



UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

Análisis y optimización de un simulador demográfico para entornos paralelos

Memoria del proyecto

Ingeniería del Software

Grado en Ingeniería Informática

Autor: Vanessa Büsing

Director: Cristina Montañola Sales

Ponente: Josep Casanovas Garcia

23 de junio de 2015



Agradecimientos

Muchas gracias a todos los que ayudaron a hacer posible este proyecto.

En especial gracias a todo el equipo del InLab que me ha brindado su apoyo y experiencia durante toda la etapa de mi carrera que he estado trabajando allí y realizando mi proyecto. Gracias a mi directora Cristina Montañola por toda la guía y el conocimiento que me ha inculcado durante la realización del proyecto. Y gracias a mi director Josep Casanovas que, con su apoyo, ha hecho posible la realización de este trabajo.

Muchas gracias a Mari Paz Linares por todo su apoyo, paciencia, sonrisas y correcciones que me ha brindado en todo momento.

Grazie mille a Alessandro Pellegrini che mi ha guidato e ha incaricato l'uso di ROOT-Sim e ha fornito tutto il supporto necessario per il progetto.

Muchas gracias a Cristina Gómez Seoane por su ayuda, asesoramiento y dedicación para la realización de este trabajo.

Gracias a mi hermano Ricardo Büsing por estar allí para escucharme todo el tiempo, y a mis padres Dagoberto Salazar y Rocio Meneses, porque aunque estén lejos, nunca me ha faltado su apoyo, consejo y cariño.

Por último, gracias a Oriol Torné por todo el apoyo, paciencia, comprensión, conocimiento y cariño que me brindas cada día y que me ha permitido llevar adelante este proyecto.



Resumen

Castellano

En la actualidad el estudio del comportamiento de las poblaciones está adquiriendo cada vez más importancia. Con los crecientes avances en muchas áreas de la vida cotidiana y la constante evolución de la sociedad, es necesario realizar planificaciones sobre cómo evoluciona la población para poder tomar las medidas necesarias para el correcto funcionamiento y distribución de los recursos y la planificación de su gestión.

Sin embargo, el estudio de los sistemas sociales presenta una gran dificultad debido a la complejidad de los propios individuos y las interacciones que tienen lugar entre ellos. Gracias a la aplicación de técnicas de simulación paralela, se pueden realizar estudios sociales a gran escala con muestras reales de poblaciones, en una menor cantidad de tiempo que en el caso de las simulaciones secuenciales, las cuales tardarían demasiado o simplemente no serían posibles.

El principal objetivo de este trabajo final de grado es el de realizar un análisis de dos herramientas de simulación paralela para la simulación de dinámicas socio-demográficas en el simulador Yades. En este trabajo, se analiza la implementación actual de Yades que emplea la librería μ sik para la paralelización y sincronización de la simulación discreta por eventos. Por otro lado, se estudia la librería de simulación paralela ROOT-Sim, que permite aplicar simulación basada en eventos discretos para entornos paralelos aprovechando el esquema de memoria compartida. Gracias a este análisis se espera realizar una optimización en el rendimiento del simulador Yades, para así aumentar su rendimiento y escalabilidad, al reducir su tiempo de computación.

English

Nowadays, the study of populations behaviour is acquiring ever more relevance. With the new advances in a lot of areas of the everyday life and the constant evolution of the society, it is necessary to perform plannings about the evolution of the population, in order to take the necessary measures to operate, to plan and to distribute properly all the resources.

However, the study of the social systems presents a big difficulty due to the complexity of the individuals and the interactions which take place among them. Thanks to the application of parallel simulation techniques, it is possible to perform social studies at a large scale, using real population samples. Moreover, those simulations can be made in lesser time than in the correspondent sequential simulation, which would be too slow or simply not possible.

The main goal of this final BSc. project is to perform an analysis of two simulation parallel tools for running socio-demography dynamic simulations in Yades simulator. In this work, we first analyse the current implementation of Yades which uses μ sik library to run and synchronize discrete event simulations in parallel environments. Second, we study the parallel simulation library ROOT-Sim, which allows to apply discrete event simulation to parallel environments profiting share-memory capabilities. Thanks to this study, we expect to optimize the performance and scalability of Yades by taking advantage of ROOT-Sim characteristics, reducing its computation time.



Índice

| | |
|---|-----------|
| 1. Introducción | 13 |
| 1.1. Contexto | 13 |
| 1.2. Formulación del problema | 18 |
| 2. Estado del arte | 21 |
| 2.1. Análisis de las herramientas de simulación | 21 |
| 2.1.1. Herramientas de simulación paralela basadas en agentes | 22 |
| 2.2. Análisis de las librerías de simulación | 26 |
| 2.2.1. Herramientas PDES para simulaciones basadas en agentes | 27 |
| 2.3. Conclusión sobre el análisis de la competencia | 31 |
| 3. Propuesta de solución | 32 |
| 4. Objetivos y alcance del proyecto | 33 |
| 4.1. Objetivos | 33 |
| 4.2. Alcance | 33 |
| 5. Planificación | 35 |
| 5.1. Calendario | 35 |
| 5.2. Metodología de trabajo | 35 |
| 5.3. Métodos de validación | 36 |
| 5.4. Descripción de las tareas | 36 |
| 5.4.1. Planificación inicial | 36 |
| 5.4.2. Iteraciones de desarrollo | 37 |
| 5.5. Diagrama de Gantt de la planificación inicial | 38 |
| 5.6. Posibles obstáculos y plan de contingencia | 41 |



| | | |
|-----------|--|------------|
| 5.7. | Diagrama de Gantt de la planificación final | 41 |
| 5.8. | Valoración de la metodología utilizada | 44 |
| 5.9. | Recursos | 45 |
| 5.9.1. | Recursos materiales y de sistema operativo | 45 |
| 5.9.2. | Recursos para documentar y de comunicación | 46 |
| 5.9.3. | Recursos de desarrollo | 46 |
| 5.9.4. | Recursos de seguimiento | 47 |
| 5.9.5. | Recursos humanos | 47 |
| 6. | Análisis del sistema actual | 48 |
| 6.1. | Descripción del funcionamiento actual de Yades con μ sik | 48 |
| 6.1.1. | Descripción de μ sik | 48 |
| 6.1.2. | Descripción de Yades | 49 |
| 6.1.3. | Escalabilidad y Overheads | 55 |
| 6.1.4. | Problemática actual | 58 |
| 6.2. | Descripción de ROOT-Sim | 59 |
| 6.2.1. | Escalabilidad y Overheads | 64 |
| 7. | Especificación del sistema | 69 |
| 7.1. | Stakeholders | 69 |
| 7.2. | Modelo conceptual | 70 |
| 7.3. | Historias de usuario | 76 |
| 8. | Diseño de la solución | 103 |
| 8.1. | Arquitectura del sistema software | 103 |
| 8.2. | Planificador de eventos | 115 |
| 8.3. | Confirmación del estado | 116 |
| 8.4. | Reportes | 117 |



| | |
|--|------------|
| 8.5. Migraciones | 118 |
| 9. Implementación de la solución | 120 |
| 9.1. Instalación previa y despliegue | 120 |
| 9.2. Estructuras de datos | 121 |
| 9.3. Redefinición de métodos | 125 |
| 9.4. Correspondencia diseño-implementación | 127 |
| 10. Diseño de experimentos | 133 |
| 11. Validación | 137 |
| 11.1. Validación por componentes | 138 |
| 11.1.1. Mortalidad | 141 |
| 11.1.2. Fertilidad | 142 |
| 11.1.3. Estados económicos | 143 |
| 11.1.4. Estados maritales | 144 |
| 11.1.5. Migraciones | 144 |
| 11.2. Testing de aplicaciones paralelas | 147 |
| 12. Comparativa de resultados | 151 |
| 12.1. Arquitectura técnica del sistema | 151 |
| 12.1.1. Marenostrom 3 | 151 |
| 12.1.2. Capitano | 151 |
| 12.2. Comparativa | 153 |
| 12.2.1. Análisis de ρ en secuencial | 153 |
| 12.2.2. Análisis del paralelismo | 158 |
| 13. Presupuesto y sostenibilidad | 166 |
| 13.1. Identificación de costes | 166 |



| | |
|---|------------|
| 13.2. Estimación de costes | 168 |
| 13.3. Viabilidad económica | 168 |
| 13.4. Sostenibilidad | 170 |
| 13.4.1. Dimensión económica | 171 |
| 13.4.2. Dimensión social | 171 |
| 13.4.3. Dimensión ambiental | 171 |
| 14. Leyes y regulación | 173 |
| 15. Conclusión | 174 |
| 16. Trabajo futuro | 176 |
| 17. Competencias técnicas trabajadas | 178 |
| Glosario | 180 |
| Acrónimos | 184 |
| Anexos | 185 |
| Referencias | 188 |



Índice de figuras

| | | |
|-----|--|----|
| 1. | Ejemplo de Rollback | 17 |
| 2. | Traza del simulador Yades en MN3. | 19 |
| 3. | Explicación Diagrama de Gantt inicial | 39 |
| 4. | Diagrama de Gantt inicial | 40 |
| 5. | Explicación Diagrama de Gantt final | 42 |
| 6. | Diagrama de Gantt inicial | 43 |
| 7. | Arquitectura μ sik | 49 |
| 8. | Diagrama UML Yades con μ sik | 50 |
| 9. | Estructura y comunicación de simulación Yades- μ sik | 51 |
| 10. | Esquema de funcionamiento de las migraciones en 3 fases con Yades | 52 |
| 11. | Diagrama de actividades actual de Yades con μ sik. Inicialización del sistema. | 53 |
| 12. | Diagrama de actividades actual de Yades con μ sik. Gestión de eventos. | 54 |
| 13. | Gráfica con el análisis weak scaling de Yades con μ sik | 57 |
| 14. | Gráfica con el análisis strong scaling de Yades con μ sik | 58 |
| 15. | Arquitectura de un PDES | 60 |
| 16. | Estructura de ROOT-Sim | 61 |
| 17. | Topologías ROOT-Sim | 63 |
| 18. | Evolución eficiencia PCS-ROOT-Sim | 66 |
| 19. | Evolución eventos PCS-ROOT-Sim | 67 |
| 20. | Evolución latencia PCS-ROOT-Sim | 68 |
| 21. | Diagrama conceptual de Yades. | 72 |
| 22. | Diagrama de estados individuo | 73 |
| 23. | Diagrama funciones de vida individuo | 74 |
| 24. | Diagrama de ES individuo | 74 |
| 25. | Diagrama de MS individuo | 75 |



| | | |
|-----|---|-----|
| 26. | Diagrama de historias de usuario de Yades. | 77 |
| 27. | Diagrama de la arquitectura del sistema. | 104 |
| 28. | Diagrama de entidades de Yades. | 105 |
| 29. | Diagrama de las entidades que representan los eventos de Yades. | 114 |
| 30. | Conflicto inconsistencia MS 1 | 116 |
| 31. | Conflicto inconsistencia MS 2 | 117 |
| 32. | Diagrama de migraciones domésticas de Yades. | 119 |
| 33. | Diagrama de estructuras de datos de Yades con ROOT-Sim. | 122 |
| 34. | Lista doblemente encadenada. | 124 |
| 35. | Lista de unidades familiares. | 125 |
| 36. | Diagrama de actividad Yades con ROOT-Sim | 126 |
| 37. | Verificación, validación y acreditación del software | 137 |
| 38. | Inicialización de la población en Yades. | 139 |
| 39. | Validación por caja blanca μ_{sik} 1 | 140 |
| 40. | Validación por caja blanca μ_{sik} 2 | 140 |
| 41. | Validación por caja blanca: mortalidad | 141 |
| 42. | Validación por caja blanca: fertilidad | 142 |
| 43. | Validación por caja blanca: ES | 143 |
| 44. | Validación por caja blanca: MS | 145 |
| 45. | Validación por caja blanca: emigración dom. | 146 |
| 46. | Validación por caja blanca: inmigración dom. | 146 |
| 47. | Validación por caja blanca: migración int. | 147 |
| 48. | Arquitectura de un nodo de Marenostrium. | 152 |
| 49. | Arquitectura de un core de Marenostrium. | 152 |
| 50. | Arquitectura de Capitano. | 153 |
| 51. | Traza Yades-ROOT-Sim en secuencial. | 154 |
| 52. | Análisis ρ secuencial 1 | 155 |



| | | |
|-----|--|-----|
| 53. | Análisis ρ secuencial 2 | 155 |
| 54. | Análisis ρ secuencial 3 | 156 |
| 55. | Análisis ρ secuencial 4 | 157 |
| 56. | Traza Yades- μ sik con 4 procesadores. | 158 |
| 57. | Análisis paralelismo 1 | 159 |
| 58. | Traza Yades-ROOT-Sim con 3 procesadores. | 159 |
| 59. | Análisis paralelismo 2 | 161 |
| 60. | Análisis paralelismo 3 | 162 |
| 61. | Análisis paralelismo 4 | 163 |
| 62. | Análisis paralelismo 5 | 164 |
| 63. | Análisis paralelismo 6 | 165 |
| 64. | Análisis paralelismo 7 | 165 |
| 65. | Arquitectura nodo Marenostrium | 185 |



Índice de tablas

| | | |
|-----|--|----|
| 1. | Análisis Mosca+ para la selección del software de simulación | 25 |
| 2. | Análisis Mosca+ para la selección de la librería de simulación | 30 |
| 3. | Épica: Inicialización del simulador | 79 |
| 4. | Historia de usuario: Inicialización del simulador 1 | 79 |
| 5. | Historia de usuario: Inicialización del simulador 2 | 80 |
| 6. | Historia de usuario: Inicialización del simulador 3 | 80 |
| 7. | Historia de usuario: Inicialización del simulador 4 | 80 |
| 8. | Historia de usuario: Inicialización del simulador 5 | 81 |
| 9. | Historia de usuario: Inicialización del simulador 6 | 81 |
| 10. | Épica: Modelización de la fertilidad | 82 |
| 11. | Historia de usuario: Fertilidad 1 | 82 |
| 12. | Historia de usuario: Fertilidad 2 | 83 |
| 13. | Historia de usuario: Fertilidad 3 | 83 |
| 14. | Historia de usuario: Fertilidad 4 | 83 |
| 15. | Historia de usuario: Fertilidad 5 | 84 |
| 16. | Épica: Modelización de la mortalidad | 85 |
| 17. | Historia de usuario: Mortalidad 1 | 85 |
| 18. | Historia de usuario: Mortalidad 2 | 86 |
| 19. | Historia de usuario: Mortalidad 3 | 86 |
| 20. | Historia de usuario: Mortalidad 4 | 86 |
| 21. | Historia de usuario: Mortalidad 5 | 87 |
| 22. | Épica: Modelización de estados económicos | 88 |
| 23. | Historia de usuario: Estados económicos 1 | 88 |
| 24. | Historia de usuario: Estados económicos 2 | 89 |
| 25. | Historia de usuario: Estados económicos 3 | 89 |



| | | |
|-----|--|-----|
| 26. | Historia de usuario: Estados económicos 4 | 89 |
| 27. | Historia de usuario: Estados económicos 5 | 90 |
| 28. | Épica: Modelización de MS | 91 |
| 29. | Historia de usuario: Estados maritales 1 | 91 |
| 30. | Historia de usuario: Estados maritales 2 | 92 |
| 31. | Historia de usuario: Estados maritales 3 | 92 |
| 32. | Historia de usuario: Estados maritales 4 | 92 |
| 33. | Historia de usuario: Estados maritales 5 | 93 |
| 34. | Épica: Modelización de migraciones | 94 |
| 35. | Historia de usuario: Migraciones domésticas 1 | 94 |
| 36. | Historia de usuario: Migraciones domésticas 2 | 95 |
| 37. | Historia de usuario: Migraciones domésticas 3 | 95 |
| 38. | Historia de usuario: Migraciones domésticas 4 | 95 |
| 39. | Historia de usuario: Migraciones domésticas 5 | 96 |
| 40. | Historia de usuario: Migraciones internacionales 1 | 96 |
| 41. | Historia de usuario: Migraciones internacionales 2 | 96 |
| 42. | Historia de usuario: Ejecutar el simulador en entorno paralelo | 98 |
| 43. | Historia de usuario: Facilidad de uso | 98 |
| 44. | Historia de usuario: Escalabilidad | 99 |
| 45. | Historia de usuario: Multiplataforma | 99 |
| 46. | Historia de usuario: Escalabilidad | 100 |
| 47. | Estructuras de simulación Yades | 107 |
| 48. | Diseño de estructuras de simulación Yades-ROOT-Sim 1 | 108 |
| 49. | Diseño de estructuras de simulación 2 | 109 |
| 50. | Diseño de estructuras de simulación 3 | 109 |
| 51. | Diseño de métodos FamilyUnit 1 | 110 |
| 52. | Diseño de métodos FamilyUnit 2 | 111 |



| | | |
|-----|--|-----|
| 53. | Diseño de métodos globales | 111 |
| 54. | Diseño de métodos PopulationSimulator | 112 |
| 55. | Diseño de métodos Region | 113 |
| 56. | Tabla con correspondencia de implementación-diseño 1 | 128 |
| 57. | Tabla con correspondencia de implementación-diseño 2 | 129 |
| 58. | Tabla con correspondencia de implementación-diseño 3 | 129 |
| 59. | Tabla con correspondencia de implementación-diseño 4 | 130 |
| 60. | Tabla con correspondencia de implementación-diseño 5 | 131 |
| 61. | Tabla con correspondencia de implementación-diseño 6 | 132 |
| 62. | Diseño de experimentos Yades 1 | 135 |
| 63. | Diseño de experimentos Yades 2 | 136 |
| 64. | Análisis de costes por recursos humanos del proyecto | 166 |
| 65. | Análisis coste por software del proyecto | 167 |
| 66. | Análisis de costes por infraestructura del proyecto | 168 |
| 67. | Análisis coste de proyecto | 169 |
| 68. | Matriz de sostenibilidad del proyecto | 170 |
| 69. | Valores para cálculo de réplicas | 187 |

1. Introducción

1.1. Contexto

En los últimos años, nuestro planeta está experimentando un fuerte incremento de la población que habita en él. Sin embargo, en algunos países desarrollados como por ejemplo en algunos países de la Unión Europea, se ha observado que la población activa está disminuyendo y se está produciendo un envejecimiento progresivo de la población. Algunos de los factores que contribuyen a este hecho son los constantes avances médicos que aumentan la esperanza de vida de las personas, el aumento de la edad a la que se tienen hijos y el ritmo de vida actual cada vez más acelerado, que influyen directamente en la tasa de natalidad, disminuyéndola. Por esta razón, las perspectivas de futuro para el viejo continente no son muy optimistas en cuanto a la evolución de su población activa.

Como resultado de esta situación, Europa deberá hacer frente a una serie de retos demográficos en los próximos años. Por ello, los gobiernos deben obtener datos sobre la población para poder entender la actual situación a la que nos enfrentamos y poder tomar medidas tales como: crear políticas de control de la población, crear políticas migratorias, estimar el número de colegios o ayudas para recién nacidos, entre otras, que permitan solucionar la creciente problemática.

El estudio de los sistemas sociales presenta una gran complejidad debido a que los individuos interactúan entre ellos y con su entorno, toman decisiones que implican cambios en su ciclo de vida y que afectan el de los demás individuos [Lozares Colina, 2004]. Por esta razón, éstos sistemas se pueden definir como sistemas no lineales, dinámicos, con un alto grado de incertidumbre [Montañola-Sales et al., 2014]. El campo de la ciencia que se encarga del estudio de las poblaciones y su comportamiento se llama demografía. Se centra especialmente en los procesos que determinan la formación, conservación y desaparición de las poblaciones. Entre los procesos más importantes que permiten la evolución de la población en el transcurso del tiempo, destacan: la natalidad, la mortalidad y los movimientos migratorios [Livi Bacci, 1993-2007].

Sin embargo, es difícil realizar predicciones en el campo de los sistemas sociales, debido a sus dinámicas y a la multitud de variables que intervienen. Por esta razón, en los últimos años se han ido introduciendo técnicas de modelización matemática y simulación en su estudio. Estas técnicas, junto con otras de desarrollo de software, permiten, a partir de la observación del sistema real, modelar el comportamiento de los diferentes grupos de poblaciones para poder observarlo, entenderlo mejor, y proponer soluciones para la problemática descrita anteriormente.

Para realizar un estudio de los sistemas sociales se debe realizar un modelo para su correspondiente estudio. A grandes rasgos, modelar un sistema consiste en realizar una representación, ya sea matemática o gráfica, de este sistema para que pueda ser tratado y estudiado.

Con el fin de modelar sistemas sociales, existen diferentes paradigmas de simulación que permiten modelar los sistemas a diferentes escalas. La ventaja principal de este enfoque es que

las variables explicativas específicas de cada individuo pueden ser incluidas en el modelo. Por ejemplo, factores como la edad, el nivel educativo, el sueldo y la etnicidad pueden ser considerados para modelar el número de hijos que una mujer pueda tener. Entre los paradigmas de simulación más conocidos encontramos la *microsimulación*, la *dinámica de sistemas (System Dynamics)*, la *simulación por eventos discretos (Discrete Event Simulation, DEV)* y la *modelización basada en agentes (Agent Based Models, ABM)*.

La *Microsimulación* [Bourguignon and Spadaro, 2006] permite simular a nivel de cada individuo (nivel microscópico) o de las entidades, su evolución a lo largo del sistema. Para ello, los modeladores necesitan especificar una función *Estocástica* para cada uno de los individuos del modelo en cada unidad de tiempo de simulación, con el fin de definir el siguiente estado de los individuos durante el tiempo de simulación. Por ello, la microsimulación requiere una gran cantidad de microdatos empíricos para poder simular el sistema. Históricamente ha sido una de las metodologías más usadas para simular sistemas sociales, y existen diferentes herramientas en el mercado que son usadas para planificar políticas sociales.

La *Dinámica de Sistemas (System Dynamics)* [Montañola Sales et al., 2015] también es un paradigma comúnmente utilizado para desarrollar modelos de simulación demográfica. Al contrario que en la microsimulación, la dinámica de sistemas no registra los cambios de estado de cada individuo en cada tiempo de simulación, sino que se enfoca en el comportamiento del grupo de individuos. Esta metodología permite observar el comportamiento de un sistema o su flujo a gran escala y sacar estadísticas de funcionamiento como un conjunto, como la cantidad de individuos que se mueven de un estado a otro, por ejemplo.

De forma similar a la microsimulación, la *Simulación por Eventos Discretos (DEV)* [Pellegrini, 2013-2014] permite registrar el movimiento de los individuos desde que llegan al sistema (por ejemplo en el caso de los movimientos demográficos a través de nacimientos y migraciones) hasta que lo abandonan (a través de muertes o migraciones, por ejemplo). Sin embargo, al contrario que en la microsimulación, la simulación por eventos discretos no inspecciona los individuos a cada tiempo de simulación sino que sólo los inspecciona cuando el estado del individuo cambia. De este modo, la simulación por eventos discretos expresa el comportamiento del sistema como una secuencia de eventos discretos que tienen lugar durante un periodo de tiempo. Los eventos ocurren en un momento determinado y producen un cambio en el estado del sistema.

El uso de los *Modelos Basados en Agentes (ABM)* [Billari and Prskawetz, 2003], [Axtell, 2000] es muy efectivo en el estudio de los sistemas sociales ya que permite modelar de una forma más natural el comportamiento social que en el caso de otras aproximaciones matemáticas [Montañola-Sales, 2014], a nivel de cada individuo. Un modelo basado en agentes está formado por un conjunto de agentes independientes que interactúan con su entorno gracias a un conjunto de reglas internas previamente definidas para conseguir sus objetivos. Actualmente la modelización basada en agentes se emplea para simular sistemas de la vida real, como por ejemplo para simular procesos demográficos [Montañola-Sales, 2014], para modelar la transmisión de enfermedades como la hepatitis A [Ajelli and Merler, 2009], simular procesos de evacuación [Wagner and Agrawal, 2014], entre otros, ya que contribuye a entender el modelo y tomar decisiones. La

simulación basada en agentes (ABS) es la implementación computarizada del ABM. La ABS se distingue del DEV en que está más orientada a los individuos que a los procesos [Siebers et al., 2010], los individuos son tomados como agentes que toman sus propias decisiones, las cuales pueden afectar al sistema, y presentan sus propias características personales.

Uno de los factores con mayor impacto de la ABS a gran escala es la *Escalabilidad* [Montañola-Sales, 2014]. La escalabilidad de un sistema es la capacidad de abarcar una gran cantidad de trabajo, a medida que el hardware aumenta, es decir, indica el grado de eficiencia de una aplicación a medida que los recursos de computación aumentan. Algunos de los factores que pueden influir directamente en la escalabilidad de un sistema ABS son:

- La complejidad del ABM, que puede afectar el rendimiento en gran medida. La complejidad del modelo puede ser debida tanto a la complejidad de cada individuo como al número de agentes que haya en el sistema.
- La complejidad y la topología de las comunicaciones entre los agentes, que tiene un impacto en la velocidad de comunicación de la simulación.
- La complejidad del espacio que ocupan los agentes.
- La complejidad de los mecanismos de sincronización del tiempo de simulación.

La duración de la simulación de un sistema ABS no depende sólo de la velocidad del procesador y de la velocidad de las comunicaciones, sino también de la capacidad de la memoria. Si el modelo requiere una gran cantidad de individuos, el uso de los recursos de memoria se verá fuertemente afectado introduciendo un uso intensivo de éstos (Memory Swapping). Por esta razón, la ejecución secuencial de un ABS de un sistema social grande, puede presentar problemas por falta de recursos y la ejecución puede ser bastante lenta.

Debido a la gran cantidad de datos con los que se trabaja y a las dificultades propias de los modelos sociales ABS, la aplicación de técnicas de simulación paralela o distribuida puede ser de gran utilidad, ya que permite distribuir los procesos en diferentes procesadores, incrementar el rendimiento de las aplicaciones y distribuir el tiempo de cálculo [Aaby et al., 2010]. El número de comunicaciones realizadas entre los agentes depende, no sólo de su distribución a lo largo del espacio, sino de la interacción o movimientos que realicen. En la mayoría de los casos, el espacio de simulación se divide entre los diferentes nodos de computación, cada uno de los cuales tiene una sección de todo el escenario de simulación. La paralelización de estos modelos se hace aún más necesaria cuando se trata de escenarios a gran escala, que comportan un gran número de agentes. Además, a medida que los modelos de toma de decisiones de los agentes adquieren mayor complejidad, el coste computacional se dispara.

En la *Computación de Altas Prestaciones* (High Performance Computing, HPC) se pueden encontrar diferentes técnicas para acelerar el rendimiento de aplicaciones, y la principal diferencia entre ellas se encuentra en la arquitectura que cada uno usa. En la computación paralela los equipos están conectados a través de una red de alta velocidad y comparten el mismo espacio físico;

la infraestructura de estos sistemas suele estar compuesta por equipos con las mismas características (homogéneos). En el caso de la computación distribuida, la capacidad de computación se encuentra distribuida en diferentes equipos que se encuentran en espacios físicos diferentes. Su infraestructura suele ser heterogénea, por lo que son necesarios protocolos software complejos para sincronizar las comunicaciones entre ellos.

Ambos paradigmas de computación se ven directamente afectados por la *Latencia*, que es el retraso (delay) al transmitir un mensaje de un computador a otro. En el caso de la computación paralela, la latencia es relativamente baja, pero en el caso de la computación distribuida es significativamente más alta, debido a que es necesario implementar complejos mecanismos para transmitir la información de un computador que se encuentra en un espacio físico a otro que está en otro diferente. Esta propiedad, además, es de gran interés para el estudio del rendimiento de los sistemas.

La distribución de una simulación en un sistema de HPC se puede hacer siguiendo dos grandes esquemas *temporal* o *espacial*. El primero consiste en dividir la simulación según su componente de tiempo, con lo que cada paso de simulación o intervalo se ejecuta en un procesador. El segundo consiste en dividirla según diferentes secciones espaciales, cada una de las cuales tiene una parte del espacio de simulación y se ejecuta en un procesador diferente. Si nos centramos en el caso particular de una simulación social, la distribución de la simulación con esquema temporal no tendría mucho sentido ya que a menudo hechos del pasado pueden afectar al presente. Por ello, se sigue un esquema espacial en la que se desean simular regiones físicas como países, estados o comunidades, asignando a cada procesador la ejecución de una región determinada con el escenario correspondiente a simular, por ejemplo.

En simulación paralela, el modelo de simulación se parte en procesos lógicos (Logical Processes, LP). Cada LP o grupos de ellos son asignados a cada procesador y se ejecutan concurrentemente. Cada uno de ellos tiene una parte del modelo de simulación y se comunica con los demás enviando mensajes o eventos. Una ejecución de simulación discreta por eventos en entornos paralelos (PDES) comporta una serie de LP concurrentes, con un identificador único, numérico, comprendido en el rango $[0, N-1]$. Para asegurar la correcta ejecución del modelo es necesario mantener la propiedad de *Locality constraint*, que establece que los eventos deben ser producidos en su correspondiente secuencia temporal. De esta manera se establece que los resultados de la simulación son equivalentes a una simulación secuencial. Por esta razón es necesario el uso de mecanismos de sincronización para asegurar esta propiedad.

Las metodologías de simulación se pueden clasificar en *basadas en el tiempo (time-stepped)* y *dirigidas por eventos (event-driven)* en DEV. En el primer caso, todos los eventos son revisados cada intervalo de tiempo definido. Esta aproximación es útil en el caso que haya muchos eventos para revisar. En el segundo caso, se emplea una lista de eventos con el fin de almacenar los diferentes eventos que tendrán lugar durante la simulación, se coge el primer elemento de la lista de eventos y se mueve el tiempo de simulación al de ese evento para simular sus efectos. En general el esquema dirigido por eventos se considera óptimo para la simulación, sobre todo en el caso de que no haya muchos eventos para procesar cada unidad de tiempo.

Los sistemas HPC, sin embargo, también encierran algunas complicaciones propias de la arquitectura [Timm and Pawlaszczyk, 2005]:

- Es necesario realizar un balanceo de carga entre las diferentes unidades de procesamiento (cores) para asegurar un buen rendimiento de la simulación paralela y evitar situaciones en las que haya una CPU ociosa.
- Sincronización de los eventos para asegurar la local causality constraint.
- Manejar las comunicaciones entre nodos.
- Monitorizar el tiempo de simulación distribuido.
- Asociación dinámica de la memoria.

