con las restricciones impuestas por el tipo de región y considerando las posibles intersecciones con las demás regiones. Además, debe ser posible el uso de diferentes librerías externas a ser determinadas por la extensión.
2. **Generar secuencias:** A partir de las variantes de cada región combinatoria, se deben construir nuevas secuencia genómicas, reemplazando en la secuencia inicial, cada región por su variante.

Esta funcionalidad representa el corazón del sistema y consiste en la construcción de un árbol de (sub)secuencias derivadas de cada una de las regiones combinatorias, donde cada hoja del árbol determinará una nueva secuencia genómica completa. Luego, el sistema debe ser capaz de generar y recorrer el árbol basándose en la información brindada por la extensión.

El criterio de terminación en la generación de secuencias (recorrido del árbol) debe ser provisto por la extensión.

3. **Validar secuencias:** Cada secuencia generada por el sistema debe ser sometida al “control de calidad” y solo aquellas que lo pasen exitosamente serán consideradas para una posterior evaluación.

El procedimiento consiste en generar y recorrer un árbol de mutaciones acumuladas por cada “región de validación”, que tendrá a lo sumo, profundidad M (dado por la extensión).

Diremos que una región pasa el “control de calidad” cuando sea posible generar el árbol completo. Es decir, cuando todos los nodos (mutantes) del árbol, pasen el criterio de prueba especificado. Si alguna región no pasa el “control de calidad”, entonces la secuencia se descarta.

Tanto el procedimiento para generar las mutaciones en cada nivel del árbol, como el criterio de prueba al que se somete cada mutación, están determinados por la “región de validación”.

3.1.4. Evaluación de secuencias

El objetivo de esta función es la evaluación y posterior construcción de un *ranking* de secuencias genómicas.

1. **Asignar puntaje a una secuencia:** El sistema solicita a la extensión cargada en memoria, un puntaje para una secuencia dada. La extensión debe evaluar la secuencia recibida comparándola con la secuencia inicial y devolver un puntaje.
2. **Construir *ranking* de secuencias:** El sistema debe construir un *ranking* de secuencias basándose en los puntajes asignados por la extensión. Este listado de secuencias es además, la salida final de la ejecución.

3.2. Restricciones de Rendimiento

No se registran requerimientos.

3.3. Base de Datos

No se registran requerimientos.

3.4. Restricciones de Diseño

3.4.1. Cumplimiento de Estándares

El producto debe cumplir con los siguientes principios de diseño de la programación orientada a objetos. Los primeros 5, son también conocidos por el acrónimo “**SOLID**”.

- Single responsibility principle (SRP)
- Open/closed principle (OCP)
- Liskov substitution principle (LSP)
- Interface segregation principle (ISP)
- Dependency inversion principle (DIP)
- Law of Demeter (LoD)

Además el producto debe cumplir con el estandar ANSI C++ y el “coding style” provisto por FuDePAN.

3.5. Atributos del Software

Se requiere que el código del producto tenga un 80 % de cobertura con pruebas automatizadas.

Apendices

A. Definiciones, Acrónimos y Abreviaturas

- **vac-o**: Combinatory Vaccine Optimizer.
- **FaMAF**: Facultad de Matemática, Astronomía y Física.
- **UNC**: Universidad Nacional de Córdoba.
- **FuDePAN**: Fundación para el Desarrollo de la Programación en Ácidos Nucleicos.
- **API**: Application Programming Interface.
- **GPL**: General Public License.
- **IEEE**: Institute of Electrical and Electronics Engineers
- **SOLID**: acrónimo nemotécnico introducido por Robert C. Martin en la década de 2000, que representa cinco principios básicos de programación y diseño orientado a objetos.
- **LoD**: Law of Demeter, principio de diseño orientado a objetos para lograr “bajo acoplamiento”.

B. Referencias

- C++: Lenguaje de programación.
<http://www.cplusplus.com>
- Python: Lenguaje de programación interpretado.
<http://www.python.org>
- BioPP: Librería C++ para biología molecular
<http://code.google.com/p/biopp>
- Boost.Python: Librería C++ para la interacción con Python.
http://www.boost.org/doc/libs/1_42_0/libs/python/doc/index.html
- QT: Librería C++ para el desarrollo de interfaces gráficas.
<http://qt.nokia.com/products>
- GPL: General Public License.
<http://www.gnu.org/licenses/gpl.html>
- IEEE STD 830-1998: Guía para la especificación de requerimientos.
http://standards.ieee.org/reading/ieee/std_public/description/se/830-1998_desc.html
- SOLID: “Design Principles and Design Patterns”, Robert C. Martin.
http://www.objectmentor.com/resources/articles/Principles_and_Patterns.pdf