Durante el proceso de simulación en paralelo, para asegurar que los eventos se ejecuten de forma equivalente a una simulación secuencial se pueden emplear diferentes esquemas de sincronización causal de los eventos [Fujimoto, 2000]. En un *esquema conservativo* las simulaciones distribuidas en los nodos esperarían a sincronizarse más allá de un cierto punto, evitando la posibilidad de una violación de la locality constraint. En cambio, en un *esquema optimista* cuando se detectan eventos que afectarían a la locality constraint, se realiza un retroceso (Rollback) para restablecer el último estado consistente. En la figura 1 se puede ver un ejemplo de una situación en la que es necesario realizar un rollback para volver a un estado de simulación consistente.

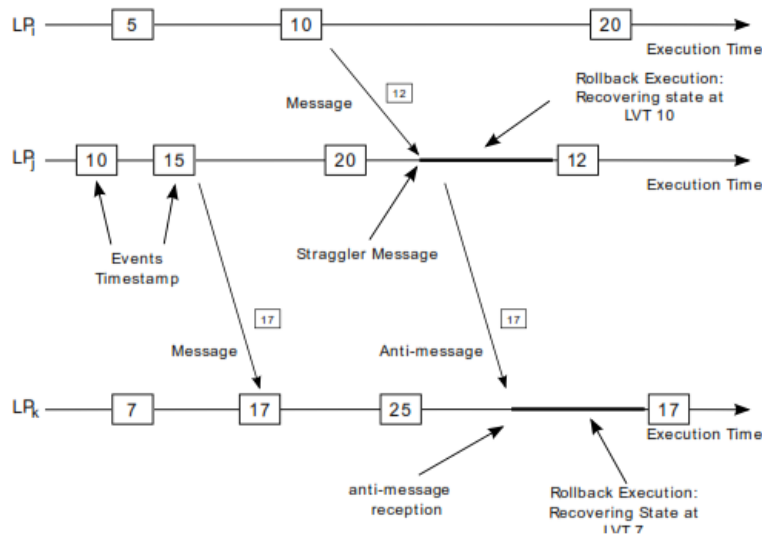


Figura 1: Ejemplo de Rollback para regresar a un estado de simulación consistente [Pellegrini and Quaglia, 2014].

Como resultado del esquema optimista, en todo momento es necesario almacenar los estados previos de simulación para poder llevar hacia atrás la ventana de simulación, si fuese necesario. En

estos casos también tiene especial interés el uso de técnicas de computación de altas prestaciones (HPC), debido a que se puede optimizar el uso de los recursos informáticos que se consumen durante la simulación. Además, la computación paralela permite realizar varias ejecuciones del mismo programa con diferentes juegos de pruebas, en una cantidad de tiempo mucho menor.

Para la realización de este trabajo se ha escogido el simulador demográfico basado en agentes Yades (Yet Another DEMographic Simulator), que se ejecuta en arquitectura paralela. Se trata de un proyecto desarrollado conjuntamente entre el Barcelona Super Computing Center (BSC), el InLab FIB de la Universitat Politècnica de Catalunya - BarcelonaTech y el Departamento de Management Science de la Universidad de Lancaster [Montañola Sales et al., 2015]. Yades emplea un esquema de simulación espacial para representar la interacción de diferentes individuos con su entorno, asignando a cada procesador una porción del espacio de simulación. Yades permite el modelado de dinámicas socio-demográficas mediante el uso de modelos basados en agentes y usa un algoritmo de sincronización optimista para la sincronización de eventos en la red.

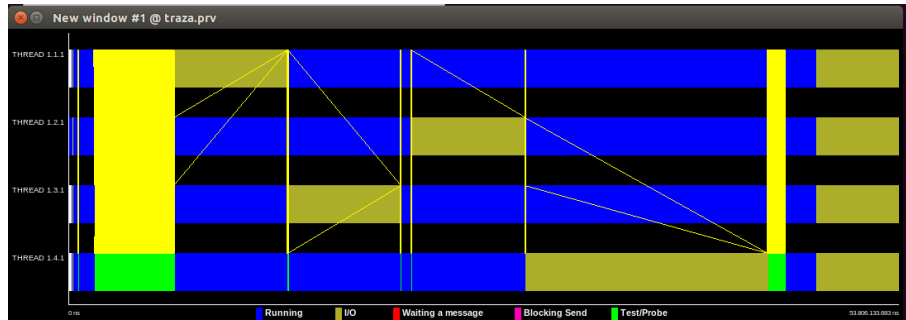
1.2. Formulación del problema

El simulador Yades emplea una librería de simulación paralela, μ sik [Perumalla, 2004], que se encarga de realizar todas las comunicaciones entre los diferentes nodos de computación y gestiona el envío, recepción y sincronización de todos los eventos de simulación. Este proceso se realiza de manera transparente para el usuario, que sólo tiene que preocuparse de la definición del modelo de simulación y del análisis de los resultados.

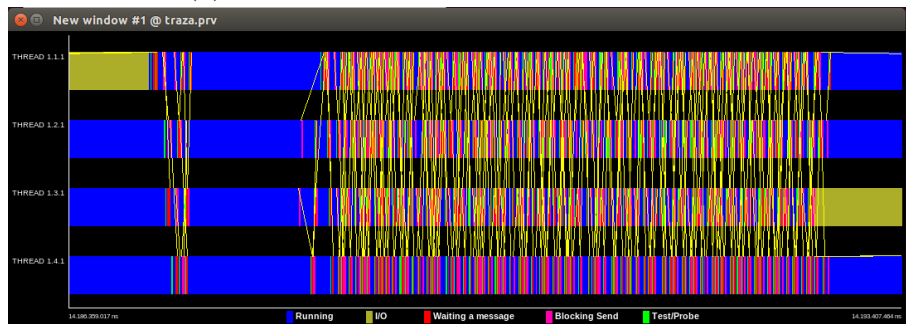
Como se vio anteriormente, en un sistema de altas prestaciones es de gran importancia el tiempo que una simulación paralela está sincronizándose, ya que la comunicación entre los diferentes nodos de computación es una de las causas de overhead que afectan directamente la escalabilidad del sistema. Después de estudiar su rendimiento, se ha observado que su implementación con la librería μ sik presenta algunos problemas de escalabilidad. Estos problemas se deben principalmente al gran número de comunicaciones que tienen lugar en el sistema, principalmente debido a la sincronización de los tiempos de simulación locales para cada procesador con el tiempo de simulación global.

En la figura 2 se muestra una traza de una ejecución del simulador Yades generada en el supercomputador Marenostrom 3 (ver arquitectura en 12.1.1) con 4 procesadores, que no incluye migraciones. En esta ejecución se simulan 4 regiones, una por cada procesador. Se puede observar que todas las zonas amarillas de la imagen representan la sincronización que se está llevando a cabo entre los diferentes hilos de computación (Threads), éstas franjas de comunicación introducen un overhead temporal importante en la aplicación que no es necesario, ya que, a menos que se estén simulando migraciones, no son necesarias tantas comunicaciones entre los diferentes threads. En esta traza, las franjas amarillas representan la comunicación entre las diferentes unidades de procesamiento del tiempo de simulación.

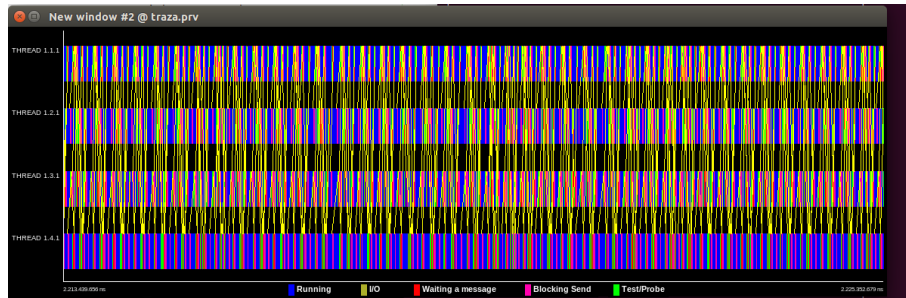
Una vez planteado el problema actual, es necesario realizar una evaluación del simulador ya



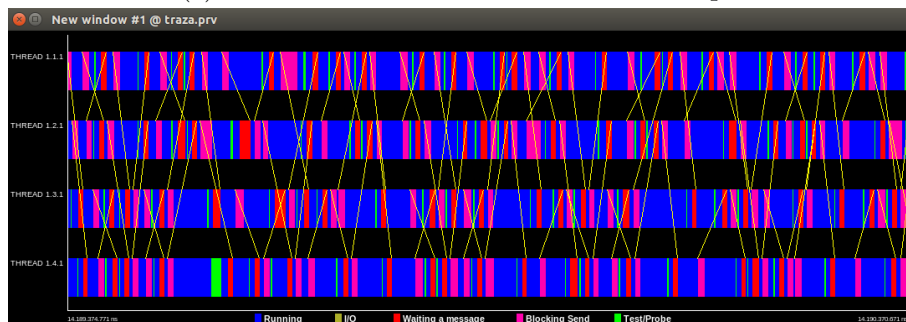
(a) Traza con ejecución de Yades completa.



(b) Traza Yades: sincronización zoom = 1pt.



(c) Traza Yades: sincronización zoom = 2pt.



(d) Traza Yades: sincronización zoom = 3pt.

Figura 2: Traza del simulador Yades en MN3.



existente Yades, del programario similar y de las librerías de simulación paralela que hay en el mercado, para proporcionar una solución a la problemática expuesta.

2. Estado del arte

Actualmente en el mercado hay varias librerías de simulación que proporcionan herramientas para realizar simulaciones sociales con modelos basados en agentes.

Asimismo, existen varias herramientas de simulación que presentan características similares a Yades y es necesario evaluar todas las alternativas existentes, tanto las librerías como el programario disponible de simulación, para tomar la decisión de implementación correcta.

Para realizar la elección de la herramienta de simulación que se empleará en este trabajo se realiza una adaptación del proceso de evaluación y selección de software llamado MOSCA+ (Modelo Sistemático de la Calidad), que permite evaluar software de simulación de eventos discretos [Rincón et al., 2003] en el que se establecen cuatro niveles (del 0 al 3) que siguen el siguiente orden: dimensiones, categorías (funcionalidad, usabilidad y eficiencia), características y criterios.

2.1. Análisis de las herramientas de simulación

A pesar de que existen numerosas herramientas en el ámbito de la simulación social que permiten simular modelos basados en agentes [Montañola-Sales et al., 2014], la mayoría de ellas ejecutan solamente simulaciones secuenciales y tienen una escalabilidad reducida. En los últimos años, con la creciente popularidad del área, han surgido algunas herramientas que aprovechan la arquitectura de altas prestaciones para llevar a cabo simulaciones de gran escala.

Para llevar a cabo el análisis MOSCA+ de las herramientas de simulación, en primer lugar se realiza una lista larga (LL) con todas las herramientas que se conocen y que se pueden usar, 10 en total, y que están en el mercado actualmente disponibles. También se contempla la posibilidad de desarrollar una herramienta propia que se adapte a las necesidades específicas. A continuación se presenta la lista:

- Repast-HPC [Collier and North, 2011]
- DMASON [Cordasco et al., 2011]
- Pandora [Rubio-Campillo]
- Flame [Richmond et al., 2010]
- Yades [Montañola-Sales, 2014]
- EPG-sim [Poulsen and Yew, 1993]
- Swarm [Minar et al., 1996]

- Netlogo [Tisue and Wilensky, 2004]
- OpenMole [Reuillon et al., 2013]
- Parsec [Bagrodia et al., 1998]
- Herramienta hecha a medida

En segundo lugar, se realiza una lista mediana (LM), en la que se descartan algunas herramientas de simulación debido a que no cumplen con algunos de los objetivos principales para la realización de este trabajo. Debido a las características mencionadas en la sección anterior sobre las ventajas de utilizar ABM para la simulación social, se desea continuar con esta aproximación por lo que se ha establecido como característica imprescindible que las herramientas a contemplar empleen un modelo basado en agentes. Además se desea probar este sistema en arquitectura paralela, por lo que la herramienta debe soportar esta posibilidad, ya sea en un supercomputador dedicado como Marenstrum o en un cluster.

Para finalizar, como se trata de un proyecto universitario, de investigación y con un presupuesto reducido, se desea emplear una herramienta que sea de código libre, para tener acceso al código fuente para modificar algunos aspectos en el caso de que sea necesario, tener una amplia red de soporte, que sea un producto probado por una amplia comunidad y que, además, se pueda obtener de manera gratuita. Como último criterio, se valora que esté acorde con los valores que hasta el presente se han seguido en esta línea de investigación.

La posibilidad de desarrollar una herramienta de simulación a medida se ha descartado debido a la fuerte inversión tanto económica como en tiempo de dedicación que requeriría.

A continuación se presenta la LM con los programas que cumplen con estas características y se realiza una descripción de estas herramientas:

- Repast-HPC
- DMASON
- Pandora
- Flame
- Yades

2.1.1. Herramientas de simulación paralela basadas en agentes

Una de las más conocidas es Repast-HPC (Repast for High Performance Computing)[Collier and North, 2011]. Se trata de una plataforma para el modelado basado en agentes a gran escala,

que permite trabajar en entornos de altas prestaciones [Collier and North, 2012]. Está pensada para usuarios con conocimientos de C++ y acceso a supercomputadores. Utiliza un esquema de sincronización conservativo, el espacio de simulación está dividido entre los procesadores, y permite el envío de información o agentes entre los procesadores contiguos.

Otra herramienta popular es DMASON [Cordasco et al., 2011], la versión paralela de la librería MASON para escribir y ejecutar modelos de simulación basados en agentes. DMASON está desarrollada en Java y permite a los usuarios crear su modelo basado en agentes y usarlo en un sistema de computación distribuido. La distribución del espacio la realiza el usuario, el cual es el responsable de mantener el balanceo de carga entre las diferentes regiones de simulación, que se comunican con las regiones vecinas en cada paso de simulación.

Con un esquema de funcionamiento similar a Repast-HPC encontramos Pandora [Rubio-Campillo], un framework basado en agentes que permite la ejecución en entornos de altas prestaciones, de código libre. Actualmente se utiliza para simular sistemas de sociedades antiguas y su relación con el medio ambiente. Utiliza un esquema de sincronización por tiempo y no por eventos. Para su utilización, el usuario debe tener cierto conocimiento de C++ o Python para poder implementar el comportamiento de los agentes. El espacio de simulación está dividido entre los diferentes procesadores o nodos de computación, que comparten la información con los procesadores contiguos a cada paso de simulación.

Mientras que Repast-HPC, DMASON y Pandora están pensadas para el prototipaje de modelos basados en agentes, FLAME [Richmond et al., 2010] es un framework pensado para la paralelización automática de estos modelos de simulación. Se trata de una herramienta de modelización basada en agentes que usa arquitectura de altas prestaciones y sistemas de unidades de procesamiento gráfico (GPU) para modelos celulares. Permite generar el modelo de agentes automáticamente, sin que el usuario deba tener conocimientos de programación específicos, automatizando además el balanceo de cargas entre las diferentes regiones de simulación. Además permite generar un sistema de visualizaciones en tiempo real.

El simulador objeto de este estudio, Yades [Montañola-Sales, 2014], no es una herramienta de uso general como las precedentes sino que está pensada para un uso específico en simulación demográfica. Yades emplea la librería de simulación paralela μ sik, que soporta diferentes algoritmos de sincronización (conservativo y optimista). En concreto, Yades emplea un algoritmo de sincronización optimista, en el que la ventana de simulación va avanzando hasta que sucede un evento que compromete el orden natural de ejecución de la simulación, en cuyo caso se realiza un retroceso del sistema hasta el momento del conflicto para asegurar la correcta consecución de eventos. Basándose en la librería μ sik, Yades emplea LP (Procesos Lógicos) para implementar los agentes. Los LP se comunican entre ellos mediante el envío de mensajes usando el protocolo estandarizado MPI (Message Passing Interface)[Pacheco, 1997]. Cada LP está mapeado en un proceso físico que puede ejecutarse en un procesador. Por tanto, es posible ejecutar múltiples LP en cada procesador.

Una vez analizadas las herramientas de simulación de la LM, se realiza una lista corta (LC)



en la que se establecen todas las métricas obligatorias para la preselección de la herramienta.

Las métricas no obligatorias servirán para poder dimensionar la idoneidad de cada herramienta para la realización de este trabajo. A cada métrica se le asigna un valor de importancia que irá del 1 (muy poco importante) al 5 (muy importante), con el objetivo de poder calcular la tasa de calidad (TC) de cada categoría y la tasa global de todo el software. En la tabla 1 aparecen resaltadas en gris dos de las métricas imprescindibles: el coste de la herramienta y si es posible ejecutarla en arquitectura paralela.

| Categoría | Características | Sub-categorías | Métricas | Peso | Repast-HPC | DMASON | Pandora | Flame | Yades |
|--------------------------------|---------------------------------------|-----------------------------------|--|------|------------|--------|---------|-------|-------|
| Funcionalidad | Ajuste a los propósitos | Coste | Gratuito | 5 | 5 | 5 | 5 | 5 | 5 |
| | | Paradigma de simulación | Genera el modelo de simulación | 5 | 0 | 0 | x | 5 | 5 |
| | | | Permite modelar sistemas sociales | 5 | 5 | 5 | 0 | 0 | 5 |
| | | Estadísticas/reports, | Posibilidad de exportar resultado/traza | 4 | x | x | x | x | 4 |
| | | Distribuciones de probabilidad | Tiene una colección estándar de distribuciones | 3 | 3 | 3 | 0 | x | 3 |
| Experimentación | Permite realizar réplicas | 3 | 0 | x | x | x | 5 | | |
| Subtotal categoría | | | | 8 | 8 | 0 | 5 | 22 | |
| Eficiencia | Interoperabilidad | Plataforma | Funciona en SO Linux | 5 | 5 | x | x | x | 5 |
| | Eficiencia | Ejecución paralela | Permite la ejecución paralela | 5 | 5 | 5 | 5 | 5 | 5 |
| | Utilización de recursos | Madurez | Casos conocidos de éxito con el software | 1 | 1 | x | x | x | 1 |
| Subtotal categoría | | | | 6 | 0 | 0 | 0 | 6 | |
| Facilidad de uso | Facilidad de compresión y aprendizaje | Ayuda y documentación | Tiene documentación/manual de usuario | 2 | 2 | 2 | 2 | 2 | 2 |
| | Interfaz gráfica | Representación gráfica del modelo | Se representa el modelo gráficamente | 1 | 1 | 1 | 1 | 1 | 0 |
| Subtotal categoría | | | | 3 | 3 | 3 | 3 | 2 | |
| Sostenibilidad | Social | Perdurable en el tiempo | Independiente del hardware | 4 | x | x | x | x | 4 |
| | | Mantenibilidad | No obsoleto | 3 | 3 | 3 | 3 | 3 | 3 |
| | Económico | Tipo de licencia | Opensource | 2 | 1 | 1 | 2 | 2 | 2 |
| Subtotal categoría | | | | 4 | 4 | 5 | 5 | 9 | |
| TCGP para cada Software | | | | 21 | 15 | 8 | 13 | 44 | |

Tabla 1: Análisis Mosca+ para la selección del software de simulación

Después de aplicar todas las métricas y asignar las diferentes puntuaciones, la herramienta que se ajusta mejor a los objetivos del proyecto ha sido Yades, debido a que es la herramienta que ha obtenido mejor puntuación en la valoración.

Como consecuencia de este análisis se puede continuar con el objetivo que se planteó inicialmente en este trabajo, que es el de analizar y optimizar el simulador Yades, ya que presenta claras ventajas frente a las otras opciones disponibles en el mercado y su mejora supondrá un avance en el área de investigación. Por lo tanto, a continuación se realizará un análisis y comparativa de las librerías de simulación que hay actualmente en el mercado con la que se utiliza actualmente, la μ sik.

2.2. Análisis de las librerías de simulación

Para esta elección se realiza una adaptación del proceso de evaluación y selección de software MOSCA+ a todas las librerías de simulación que hay en el mercado actualmente.

En primer lugar, se realiza una lista larga (LL) en la que se evalúan 11 librerías de simulación paralela. También se contempla la posibilidad de realizar una librería de simulación paralela hecha a medida.

- μ sik [Perumalla, 2004]
- ROSS [Carothers et al., 2002]
- GTW [Das et al., 1994]
- HLA [Fujimoto, 2003]
- SPEEDES [Steinman, 1992]
- WARPED [Martin et al., 1996]
- ClusterSim [Goes et al., 2004]
- OMNET++ [Varga, 2001]
- SimX [Thulasidasan et al., 2012]
- SimPy [Muller and Vignaux, 2003]
- ROOT-Sim [Pellegrini et al., 2011]
- Librería hecha a medida

A esta lista se le aplican unas métricas básicas con las que se descartan varias librerías que no cumplen con los parámetros básicos. La lista mediana tiene como requisito principal que la librería de simulación empleada permita la ejecución de la simulación en paralelo y sea de licencia gratuita. La posibilidad de hacer una librería de simulación paralela a medida queda descartada por falta de tiempo y por la importante inversión de recursos que requeriría.

A continuación se presenta la LM de librerías que han pasado el primer filtro de selección:

- μ sik
- ROOT-Sim
- ROSS
- HLA
- GTW
- Warped
- Omnet++
- SimX
- ClusterSim

2.2.1. Herramientas PDES para simulaciones basadas en agentes

μ sik es una librería para la implementación de simuladores discretos en paralelo que permite cambiar el método de sincronización empleado, ya que está especialmente diseñada para sistemas en los que no se puede predecir qué mecanismo de sincronización tendrá un rendimiento mejor [Perumalla, 2004]. Como se ha podido observar después de aplicar estudios de rendimiento al simulador Yades [Montañola Sales et al., 2015], su implementación con la librería μ sik presenta algunos problemas de escalabilidad debido al gran número de comunicaciones que tienen lugar en el sistema, principalmente debido a la sincronización de los tiempos de simulación locales para cada procesador con el tiempo de simulación global.

Existen numerosas alternativas a μ sik que permiten la paralelización de simulaciones discretas por eventos en entornos distribuidos y paralelos. Los frameworks de simulación paralela por eventos discretos más representativos son ROSS (Rensselaer's Optimistic Simulation System) [Carothers et al., 2002] y GTW [Das et al., 1994]. Ambos están restringidos a sistemas homogéneos de multiprocesadores con memoria compartida, cosa que restringe la arquitectura en la que se pueden ejecutar los modelos. Están orientados para trabajar con modelos de simulación a gran escala, utilizando algoritmos de sincronización optimista y conservativa. Para alcanzar un

alto rendimiento a nivel de paralelismo, utilizan una técnica llamada computación inversa (Reverse computation) al igual que μsik . Esta técnica para implementar la sincronización optimista consiste en que, en el momento de realizar un rollback por un estado inconsistente, en lugar de ir almacenando los estados anteriores se recalculan los eventos en orden inverso [Yaun et al., 2003]. Uno de sus inconvenientes es la dificultad que representa para los usuarios este hecho, ya que deben escribir los eventos en orden inverso. Por ello, μsik proporciona también la opción de hacer rollback basándose en almacenar los distintos cambios de estado, con la consiguiente necesidad de memoria que esto implica.

Desde el Departamento de Defensa de los Estados Unidos existe la librería HLA [Fujimoto, 2003]. Sin embargo, esta solución no se ajusta bien a la integración de entidades autónomas como son los procesos lógicos, ni proporciona primitivas para facilitar la planificación eficiente de procesos. [Jha and Bagrodia, 1994] plantearon un framework para la paralelización de sistemas discretos por eventos que permitía escoger el mecanismo de sincronización, pero no evaluaron las implicaciones en el rendimiento de la herramienta que esto conllevaba.

SPEEDES [Steinman, 1991] es un software comercial para la simulación optimista en entornos distribuidos pero que no ha sido probado a gran escala, al igual que WARPED [Martin et al., 1996]. Otras librerías son ClusterSim [Goes et al., 2004] y OMNET++ [Varga, 2001], que permiten especificar modelos de simulación paralela en Java y C++ respectivamente.

Una alternativa basada en Python es la librería SimX [Thulasidasan et al., 2012], que permite el desarrollo de prototipos de modelos de simulación discreta por eventos genéricos para ser ejecutados tanto en arquitectura multi-core como en clústeres de computadores. La ventaja de SimX es que, a pesar de que su motor está programado en C++, permite especificar los modelos enteramente en Python, lenguaje más cercano a los desarrolladores de modelos sociales. Además de la paralelización de objetos y eventos, SimX permite también la ejecución de procesos de forma concurrente con poco overhead. También basada en Python se encuentra SimPy [Muller and Vignaux, 2003], que permite la paralelización de modelos de simulación, aunque de forma poco transparente.

Finalmente, se encuentra ROOT-Sim (ROme OpTimistic Simulator) [Pellegrini et al., 2011], un kernel de simulación basado en PDES que utiliza un paradigma optimista de sincronización de eventos. Es una librería estática que puede ser vinculada por ejecutables que usan modelos de simulación utilizando el estándar de programación ANSI-C, como si fueran completamente secuenciales. El modelo de simulación está basado en eventos: cada LP procesa eventos, su avance en el Logical Virtual Time (LVT) está conectado a su ejecución y los LP se comunican a través de mensajes. Cada evento se identifica por un código numérico, que está definido por el nivel lógico de la aplicación. La ventaja de ROOT-Sim es que implementa mecanismos de balanceo de carga automáticos, de forma que el desarrollador no necesita tenerlo en cuenta, como sucede con μsik . Como contrapartida, actualmente ROOT-Sim está diseñada para ejecutarse en paralelo en varios procesadores de clusters o en supercomputadores pero siempre dentro del mismo nodo de computación, debido a que no utiliza MPI, sino pthreads y por tanto no pueden establecer comunicaciones entre diferentes nodos. Sin embargo, los desarrolladores están



actualmente trabajando en una versión que permita la explotación de múltiples nodos con MPI.

Según los análisis de rendimientos aplicados a ROOT-Sim en comparación con otras librerías de simulación [Vitali et al., 2012], ejecutando bajo la misma arquitectura, ROOT-Sim ha presentado una buena escalabilidad.

Una vez analizadas las librerías de simulación de la LM, se realiza una lista corta (LC) en la que se establecen todas las métricas obligatorias para la preselección de la librería.

| Categoría | Características | Sub-categorías | Métricas | Peso | μ_{sik} | ROOT-Sim | HLA | Warped | Omnet++ | ROSS | GTV | SimX | Cluster Sim |
|--------------------------------|--|-------------------------|---------------------------------------|------|-------------|----------|-----|--------|---------|------|-----|------|-------------|
| Funcionalidad | Ajuste a los propósitos | Cost | Gratuit | 5 | 5 | 5 | X | 5 | 5 | 5 | 5 | 5 | 5 |
| | | Paradigma de simulación | Algoritmo sincronización optimista | 5 | 5 | 5 | 5 | 5 | X | 5 | 5 | 5 | X |
| | | | DEV | 5 | 0 | 5 | 0 | 0 | X | 0 | 0 | 5 | X |
| | | Lenguaje | C/C++ | 5 | 5 | 5 | X | 5 | 5 | X | X | 0 | 0 |
| Subtotal categoría | | | | | 10 | 15 | 5 | 10 | 5 | 5 | 5 | 10 | 0 |
| Eficiencia | Interoperabilidad | Plataforma | Funciona en SO Linux | 5 | 5 | 5 | X | X | 5 | 5 | 5 | X | X |
| | Eficiencia | Ejecución paralela | Permite la ejecución paralela | 5 | 5 | 5 | 3 | 5 | 5 | 3 | 3 | 5 | 5 |
| | | Utilización de recursos | Balanceo de carga | 5 | 0 | 5 | 0 | 0 | X | 0 | 0 | 5 | X |
| Subtotal categoría | | | | | 10 | 15 | 3 | 5 | 10 | 8 | 8 | 10 | 5 |
| Facilidad de uso | Facilidad de comprensión y aprendizaje | Ayuda y documentación | Tiene documentación/manual de usuario | 2 | 2 | 2 | 0 | X | X | 2 | 2 | 2 | X |
| | | Madurez | Casos de éxito conocidos | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| Subtotal categoría | | | | | 3 | 3 | 0 | 0 | 0 | 3 | 3 | 3 | 0 |
| Sostenibilidad | Social | Perdurable en el tiempo | Independiente del hardware | 3 | 3 | 3 | X | X | X | X | X | 3 | X |
| | | Mantenibilidad | No obsoleto | 3 | 3 | 3 | X | X | X | 3 | 3 | 0 | X |
| | Económico | Tipo de licencia | OpenSource | 2 | 1 | 2 | 0 | 1 | 1 | X | X | 2 | 1 |
| Subtotal categoría | | | | | 7 | 8 | 0 | 1 | 1 | 3 | 3 | 5 | 1 |
| TCGP para cada Software | | | | | 30 | 41 | 8 | 16 | 16 | 19 | 19 | 28 | 6 |

Tabla 2: Análisis Mosca+ para la selección de la librería de simulación

Finalmente, una vez aplicada la última fase del proceso de selección para la librería de simulación paralela, como se muestra en la tabla 2, se ha podido escoger entre las diferentes librerías de simulación paralela. Una vez analizadas las herramientas que permiten desarrollar las mismas funcionalidades que la actual librería μ sik, se ha decidido utilizar ROOT-Sim como alternativa a μ sik para mejorar el rendimiento de Yades, ya que es la que presentó mejores resultados en el proceso de selección de software para su integración con Yades.

2.3. Conclusión sobre el análisis de la competencia

El simulador Yades surge como un proyecto parte de una tesis doctoral desarrollada en el Barcelona Super Computing Center (BSC) y la Universidad de Lancaster, en colaboración con el InLab-FIB de la Universitat Politècnica de Catalunya - BarcelonaTech. El código de este trabajo aún no está publicado, por lo que, en este momento, no hay otros grupos de investigación trabajando en posibles soluciones o implementaciones alternativas que den solución a los problemas de rendimiento que presenta Yades. Además, después de realizar un análisis del mercado y debido a la complejidad que el simulador encierra al estar pensado para ejecutarse en arquitecturas paralelas que realicen una repartición de los recursos en los diferentes nodos de cálculo, se ha podido observar que actualmente no hay una solución que permita integrarse de manera sencilla y directa con Yades o una solución que presente mejores características que Yades, sino que para solucionar la problemática existente es necesario integrar Yades con ROOT-Sim para desarrollar una solución que se adapte a las necesidades actuales.

3. Propuesta de solución

Para solucionar la problemática descrita anteriormente en la sección 1.2, en este proyecto se ha decidido realizar un análisis de mercado en cuanto a librerías de simulación paralela y a herramientas de simulación. La herramienta de simulación social escogida fue Yades y la librería de simulación fue la librería paralela por eventos discretos de propósito general ROOT-Sim, debido a que presenta más y mejores características en comparación con la librería actual, μsik . El objetivo del proyecto es realizar un análisis de ambas herramientas de simulación, Yades y ROOT-Sim, con el fin de evaluar la posibilidad de realizar un cambio en Yades con el objetivo de integrar las capacidades de ROOT-Sim para la simulación de dinámicas socio-demográficas.

Para realizar la integración entre la arquitectura de Yades y el funcionamiento de ROOT-Sim, se ha decidido descartar la librería μsik , encargada de gestionar las comunicaciones entre nodos y procesadores usados por el simulador Yades para usar aquellas funcionalidades que ROOT-Sim provee en cuanto a paralelización y sincronización de la simulación.

Este cambio, implicará en primer lugar, un análisis en profundidad de ambas herramientas y el diseño de una posible solución software. En segundo lugar, implicará la reimplementación del simulador Yades y adaptación de su modelo a esta nueva librería que será la encargada de gestionar la planificación y entrega de los eventos de simulación y las comunicaciones entre las diferentes unidades de procesamiento. Por último, para evaluar la mejora de la solución aportada se realizará un análisis de rendimiento con la nueva implementación, junto con la validación de los resultados y desarrollo de una comparativa de rendimiento frente a su versión inicial, para evaluar si el cambio se traduce en una mejora en la escalabilidad del simulador Yades.

4. Objetivos y alcance del proyecto

4.1. Objetivos

El objetivo principal de este trabajo final de grado es el de realizar un análisis de dos herramientas de simulación paralela para la simulación de dinámicas socio-demográficas en el simulador Yades.

En primer lugar, se analiza la implementación actual de Yades que emplea la librería μ sik para la paralelización y sincronización de la simulación por eventos discretos. En segundo lugar, se estudia la librería de simulación paralela ROOT-Sim, que permite aplicar simulación basada en eventos discretos para entornos paralelos aprovechando el esquema de memoria compartida.

El objetivo de este análisis es el de evaluar la posibilidad de integrar Yades con ROOT-Sim para realizar una optimización de rendimiento y aumentar la escalabilidad del simulador Yades, reduciendo así su tiempo de computación.

Los objetivos derivados de este trabajo son, por un lado, abrir una línea de investigación que permita optimizar el rendimiento del simulador Yades. Por otro lado, probar el funcionamiento de la herramienta de simulación ROOT-Sim en un modelo socio-demográfico, y a gran escala.

Por último, como resultado de este proyecto se inicia una colaboración entre el inLab FIB y el grupo de High Performance and Dependable Computing Systems del departamento de Computer Science and System Engineering de la Università di Sapienza, Roma.

4.2. Alcance

Para el desarrollo de este proyecto se emplea la herramienta de simulación Yades, pero solo un subconjunto de ésta. El simulador Yades cuenta con dos partes bien diferenciadas, la primera es la librería que genera la simulación y la segunda es una interfaz gráfica, disponible en <https://yades.fib.upc.es/>, en la que los modeladores y científicos sociales pueden definir su modelo sin necesidad de tener conocimientos específicos de programación. Actualmente ésta interficie genera un código en C++ que puede ser compilado y ejecutado en arquitectura paralela que precisa μ sik. Esta herramienta queda fuera del alcance de este proyecto y solo se empleará la librería de la herramienta para el modelado de dinámicas socio-demográficas.

Así mismo, se emplean las funcionalidades que proporciona la librería de simulación paralela ROOT-Sim sin entrar a analizar en profundidad diferentes opciones de optimización a bajo nivel que incluye y que se dejarían para un trabajo futuro.

La realización de este proyecto consta de 3 fases. La primera fase se basa en el análisis en profundidad del simulador Yades, así como de las herramientas ROOT-Sim y μ sik. Su objetivo

es el de comprender a fondo el funcionamiento e implementación de cada una para, en las fases siguientes, poder desvincular Yades de μsik .

En la segunda fase se realiza un estudio en profundidad de la posibilidad de integrar ROOT-Sim con Yades, de manera que se pueda sustituir la librería μsik , y para evaluar si esta nueva implementación puede ser una solución a los problemas de rendimiento y escalabilidad que Yades presenta actualmente. En esta fase es necesario redefinir todas las estructuras de datos, métodos, etc. de Yades, debido a que Yades está escrito en C++ y ROOT-Sim obliga a implementar el código en el lenguaje C. Como la librería μsik gestiona el envío y recepción de eventos y las comunicaciones que tienen lugar entre las diferentes regiones de simulación, este comportamiento pasa a ser responsabilidad de la nueva librería y es necesario adaptar el modelo a las características específicas de ROOT-Sim.

En la última fase se realiza la implementación de una solución software que sustituye la librería μsik con la ROOT-Sim en Yades.

Para comprobar la mejora del rendimiento del simulador se realiza una validación de los resultados obtenidos, un análisis de rendimiento y escalabilidad de la nueva implementación del simulador que permitirá evaluar si la solución representa una optimización de Yades.

A pesar de que queda fuera del alcance de este proyecto, si los resultados obtenidos con ROOT-Sim son buenos, se contempla la posibilidad de continuar con esta línea de investigación en el futuro y ampliar el simulador Yades hacia una solución software que sea adaptable a diversas tecnologías.

5. Planificación

5.1. Calendario

La duración de este proyecto se ha estimado en 6 meses, comenzando el 7 de enero y acabando el 7 de junio. Este proyecto consta de 18 créditos ECTS (Sistema Europeo de Transferencia y Acumulación de Créditos), de los cuales 15 pertenecen al proyecto íntegramente y 3 a la asignatura semipresencial de GEP (Gestió de Projectes).

Para la planificación de todo el proyecto, se ha estimado una carga de trabajo de unas 540 horas aproximadamente, que engloban todas las tareas necesarias para la realización del proyecto, desde la documentación inicial previa a la realización del mismo, así como toda la documentación necesaria para su planificación, contenida dentro de la asignatura de GEP y la documentación de la fase final de la memoria. El cálculo de la dedicación se ha basado en las cifras dadas por la FIB (Facultat d'Informàtica de Barcelona) de 30 horas de dedicación por cada crédito ECTS.

La fecha de finalización se ha estimado teniendo en cuenta las posibles desviaciones que puedan haber en el proyecto, dejando algunas semanas de margen a la fecha de entrega del proyecto que será a finales de junio, por si fuese necesario realizar un reajuste del calendario.

La dedicación de recursos de este proyecto es principalmente de una persona, dedicada 20 horas semanales a su realización. Sin embargo, se contará con la participación de dos personas más implicadas, que realizarán aportaciones de tiempo variable al proyecto.

Debido a la complejidad del proyecto, que implica el desarrollo en diferentes equipos con arquitectura paralela y que engloba diversas áreas de estudio diferentes entre ellas, es posible que se requiera la participación de equipos de investigación externos a éste para posibles consultas y soporte.

5.2. Metodología de trabajo

El desarrollo del proyecto se ha planteado usando una adaptación de la metodología ágil scrum, la cual se basa en el desarrollo del software mediante iteraciones de una duración determinada, conocidas como sprint. En el caso de este proyecto, la duración de cada sprint es de tres semanas. En cada iteración de desarrollo se ejecutan las tareas de análisis, especificación, diseño y test de las diferentes partes de la solución software, aunque solapando las diferentes fases del desarrollo en algunos casos.

El objetivo de la metodología es conseguir, al final de cada sprint, un producto potencialmente entregable al cliente. Estas versiones previas a la solución definitiva del producto, permiten llevar un control exhaustivo del estado del proyecto y de su correcto desarrollo, validando cons-

tantemente que se cumplan las expectativas del cliente, si hay desviaciones temporales, que el producto tenga la calidad esperada, entre otras.

Otro objetivo de la metodología escogida es que se realicen las tareas de manera priorizada, desarrollando en las primeras iteraciones los objetivos más importantes. En consecuencia, se ha conseguido tener una gran capacidad de adaptación a algunas desviaciones que surgen a lo largo del proyecto, haciendo los reajustes necesarios.

En algunas fases de este proyecto también se contempla la utilización de la metodología ágil Kanban, que se basa en trabajar bajo demanda y que el flujo de trabajo se vaya adecuando al ritmo de trabajo del equipo, para de esta manera, poder detectar los cuellos de botella o las principales dificultades que puedan aparecer en el desarrollo del proyecto.

5.3. Métodos de validación

Como consecuencia de desarrollar el proyecto siguiendo una metodología ágil, la validación del sistema se realiza de manera iterativa e incremental, al final de cada iteración de desarrollo. Al finalizar cada sprint, se valida el producto desarrollado contra el ya existente. Las pruebas incrementales permiten llevar un control exhaustivo del software. Asimismo, las revisiones periódicas con las distintas partes interesadas en el proyecto permiten validar los requisitos de cada fase.

Se aplican pruebas funcionales y no funcionales al sistema con el objetivo de validarlo completamente. Entre las pruebas no funcionales más importantes se encuentran las pruebas de rendimiento y de coherencia, corrección e integridad de los datos de salida del simulador, así como un análisis y comparativa de los resultados, tanto finales como intermedios.

Al final del proyecto se reserva un tiempo para realizar test adicional del sistema, que incluye pruebas de estrés y comprobar que todo funciona correctamente y cumple con los requisitos establecidos antes de entregar el producto final.

5.4. Descripción de las tareas

5.4.1. Planificación inicial

Para llevar a cabo la planificación temporal del proyecto, en primer lugar se define el alcance de éste, junto con los objetivos y los posibles obstáculos que se puedan encontrar. Seguidamente se realiza un análisis de requerimientos, tanto funcionales como no funcionales. En este análisis inicial también se define una planificación temporal inicial y un estudio de la viabilidad económica del proyecto.

Toda la planificación temporal, definición de objetivos y alcance del proyecto, se realiza al

comienzo de éste para la asignatura de GEP, que ocupa 5 semanas del calendario lectivo del cuatrimestre de primavera 2014-2015, comenzando el 16 de febrero y terminando el 22 de marzo con la presentación final de la planificación del proyecto.

Esta fase es de vital importancia para poder estimar la duración temporal del proyecto y poder delimitar correctamente los recursos, tanto físicos como materiales, que se usan en su realización.

Para la realización de la planificación del proyecto, se realiza una planificación por sprints de duración de 3 semanas cada uno. La planificación total se estima en 7 sprints que incluyen todas las fases de realización del proyecto. Sin embargo, hasta el 3^o no se tiene un producto potencialmente entregable.

- 1^o Sprint: el objetivo principal de esta primera fase es el de analizar en profundidad la problemática, las posibles soluciones software que hay y la elección de una de ellas para su implementación con el simulador Yades. Además en esta fase se contempla también la instalación del entorno de trabajo necesario para el desarrollo del proyecto en el equipo de desarrollo local.
- 2^o Sprint: en esta fase se ha priorizado la instalación del entorno de desarrollo en MN3 (Marenostum 3), junto a la familiarización con éste, que requerirá de un fuerte aprendizaje autónomo por parte del alumno. También se realiza la instalación del entorno en Capitano, una de las máquinas del cluster de la Università di Sapienza, Roma.

5.4.2. Iteraciones de desarrollo

La finalidad de la metodología empleada es la de poder ir contrastando con el cliente si el proyecto va bien encaminado, si hay desviaciones temporales y si cumple con sus expectativas, tanto en resultado final como en calidad. Además al final de cada sprint se contempla una fase de test de las nuevas funcionalidades añadidas, que permite verificar el correcto funcionamiento de la aplicación.

Otro de los objetivos de la metodología escogida es el de realizar las tareas de manera priorizada por el cliente en función del valor que le aportan a él. De esta manera, en las primeras iteraciones se pretende alcanzar los objetivos más importantes del proyecto, y dejar para las fases posteriores los objetivos que dan un valor añadido al proyecto.

Las iteraciones de desarrollo se han definido de la siguiente manera:

- 3^o Sprint: se comienza la integración del simulador Yades con ROOT-Sim definiendo las estructuras de datos que se usarán en las fases siguientes de desarrollo y se definen los métodos principales que permiten la interacción entre los diferentes objetos de simulación. Estos son: nacimientos, defunciones y el “dispatcher” o planificador de eventos. El entregable de

esta fase consiste en una versión del simulador Yades capaz de simular correctamente los eventos básicos de simulación: nacimientos y defunciones, a partir de una población inicial.

- 4º Sprint: se desarrollan los métodos secundarios de simulación: inmigraciones, estados maritales y económicos de la población. Además se comienza a implementar el modelo para el caso de estudio de Gambia. El entregable de esta fase consiste en una versión del simulador Yades capaz de simular correctamente a partir de una población inicial, los eventos secundarios de simulación: estados maritales y económicos y migraciones, además de una primera aproximación del modelo de Gambia.
- 5º Sprint: se termina de implementar el modelo para el caso de estudio de Gambia, añadiendo todas las funcionalidades que éste ha de tener. El entregable de esta fase consiste en una versión del simulador Yades capaz de reproducir todos los eventos de simulación descritos en la planificación. Además se completa el modelo de Gambia para su validación.
- 6º Sprint: se realiza una comparativa de los resultados obtenidos durante los experimentos de simulación realizados. Además se realizan diversos test de la integración del sistema final. El entregable de esta fase consiste en el simulador completo, con todas las funcionalidades añadidas, testeadas y funcionando tanto en arquitectura paralela, como en local.
- Fase final: una vez se han completado los sprints de desarrollo, se han validado los resultados y aplicado los test de conjunto finales a la aplicación, se termina de escribir la memoria del proyecto y se prepara la presentación final. Para esta fase se dispone de un periodo de 4 semanas. Sin embargo, como se han dejado algunas semanas de margen para la finalización del proyecto, si se acaba en el tiempo previsto, es posible ampliar esta fase.

5.5. Diagrama de Gantt de la planificación inicial

En la figura 4 se presenta el diagrama de Gantt de la planificación temporal inicial del proyecto.

| | Nombre de tarea | Duración | Comienzo | Fin |
|----|---|----------|--------------|--------------|
| 1 | ▲ Análisis de requisitos inicial | 13 días | mié 07/01/15 | vie 23/01/15 |
| 2 | Estudio de la problemática | 3 días | mié 07/01/15 | vie 09/01/15 |
| 3 | Análisis de la solución actu | 2 días | lun 12/01/15 | mar 13/01/15 |
| 4 | Definición de la solución | 2 días | mié 14/01/15 | jue 15/01/15 |
| 5 | Estudio de la viabilidad económica | 2 días | vie 16/01/15 | lun 19/01/15 |
| 6 | Instalación del entorno de trabajo en local | 4 días | mar 20/01/15 | vie 23/01/15 |
| 7 | Familiarización con el entorno, aprendizaje autónomo e instalación en MN3 | 15 días | lun 26/01/15 | vie 13/02/15 |
| 8 | Definición de estructuras y métodos primarios | 15 días | lun 16/02/15 | vie 06/03/15 |
| 9 | Definición de métodos secundarios, modelo del sistema. Modelo Gambia I | 15 días | lun 09/03/15 | vie 27/03/15 |
| 10 | Modelo de Gambia II | 14 días | lun 30/03/15 | jue 16/04/15 |
| 11 | Comparativa resultados y test integrado final. | 15 días | vie 17/04/15 | jue 07/05/15 |
| 12 | Finalización documentación (Memoria del proyecto) | 15 días | vie 08/05/15 | jue 28/05/15 |
| 13 | Preparación de presentación | 5 días | vie 29/05/15 | jue 04/06/15 |

Figura 3: Explicación del Diagrama de Gantt de la planificación inicial del proyecto

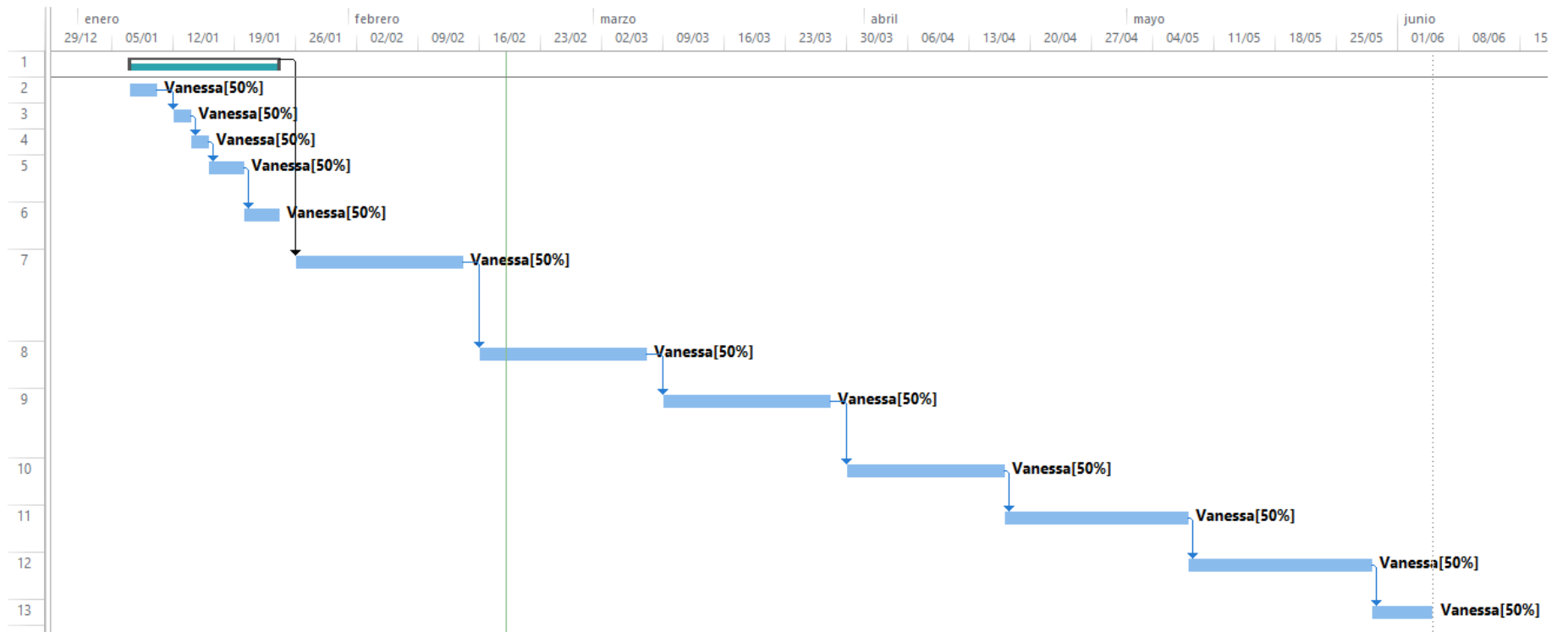


Figura 4: Diagrama de Gantt de la planificación inicial del proyecto

5.6. Posibles obstáculos y plan de contingencia

En la planificación temporal del proyecto se han tenido en cuenta los posibles obstáculos con los que se puede encontrar el equipo de desarrollo, con el fin de mitigar al máximo sus efectos. Sin embargo debido a las características del proyecto, es probable que se produzcan alteraciones significativas en su planificación y por esta razón su finalización se ha planificado con tiempo de antelación a la fecha de entrega, para contrarrestar una posible desviación imprevista y poder tener finalizado el proyecto a tiempo.

Uno de los principales obstáculos del proyecto es la dificultad para entender el contexto de éste, debido a que engloba varias y diferentes áreas de investigación nuevas para el equipo y que requieren de un elevado tiempo de aprendizaje.

Además de la dificultad de comprender la problemática asociada con el proyecto, también se tiene en cuenta: el tiempo a dedicar, la dificultad en el aprendizaje de las herramientas que se emplean y las características específicas y particulares del entorno de trabajo que consiste principalmente en arquitectura paralela, tanto en los equipos de desarrollo como en los equipos donde se despliega finalmente la aplicación.

Otro de los obstáculos destacables está en la dificultad para integrar las herramientas ROOT-Sim y Yades entre ellas, ya que al inicio del proyecto no se tiene constancia que sea factible y aporte buenos resultados. Además estas herramientas pertenecen a dos líneas de investigación diferentes entre ellas y que aún están en marcha y han sido desarrolladas en diferentes partes de Europa con lo que los protocolos de validación de software usados pueden no ser los mismos o no ser estándar, es posible también que aún no hayan sido testeadas intensivamente o que no sean versiones completamente estables.

Por esta razón, en la planificación temporal se dedica una cantidad de horas suficiente a la familiarización con las herramientas y su estudio.

Finalmente el equipo de trabajo está formado por personas que se encuentran físicamente lejos del lugar de desarrollo y entre ellas y esto puede derivar en dificultades de comunicación durante el proyecto, además algunas trabajan actualmente en otros proyectos, hecho que dificulta la comunicación fluida de los grupos y su disponibilidad.

5.7. Diagrama de Gantt de la planificación final

En la figura 6 se muestra el diagrama de Gantt de la planificación final del proyecto.



| | Nombre de tarea | Duración | Comienzo | Fin |
|----|---|----------|--------------|--------------|
| 1 | ▲ Análisis de requisitos inicial | 20 días | mié 07/01/15 | mar 03/02/15 |
| 2 | Estudio de la problemática | 3 días | mié 07/01/15 | vie 09/01/15 |
| 3 | Análisis de la solución actual | 5 días | lun 12/01/15 | vie 16/01/15 |
| 4 | Definición de la solución | 4 días | lun 19/01/15 | jue 22/01/15 |
| 5 | Estudio de la viabilidad económica | 2 días | vie 23/01/15 | lun 26/01/15 |
| 6 | Instalación del entorno de trabajo en local | 6 días | mar 27/01/15 | mar 03/02/15 |
| 7 | Familiarización con el entorno, aprendizaje autónomo e instalación en MN3 | 20 días | mié 04/02/15 | mar 03/03/15 |
| 8 | Definición de estructuras y métodos primarios | 23 días | mié 04/03/15 | vie 03/04/15 |
| 9 | Definición de métodos secundarios, modelo del sistema. Modelo Gambia I | 19 días | lun 06/04/15 | jue 30/04/15 |
| 10 | Modelo de Gambia II | 7 días | vie 01/05/15 | lun 11/05/15 |
| 11 | Comparativa resultados y test integrado final. | 8 días | mar 12/05/15 | jue 21/05/15 |
| 12 | Finalización documentación (Memoria del proyecto) | 21 días | vie 22/05/15 | vie 19/06/15 |
| 13 | Preparación de presentación | 4 días | lun 22/06/15 | jue 25/06/15 |

Figura 5: Explicación del Diagrama de Gantt de la planificación final del proyecto

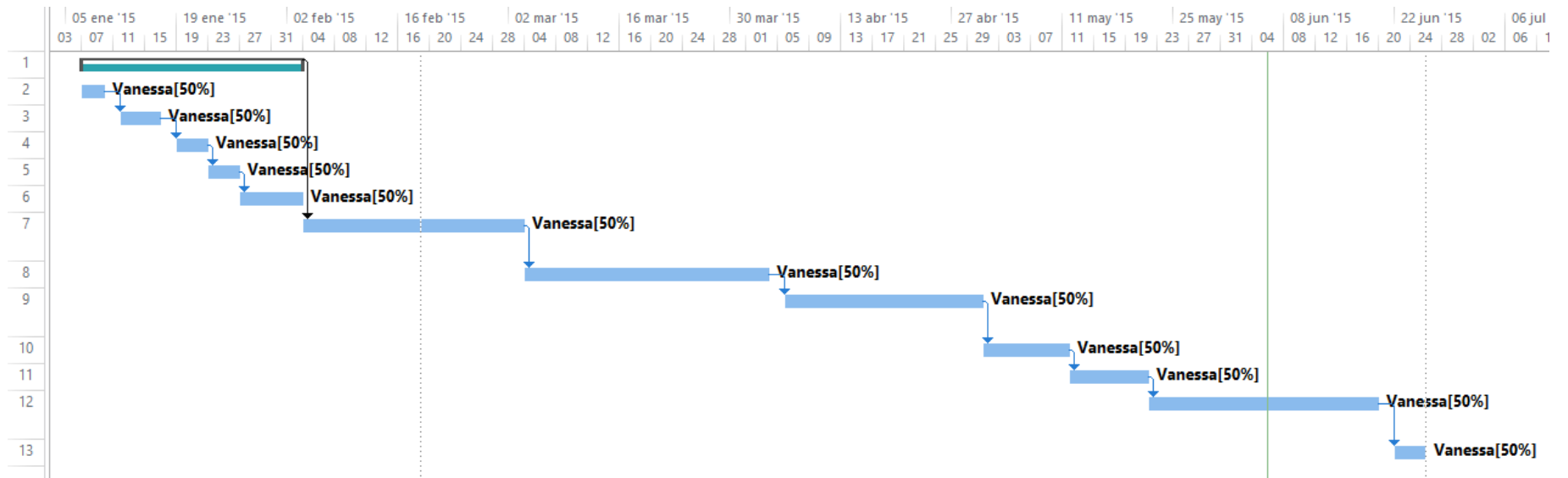


Figura 6: Diagrama de Gantt de la planificación final del proyecto

5.8. Valoración de la metodología utilizada

Para llevar a cabo el proceso de implementación de las historias de usuario relacionadas con las funcionalidades que debe cumplir la aplicación, es decir, para la implementación de los 5 componentes demográficos, se ha seguido un proceso *Walking skeleton* [Varios, c] en el que se selecciona una funcionalidad y se desarrolla de principio a fin.

Dentro de este proceso ágil, además, se han priorizado las tareas que aportan “más valor comercial” a la aplicación, y se ha dejado para el final la tarea que se acordó con el cliente que aportaba “menos valor comercial”. Esta tarea se ha valorado de esta manera ya que el cliente puede prescindir de esa funcionalidad en caso que no se pueda realizar y representa un mayor riesgo tecnológico.

Por esta razón, se decidió comenzar con la inicialización de la población. El objetivo de la primera fase de desarrollo es que se pueda introducir una población inicial en el simulador y que éste la pueda representar correctamente. Seguidamente, se continúa con los nacimientos, muertes y el planificador de eventos. Es primordial que la solución software sea capaz de generar un escenario básico de simulación y que sea capaz de entregar eventos sencillos de tipo nacimiento y muerte.

En la siguiente iteración se desarrollaron los estados económicos, maritales y las migraciones. Una vez se ha superado la cuesta de aprendizaje de la primera iteración se quiere que el producto sea capaz de generar los componentes demográficos que permiten un cambio en el estado de los individuos durante su periodo de vida.

Para finalizar, se ha sustituido la población sintética inicial (población que no contiene datos reales, sino que están adaptados al modelo) por el modelo de Gambia, para poder simular los cambios concretos en este grupo de población.

De esta manera en cada iteración de desarrollo, se dispone de un producto potencialmente entregable que aporta valor al cliente y que cumple con sus expectativas, al haber priorizado los requisitos de más valor al comienzo del desarrollo y de cada fase.

Durante el desarrollo del proyecto se han producido diversas alteraciones que han implicado modificaciones en la planificación inicial. Además de las dificultades mencionadas en el apartado 5.6 también se han encontrado otras, como por ejemplo la falta de compatibilidad de las herramientas empleadas con las diferentes arquitecturas de desarrollo. Este problema se vio cuando se instaló una versión de ROOT-Sim actualizada en Marenostrium 3 y la librería había actualizado su versión de automake para incluir nuevas funcionalidades, sin embargo, en Marenostrium no se contaba con esa versión y no era posible actualizarla. Se tuvo que generar una versión de ROOT-Sim que mantuviera las funcionalidades de la última versión y que a la vez, fuera compatible con la versión 1.10.1 de automake. Este contratiempo supuso un retraso de dos días en el desarrollo.

Otro inconveniente fue las particularidades de la simulación socio-demográfica que requieren

de un uso intensivo de memoria y generan una gran cantidad de eventos, muchos más que otros modelos a medida que la población aumenta. En el caso de Yades y ROOT-Sim, se generaban muchos más eventos que los requeridos en otros modelos de simulación por lo que hubo que actualizar la ventana de eventos de ROOT-Sim para poder contener todos los eventos que el modelo requería. Estas dificultades, aunque han supuesto un retraso en la planificación inicial, se pudieron superar y han significado un valor añadido al proyecto y para las herramientas que se emplean.

Las metodologías ágiles empleadas para el desarrollo del proyecto, en particular scrum y toda la filosofía de testing que representan, han sido de mucha ayuda para la fase de desarrollo, ya que se han podido ir introduciendo los diferentes componentes de la simulación y validando cada una de ellas a medida que se van realizando. Pero sobre todo la principal característica de las metodologías ágiles, la adaptación al cambio, ha sido fundamental para poder desarrollar el proyecto en el tiempo establecido.

Además, durante el transcurso del proyecto, las metodologías ágiles empleadas han permitido ir comprobando que los requisitos marcados por el cliente se han ido cumpliendo, y se han podido corregir las diversas desviaciones temporales que han ido surgiendo a lo largo del proyecto sin necesidad de alterar la fecha de entrega del mismo.

5.9. Recursos

A continuación se presentan los recursos materiales y software que han sido necesarios para el desarrollo del proyecto.

5.9.1. Recursos materiales y de sistema operativo

- Cuenta en Marenostrom 3: entorno de producción (super ordenador dedicado) en el que se ejecuta y se prueba el software.
- Cuenta en Capitano: entorno de producción (cluster) en el que se ejecuta y se prueba el software.
- Ordenador de sobremesa con procesador Intel Core i5 con 8GB de RAM, de 64 bits: entorno local en el que se instala la máquina virtual necesaria para el desarrollo del software.
- Ordenador portátil con procesador Intel Core i3 con 4GB de RAM, de 64 bits: entorno local en el que se instala la máquina virtual necesaria para el desarrollo del software.
- 4 Máquinas virtuales con Ubuntu 14.04 con 4 cores: entornos virtuales en los que se desarrolla el software. Durante la realización del proyecto se utiliza un cluster de máquinas virtuales para probar en local la aplicación pero ejecutando en arquitectura paralela.

- Conexión ADSL
- Electricidad

5.9.2. Recursos para documentar y de comunicación

- Microsoft Office para estudiantes: paquete de software de Microsoft con herramientas para hacer la presentación y documentación.
- Texmaker: programa opensource de escritorio para editar documentos latex.
- ShareLatex online: programa online opensource para editar documentos latex de manera colaborativa.
- Cacao: herramienta gratuita online para realizar diagramas.
- Thunderbird: herramienta gratuita para la gestión del correo electrónico.
- Skype: software que permite la comunicación vía texto, voz y con imagen a través de internet de manera gratuita a todas partes del mundo.

5.9.3. Recursos de desarrollo

- Gdb: debugger opensource para C/C++.
- Valgrind: conjunto de herramientas gratuitas que permiten la depuración de código a nivel de memoria y análisis de rendimiento.
- Paraver-Extrac: herramienta gratuita para realizar estudios de análisis de rendimiento de aplicaciones paralelas.
- Vim: editor gratuito de texto para entornos Unix.
- Geany: editor de texto gratuito con interfaz de ventanas básica.
- Yades: simulador paralelo basado en eventos discretos.
- ROOT-Sim: librería paralela por eventos discretos de propósito general.
- μ sik: librería paralela de propósito general.

5.9.4. Recursos de seguimiento

- GIT: herramienta para el control de versiones. Se contará con un repositorio local en cada punto de trabajo del desarrollo, y un repositorio central donde se confirmarán todos los cambios en el código una vez han sido validados exhaustivamente.
- Bitbucket: servicio de alojamiento basado en web para sistemas de control de versiones opensource online.
- Trello: herramienta online para la gestión del proyecto, control y priorización de las tareas. Se basa en la creación y la gestión de tareas en forma de tarjetas, las cuales seguirán el workflow definido en la metodología ágil escogida:
 - Backlog: tareas a realizar antes de la finalización del proyecto, ordenadas según su prioridad por el product owner.
 - To do: tareas a realizar en el sprint actual. Antes de terminar la iteración vigente todas las tareas en este estado deben finalizarse.
 - Done: tareas terminadas durante el desarrollo del proyecto.

5.9.5. Recursos humanos

- Jefe de proyecto
- Analista
- Diseñador
- Técnico programador
- Tester
- Documentador

6. Análisis del sistema actual

En esta sección se presenta un análisis de las herramientas de simulación empleadas para este proyecto, Yades y ROOT-Sim.

6.1. Descripción del funcionamiento actual de Yades con μsik

En esta sección se presenta un análisis de Yades con su implementación actual con la librería de simulación paralela μsik . Este análisis contiene una explicación de la herramienta de simulación a nivel conceptual y a nivel de rendimiento (ρ).

6.1.1. Descripción de μsik

μsik es una librería de propósito general para la simulación de eventos discretos que se puede ejecutar en entornos paralelos o distribuidos, está construida sobre una arquitectura *Micro Kernel* que consiste en procesos de simulación autónomos. Su autonomía se basa en que cada uno controla y maneja sus propios eventos y decide cómo realiza su sincronización dinámicamente.

Para realizar la sincronización entre eventos, μsik permite emplear un enfoque optimista (state saving-based optimistic) o conservativo (lookahead-based conservative). También permite otras formas más complejas y novedosas de sincronización como la computación inversa (reverse computation) y procesamiento de eventos basado en agregación (aggregation-based event processing).

μsik fue desarrollada por el grupo de Sistemas de Computación Discretos del Oak Ridge National Laboratory (Estados Unidos) [Perumalla, 2004]. Está escrita en C++ y es portable a sistemas Unix y Windows. Emplea una librería llamada libSynk que provee la gestión de la sincronización del tiempo distribuido. μsik adopta un proceso de interacción con visión global donde la simulación está formada por una serie de procesos lógicos, LP, que interaccionan entre ellos y están alojados en un procesador. Los LP tienen un comportamiento totalmente autónomo y pueden determinar por ellos mismos cómo procesar los eventos que reciben. Por ello, Yades utiliza los LP para implementar los agentes del modelo de simulación. Yades utiliza MPI [Pacheco, 1997] para gestionar el envío de mensajes entre los diferentes procesadores a través de eventos. Cada LP está mapeado en un proceso físico (physical process, PP) que puede ejecutarse en un procesador. Por tanto, es posible ejecutar múltiples LP en cada procesador. En la figura 7 se presenta la arquitectura que debe tener cualquier aplicación que use la librería de simulación paralela μsik .

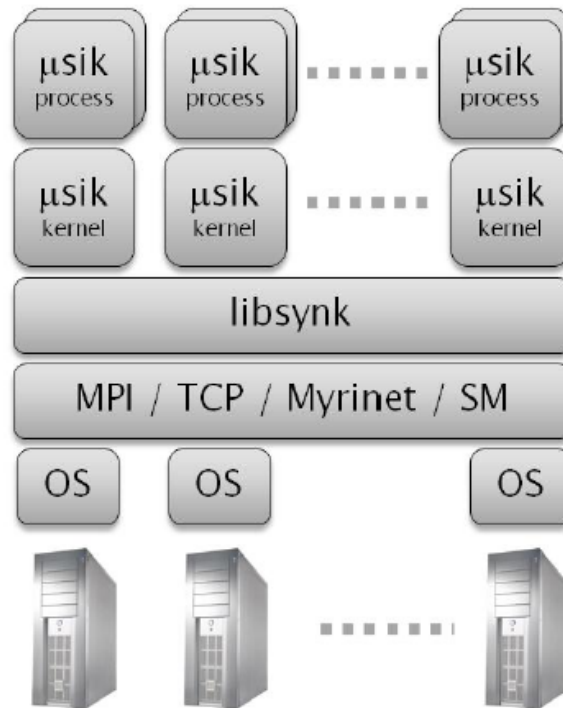


Figura 7: Arquitectura de una simulación paralela usando la librería μsik [Montañola Sales et al., 2015]

6.1.2. Descripción de Yades

Yades (Yet Another DEMographic Simulator) es un simulador de propósito específico para simulación socio-demográfica basado en agentes, diseñado para ejecutarse en arquitectura paralela. Emplea la librería μsik , que permite gestionar los eventos y comunicaciones entre los diferentes LP y nodos de simulación, empleando un algoritmo de sincronización optimista. Al igual que μsik , Yades está desarrollado en C++ y utiliza orientación a objetos. Para esto, todas las clases definidas en Yades heredan de las clases de μsik : `Simulator`, `SimEvent`, `SimProces`, `SimTime`, `SimPID` y `SimEvent` como se puede ver en la figura 8. En ella, se presenta el diagrama de clases UML de la aplicación Yades con la librería de simulación paralela μsik .

Yades usa dos tipos de agentes: las regiones (la clase `Region`) y las unidades familiares (la clase `FamilyUnit`). La clase `FamilyUnit` representa una unidad familiar formada por como máximo dos adultos y un número de hijos definido por el modelador así como parámetros específicos para cada uno.

En la figura 9 se puede observar cómo se realiza el mapeo de los dos tipos de agentes que emplea Yades, regiones y unidades familiares, como procesos lógicos y cómo están distribuidos entre los diferentes procesos físicos (PP) del simulador.

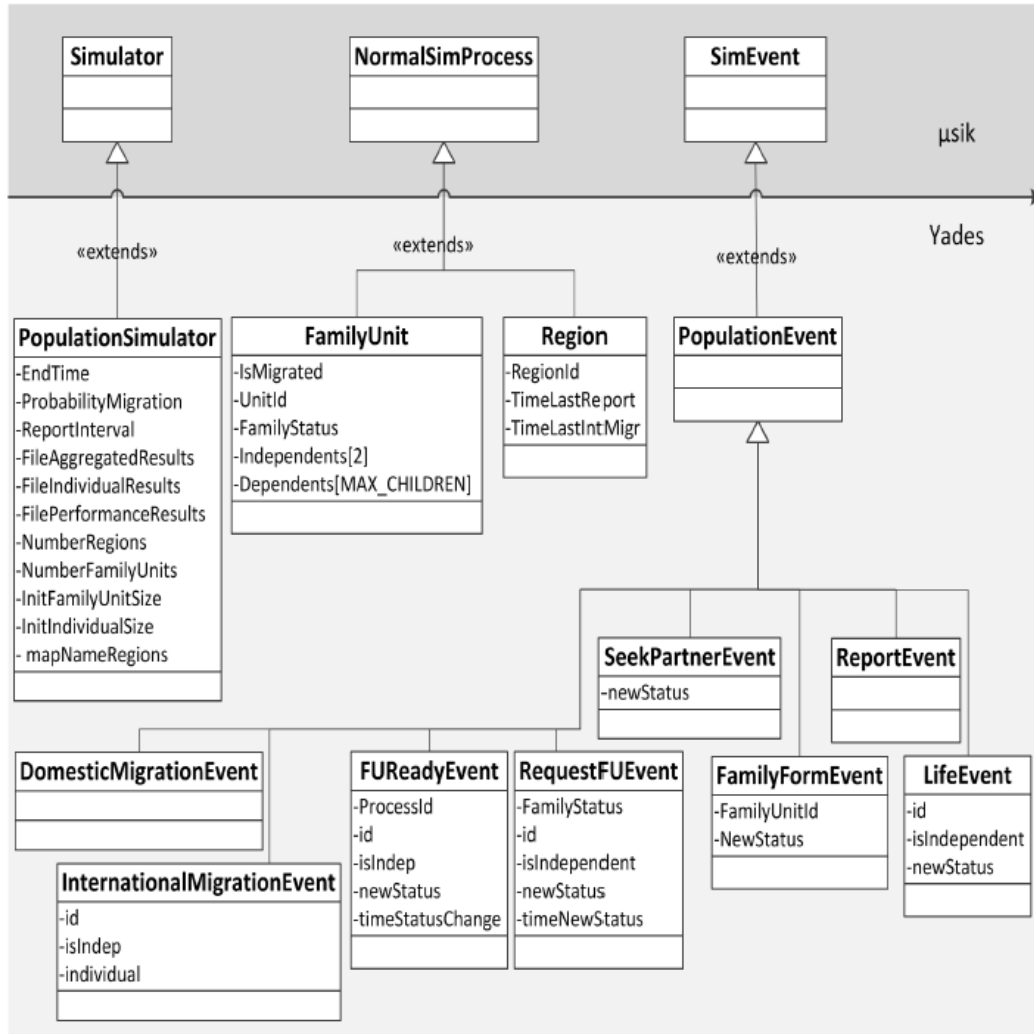


Figura 8: Diagrama de clases UML de Yades [Montañola Sales et al., 2015]

La definición de unidad familiar que emplea Yades contempla la posibilidad de tener familias monoparentales o de un solo individuo. Las unidades familiares evolucionan a lo largo del tiempo de simulación gracias a 5 componentes demográficos diferentes: fertilidad, mortalidad, cambios en el estado económico, cambios en el estado marital y migraciones (que incluye tanto emigraciones como inmigraciones).

Existen eventos que gestionan los cambios de estado que tienen lugar como consecuencia de estos componentes. Los cambios de estado pueden afectar a la familia entera (por ejemplo, al emigrar) o bien a individuos específicos de la misma (por ejemplo, que alguien encuentre trabajo). Por tanto, para todos los componentes es necesario establecer determinadas políticas para determinar cómo se realizarán los cambios de estado, dependiendo de las regiones que se simulen.

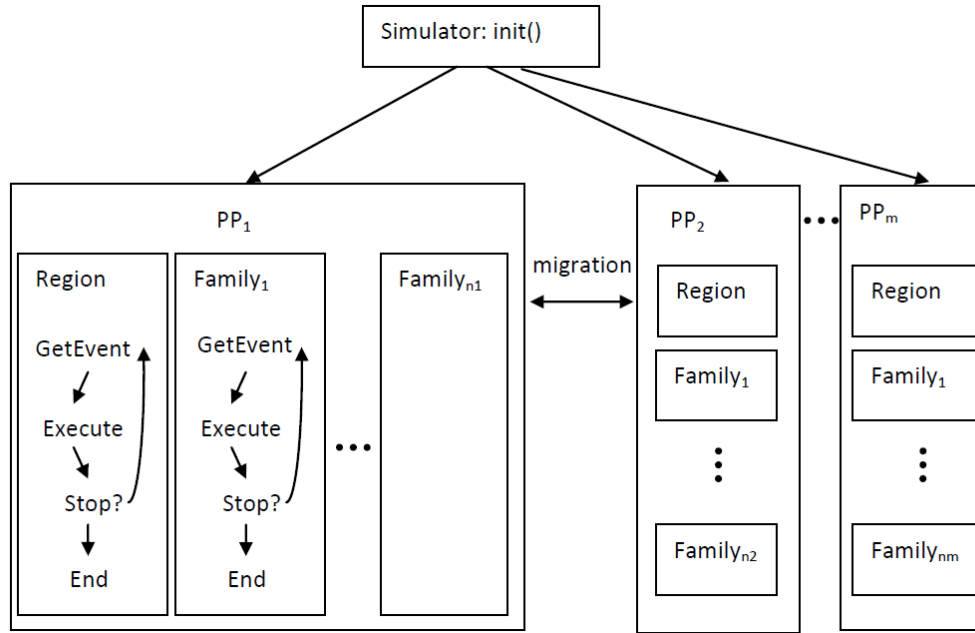


Figura 9: Estructura y comunicación de la simulación en Yades, que consiste en procesos lógicos distribuidos en procesos físicos [Montañola Sales et al., 2015].

Por ejemplo, en el caso del cambio en los estados maritales, la política establece si los individuos encontrarán o no pareja, y qué harán a continuación: casarse, cohabitar, formar una familia, etc.

En el caso de la mortalidad se emplean datos de tablas de vida o funciones de supervivencia para modelar el comportamiento de la mortalidad.

La clase Region representa una región física donde los individuos viven e interaccionan. Como se comentó anteriormente, cada región está implementada como un LP, y de ella dependen las migraciones locales (de una región a otra), las inmigraciones, los cambios en los parámetros de simulación y los datos que se imprimen en los informes del estado de simulación, tanto periódicos como finales.

La comunicación entre los LP de tipo región sucede cuando una familia, que se encuentra en una región y por tanto en un proceso físico, quiere emigrar a otra región que se encuentra en otro proceso físico. En este caso se envía un evento RequestFUEvent a la región destino para ver si es posible realizar la migración. La región destino contesta con el identificador de una familia que está libre para ser aprovechada, y la familia migra con todos sus miembros a la región destino. En este caso la familia es removida de la lista de familias de su región de origen y se envía a la región de destino todo el LP de la unidad familiar. Este tipo de evento se realiza en 3 fases y requiere el envío de 3 eventos diferentes: RequestFUEvent, FUEvent y DomMigrEvent. En la figura 10 se puede observar como se realiza el proceso de migración.

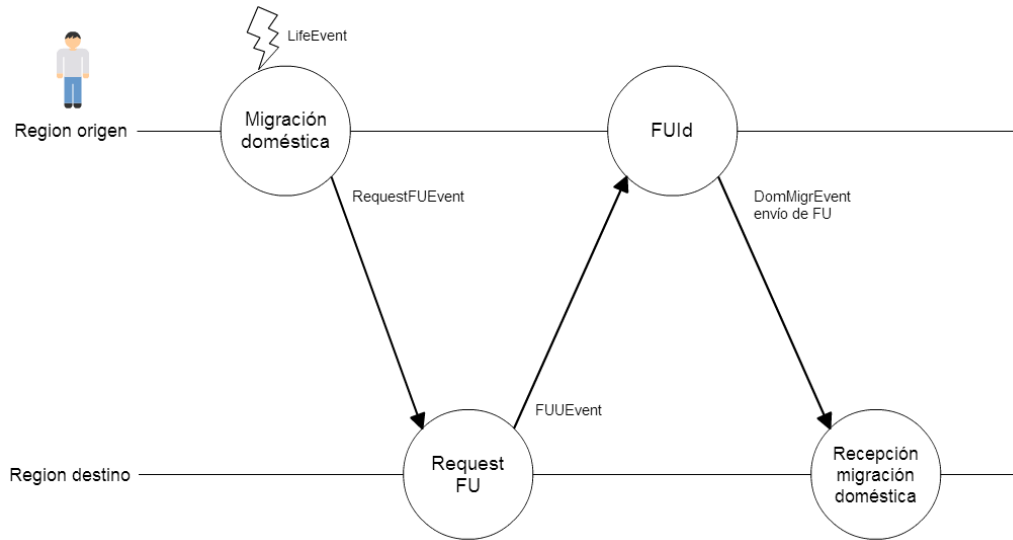


Figura 10: Esquema de funcionamiento de las migraciones en 3 fases con Yades

En el caso de las inmigraciones internacionales en una región, se simula el comportamiento de familias llegando a la región, se construye el número de individuos y familias entrantes que se añaden a la población actual.

La clase PopulationSimulator implementa los procesos físicos y tiene como principal objetivo establecer los parámetros de simulación, controlar los LP y generar los informes de simulación, tanto los periódicos como los finales. La versión actual de Yades simula, para cada proceso físico, una sola región y varias unidades familiares. Actualmente, el modelo especificado garantiza que una familia solo puede tener un evento que la afecte a la vez, que será el que sea lanzado con un tiempo de simulación más cercano al actual. Sin embargo, se puede dar el caso que se lance un evento para una familia que ha migrado o que se ha modificado a causa de un evento marital y no se encuentre, en cuyo caso el evento es desechado.

En las figuras 11 y 12 se presentan los diagramas de actividades con el flujo de vida actual de la aplicación Yades con μ sik. En la figura 11 se presenta como se realiza la inicialización del sistema. La población del sistema está definida como grupos independientes y dependientes, donde cada grupo tiene unas características específicas que los detallan. Cada proceso de inicialización de tipo de población se realiza para cada grupo de edad independiente que compone el sistema.

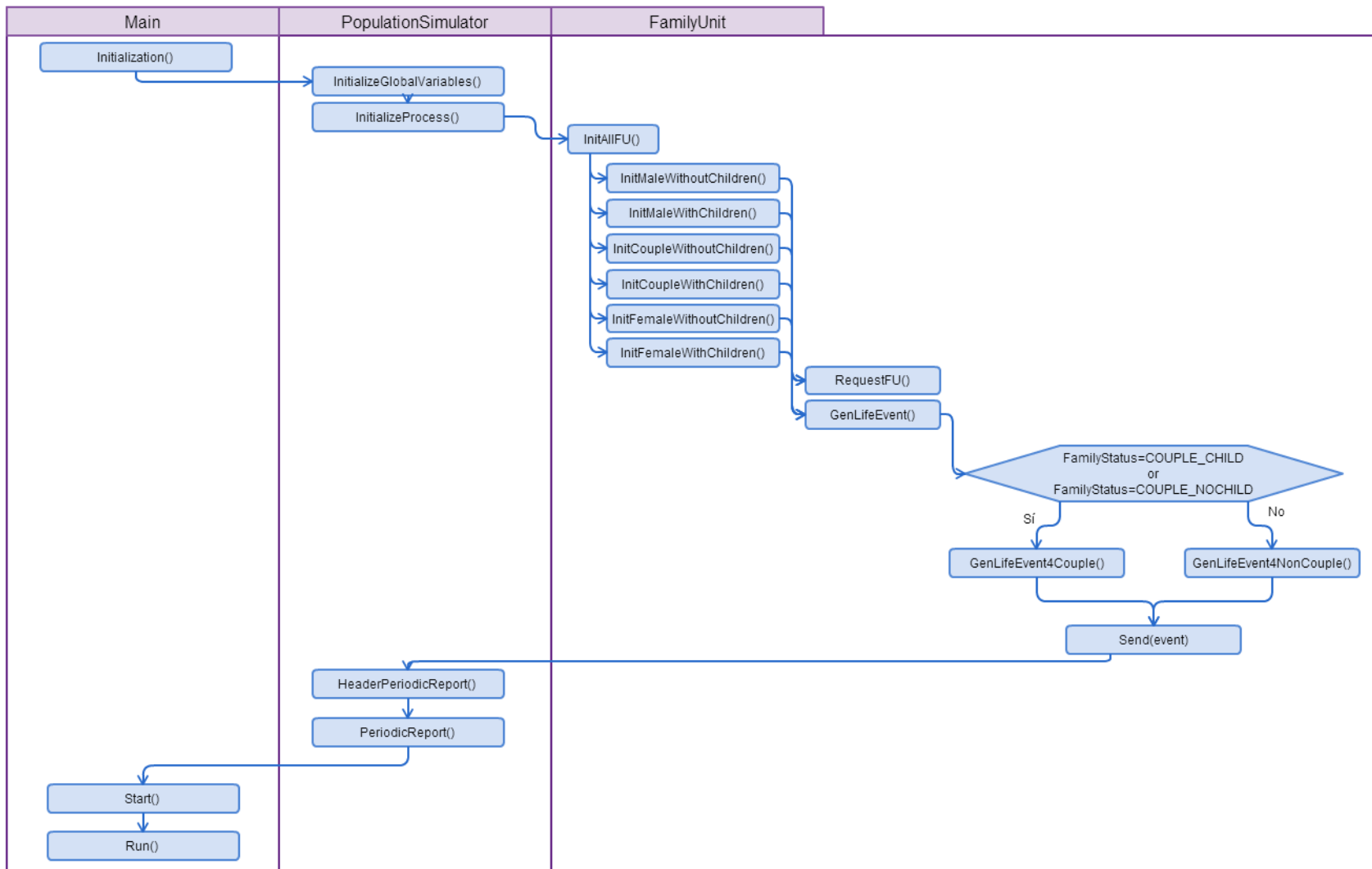


Figura 11: Diagrama de actividades actual de Yades con musik. Inicialización del sistema.

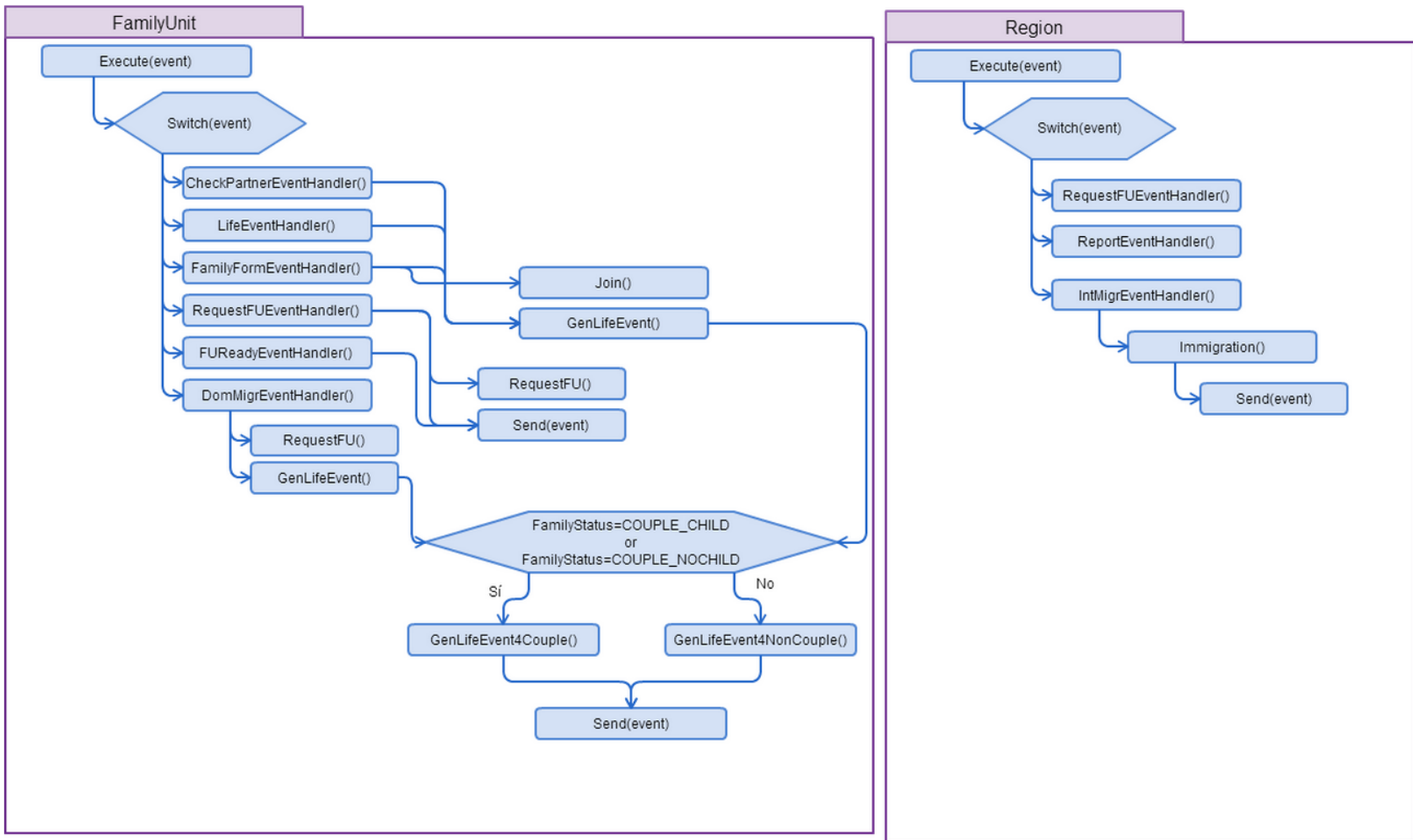


Figura 12: Diagrama de actividades actual de Yades con μ sik. Gestión de eventos.

En el diagrama 12 se puede observar como se gestionan los eventos en el sistema. Los eventos que se generan, envían y reciben durante la simulación pueden afectar a la clase FamilyUnit o a la clase Region, dependiendo de que tipo de evento se trate.

6.1.3. Escalabilidad y Overheads

Antes de presentar los análisis de escalabilidad y de rendimiento de la herramienta Yades, es necesario introducir algunos conceptos en esta materia:

- *Speedup* [Ayguade and Jimenez, 2012-2013]: métrica que permite evaluar la reducción temporal en la ejecución de un problema resuelto de forma paralela, usando P procesadores, con respecto a su correspondiente versión en secuencial.

$$S_p = \frac{T_1}{T_P} \quad (1)$$

Dónde:

- T_1 : tiempo de ejecución del programa con un procesador.
 - T_P : tiempo de ejecución del programa con P procesadores.
- *Weak scaling* [Merz et al., 2015]: análisis en el que el tamaño del problema (workload) por procesador permanece constante y se varían otros parámetros, como el número de procesadores a usar. Este tipo de cálculos y gráficos permiten analizar sistemas que requieren una gran cantidad de memoria. Se considera que un programa escala linealmente cuando el tiempo de ejecución permanece constante mientras que el tamaño del problema aumenta proporcionalmente al número de procesadores. Para calcular la eficiencia en weak scaling, se sigue la fórmula:

$$E_{ws} = \frac{T_1}{T_N} * 100 \% \quad (2)$$

Dónde:

- T_1 : tiempo de ejecución del programa con un procesador.
 - T_P : tiempo de ejecución del programa con P procesadores.
- *Strong scaling* [Merz et al., 2015]: análisis en el que el tamaño del problema permanece fijo, y se aumenta el número de procesadores. Esta medida se usa en programas que toman una gran cantidad de tiempo en ejecutarse. Se considera que un programa escala linealmente si su speedup es igual al número de procesadores usados. Para calcular la eficiencia en strong scaling, se sigue la fórmula:

$$E_{ss} = \frac{T_1}{P * T_P} * 100 \% \quad (3)$$

Dónde:

- T_1 : tiempo de ejecución del programa con un procesador.
 - P : número de procesadores.
 - T_P : tiempo de ejecución del programa con P procesadores.
- *Ejecución Concurrente* [Ayguade and Jimenez, 2012-2013]: romper el programa en trozos más pequeños, llamados tareas y organizarlos para que se ejecuten simultáneamente, o den esa impresión.
 - *Ejecución Paralela* [Ayguade and Jimenez, 2012-2013]: uso simultáneo de varios procesadores (CPUs) para ejecutar las tareas identificadas para la ejecución concurrente. Idealmente cada procesador P podría recibir $\frac{1}{P}$ partes del programa, reduciendo el tiempo de ejecución en P .
 - *Eficiencia* [Ayguade and Jimenez, 2012-2013]: fracción de tiempo en la cual un elemento de procesado está trabajando.

$$E_{ffp} = \frac{S_p}{P} \quad (4)$$

Dónde:

- S_p : Speedup en P procesadores.
 - P : número de procesadores.
- *Sincronización* [Ayguade and Jimenez, 2012-2013]: procesamiento que se añade a la ejecución del programa para asegurarse que se satisfacen las dependencias del grafo de dependencias de la aplicación.
 - *Overhead* [Ayguade and Jimenez, 2012-2013]: tiempo que se añade debido a la sincronización.
 - *Miss de cache*: error de lectura o escritura de un dato en la memoria cache.

A continuación se presenta el análisis de rendimiento de la aplicación Yades con su actual implementación con la librería μ sik. En este análisis se evalúa la escalabilidad de Yades; se evalúa su eficiencia a medida que se aumenta el número de procesadores o elementos de paralelismo empleados, a través de las técnicas *Weak scaling* y *Strong scaling*. Este análisis ha sido publicado y extraído de [Montañola Sales et al., 2015].

Weak scaling

En este análisis se muestra cómo se comporta la aplicación cuando variamos el tamaño de la población, con respecto al tiempo de ejecución y la escalabilidad.

Para este experimento, se ha aumentado proporcionalmente la cantidad de población por cada procesador, a la vez que se aumenta el número de procesadores. En este caso, ya que se quiere ver el impacto del aumento de la población sobre el rendimiento, las migraciones están desactivadas.

Estos experimentos se realizaron sobre las arquitecturas de Marenostrom 2 y 3.

Como se puede ver en la figura 13 [Montañola Sales et al., 2015] el tiempo de ejecución de la aplicación permanece estable mientras se aumenta la población y el número de procesadores. Por lo tanto, se observa un buen escalado de la aplicación a medida que se aumenta el tamaño del problema. Además en este experimento se puede observar como afectan los misses de cache al tiempo de ejecución al observar que el tiempo de ejecución para 40,000 familias por nodo es mayor que el doble que simular 20,000 por nodo.

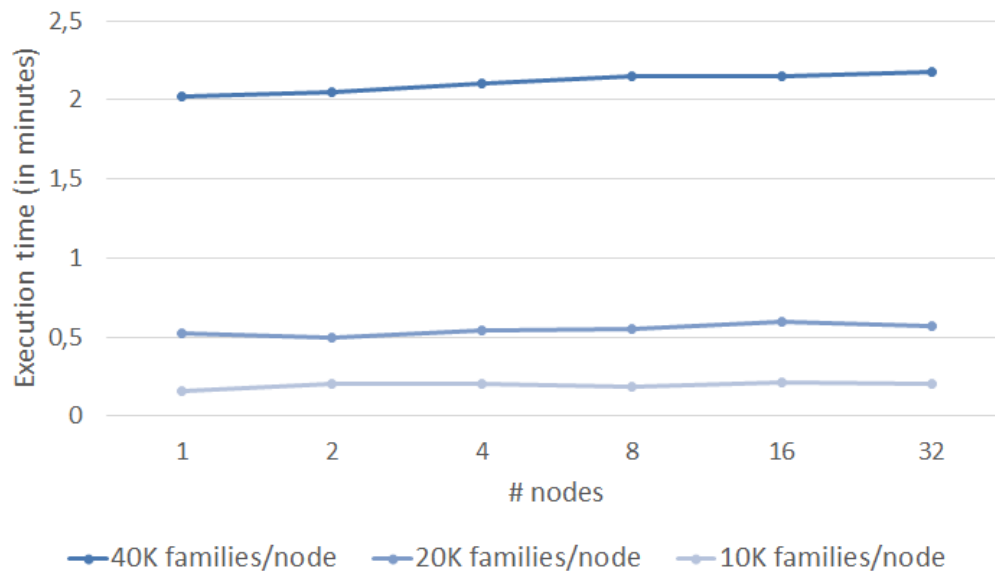


Figura 13: Gráfica con el análisis weak scaling de Yades con μ sik [Montañola Sales et al., 2015]. Nota: en estas pruebas se usa sólo un procesador por cada nodo

Strong scaling

En este análisis se ha evaluado el efecto en el speedup de la aplicación mientras se aumenta el número de procesadores a usar cuando el tamaño de la población a simular permanece estable.

Para este experimento se han tenido en cuenta las migraciones. Se ha observado que a medida que se aumentan los procesadores, la capacidad de computación de la aplicación aumenta. Sin

embargo, en la figura 14 se observa una superlinealidad de la aplicación debido a los misses de cache que ocasionan un aumento de las sincronizaciones de la aplicación y que hacen que su rendimiento disminuya. Las migraciones aumentan las comunicaciones entre procesadores y por lo tanto, los rollbacks.

Como se puede observar, tasas de migraciones más elevadas reducen el speedup de la aplicación.

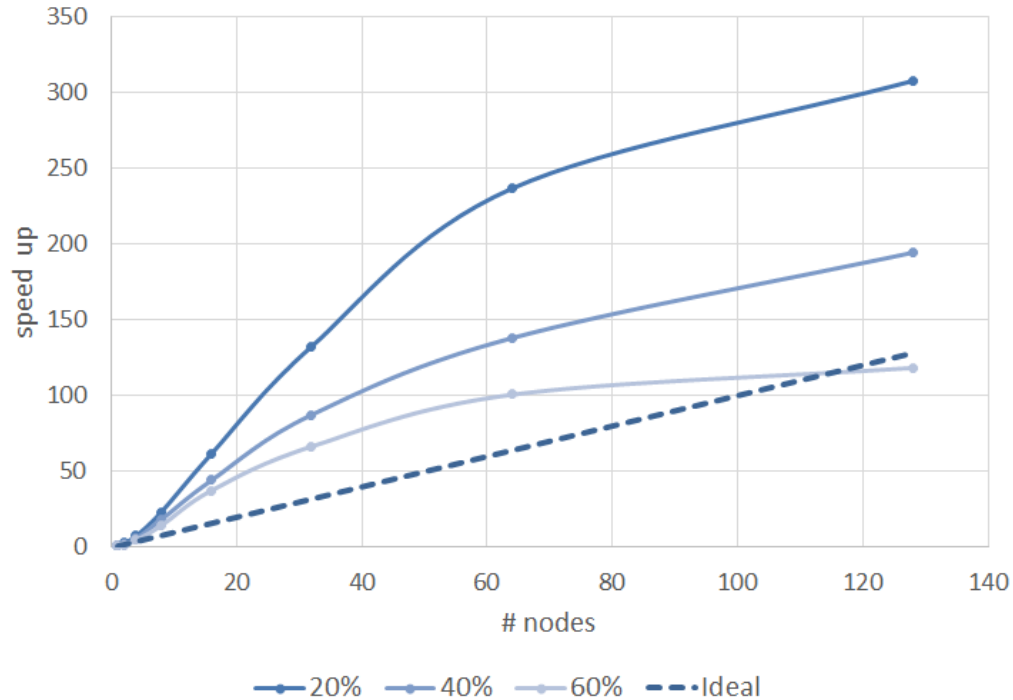


Figura 14: Gráfica con el análisis strong scaling de Yades con μsik [Montañola Sales et al., 2015]. Nota: en estas pruebas se usa sólo un procesador por cada nodo. Los porcentajes de la leyenda se refieren a la probabilidad de migración interna.

6.1.4. Problemática actual

Como se ha podido observar en la sección 6.1.3 y en la 1.2, la implementación actual del simulador Yades con la librería μsik presenta algunos problemas de rendimiento que afectan directamente a la ejecución del simulador y, en especial, a su tiempo de ejecución.

Estos problemas dificultan que el simulador pueda ser empleado para su propósito original con total comodidad por parte de los interesados, y no permiten aprovechar en su totalidad los beneficios que brinda la arquitectura paralela. En la sección 6.1.3, se puede observar que a medida que la población en cada región de simulación aumenta y aumenta la probabilidad de

migración, el simulador empieza a presentar importantes overheads a causa de las comunicaciones internas y la gestión de la memoria que realiza. Además, la librería μ sik es un proyecto que se ha descontinuado y actualmente no cuenta con soporte técnico ni con nuevas actualizaciones, por lo que no se espera que en un futuro cercano puedan ser solucionados los problemas de comunicaciones que presenta debido, principalmente, a que no aprovecha las características de la arquitectura multicore con memoria compartida para optimizar las simulaciones.

Además, el simulador Yades no ha pasado por una fase de test exhaustivo durante su desarrollo y se han detectado algunas funcionalidades que podrían mejorarse, para incrementar su rendimiento.

Por estas razones, se ha decidido desarrollar una solución que aproveche las capacidades que ROOT-Sim proporciona para entornos multicore con memoria compartida y que permite optimizar el rendimiento de Yades en entornos paralelos.

6.2. Descripción de ROOT-Sim

ROme OpTimistic Simulator (ROOT-Sim) es una librería estática de simulación paralela de propósito general por eventos discretos (Parallel Discrete Event Simulator - PDES), construida de acuerdo al protocolo de sincronización optimista. Es una librería estática que puede ser vinculada por ejecutables que usan modelos de simulación utilizando el estándar de programación ANSI-C, como si fueran completamente secuenciales.

En ROOT-Sim el modelo de simulación se describe mediante:

- El estado de simulación: contiene el estado actual del sistema.
- Eventos: están asociados a cambios o acciones dentro del sistema. Un evento discreto ocurre en un instante temporal concreto y produce un cambio en el sistema. Un evento no puede ser enviado al pasado, ya que se debe seguir un orden temporal.
- Transiciones de estados: la acción de los eventos puede ocasionar cambios en los estados de los objetos de simulación.

De este modo, el modelo de simulación se compone de objetos de simulación, cada uno de los cuales modela una parte del escenario o de los agentes.

Los objetos de simulación son estructuras de datos manejadas por los LP. Los LP son las estructuras de datos principales de la simulación. Cada una de ellas recibe, envía y procesa eventos, su avance en el Logical Virtual Time (LVT) está conectado a su ejecución y los LP, que consisten en una pareja de $\langle \text{id}, \text{estado} \rangle$, se comunican entre ellos vía eventos. Cada evento se identifica por un código numérico, que está definido por el nivel lógico de la aplicación.

El kernel de simulación se encarga de programar la ejecución de cada LP, de entregar los eventos a los LP receptores y de propagar las consecuencias de estos eventos en la simulación.

ROOT-Sim sigue un paradigma basado en PDES con un algoritmo de sincronización optimista, donde los eventos se ejecutan de manera especulativa, esto significa que se puede hacer confirmación (commit) o no de los eventos procesados. La decisión de confirmar el estado o no, depende de la consistencia del estado de simulación y de la ventana de simulación. En la figura 15 se muestra la arquitectura de un simulador paralelo basado en PDES.

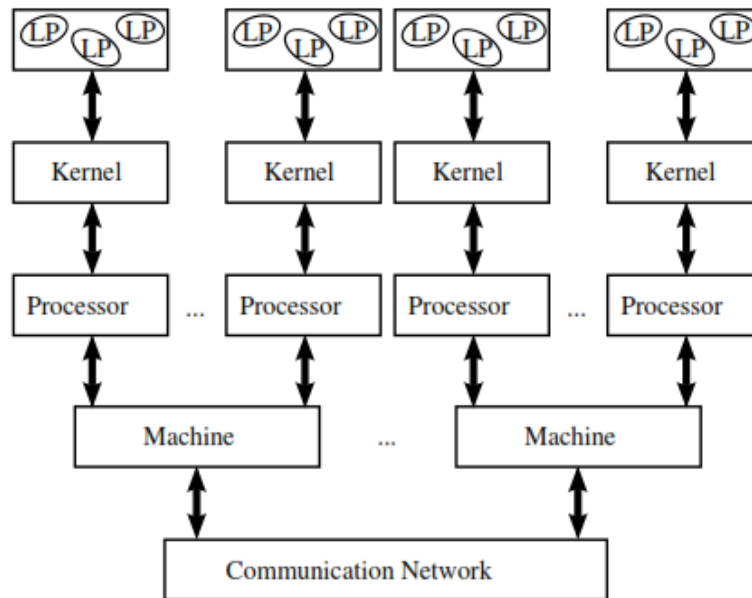


Figura 15: Arquitectura de un PDES [Pellegrini and Quaglia, 2014].

La librería de simulación ROOT-Sim proporciona 4 funciones a nivel de *API* (Application Programming Interface) que permiten interactuar con la aplicación. En la figura 16 se muestra la estructura que sigue ROOT-Sim. Estas estructuras son las que se detallan a continuación:

- *void ProcessEvent(unsigned int me, simtime_t now, int event_type, event_content_type *event_content, unsigned int size, void *ptr)*: esta función permite especificar el planificador (dispatcher o scheduler) de eventos para cada LP otorgando el control de la simulación al modelo, que debe ser implementado dentro de esta función. Además, es la encargada de la parte especulativa de la aplicación, es decir, el modelo de simulación se va ejecutando pero en el momento que se llega a un estado inconsistente de simulación se realiza una marcha atrás (rollback) y se restablece todo el estado de simulación previo.

Para cualquier modelo de simulación que se desee especificar, es necesario definir el evento INIT (que tiene el identificador 0 dentro de ROOT-Sim), ya que es donde se inicializan todas las estructuras de datos, se reserva el espacio de memoria para cada estructura, se

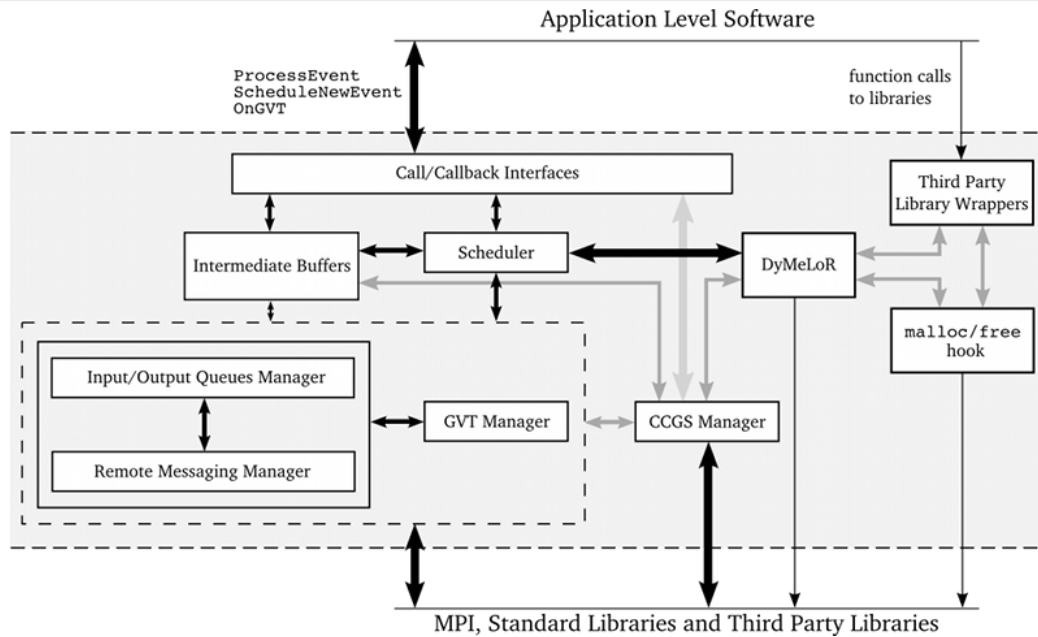


Figura 16: Estructura de ROOT-Sim [Pellegrini, 2013-2014].

informa a la librería cual será el estado inicial de simulación que tendrá ese LP y se generan los eventos de vida de la simulación.

En el evento INIT además, tienen lugar las comprobaciones de parámetros que se pasan por línea de comandos. Para esto, ROOT-Sim proporciona una serie de funciones que permiten obtener los parámetros de entrada de manera similar a una aplicación tradicional en C, especificando por línea de comandos el nombre del parámetro que se desea pasar antes de su valor.

- *extern void (*SetState)(void *new_state)*: a través de esta función se le informa al compilador del estado inicial de simulación para cada LP y se reserva el espacio para la ejecución. Es llamada desde el INIT de ProcessEvent(..).
- *extern void (*ScheduleNewEvent)(unsigned int receiver, simtime_t timestamp, unsigned int event_type, void *event_content, unsigned int event_size)*: esta función permite inyectar cualquier tipo de evento nuevo en el sistema. Se debe indicar hacia qué LP va dirigido el evento, qué hará este evento y el tiempo en el que éste sucederá.
- *bool OnGVT(unsigned int me, lp_state_type *snapshot)*: esta función es la que se ejecuta cuando se hace commit del estado de simulación, es decir, aquí es donde tienen lugar todas las acciones no especulativas e irrevocables que se toman durante la simulación (como por ejemplo escribir datos en un fichero).

Esta función recibe un LP con un estado consistente, ya confirmado. La decisión de cuándo comprobar el estado de simulación y en caso de que sea consistente, hacer el commit, es

tomada por ROOT-Sim. Además es la encargada de controlar el tiempo de finalización de la aplicación. Durante la ejecución se va comprobando el estado de los LP y cuando todos están de acuerdo con terminar, la aplicación acaba. Este comportamiento se puede alterar y marcar una condición de finalización, como por ejemplo un número específico de pasos de simulación que se deben dar antes de finalizar. En el momento que esta condición se cumpla, la simulación terminará.

Para trabajar con ROOT-Sim es necesario instalar todo el paquete que viene con la librería en todos los equipos en los que se desea trabajar. Una vez se ha instalado, se dispone de un compilador *rootsim-cc* que permitirá compilar las aplicaciones que usan la librería. Este compilador es una especie de envoltorio sencillo del compilador estándar de C, GCC, que realiza algunos controles, gestiones de memoria, modificaciones del código y enlaces a otras librerías de paralelismo, como pthread, que permiten aumentar el rendimiento de las aplicaciones y ejecutarlas en paralelo.

La librería pthread es un POSIX (Portable Operating System Interface) que proporciona herramientas para trabajar con threads [Ayguade, 2014-2015]. Esta aplicación de bajo nivel proporciona un control de *Granularidad fina* que permite realizar operaciones con los threads tales como: creates, joins, operaciones de sincronización, entre otras. Es empleada para realizar comunicaciones “intra-nodo” en sistemas de memoria compartida o *Multiprocesamiento genérico* (SMP). Por el contrario, la librería MPI que actualmente es el protocolo estándar de comunicación para ejecutar aplicaciones concurrentes en sistemas multiprocesadores de memoria distribuida, es usada para comunicaciones “inter-nodo”. Los threads tienen una menor latencia y en sistemas SMP, permiten acelerar códigos que la librería MPI no puede, por ejemplo, aliviando la contención de una aplicación con MPI. La librería ROOT-Sim emplea pthreads para resolver las comunicaciones intra-nodo.

Además ROOT-Sim cuenta con su propio main, que es totalmente transparente al usuario, que no tiene que usarlo para desarrollar su modelo y trabajar con el.

ROOT-Sim también implementa varias funciones matemáticas que permiten trabajar con las distribuciones estadísticas más usadas en el campo de la simulación basada en agentes como por ejemplo la distribución Exponencial, la Poisson, la generación de números aleatorios siguiendo una distribución Random o generación de números comprendidos en un intervalo específico, entre otros. Con ellas se garantiza que en el caso de hacer un rollback de la aplicación, se tienen almacenadas todas las acciones realizadas por estas funciones, como por ejemplo todos los números aleatorios generados, y se puede restablecer el estado de simulación completo.

ROOT-Sim dispone de una librería de topologías que permite interactuar con los diferentes LP del sistema. En la figura 17 se muestran estas topologías.

De este modo, la librería de simulación ROOT-Sim presenta las siguientes ventajas frente a otras librerías de simulación paralela:

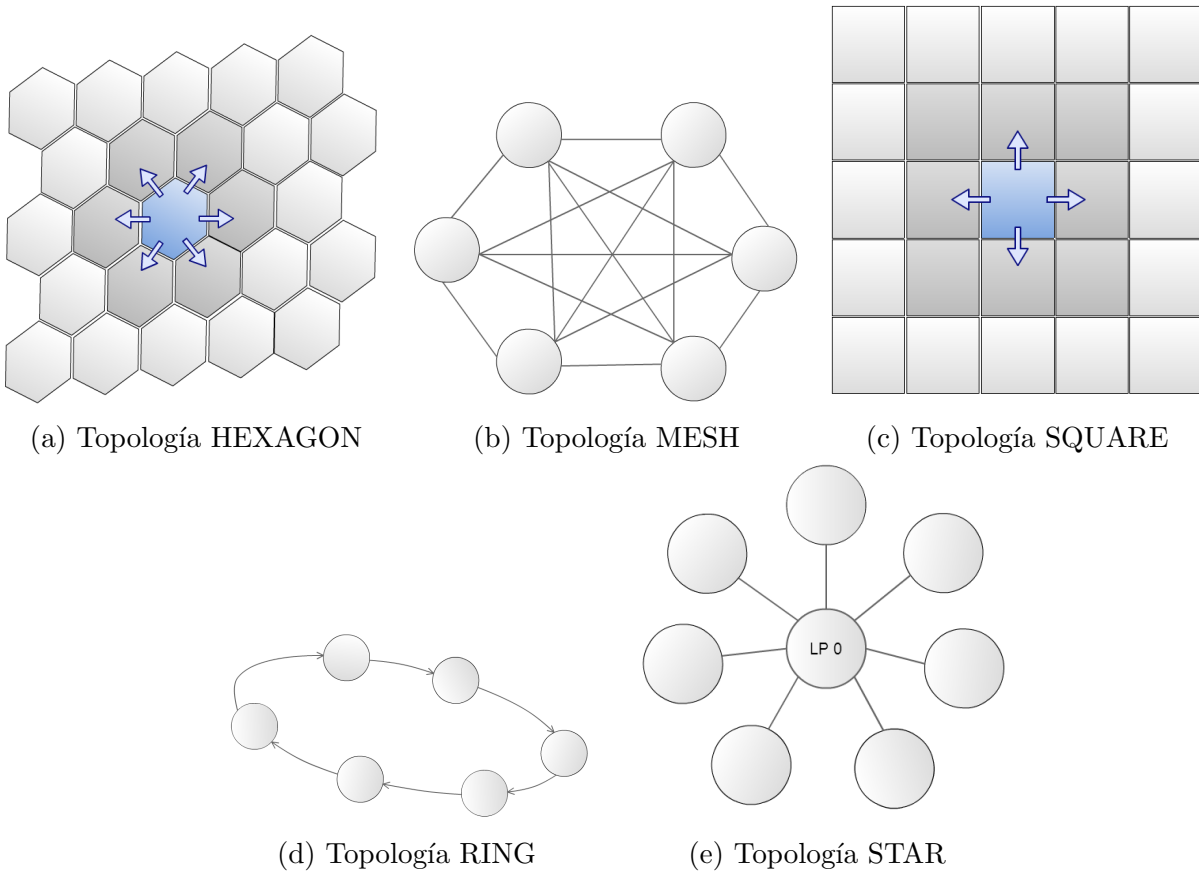


Figura 17: Topologías proporcionadas por la librería ROOT-Sim.

- El estado de simulación puede ser repartido a través de una asignación dinámica de la memoria.
- Emplea un manejo consistente con commits del estado global y tiene sistema de detección de final.
- La librería de rollbacks es transparente al usuario y permite obtener una gran cantidad de estadísticas sobre la confirmación o no de los estados de simulación.
- Contiene una librería de topologías.

Actualmente ROOT-Sim es un proyecto en desarrollo y se están implementando otras características como son:

- Compartición del estado de memoria (vía variables globales)
- Manejo consistente de la salida (output)
- Balanceo de comunicaciones (via multithreading)

ROOT-Sim ha sido probado con modelos que implementan redes de comunicación (PCS, [Pellegrini, 2013-2014]), modelo de almacenamiento de datos de objetos persistentes (NoSQL Data-Store Simulator, [Pellegrini, 2013-2014]) y un modelo de una red de hormigas robot (Terrain-Covering Ant Robot, [Pellegrini, 2013-2014]), pero hasta el momento no se ha probado en grandes modelos de simulación social como el caso de Yades.

6.2.1. Escalabilidad y Overheads

Para evaluar la escalabilidad de ROOT-Sim, se ha empleado el modelo PCS [Pellegrini, 2013-2014] que proporciona la librería como ejemplo y que es un benchmark, conocido en el área de PDES. Además, gracias a las pruebas realizadas con este modelo, se ha adquirido un importante conocimiento sobre el funcionamiento de la librería para las fases futuras de implementación.

El modelo PCS, Personal Communication System, modela una red móvil que funciona con tecnología GSM. Cada LP modela la evolución del estado de una celda hexagonal individual y el conjunto total de celdas proporciona cobertura wireless en regiones cuadradas de tamaño variable.

Cada celda gestiona un número N parametrizable de canales de la red wireless que son modelados de una forma explícita donde hay interferencias.

Durante la simulación en cada celda se llevan a cabo una serie de llamadas con una cierta frecuencia y se simulan interferencias y caídas del servicio debidas a un creciente número de llamadas.

Para observar el rendimiento de ROOT-Sim se redefine el rendimiento con esta fórmula:

$$\eta = \left(1 - \frac{N_{rb}}{N_{evt}} \cdot \frac{N_{evt} - N_{comm}}{N_{rb}} \right) \cdot 100 \quad (5)$$

Donde:

- N_{rb} es el número total de operaciones de rollback en una ejecución paralela.
- N_{evt} es el número total de eventos, confirmados y no confirmados.
- N_{comm} es el número total de eventos confirmados.

Por lo tanto, si η es bajo, significa que la ejecución paralela de la simulación provoca un número elevado de operaciones de rollback, o *la longitud* de estas operaciones es muy elevada.

Como se puede ver en la figura 18, a medida que se aumenta el número de regiones o celdas, la eficiencia disminuye sobre todo cuando se aumenta la comunicación hasta el 120 %.

Esto es debido a que el número de eventos de comunicación que tienen lugar aumenta a medida que se incrementa la cantidad de regiones, especialmente cuando la tasa de comunicaciones es muy elevada, como se observa en la figure 19

Al observar el efecto de variar la frecuencia de llamadas en PCS, podemos ver en la figura 20, que la latencia de los eventos aumenta de forma exponencial.

Los resultados de ROOT-Sim con este benchmark nos muestran que la librería presenta una buena escalabilidad para un modelo de regiones con un alto número de comunicaciones entre ellas. Este benchmark es interesante para nuestro trabajo ya que se asemeja al modelo de Yades.

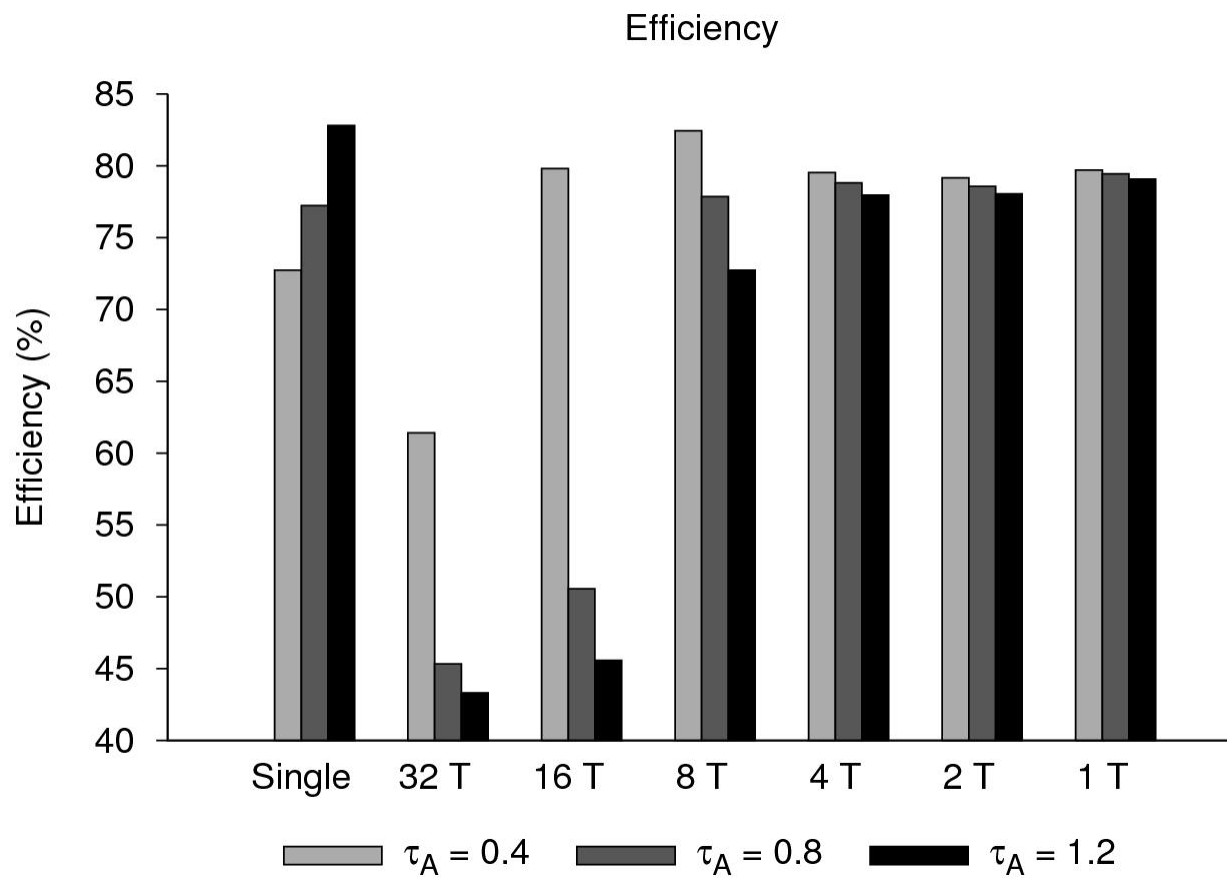


Figura 18: Evolución de la eficiencia de la implementación de PCS con ROOT-Sim [Pellegrini, 2013-2014].

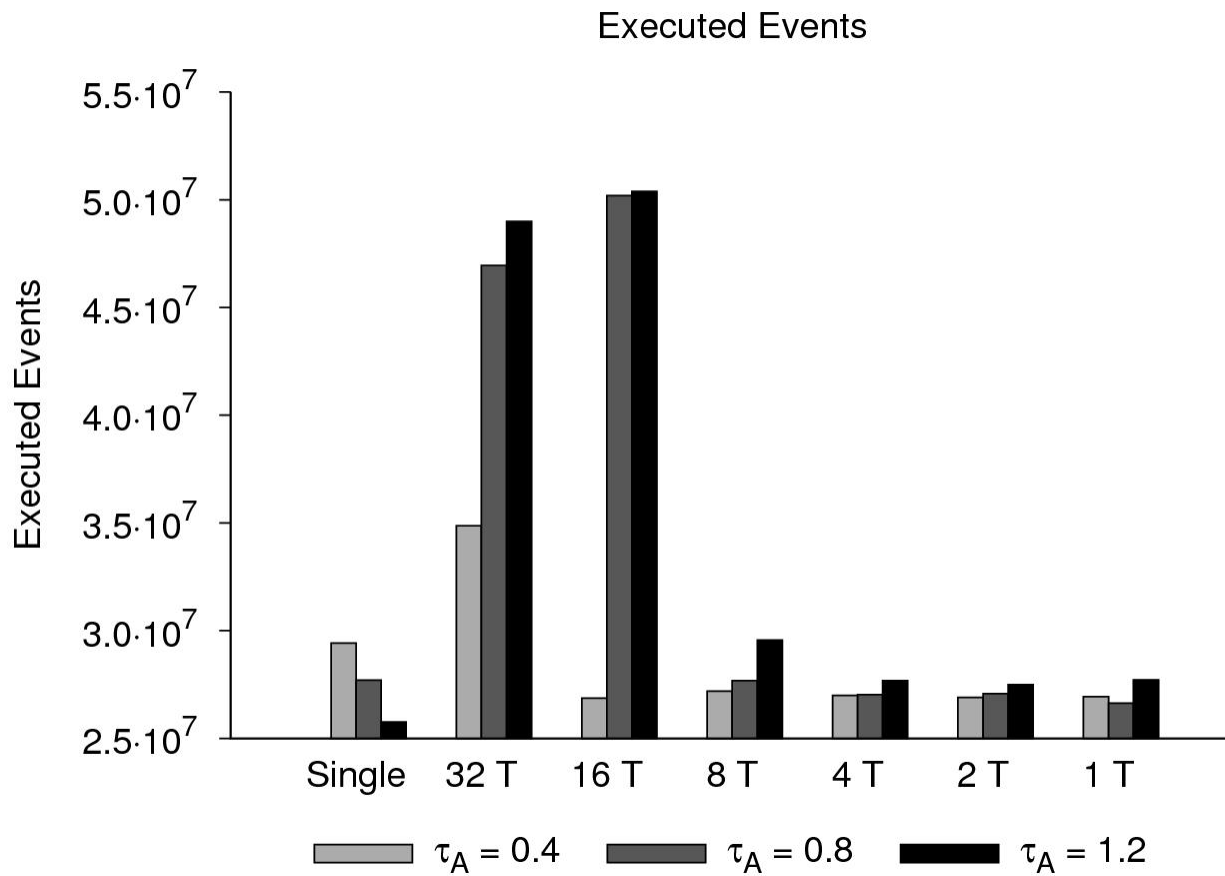


Figura 19: Evolución del número de eventos a medida que aumentan las comunicaciones en la implementación de PCS con ROOT-Sim [Pellegrini, 2013-2014].

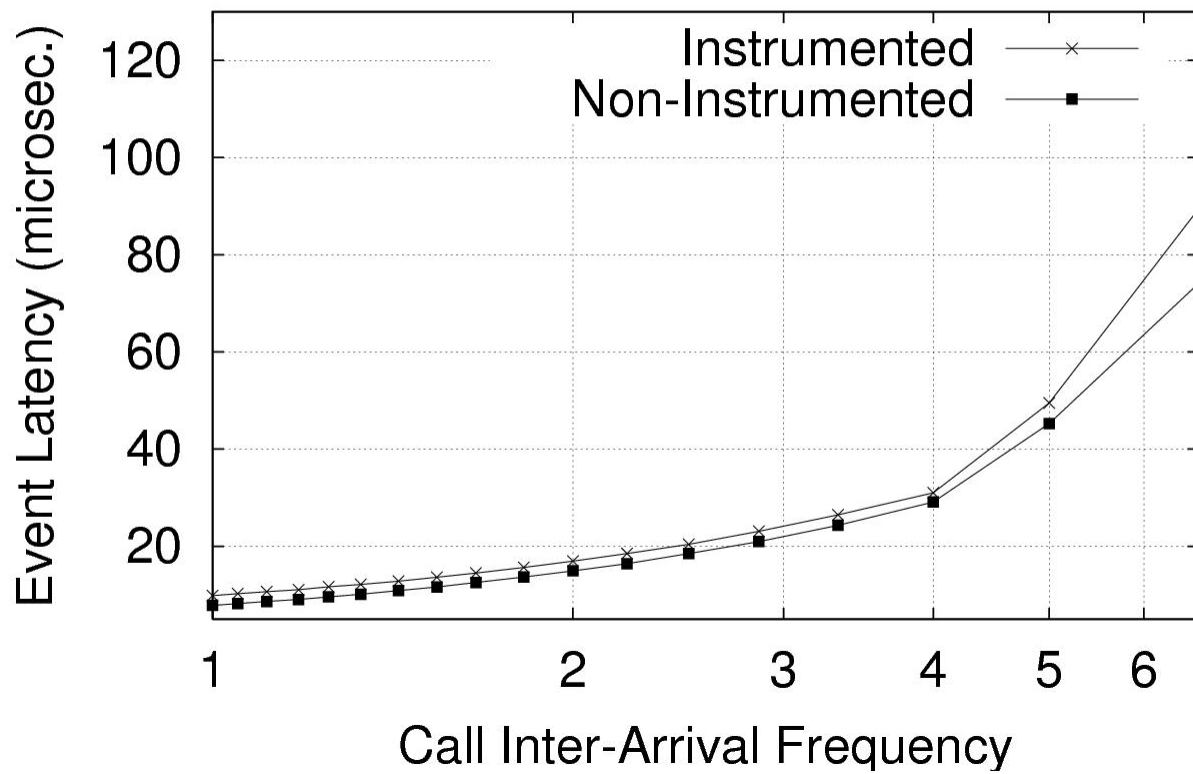


Figura 20: Evolución de la latencia de los eventos a medida que la frecuencia de llamadas aumenta en la implementación de PCS con ROOT-Sim [Pellegrini, 2013-2014].

7. Especificación del sistema

El objetivo de la especificación del sistema es el de definir de manera clara y precisa todas las características y restricciones que debe tener el sistema a desarrollar. También es necesario que contemple todas las necesidades que debe cubrir el sistema y que han sido acordadas previamente con el cliente.

Este trabajo consiste en analizar una aplicación ya existente y evaluar su funcionamiento con otra librería de simulación paralela. Por esta razón no se ha realizado una especificación del sistema desde cero, sino que se han tomado las características y funcionalidades del sistema Yades ya existente. Sin embargo, una de las aportaciones que se realizan en este trabajo es la de definir de manera formal la especificación del sistema, ya que el sistema original Yades no contiene esta clase de documentación.

El primer paso en la especificación de un sistema es definir las partes interesadas en el proyecto. Estas partes son conocidas como *stakeholders*. A continuación se presenta una explicación sobre quiénes son las partes interesadas en este proyecto y todas las historias de usuario definidas para este sistema.

7.1. Stakeholders

Los *stakeholders* son todas aquellas personas afectadas directa o indirectamente por el desarrollo de este proyecto. Los requisitos que debe cumplir el software deben ser pactados con todos ellos y se debe comprobar que se han cumplido al final del proyecto.

Desde el inicio del proyecto hasta su finalización, están implicados tanto el desarrollador como el tutor del proyecto. Además, en este proyecto también está implicado un asesor externo de la Università di Sapienza, Roma (Italia) responsable del desarrollo de la librería de simulación ROOT-Sim. La persona con el rol de desarrollador llevará a cabo el proyecto en todas sus fases y el director del proyecto será el encargado de supervisar que se vayan cumpliendo los objetivos y requisitos previamente definidos, asegurar que se sigue la planificación establecida, así como asesorar en los aspectos más técnicos del proyecto.

El cliente de este proyecto es, en primer lugar, el equipo de investigación que trabaja con la aplicación Yades y desea tener una aplicación que escale mejor y presente mejor rendimiento para poder mejorar las investigaciones que se realizan con él. En segundo lugar, está el equipo de investigación que desarrolló ROOT-Sim y que está interesado en probar su librería en un caso real a gran escala.

Para finalizar, los usuarios finales de este proyecto serán los demógrafos o modeladores, para quienes está destinada la aplicación Yades. Estas personas no tendrán un amplio conocimiento técnico en el área de la simulación paralela o el uso del lenguaje de programación C, pero deben

ser capaces de poder ejecutar el modelo de simulación con la menor cantidad de complicaciones posibles y obtener unos resultados que se ajusten al modelo y a los datos previamente definidos.

7.2. Modelo conceptual

El modelo conceptual permite especificar los elementos que pertenecen al dominio del problema, permite representar ideas o conceptos del mundo real. Es muy útil en la definición de un sistema y se utilizan para comprender las entidades que conforman el contexto del proyecto.

Para la representación del diagrama conceptual se ha escogido el lenguaje de modelado de sistemas software UML (Unified Modeling Language) [Group], que permite, a través de un lenguaje gráfico, especificar, documentar y construir el comportamiento del sistema. Se ha escogido el lenguaje de especificación UML, que es orientado a objetos, debido a que la aplicación inicial Yades, de la cual se está realizando la especificación para la siguiente definición del proyecto actual es orientado a objetos, aunque la aplicación final que se desarrolla en este proyecto no lo es.

El proyecto del cual forma parte este TFG se ha estructurado en dos grandes fases, dependientes entre ellas. La fase realizada en este TFG solo incluye su primera fase, que consiste en la investigación y evaluación de la integración de Yades con ROOT-Sim y el correspondiente análisis de rendimiento. Si esta fase proporciona buenos resultados, se podrá proseguir con la segunda fase del proyecto, en la que se realizaría una adaptación de la aplicación realizada en este TFG aplicando patrones para aumentar la cambiabilidad del código. Por esta razón, se ha considerado que, en pos de aumentar la cambiabilidad de la aplicación resultante para futuros proyectos que se puedan realizar con ella, es recomendable realizar la especificación en un lenguaje estándar de especificación que incluya la orientación a objetos.

En la figura 21 se presenta el diagrama conceptual de las entidades que intervienen en el contexto del sistema. En la figura se muestra cómo deben ser las relaciones entre las entidades: Región, Unidad Familiar e Individuo, los cuales pueden ser dependientes o independientes.

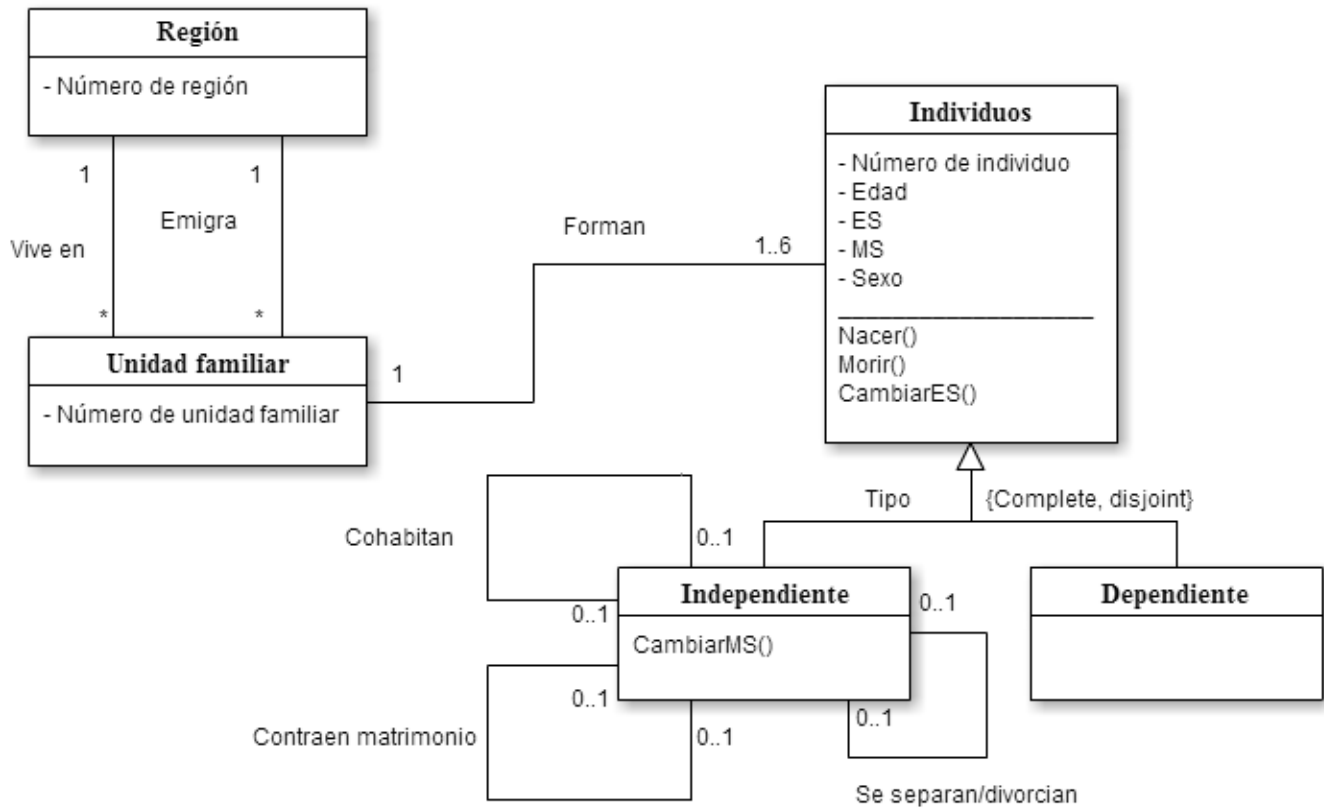
La entidad *Región* representa el emplazamiento físico en el que habitan familias y evolucionan durante el transcurso del tiempo. Cada región es única.

La entidad *Unidad Familiar*, representa una familia dentro de la comunidad que habita en una región física. La razón por la cual se ha cogido la unidad familiar como unidad básica se debe a que en algunos países las decisiones demográficas o políticas públicas que surgen a raíz de aplicar estudios de simulación social, se toman a nivel de unidades familiares [Montañola Sales et al., 2015]. Estas unidades familiares pueden estar formadas por uno o más individuos, con lo que se contempla la posibilidad de tener un individuo soltero o incluso niños sin padres.

La entidad *Individuo* pertenece a una unidad familiar. Una familia puede estar formada por un máximo de dos individuos adultos y un número predeterminado de hijos. A estos individuos les pueden suceder diversos eventos como fertilidad, mortalidad, cambiar de estado marital o de



estado económico, e incluso migraciones, aunque estas están definidas a nivel de unidad familiar: si una persona migra, migrará toda su familia con ella. Cada individuo tiene características propias tales como edad, sexo, estado económico o estado marital.



Restricciones de integridad:

- Una unidad familiar no puede emigrar a la misma región donde vive
- Un individuo no puede contraer matrimonio consigo mismo
- Un individuo no puede cohabitar consigo mismo
- Un individuo solo se divorcia si estaba casado y se divorcia únicamente de la persona con la que estaba casado
- Un individuo solo se separa si estaba casado o cohabitando y se separa únicamente de la persona con la que estaba casado o cohabitando
- La edad de un individuo debe ser ≥ 0

Figura 21: Diagrama conceptual de Yades.

En la figura 22 se muestra un diagrama con los cambios de estados que puede realizar un individuo a lo largo de su vida.

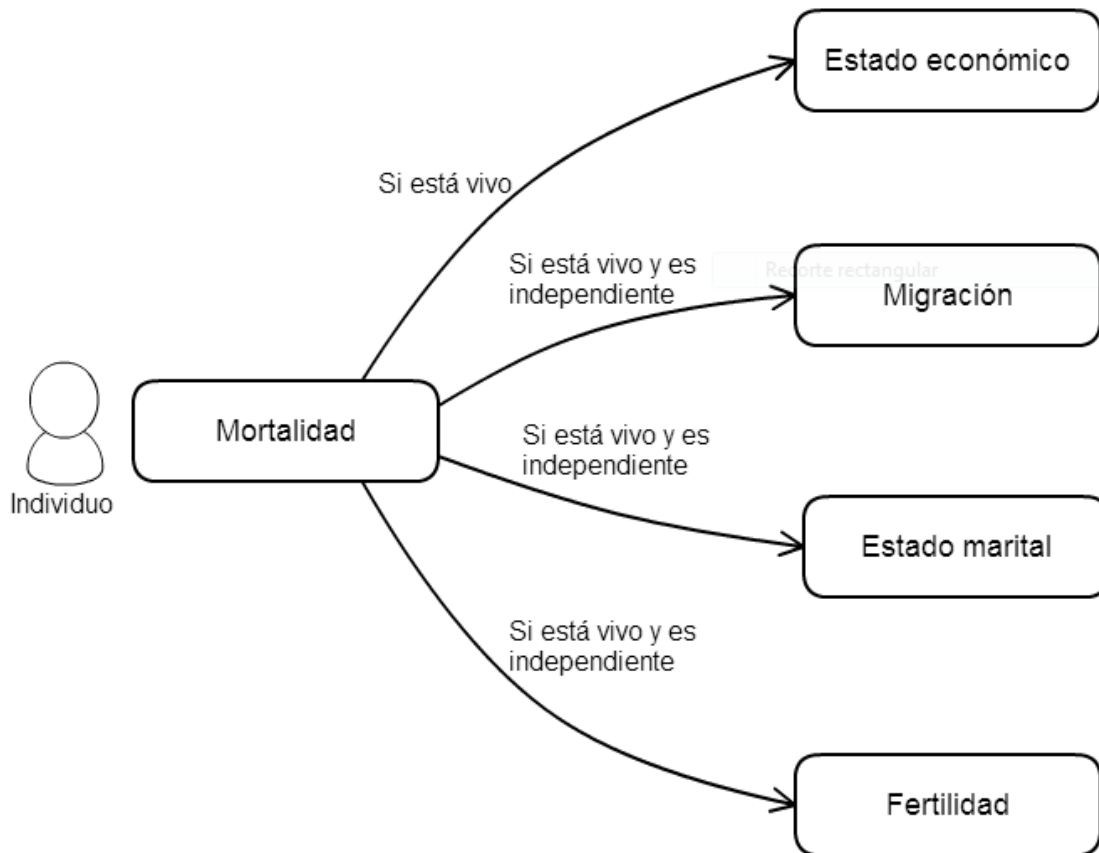


Figura 22: Diagrama de estados por los que puede pasar un individuo durante su vida

En la figura 23 se muestra un diagrama con las funciones de vida de un individuo durante el transcurso de su vida.

En la figura 24 se muestra un diagrama con los cambios de estados económicos que puede realizar un individuo a lo largo de su vida.

En la figura 25 se muestra un diagrama con los cambios de estados maritales que puede realizar un individuo a lo largo de su vida.

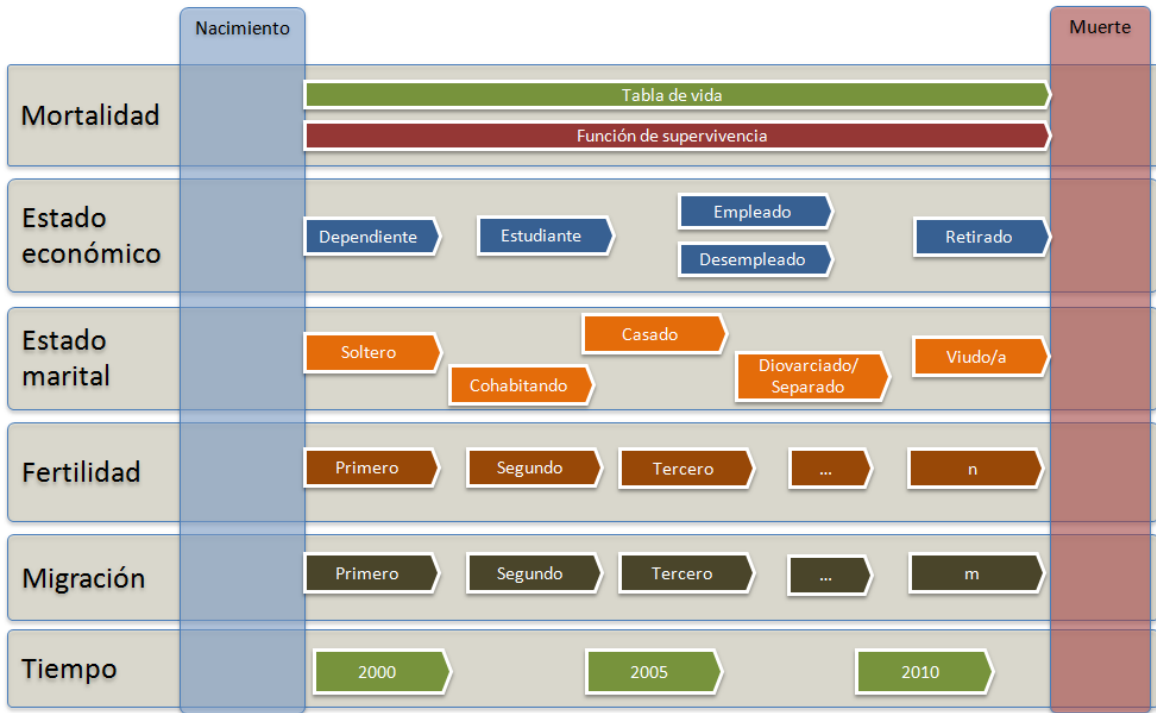


Figura 23: Diagrama con las funciones de vida de un individuo durante su vida

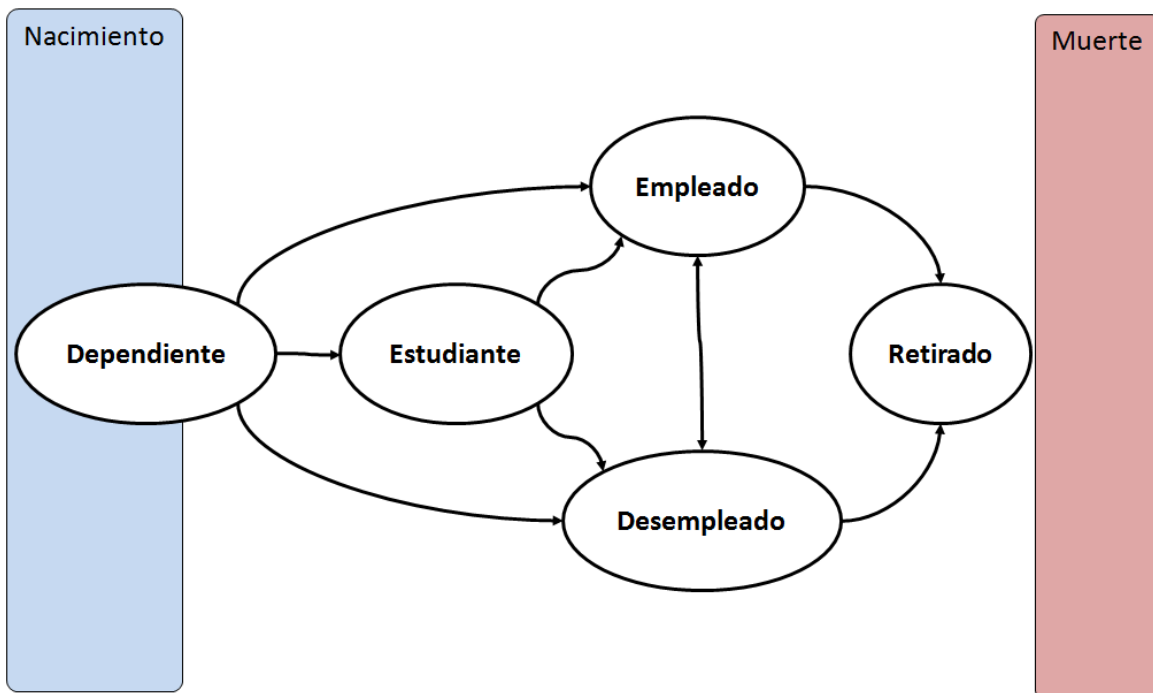


Figura 24: Diagrama de estados económicos por los que puede pasar un individuo

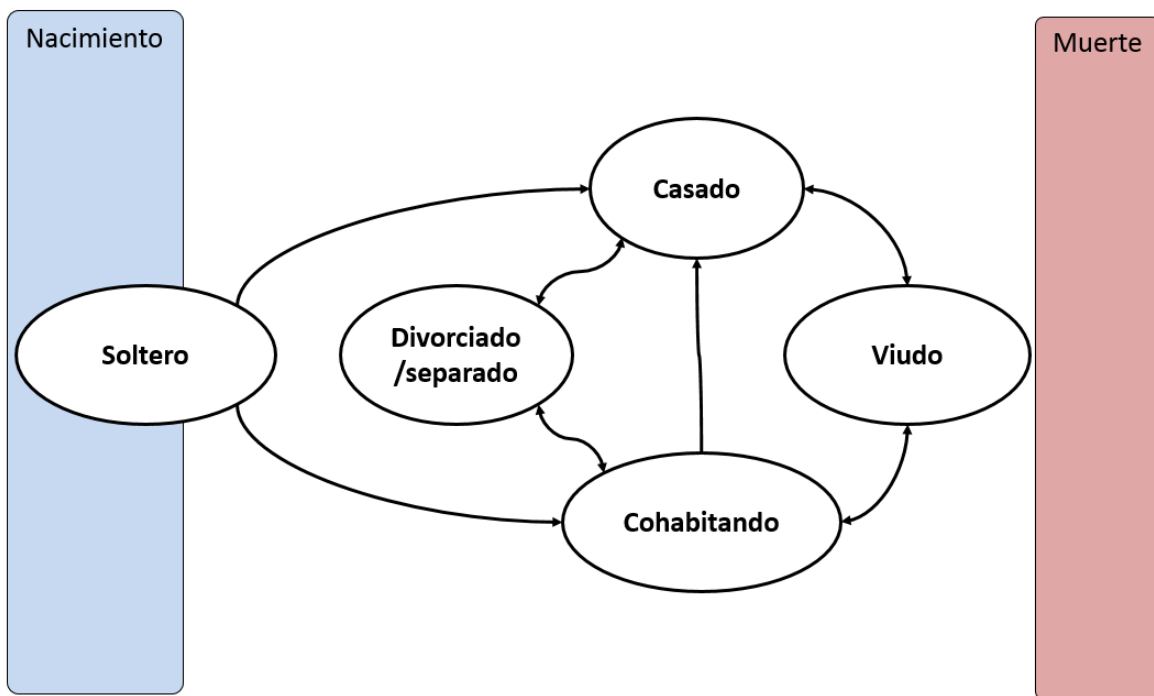


Figura 25: Diagrama de estados maritales por los que puede pasar un individuo

7.3. Historias de usuario

Como en este proyecto se ha seguido una metodología ágil, para definir los requisitos funcionales de la aplicación se ha decidido utilizar historias de usuario [Cohn, 2005] en lugar de casos de uso para la documentación del proyecto. Por esta razón, en lugar de utilizar herramientas clásicas para especificar los requisitos del software, como puede ser la plantilla Volere, se han empleado las historias de usuario, que permiten definir en lenguaje común y con mayor agilidad lo que se espera de la aplicación. Para definir las historias de usuario se han seguido las directrices: Como <rol> quiero <algo> para poder <beneficio>. Además, van acompañadas de discusiones con los usuarios y la definición de pruebas de validación para cada una.

En las bases del “Manifiesto Agile” [Varios, a], se establece que las historias de usuario deben tener las siguientes características: ser *independientes* unas de otras, y referirse a las características y funcionalidades que debe cumplir la aplicación, ser *negociables* y *valoradas* por los clientes y equipo de desarrollo antes de llevarlas a cabo, además se establece la prioridad de cada una y en que iteración de desarrollo se realizarán.

Además estas historias deben ser *estimables* en cuanto al tiempo que lleva desarrollarlas y deben atomizarse para llevarlas a cabo con mayor facilidad. Por último deben ser *verificables*, ya que se definen métodos de validación que permiten chequear que las funcionalidades añadidas con anterioridad sigan funcionando en el momento de añadir una nueva. De esta manera se puede alcanzar con mayor facilidad la meta que se marcó al principio del proyecto, en la fase de planificación, de ir incorporando funcionalidades a la aplicación en cada iteración y tener un producto potencialmente entregable al final de cada una de ellas.

Las historias de usuario deben cumplir con su definición original presente en la palabra INVEST (siglas en inglés con las iniciales de cada característica que debe tener una historia de usuario) y que significa “inversión”, ya que se espera que la correcta definición de las historias de usuario de una aplicación represente una inversión para ésta.

Este proyecto parte de dos aplicaciones que ya están implementadas y que se deben integrar, Yades y ROOT-Sim, buscando mejorar el funcionamiento de la primera incorporándole la segunda. Por esta razón, a partir de las funcionalidades de Yades y de sus características y lo que el cliente espera que cumpla este sistema, se han definido las historias de usuario que aparecen en esta documentación. Se debe cumplir que una vez realizada la modificación, el software continúe teniendo todas sus características y funcionalidades.

En cuanto a los requisitos funcionales de Yades, el proyecto actual debe mantener los 5 componentes demográficos que se le proporcionan a los clientes finales, los demógrafos, para que puedan modelar el comportamiento y evolución de la población.

En la figura 26 se muestra un diagrama con las historias de usuario que debe cumplir la aplicación Yades.

Una característica importante de las historias de usuario son los puntos de historia, que

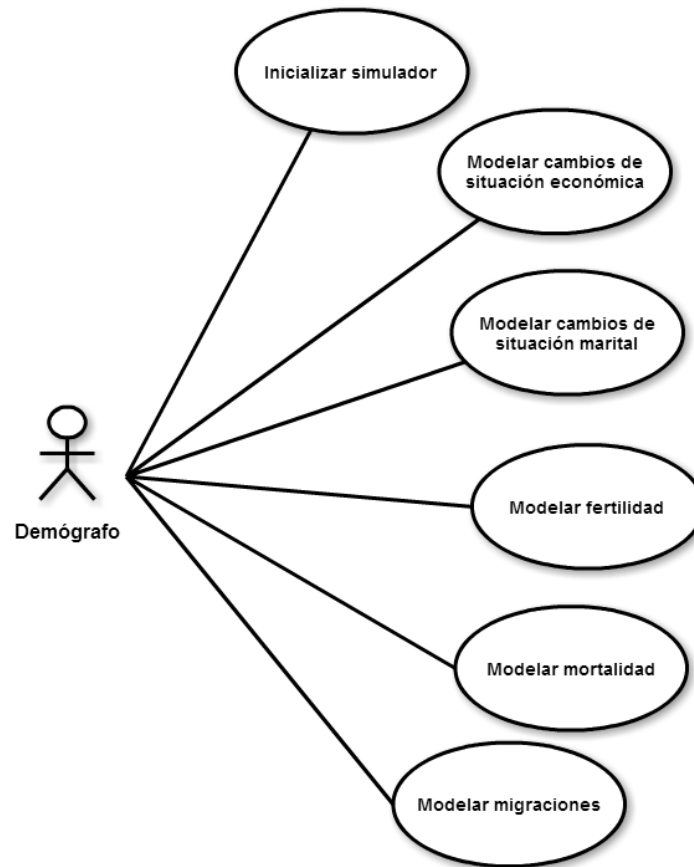


Figura 26: Diagrama de historias de usuario de Yades.

representan la dificultad de cada tarea. En este proyecto se ha estimado que un punto de historia es equivalente a un día de trabajo en exclusividad de jornada completa para un miembro del equipo. Si una tarea requiere más de dos puntos y medio, será necesario atomizarla.

En las metodologías ágiles, las épicas representan un nivel de agregación superior a las historias de usuario, y permiten agruparlas por funcionalidades, temáticas, módulos, etc. Siguiendo este modelo, los componentes demográficos de Yades se han dividido en épicas, ordenadas en función de la importancia y el valor que representan para el cliente. Éstas se han atomizado en diferentes historias de usuario que serán realizadas durante los diferentes sprints de desarrollo. Debido a que cada épica se realizará en un mismo sprint, en las tablas de las historias de usuario que se muestran a continuación no se han incorporado algunos parámetros globales para toda la épica, como por ejemplo: la persona responsable de su desarrollo, el usuario, la iteración asignada o su riesgo.

Las tablas que se presentan a continuación, se corresponden con las historias de usuario originales de Yades adaptadas a la reimplementación con ROOT-Sim. Éstas describen las funcionalidades que debe cumplir la aplicación. En un sistema especificado de manera clásica (no



ágil), esas historias serían los “Requisitos funcionales” de la aplicación.



| | |
|--|-----------------------------------|
| Número: 1 | Usuario: Cliente |
| Épica: Inicialización del simulador | |
| Prioridad en negocio: Alta | Riesgo en desarrollo: Alta |
| Iteración asignada: 3 | |
| Programador responsable: Vanessa Büsing Meneses | |

Tabla 3: Épica: Inicialización del simulador

| |
|---|
| Número: 1.1 |
| Historia de Usuario: Estructuras de simulación |
| Puntos estimados: 0.75 |
| Descripción COMO demógrafo QUIERO disponer de contenedores para almacenar todos los componentes de simulación PARA poder trabajar con datos iniciales concretos de diferentes grupos poblacionales. |
| Validación: Hay estructuras en el modelo capaces de almacenar los datos iniciales. |

Tabla 4: Historia de usuario: Inicialización del simulador 1

Número: 1.2

Historia de Usuario: Métodos de inicialización del simulador

Puntos estimados: 0.5

Descripción
COMO demógrafo
QUIERO disponer de métodos que inicialicen las estructuras de simulación
PARA tener reseteadas las estructuras de simulación y no arrastrar resultados incorrectos.

Validación: Todas las estructuras de simulación contienen los valores establecidos como por defecto.

Tabla 5: Historia de usuario: Inicialización del simulador 2

Número: 1.3

Historia de Usuario: Carga de datos iniciales

Puntos estimados: 0.25

Descripción
COMO demógrafo
QUIERO disponer de métodos que permitan cargar los datos iniciales de manera automática
PARA poder cargar diferentes casos de estudio de manera automática, con sus correspondientes datos.

Validación: al ejecutar los métodos de carga de datos, se cuenta con los datos en el formato esperado por la aplicación.

Tabla 6: Historia de usuario: Inicialización del simulador 3

Número: 1.4

Historia de Usuario: Métodos de inicialización de la población

Puntos estimados: 0.75

Descripción
COMO demógrafo
QUIERO disponer de métodos que inicialicen la población
PARA disponer de una población inicial con unas características previamente definidas.

Validación: Las estructuras de simulación que corresponden a los individuos contienen los valores iniciales.

Tabla 7: Historia de usuario: Inicialización del simulador 4

Número: 1.5

Historia de Usuario: Métodos de validación de inicialización de población

Puntos estimados: 0.5

Descripción
COMO demógrafo
QUIERO disponer de métodos que me permitan comprobar que se han inicializado correctamente los contenedores de datos y validar este componente
PARA trabajar con datos iniciales correctos.

Validación: Los métodos de validación de los datos iniciales permitirán comprobar que el estado de la población inicial sea correcto.

Tabla 8: Historia de usuario: Inicialización del simulador 5

Número: 1.6

Historia de Usuario: Desarrollo del planificador de eventos

Puntos estimados: 0.5

Descripción
COMO demógrafo
QUIERO disponer de un método que permita asignar eventos a los individuos.
PARA observar la influencia de los diferentes componentes demográficos sobre la población.

Validación: se pueden asignar eventos sintéticos a la población.

Tabla 9: Historia de usuario: Inicialización del simulador 6



| | |
|---|-----------------------------------|
| Número: 2 | Usuario: Cliente |
| Épica: Modelización del componente demográfico de fertilidad | |
| Prioridad en negocio: Alta | Riesgo en desarrollo: Alta |
| Iteración asignada: 3 | |
| Programador responsable: Vanessa Büsing Meneses | |

Tabla 10: Épica: Modelización de la fertilidad

| |
|---|
| Número: 2.1 |
| Historia de Usuario: Métodos para calcular la fertilidad |
| Puntos estimados: 0.25 |
| Descripción COMO demógrafo QUIERO calcular la fertilidad que le corresponde a un individuo en función de sus características personales PARA simular el comportamiento del componente demográfico de natalidad y evaluar su influencia en la evolución de la población. |
| Validación: Se prueban diferentes individuos y para todos ellos la fertilidad asignada es la que corresponde con sus características personales. |

Tabla 11: Historia de usuario: Fertilidad 1

Número: 2.2

Historia de Usuario: Métodos para aplicar la fertilidad

Puntos estimados: 0.25

Descripción
COMO demógrafo
QUIERO aplicar la fertilidad a cada individuo, según la que le corresponde.
PARA simular el comportamiento del componente demográfico de natalidad y evaluar su influencia en la evolución de la población.

Validación: Se prueban diferentes individuos y a todos ellos se les aplica la fertilidad que les corresponde con sus características personales.

Tabla 12: Historia de usuario: Fertilidad 2

Número: 2.3

Historia de Usuario: Planificar el evento fertilidad

Puntos estimados: 0.25

Descripción
COMO demógrafo
QUIERO observar el impacto de la fertilidad en los individuos.
PARA simular el comportamiento del componente demográfico de natalidad y evaluar su influencia en la evolución de la población.

Validación: Se envían eventos de fertilidad para los individuos.

Tabla 13: Historia de usuario: Fertilidad 3

Número: 2.4

Historia de Usuario: Propagar el evento fertilidad

Puntos estimados: 0.5

Descripción
COMO demógrafo
QUIERO observar el impacto de la fertilidad en los individuos.
PARA simular el comportamiento del componente demográfico de natalidad y evaluar su influencia en la evolución de la población.

Validación: Se reciben eventos de fertilidad para los individuos y se observa un incremento en la población.

Tabla 14: Historia de usuario: Fertilidad 4



| |
|--|
| Número: 2.5 |
| Historia de Usuario: Métodos de validación de fertilidad |
| Puntos estimados: 0.5 |
| Descripción COMO demógrafo QUIERO disponer de métodos para validar el componente demográfico fertilidad. PARA comprobar que la natalidad simulada se corresponde con las tasas de natalidad reales para el conjunto de población simulada. |
| Validación: Se dispone de gráficos para comparar el componente demográfico de natalidad con el válido, ya sea teórico o sacado de un simulador ya validado. |

Tabla 15: Historia de usuario: Fertilidad 5



| | |
|---|-----------------------------------|
| Número: 3 | Usuario: Cliente |
| Épica: Modelización del componente demográfico de mortalidad | |
| Prioridad en negocio: Alta | Riesgo en desarrollo: Alta |
| Iteración asignada: 3 | |
| Programador responsable: Vanessa Büsing Meneses | |

Tabla 16: Épica: Modelización de la mortalidad

| |
|---|
| Número: 3.1 |
| Historia de Usuario: Métodos para calcular la mortalidad |
| Puntos estimados: 0.25 |
| Descripción COMO demógrafo QUIERO calcular la mortalidad que le corresponde a un individuo en función de sus características personales PARA simular el comportamiento del componente demográfico de mortalidad y evaluar su influencia en la evolución de la población |
| Validación: Se prueban diferentes individuos y para todos ellos la mortalidad asignada es acorde con sus características personales. |

Tabla 17: Historia de usuario: Mortalidad 1

Número: 3.2

Historia de Usuario: Métodos para aplicar la mortalidad

Puntos estimados: 0.25

Descripción

COMO demógrafo

QUIERO aplicar la mortalidad a cada individuo, según la que le corresponde.

PARA simular el comportamiento del componente demográfico de mortalidad y evaluar su influencia en la evolución de la población.

Validación: Se prueban diferentes individuos y a todos ellos se les aplica la mortalidad que les corresponde.

Tabla 18: Historia de usuario: Mortalidad 2

Número: 3.3

Historia de Usuario: Planificar el evento mortalidad

Puntos estimados: 0.25

Descripción

COMO demógrafo

QUIERO observar el impacto de la mortalidad en los individuos.

PARA simular el comportamiento del componente demográfico de mortalidad y evaluar su influencia en la evolución de la población.

Validación: Se envían eventos de mortalidad para los individuos.

Tabla 19: Historia de usuario: Mortalidad 3

Número: 3.4

Historia de Usuario: Propagar el evento mortalidad

Puntos estimados: 0.75

Descripción

COMO demógrafo

QUIERO observar el impacto de la mortalidad en los individuos.

PARA simular el comportamiento del componente demográfico de mortalidad y evaluar su influencia en la evolución de la población.

Validación: Los individuos reciben eventos de mortalidad y hay un descenso en la población.

Tabla 20: Historia de usuario: Mortalidad 4



| |
|---|
| Número: 3.5 |
| Historia de Usuario: Métodos de validación de mortalidad |
| Puntos estimados: 0.5 |
| Descripción COMO demógrafo QUIERO disponer de métodos para validar el componente demográfico de mortalidad. PARA comprobar que la mortalidad simulada se corresponde con las tasas de mortalidad reales para el conjunto de población simulada. |
| Validación: Se comparara mediante gráficos, el componente demográfico de mortalidad con uno ya validado. |

Tabla 21: Historia de usuario: Mortalidad 5



| | |
|--|-----------------------------------|
| Número: 4 | Usuario: Cliente |
| Épica: Modelización de estados económicos | |
| Prioridad en negocio: Alta | Riesgo en desarrollo: Alta |
| Iteración asignada: 4 | |
| Programador responsable: Vanessa Büsing Meneses | |

Tabla 22: Épica: Modelización de estados económicos

| |
|---|
| Número: 4.1 |
| Historia de Usuario: Métodos para determinar los cambios de estados económicos entre los individuos |
| Puntos estimados: 0.25 |
| Descripción COMO demógrafo QUIERO definir cambios entre los diferentes estados económicos para los individuos PARA simular el comportamiento del componente demográfico de estados económicos y evaluar su influencia en la evolución de la población |
| Validación: Se prueban diferentes individuos y a todos ellos se les asigna un estado económico que se corresponde con los valores teóricos previamente definidos. |

Tabla 23: Historia de usuario: Estados económicos 1

Número: 4.2

Historia de Usuario: Métodos para aplicar los estados económicos

Puntos estimados: 0.25

Descripción

COMO demógrafo

QUIERO aplicar los estados económicos a cada individuo, según la que le corresponde.

PARA simular el comportamiento del componente demográfico de estados económicos y evaluar su influencia en la evolución de la población.

Validación: Se prueban diferentes individuos y a todos ellos se les aplica el estado económico que les corresponde.

Tabla 24: Historia de usuario: Estados económicos 2

Número: 4.3

Historia de Usuario: Planificar eventos de estados económicos

Puntos estimados: 0.25

Descripción

COMO demógrafo

QUIERO observar el impacto de los estados económicos en los individuos.

PARA simular el comportamiento del componente demográfico de estados económicos y evaluar su influencia en la evolución de la población.

Validación: Se envían eventos de estados económicos para los individuos.

Tabla 25: Historia de usuario: Estados económicos 3

Número: 4.4

Historia de Usuario: Propagar eventos de estados económicos

Puntos estimados: 0.5

Descripción

COMO demógrafo

QUIERO observar el impacto de los estados económicos en los individuos.

PARA simular el comportamiento del componente demográfico de estados económicos y evaluar su influencia en la evolución de la población.

Validación: Se reciben eventos de estados económicos para los individuos y se observan cambios entre los estados de la población.

Tabla 26: Historia de usuario: Estados económicos 4

| |
|---|
| Número: 4.5 |
| Historia de Usuario: Métodos de validación de estados económicos |
| Puntos estimados: 0.5 |
| Descripción COMO demógrafo QUIERO disponer de métodos para validar el componente demográfico de estados económicos. PARA comprobar que los cambios en los estados económicos simulados se corresponden con los reales para el conjunto de población simulada. |
| Validación: Se dispone de gráficos para comparar el componente demográfico de estados económicos con el válido, ya sea teórico o sacado de un simulador ya validado. |

Tabla 27: Historia de usuario: Estados económicos 5



| | |
|--|-----------------------------------|
| Número: 5 | Usuario: Cliente |
| Épica: Modelización de estados maritales | |
| Prioridad en negocio: Alta | Riesgo en desarrollo: Alta |
| Iteración asignada: 4 | |
| Programador responsable: Vanessa Büsing Meneses | |

Tabla 28: Épica: Modelización de MS

| |
|---|
| Número: 5.1 |
| Historia de Usuario: Métodos para determinar los cambios de estados maritales entre los individuos |
| Puntos estimados: 0.25 |
| Descripción COMO demógrafo QUIERO definir cambios entre los diferentes estados maritales para los individuos PARA simular el comportamiento del componente demográfico de estados maritales y evaluar su influencia en la evolución de la población |
| Validación: Se prueban diferentes individuos y a todos ellos se les asigna un estado marital que se corresponde con los valores teóricos previamente definidos. |

Tabla 29: Historia de usuario: Estados maritales 1

Número: 5.2

Historia de Usuario: Métodos para aplicar los estados maritales

Puntos estimados: 0.5

Descripción

COMO demógrafo

QUIERO aplicar los estados maritales a cada individuo, según la que le corresponde.

PARA simular el comportamiento del componente demográfico de estados maritales y evaluar su influencia en la evolución de la población.

Validación: Se prueban diferentes individuos y a todos ellos se les aplica el estado marital que les corresponde.

Tabla 30: Historia de usuario: Estados maritales 2

Número: 5.3

Historia de Usuario: Planificar eventos de estados maritales

Puntos estimados: 0.5

Descripción

COMO demógrafo

QUIERO observar el impacto de los estados maritales en los individuos.

PARA simular el comportamiento del componente demográfico de estados maritales y evaluar su influencia en la evolución de la población.

Validación: Se envían eventos de estados maritales para los individuos.

Tabla 31: Historia de usuario: Estados maritales 3

Número: 5.4

Historia de Usuario: Propagar eventos de estados maritales

Puntos estimados: 0.5

Descripción

COMO demógrafo

QUIERO observar el impacto de los estados maritales en los individuos.

PARA simular el comportamiento del componente demográfico de estados maritales y evaluar su influencia en la evolución de la población.

Validación: Se reciben eventos de estados maritales para los individuos y se observan cambios en la población como creación de familias o separaciones.

Tabla 32: Historia de usuario: Estados maritales 4

| |
|---|
| Número: 5.4 |
| Historia de Usuario: Métodos de validación de estados maritales |
| Puntos estimados: 0.5 |
| Descripción COMO demógrafo QUIERO disponer de métodos para validar el componente demográfico de estados maritales. PARA comprobar que los cambios en los estados maritales simulados se corresponden con los reales para el conjunto de población simulada. |
| Validación: Se dispone de gráficos para comparar el componente demográfico de estados maritales con el válido, ya sea teórico o sacado de un simulador ya validado. |

Tabla 33: Historia de usuario: Estados maritales 5



| | |
|--|-----------------------------------|
| Número: 6 | Usuario: Cliente |
| Épica: Modelización de migraciones | |
| Prioridad en negocio: Alta | Riesgo en desarrollo: Alta |
| Iteración asignada: 4 | |
| Programador responsable: Vanessa Büsing Meneses | |

Tabla 34: Épica: Modelización de migraciones

| |
|---|
| Número: 6.1 |
| Historia de Usuario: Métodos para determinar las migraciones domésticas en los individuos |
| Puntos estimados: 0.25 |
| Descripción COMO demógrafo QUIERO definir la posibilidad de realizar migraciones para los individuos PARA simular el comportamiento del componente demográfico de migraciones y evaluar su influencia en la evolución de la población |
| Validación: Se prueban diferentes individuos y a todos ellos se les asigna un estado marital que se corresponde con los valores teóricos previamente definidos. |

Tabla 35: Historia de usuario: Migraciones domésticas 1

Número: 6.2

Historia de Usuario: Métodos para aplicar las migraciones domésticas

Puntos estimados: 0.5

Descripción

COMO demógrafo

QUIERO aplicar las migraciones domésticas a cada individuo, según la que le corresponde.

PARA simular el comportamiento del componente demográfico de migraciones domésticas y evaluar su influencia en la evolución de la población.

Validación: Se prueban diferentes individuos y a todos ellos se les aplica las migraciones domésticas que les corresponde.

Tabla 36: Historia de usuario: Migraciones domésticas 2

Número: 6.3

Historia de Usuario: Planificar eventos de migraciones domésticas

Puntos estimados: 0.5

Descripción

COMO demógrafo

QUIERO observar el impacto de las migraciones domésticas en los individuos.

PARA simular el comportamiento del componente demográfico de las migraciones domésticas y evaluar su influencia en la evolución de la población.

Validación: Se envían eventos de migraciones domésticas para los individuos.

Tabla 37: Historia de usuario: Migraciones domésticas 3

Número: 6.4

Historia de Usuario: Propagar eventos de migraciones domésticas

Puntos estimados: 1

Descripción

COMO demógrafo

QUIERO observar el impacto de las migraciones domésticas en los individuos.

PARA simular el comportamiento del componente demográfico de las migraciones domésticas y evaluar su influencia en la evolución de la población.

Validación: Se reciben eventos de migraciones domésticas para los individuos y se observan cambios en la población como aumentos de población en las regiones destino y descenso de la población en las regiones origen.

Tabla 38: Historia de usuario: Migraciones domésticas 4



Número: 6.5

Historia de Usuario: Métodos de validación de migraciones domésticas

Puntos estimados: 0.5

Descripción

COMO demógrafo

QUIERO disponer de métodos para validar el componente demográfico de migraciones domésticas.

PARA comprobar que los cambios en las migraciones domésticas simulados se corresponden con los reales para el conjunto de población simulada.

Validación: Se dispone de gráficos para comparar el componente demográfico de las migraciones domésticas con el válido, ya sea teórico o sacado de un simulador ya validado.

Tabla 39: Historia de usuario: Migraciones domésticas 5

Número: 6.6

Historia de Usuario: Métodos para determinar las migraciones internacionales

Puntos estimados: 0.25

Descripción

COMO demógrafo

QUIERO definir la posibilidad de realizar migraciones internacionales para los individuos

PARA simular el comportamiento del componente demográfico de migraciones internacionales y evaluar su influencia en la evolución de la población

Validación: Se observan llegadas a una región y salidas que se corresponden con las tasas definidas para el componente demográfico de migraciones internacionales.

Tabla 40: Historia de usuario: Migraciones internacionales 1

Número: 6.7

Historia de Usuario: Métodos de validación de migraciones internacionales

Puntos estimados: 0.5

Descripción

COMO demógrafo

QUIERO disponer de métodos para validar el componente demográfico de migraciones internacionales.

PARA comprobar que los cambios en las migraciones internacionales simulados se corresponden con los reales para el conjunto de población simulada.

Validación: Se dispone de gráficos para comparar el componente demográfico de las migraciones internacionales con el válido, ya sea teórico o sacado de un simulador ya validado.

Tabla 41: Historia de usuario: Migraciones internacionales 2



Las tablas 42, 43, 44, 45 y 46 que se muestran a continuación, presentan las historias de usuario originales de Yades adaptadas a la reimplementación con ROOT-Sim que tienen que ver con los atributos de calidad que debe cumplir la aplicación. En un sistema especificado de manera clásica (no ágil), esas historias serían los “Requisitos no funcionales” de la aplicación.



| | |
|--|---|
| Número: 7 | Usuario: Equipo de investigación |
| Historia de usuario: Ejecutar el simulador en un entorno paralelo | |
| Prioridad en negocio: Alta | Riesgo en desarrollo: Alta |
| Puntos estimados: 2 | Iteración asignada: 1 |
| Programador responsable: Vanessa Büsing Meneses | |
| Descripción COMO Equipo de investigación QUIERO ejecutar el simulador en un entorno paralelo PARA aprovechar las características que brindan estos sistemas y poder ejecutar simulaciones con datos reales | |
| Validación: El simulador se ejecuta en arquitectura paralela y produce resultados correctos. | |

Tabla 42: Historia de usuario: Ejecutar el simulador en entorno paralelo

| | |
|---|-----------------------------------|
| Número: 8 | Usuario: Desarrollador |
| Historia de usuario: El código fuente estará bien documentado | |
| Prioridad en negocio: Bajo | Riesgo en desarrollo: Bajo |
| Puntos estimados: 0.5 | Iteración asignada: 1 |
| Programador responsable: Vanessa Büsing Meneses | |
| Descripción COMO Desarrollador QUIERO tener el código fuente del simulador completamente comentado PARA que el código sea mantenible en el tiempo | |
| Validación: El código del simulador cuenta con información suficiente y detallada como para que otro programador sea capaz de entenderlo. | |

Tabla 43: Historia de usuario: Facilidad de uso

| | |
|--|-----------------------------------|
| Número: 9 | Usuario: Demógrafo |
| Historia de usuario: Buena escalabilidad del simulador | |
| Prioridad en negocio: Alta | Riesgo en desarrollo: Alta |
| Puntos estimados: 2 | Iteración asignada: 1 |
| Programador responsable: Vanessa Büsing Meneses | |
| Descripción COMO equipo de investigación QUIERO que al aumentar la cantidad y capacidad de los recursos, el simulador aumente en proporción su rendimiento PARA poder modelar escenarios reales de tamaño variable | |
| Validación: El simulador presenta buena escalabilidad en las pruebas de escalabilidad. | |

Tabla 44: Historia de usuario: Escalabilidad

| | |
|--|--|
| Número: 10 | Usuario: Demógrafo Desarrollador |
| Historia de usuario: Ejecutar el simulador en diferentes plataformas | |
| Prioridad en negocio: Alta | Riesgo en desarrollo: Alta |
| Puntos estimados: 2 | Iteración asignada: 1 |
| Programador responsable: Vanessa Büsing Meneses | |
| Descripción COMO Demógrafo y equipo de investigación QUIERO poder ejecutar el simulador en cualquier arquitectura multicore PARA que el simulador pueda dar servicio en diferentes arquitecturas y sea multiplataforma | |
| Validación: El simulador se puede ejecutar tanto en local como en el superordenador dedicado Marenostum y en el cluster Capitano. | |

Tabla 45: Historia de usuario: Multiplataforma



| | |
|---|--|
| Número: 11 | Usuario: Demógrafo Desarrollador |
| Historia de usuario: Ejecución eficiente del simulador | |
| Prioridad en negocio: Alta | Riesgo en desarrollo: Alta |
| Puntos estimados: 2 | Iteración asignada: 1 |
| Programador responsable: Vanessa Büsing Meneses | |
| Descripción COMO Demógrafo y equipo de investigación QUIERO que la ejecución de la simulación consuma los recursos mínimos durante el tiempo de simulación PARA que sea más escalable | |
| Validación: Al ejecutar el simulador y sacar una traza de la ejecución se observa que solo se consumen los recursos necesarios. | |

Tabla 46: Historia de usuario: Escalabilidad



Este proyecto debe mantener todos los requisitos funcionales y no funcionales de Yades, representados como historias de usuario en este proyecto y, además, debe dar respuesta a un requisito no funcional nuevo, que es el motivo principal por el cual se ha puesto en marcha este proyecto y que pretende resolver la problemática mencionada en las secciones 1.2 y 6.1.4: aumentar el rendimiento de Yades, disminuyendo el tiempo de sincronización de la aplicación. Debido a esto, los requisitos no funcionales adquieren más importancia en el desarrollo de este proyecto.

Como el proyecto se ha planteado de manera ágil de principio a fin, en cada iteración de desarrollo se han llevado a cabo todas las fases de desarrollo de una aplicación, para asegurar que se cumpla con todas las metas establecidas en el tiempo previsto.

En cada iteración de desarrollo (de la 3 a la 6) se realizan las siguientes fases:

- 25 % Análisis
- 15 % Especificación
- 15 % Diseño
- 35 % Implementación/test/documentación
- 10 % Demostración

También se han establecido unos requisitos de hardware y software que los usuarios deben cumplir para poder utilizar la aplicación de simulación. Es necesario que los usuarios dispongan de acceso a un ordenador con alguna versión de Linux instalada y que dispongan de las librerías previamente instaladas:

- autotools (de la versión 1.10 en adelante)
- gcc (viene en el paquete “build-essential” de Linux)
- pthread (viene en el paquete “build-essential” de Linux)
- mpicc
- automake

Actualmente, la aplicación se puede ejecutar tanto en secuencial como en paralelo. En el caso que se desee ejecutar en secuencial no es necesario ningún tipo de hardware adicional, pero con este modo de ejecución no se podrá aprovechar el paralelismo que brinda la aplicación.

En el caso de las ejecuciones en paralelo será necesario disponer de una máquina con arquitectura multicore, ya sea un cluster, un superordenador dedicado o un ordenador de sobremesa.



Actualmente la aplicación se ejecuta en un nodo de computación y puede aprovechar tantos cores como se le indique, siempre y cuando el número de cores sea menor que el número de regiones o LP indicados.

8. Diseño de la solución

En la fase de diseño se especifica cómo se llevará a cabo el desarrollo del software para que cumpla con todas las historias de usuario definidas en la fase de especificación.

8.1. Arquitectura del sistema software

La versión inicial de Yades con la librería de simulación μ sik, empleaba dicha librería como núcleo de simulación. Era la encargada de la gestión de la cola de eventos, de su planificación y entrega a los diferentes destinatarios, de la gestión del tiempo de simulación y de la ventana de simulación así como de la sincronización de las instancias de simulación paralelas.

La versión de Yades definida en la sección de 6.1.2, es desvinculada de la librería μ sik para probar su integración con la librería de simulación paralela ROOT-Sim.

En la figura 27 se presenta la nueva arquitectura de Yades propuesta para este proyecto. En la figura se muestra la aplicación Yades en su totalidad, incluyendo la interfaz gráfica. Sin embargo, las partes que se encuentran en gris, quedan fuera del alcance de este proyecto. En la figura se observa que Yades utiliza ROOT-Sim, de modo que el planificador de eventos de Yades es sustituido por el planificador de eventos que proporciona la librería ROOT-Sim y que permite trabajar con cada uno de los eventos que pueden afectar a las estructuras de simulación de una manera transparente para el usuario. El responsable de controlar la consistencia de estados es sustituido también por el que proporciona la librería ROOT-Sim y que le permite al modelador centrarse en la definición del modelo y dejar la gestión de la consistencia de estados a la aplicación.

En esta primera fase del proyecto, la implementación de la aplicación Yades con ROOT-Sim no es cambiable, ya que no es independiente de la librería de simulación por las razones explicadas en la sección 7.2. Sin embargo, en esta fase de desarrollo del proyecto el objetivo es evaluar la escalabilidad del sistema y valorar si es factible desarrollar una segunda fase de desarrollo, que queda fuera de este trabajo final de carrera, que implique la cambiabilidad del simulador usando diferentes librerías, ya sea escogidas por el usuario o definidas por el implementador, y la adaptación con la interfaz gráfica de la que actualmente dispone Yades, que queda fuera del alcance de este proyecto también.

Debido a que la librería ROOT-Sim está implementada en C y en este momento no es compatible su utilización con un modelo en C++, se ha decidido para esta fase del proyecto sacrificar la orientación a objetos del simulador en pos de evaluar el rendimiento de la aplicación Yades, y si es posible, aumentarlo. Sin embargo, se procurará mantener algunas buenas prácticas de programación para asegurar, dentro de lo posible, la modularidad del código para su posterior refactorización.

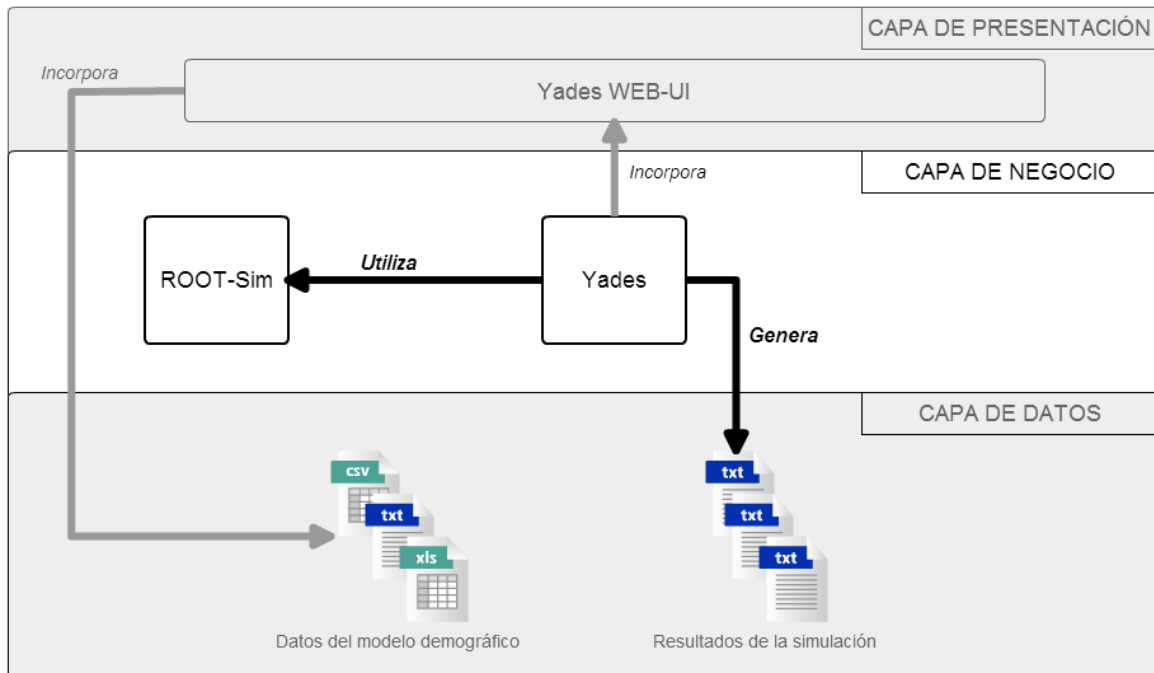


Figura 27: Diagrama de la arquitectura del sistema.

Se ha decidido sustituir las clases existentes en la versión original de Yades por las estructuras de datos que se muestran en la figura 28, donde se presenta el diagrama de las entidades del sistema, que representan los conceptos: región, unidad familiar, individuo y estado de simulación. El último se ha diseñado con el objetivo de poder almacenar el estado de simulación de una iteración a otra, para cada LP.

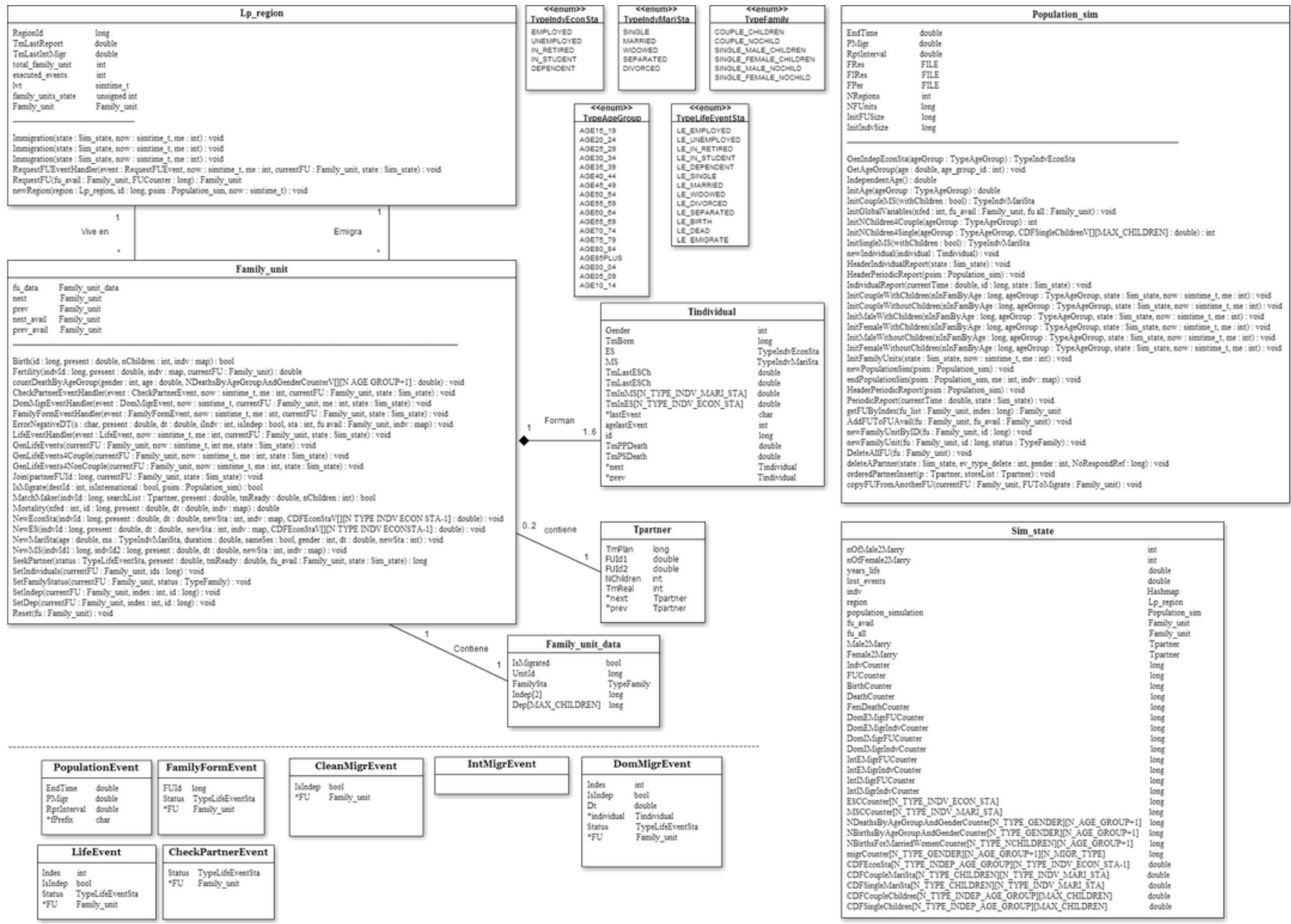


Figura 28: Diagrama de entidades de Yades.



A continuación se presenta las tablas 47, 48, 49, 50, 51, 52, 53, 54 y 55 con las clases, estructuras y métodos de simulación de la versión original de Yades y como se traducen a la nueva versión con ROOT-Sim. estructuras de simulación.

| Yades | Tipo | Función | Visibilidad |
|--|-----------------------|-------------------------------|-------------|
| FUCounter | long | # total FUs creadas | Global |
| FUList | lista con doble final | Ids FU disponibles | Región |
| FU{0...MAX_FU} | lista | Todas las FU | Región |
| BirthCounter | long | # total ★ | Global |
| DeathCounter | long | # total † | Global |
| FemDeathCounter | long | # total ♀ muertas | Global |
| DomEMigrFUCounter | long | # total FU emigr. dom. | Global |
| DomEMigrIndvCounter | long | # total Indv. emigr. dom. | Global |
| DomIMigrFUCounter | long | # total FU. inmigr. dom. | Global |
| DomIMigrIndvCounter | long | # total Indv. inmigr. dom. | Global |
| IntEMigrFUCounter | long | # total FU emigr. inter. | Global |
| IntEMigrIndvCounter | long | # total Indv emigr. inter. | Global |
| IntIMigrFUCounter | long | # total FU inmigr. inter. | Global |
| IntIMigrIndvCounter | long | # total Indv inmigr. inter. | Global |
| ESCCounter{FU FU ∈ Estado económico} | long | # total Indv. en cada ES | Global |
| MSCCounter{FU FU ∈ Estado marital} | long | # total Indv. en cada MS | Global |
| IndvCounter | long | # total Indv creados | Global |
| IndvList | mapa | Indv vivos | Región |
| Male2Marry | long | L. ♂ potenciales | Global |
| Female2Marry | long | L. ♀ potenciales | Global |
| CDFEconSta{0..N_TYPE_INDEP_AGE_GROUP}{0..N_TYPE_INDV_ECON_STA} | double | F. dist. acum. ES | Global |
| CDFCoupleMariSta{0..N_TYPE_CHILDREN}{0..N_TYPE_INDV_MARI_STA} | double | F. dist. acum. ∞ | Global |
| CDFSingleMariSta{0..N_TYPE_CHILDREN}{0..N_TYPE_INDV_MARI_STA} | double | F. dist. acum. solteros | Global |
| CDFCoupleChildren{0..N_TYPE_INDEP_AGE_GROUP}{0..MAX_CHILDREN} | double | F. dist. acum. ∞+hijos | Global |
| CDFSingleChildren{0..N_TYPE_INDEP_AGE_GROUP}{0..MAX_CHILDREN} | long | F. dist. acum. solteros+hijos | Global |

Tabla 47: Estructuras de simulación Yades

| Yades-ROOT-Sim | Tipo | Función | Visibilidad |
|--|--------|-------------------------------|-------------|
| FUCounter | long | # total FUs creadas | LP |
| fu_avail | lista | FU disponibles | LP |
| fu_all | lista | Todas las FU | LP |
| BirthCounter | long | # total ★ | LP |
| DeathCounter | long | # total † | LP |
| FemDeathCounter | long | # total ♀ muertas | LP |
| DomEMigrFUCounter | long | # FU emigr. dom. | LP |
| DomEMigrIndvCounter | long | # total Indv. emigr. dom. | LP |
| DomIMigrFUCounter | long | # total FU. inmigr. dom. | LP |
| DomIMigrIndvCounter | long | # total Indv. inmigr. dom. | LP |
| IntEMigrFUCounter | long | # total FU emigr. inter. | LP |
| IntEMigrIndvCounter | long | # total Indv emigr. inter. | LP |
| IntIMigrFUCounter | long | # total FU inmigr. inter. | LP |
| IntIMigrIndvCounter | long | # total Indv inmigr. inter. | LP |
| ESCCounter{FU FU ∈ Estado económico} | long | # total Indv. en cada ES | LP |
| MSCCounter{FU FU ∈ Estado marital} | long | # total Indv. en cada MS | LP |
| IndvCounter | long | # total Indv creados | LP |
| Indv | mapa | Indv. vivos | LP |
| Male2Marry | lista | L. ♂ potenciales | LP |
| Female2Marry | lista | L. ♀ potenciales | LP |
| CDFEconSta{0..N_TYPE_INDEP_AGE_GROUP }{0..N_TYPE_INDV_ECON_STA } | double | F. dist. acum. ES | LP |
| CDFCoupleMariSta{0..N_TYPE_CHILDREN}{0..N_TYPE_INDV_MARI_STA } | double | F. dist. acum. ∞ | LP |
| CDFSinglMariSta{0..N_TYPE_CHILDREN}{0..N_TYPE_INDV_MARI_STA } | double | F. dist. acum. solteros | LP |
| CDFCoupleChildren{0..N_TYPE_INDEP_AGE_GROUP}{0..MAX_CHILDREN } | double | F. dist. acum. ∞+hijos | LP |
| CDFSinglChildren{0..N_TYPE_INDEP_AGE_GROUP}{0..MAX_CHILDREN } | long | F. dist. acum. solteros+hijos | LP |

Tabla 48: Diseño de estructuras de simulación Yades-ROOT-Sim 1

| Yades | Tipo | Función | Visibilidad | Yades-ROOT-Sim | Tipo | Función | Visibilidad |
|-------------------|-------|------------------------|-------------|-------------------|------------|--------------------------|-------------|
| PopulationEvent | clase | Evento vacío | Region | PopulationEvent | estructura | Evento vacío | LP |
| ReportEvent | clase | Evento reporte | Region | | | | |
| LifeEvent | clase | Evento de vida | Region | LifeEvent | estructura | Evento de vida | LP |
| FamilyFormEvent | clase | Evento formar familia | Region | FamilyFormEvent | estructura | Evento formar familia | LP |
| CheckPartnerEvent | clase | Evento buscar pareja | Region | CheckPartnerEvent | estructura | Evento buscar pareja | LP |
| RequestFUEvent | clase | Evento solicitar migr. | Region | | | | |
| FUReadyEvent | clase | Evento confirmar migr. | Region | | | | |
| DomMigrEvent | clase | Evento migrar | Region | DomMigrEvent | estructura | Evento migrar | LP |
| IntMigrEvent | clase | Evento migración int. | Region | IntMigrEvent | estructura | Evento migración int. | LP |
| | | | | ClearEvent | estructura | Limpiar estructura migr. | LP |

Tabla 49: Diseño de estructuras de simulación 2

| Yades | Tipo | Función | Visibilidad | Yades-ROOT-Sim | Tipo | Función | Visibilidad |
|---------------------|------------|--------------------|-------------|----------------|------------|---------------------------|-------------|
| FamilyUnit | clase | Unidad familiar/LP | Region | Family_unit | estructura | P. a FU | LP |
| Region | clase | Región Federal/LP | Region | region | estructura | Región Federal/LP | LP |
| PopulationSimulator | clase | Parám. simulación | Region | Population_sim | estructura | Parám. simulación | LP |
| TIndividual | estructura | Individuo | Region | Tindividual | estructura | Individuo | LP |
| TPartner | estructura | Soltero disponible | Region | Tpartner | estructura | Soltero disponible | LP |
| | | | | Sim_state | estructura | Almacenar estado de la LP | LP |

Tabla 50: Diseño de estructuras de simulación 3

Métodos de la clase FamilyUnit

| Yades | Yades-ROOT-Sim |
|--|--|
| Birth(id: long, present: double , nChildren: int) | bool Birth(id: long, present: double, nChildren: int, indiv: mapa) |
| double Fertility(indvId: long, present: double) | double Fertility(indvId: long, present: double, indiv: mapa, currentFU: Family_unit) |
| void countDeathByAgeGroup(gender: int, age: double) | void countDeathByAgeGroup(gender: int, age: double, NDeathsByAgeGroupAndGenderCounterV{0..N_TYPE_GENDER}{0..N_AGE_GROUP+1}: long) |
| void CheckPartnerEventHandler(event: CheckPartnerEvent) | void CheckPartnerEventHandler(event: CheckPartnerEvent, now: tiempo, me: int, currentFU: Family_unit, state: Sim_state) |
| void FUReadyEventHandler(event: FUReadyEvent) | |
| void DomMigrEventHandler(event: DomMigrEvent) | void DomMigrEventHandler(event: DomMigrEvent, now: tiempo, currentFU: Family_unit, me: int, state: Sim_state) |
| void FamilyFormEventHandler(event: FamilyFormEvent) | void FamilyFormEventHandler(event: FamilyFormEvent, now: tiempo, me: int, currentFU: Family_unit, state: Sim_state) |
| void ErrorNegativeDT(tipo: char, present: double, dt: double, iIndv: int, isIndep: bool, sta: int) | void ErrorNegativeDT(s: char, present: double, dt: double, iIndv: int, isIndep: bool, sta: int, fu_avail: Family_unit, indiv: mapa) |
| virtual void execute(event: SimEvent) | |
| void LifeEventHandler(event: LifeEvent) | void LifeEventHandler(event: LifeEvent, now: tiempo, me: int, currentFU: Family_unit, state: Sim_state) |
| void GenLifeEvents(void) | void GenLifeEvents(currentFU: Family_unit, now: tiempo, me: int, state: Sim_state) |
| void GenLifeEvents4Couple(void) | void GenLifeEvents4Couple(currentFU: Family_unit, now: tiempo, me: int, state: Sim_state) |
| void GenLifeEvents4NonCouple(void) | void GenLifeEvents4NonCouple(currentFU: Family_unit, now: tiempo, me: int, state: Sim_state) |
| long getId() | |
| long getIndep(index: int) | |
| virtual void init(void) | |
| void Join(unitId: long) | void Join(partnerFUId: long, currentFU: Family_unit, state: Sim_state) |
| virtual bool IsMigrate(destId: int, isInternational: bool) | bool IsMigrate(destId: int, isInternational: bool, psim: Population_sim) |
| virtual bool MatchMaker(indvId: long, iter: Tpartner, present: double, tmReady: double, nChildren: int) | bool MatchMaker(indvId: long, searchList: Tpartner, present: double, tmReady: double, nChildren: int) |
| virtual double Mortality(id: long, present: double, dt: double) | double Mortality(nfed: int, id: long, present: double, dt: double, indiv: mapa) |
| void NewEconSta(indvId: long, present: double, dt: double, newSta: int) | void NewEconSta(indvId: long, present: double, dt: double, newSta: int, indiv: mapa, CDFEconSta{0..N_TYPE_INDEP_AGE_GROUP}{0..N_TYPE_INDV_ECON_STA}: double) |
| virtual void NewES(id: long, present: double, evTime: double, newSta: int) | void NewES(indvId: long, present: double, dt: double, newSta: int, indiv: mapa, CDFEconSta{0..N_TYPE_INDEP_AGE_GROUP}{0..N_TYPE_INDV_ECON_STA}: double) |
| void NewMariSta(age: double, ms: TypeIndvMariSta, duration: double, sameSex: bool, gender: int, dt: double, newSta: int) | void NewMariSta(age: double, ms: TypeIndvMariSta, duration: double, sameSex: bool, gender: int, dt: double, newSta: int) |
| virtual void NewMS(id1: long, id2: long, present: double, evTime: double, newSta: int) | void NewMS(indvId1: long, indivId2: long, present: double, dt: double, newSta: int, indiv: mapa) |

Tabla 51: Diseño de métodos FamilyUnit 1

Métodos de la clase FamilyUnit

| Yades | Yades-ROOT-Sim |
|--|---|
| long SeekPartner(status: TypeLifeEventSta, present: double, tmReady: double) | long SeekPartner(status: TypeLifeEventSta, present: double, tmReady: double, fu_avail: Family_unit, state: Sim_state) |
| void SetIndividuals(ids: long) | void SetIndividuals(currentFU: Family_unit, ids: long) |
| SetFamilyStatus(status: TypeFamily) | void SetFamilyStatus(currentFU: Family_unit, status: TypeFamily) |
| SetIndep(index: int, id: long) | void SetIndep(currentFU: Family_unit, index: int, id: long) |
| SetDep(index: int, id: long) | void SetDep(currentFU: Family_unit, index: int, id: long) |
| Reset(void) | void Reset(fu: Family_unit) |
| undo_event(SimEventBase) | |
| wrapup(void) | |

Tabla 52: Diseño de métodos FamilyUnit 2

Métodos globales

| Yades | Yades-ROOT-Sim |
|---|--|
| TypeIndvEconSta GenIndepEconSta(ageGroup: TypeAgeGroup) | TypeIndvEconSta GenIndepEconSta(ageGroup: TypeAgeGroup) |
| GetAgeGroup(age: double, age_group_id: int) | void GetAgeGroup(age: double, age_group_id: int) |
| inline double IndependentAge() | inline double IndependentAge() |
| double InitAge(ageGroup: TypeAgeGroup) | double InitAge(ageGroup: TypeAgeGroup) |
| TypeIndvMariSta InitCoupleMS(withChildren: bool) | TypeIndvMariSta InitCoupleMS(withChildren: bool) |
| void InitGlobalVariables(nfed: int) | void InitGlobalVariables(nfed: int, fu_avail: Family_unit, fu_all: Family_unit) |
| int InitNChildren4Couple(ageGroup: TypeAgeGroup) | int InitNChildren4Couple(ageGroup: TypeAgeGroup) |
| int InitNChildren4Single(ageGroup: TypeAgeGroup) | int InitNChildren4Single(ageGroup: TypeAgeGroup, CDFSsingleChildren{N_TYPE_INDEP_AGE_GROUP}{MAX_CHILDREN}: double) |
| TypeIndvMariSta InitSingleMS(withChildren: bool) | TypeIndvMariSta InitSingleMS(withChildren: bool) |
| | void newIndividual(individual: Tindividual) |

Tabla 53: Diseño de métodos globales

Métodos de clase PopulationSimulator

| Yades | Yades-ROOT-Sim |
|---|--|
| HeaderIndividualReport(void) | void HeaderIndividualReport(state: Sim_state) |
| HeaderPeriodicReport(void) | extern void HeaderPeriodicReport(psim: Population_sim) |
| IndividualReport(currentTime: double, id: long) | void IndividualReport(currentTime: double, id: long, state: Sim_state) |
| init(ac: int, av[]: char) | |
| void InitCoupleWithChildren(nInFamByAge: long, TypeAgeGroup: ageGroup) | void InitCoupleWithChildren(nInFamByAge: long, TypeAgeGroup: ageGroup, state: Sim_state, now: tiempo, me: int) |
| void InitCoupleWithoutChildren(nInFamByAge: long, const TypeAgeGroup) | void InitCoupleWithoutChildren(nInFamByAge: long, const TypeAgeGroup ageGroup, state: Sim_state, now: tiempo, me: int) |
| void InitMaleWithChildren(nInFamByAge: long, TypeAgeGroup: ageGroup) | void InitMaleWithChildren(nInFamByAge: long, TypeAgeGroup: ageGroup, state: Sim_state, now: tiempo, me: int) |
| void InitFemaleWithChildren(nInFamByAge: long, TypeAgeGroup: ageGroup) | void InitFemaleWithChildren(nInFamByAge: long, TypeAgeGroup: ageGroup, state: Sim_state, now: tiempo, me: int) |
| void InitMaleWithoutChildren(nInFamByAge: long, TypeAgeGroup: ageGroup) | void InitMaleWithoutChildren(nInFamByAge: long, TypeAgeGroup: ageGroup, state: Sim_state, now: tiempo, me: int) |
| void InitFemaleWithoutChildren(nInFamByAge: long, TypeAgeGroup: ageGroup) | void InitFemaleWithoutChildren(nInFamByAge: long, TypeAgeGroup: ageGroup, state: Sim_state, now: tiempo, me: int) |
| void InitFamilyUnits(void) | void InitFamilyUnits(state: Sim_state, now: tiempo, me: int) |
| void InitProcesses(nfed: int) | |
| void pre_init(void) | |
| | void newPopulationSim(psim: Population_sim) |
| | void endPopulationSim(psim: Population_sim, me: int, indv: Hashmap) |
| void HeaderPeriodicReport(void) | extern void HeaderPeriodicReport(psim: Population_sim) |
| void PeriodicReport(tm: double) | void PeriodicReport(currentTime: double, state: Sim_state) |
| void ReturnFU(index: long) | Family_unit *getFUByIndex(fu_list: Family_unit, index: long) |
| virtual void run(void) | |
| | void AddFUToFUAvail(fu: Family_unit, fu_avail: Family_unit) |
| | void newFamilyUnitByID(fu: Family_unit, id: long) |
| | void newFamilyUnit(fu: Family_unit, id: long, status: TypeFamily) |
| | void DeleteAllFU(fu: Family_unit) |
| | void deleteAPartner(state: Sim_state, ev_type_delete: int, gender: int, NoRespondRef: long) |
| | void orderedPartnerInsert(partner: Tpartner, storeList: Tpartner) |
| | void copyFUFromAnotherFU(currentFU: Family_unit, FUMigrate: Family_unit) |

Tabla 54: Diseño de métodos PopulationSimulator

Métodos de clase región

| Yades | Yades-ROOT-Sim |
|---|---|
| virtual void execute(event: SimEvent) | |
| virtual void Immigration(void) | void Immigration(state: Sim_state, now: tiempo, me: int) |
| virtual void init(void) | |
| void IntMigrEventHandler(event: IntMigrEvent) | void IntMigrEventHandler(event: IntMigrEvent, now: tiempo, me: int, state: Sim_state) |
| void RequestFUEventHandler(event: RequestFUEvent) | void RequestFUEventHandler(event: RequestFUEvent, now: tiempo, me: int, currentFU: Family_unit, state: Sim_state) |
| void ReportEventHandler(event: ReportEvent) | |
| virtual void undo_event(e: SimEventBase) | |
| virtual void wrapup(void) | |
| void report_status(int: SimTime, et: SimTime) | |
| long RequestFU(void) | Family_unit *RequestFU(fu_avail: Family_unit, FUCounter: long) |
| | void newRegion(region: Lp_region, id: long, psim: Population_sim, now: tiempo) |

Tabla 55: Diseño de métodos Region

Gracias a la definición de la estructura de estado de simulación (*Sim_state*) se puede aprovechar la pila de estados (`void *ptr`) que proporciona ROOT-Sim y que está disponible en la función especulativa: *void ProcessEvent(...)*. Con ella se evita el uso de variables globales, que representan un peligro para la ejecución de la simulación al no poder asegurar que se mantiene el estado de cada región en cada paso de simulación. Cada LP tiene su espacio de variables reservado, y gracias a la pila proporcionada por ROOT-Sim se puede acceder a ellas. Al comienzo de la simulación, en el evento INIT, se le indica al simulador que la variable que contendrá toda la información de la simulación para cada LP será la designada en la función: *extern void (*SetState)(void *new_state)*. Si no existiera este elemento, como las variables serían globales para toda la simulación, todos los LP podrían acceder a ellas y modificarlas con sus datos, modificando los datos almacenados por los otros LP y creando un problema de inconsistencia de datos conocido como *condición de carrera* (race condition).

La estructura *Sim_state* incluye todas las variables de Yades que permiten definir el estado de simulación de una región, entre ellas destacan: la lista de unidades familiares, la lista de individuos, la lista de individuos que buscan pareja dentro de una región, las variables auxiliares que permiten almacenar las estadísticas de la población que van sucediendo a lo largo de la simulación, entre otros.

En la figura 29 se muestra un diagrama del diseño de las entidades de Yades con ROOT-Sim que representan los eventos que se envían para modificar el comportamiento de las familias a lo largo de la simulación.

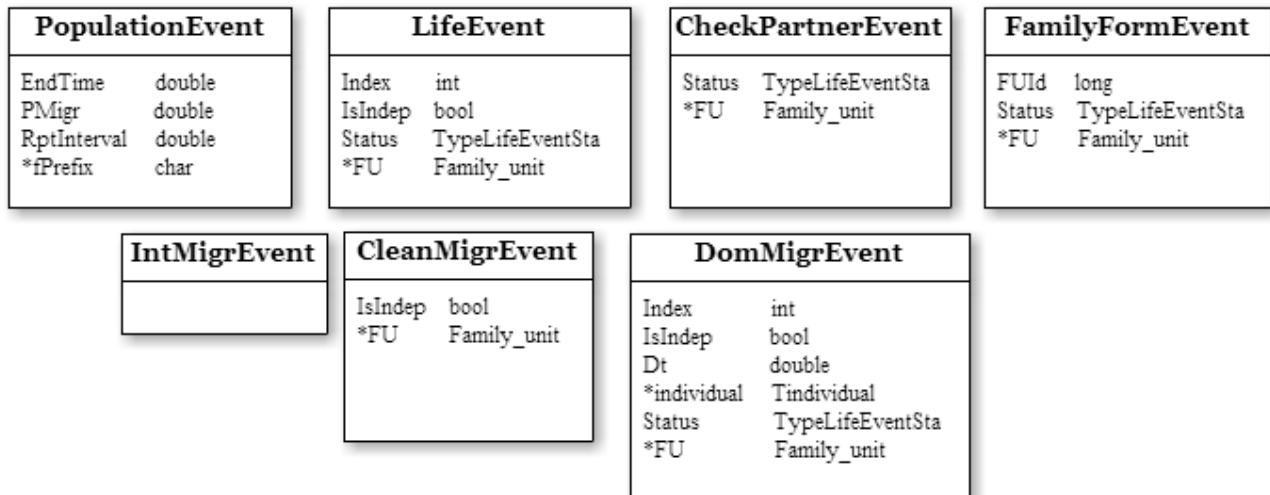


Figura 29: Diagrama de las entidades que representan los eventos de Yades.

8.2. Planificador de eventos

Para implementar el planificador de eventos de Yades con ROOT-Sim, es necesario usar la función de ROOT-Sim *void ProcessEvent(...)* que permite recibir un evento entrante concreto destinado a un LP específico.

Para cada tipo de evento que se puede recibir en Yades, se creará un caso para recibirlo y aplicarlo sobre el LP y la familia indicada por el evento. Esta función tiene un comportamiento especulativo, por lo que sólo se realizarán acciones que permitan realizar una marcha atrás si es necesario.

Los eventos de simulación pueden ser de tipo: *EV_LIFE*, *EV_FAMILY*, *EV_CHECK*, *EV_CLEAN_MIGR*, *EV_DOM_MIGR* y *EV_INT_MIGR*. Los eventos de tipo vida pueden ser: fertilidad, mortalidad, cambio de estado marital o cambio de estado económico.

Los eventos de tipo marital se realizan en varios pasos, en el momento en que se planifica que un individuo quiere encontrar pareja, se miran los individuos disponibles en el sistema, si hay algún individuo disponible para emparejarse con el primero, se planifica un evento de tipo *EV_FAMILY* donde se juntará con la pareja y con sus hijos, en caso de que tenga. Al contrario, si no encuentra a nadie para contraer matrimonio, se planifica un evento de tipo *EV_CHECK* que significa que en el próximo evento se mirará si hay alguien para contraer matrimonio y en caso de que no haya nadie disponible, saldrá de la lista de candidatos para emparejarse.

Cada evento de simulación se envía a una familia específica. A esta familia no se le planificará otro evento hasta que no se haya entregado el primero. De esta manera se evitan posibles problemas de inconsistencia temporal en el momento de entregar los eventos. Si no se toman estas medidas se podría enviar un evento a una familia con unas condiciones específicas de sus individuos, y en el momento en que reciban esos eventos ya no las tengan, generando problemas de consistencia de datos o de estado.

Estos problemas podrían suceder si, por ejemplo, se diera la siguiente situación y no se tomarán las medidas antes mencionadas: en el instante $t=1$ se envía un evento de matrimonio a una persona de la *FU1*, *Indep0* (de ahora en adelante *fu10*) para un instante $t=2$. En ese instante, la persona *fu10* mira la lista de parejas disponibles y encuentra que la persona de la *FU2*, *Indep0* (de ahora en adelante *fu20*) está disponible para formar familia. A continuación, se planifica un evento *EV_FAMILY* para el instante $t=5$. En el instante $t=3$, como la persona *fu20* aún no ha encontrado pareja (si la ha encontrado pero no se casa aún), se le planifica un evento de muerte para el instante $t=4$. Cuando llega a este tiempo, el evento se entrega y la persona *fu20* muere. En el instante $t=5$ le tocaba casarse, y el evento se realiza sobre el individuo *fu10* y *fu20*, aunque ya no se encuentra en el mismo estado sino que está muerto, por lo que se genera un error. Este proceso se puede observar en la figura 30.

Además, se ha definido un control adicional para evitar inconsistencias de estados al entregar un evento de tipo estado marital o de tipo migración. El problema que se describió anteriormente

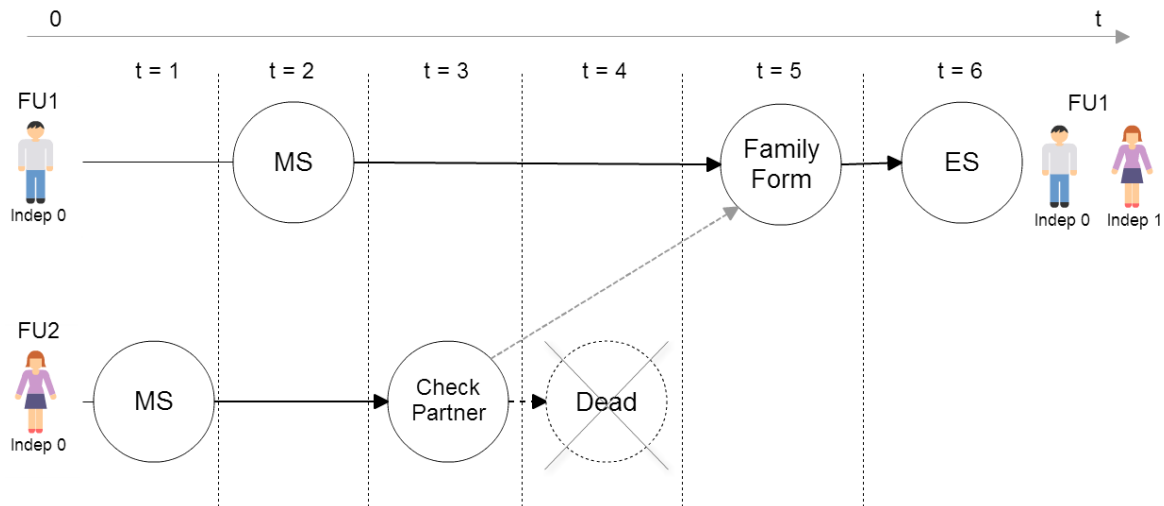


Figura 30: Conflicto de inconsistencia de estados en el MS para el caso que un evento anterior al matrimonio afecte a uno de los miembros

no podría suceder con los filtros ya definidos en la versión original de Yades. Sin embargo se podría dar esta situación: en el instante $t=1$ se envía un evento de matrimonio a una persona de la $FU1$, $Indep0$ (de ahora en adelante $fu10$) para un instante $t=2$. En ese instante, la persona $fu10$ mira la lista de parejas disponibles y encuentra que la persona de la $FU2$, $Indep0$ (de ahora en adelante $fu20$) está disponible para formar familia. Se planifica este evento para el instante $t=5$. En el instante $t=3$, la persona $fu20$ como aún no ha encontrado pareja (si la encontró, pero aún no se ha casado), por lo que se le planifica un evento de cambio de estado económico para el instante $t=6$. Llegado el instante $t=5$, la persona $fu10$ y la $fu20$ forman familia. En el instante $t=6$ se entrega el evento económico que afecta a la persona $fu20$ aunque ya no se encuentra en el mismo estado sino que se casó (cuando una persona se casa se “muda” a la unidad familiar de su pareja si esta fue la que planificó casarse), por lo que se genera un error. Este proceso se puede observar en la figura 31. Por esta razón se ha definido un control para eventos perdidos en el que, antes de entregar un evento se evalúa si los datos del evento se corresponden con los de la familia actual. En caso contrario el evento no se entrega y se contabiliza el número de eventos perdidos o no entregados.

8.3. Confirmación del estado

Para implementar la confirmación de estado y el lugar donde se realizarán las acciones finales del simulador, se emplea la función de ROOT-Sim $bool OnGVT(...)$ que permite realizar acciones que requieran un estado confirmado. El propio sistema de ROOT-Sim comprueba el estado de la simulación cada 2 segundos y ejecuta esta función.

Además esta función se utiliza para terminar la simulación, aquí se mira el estado de la

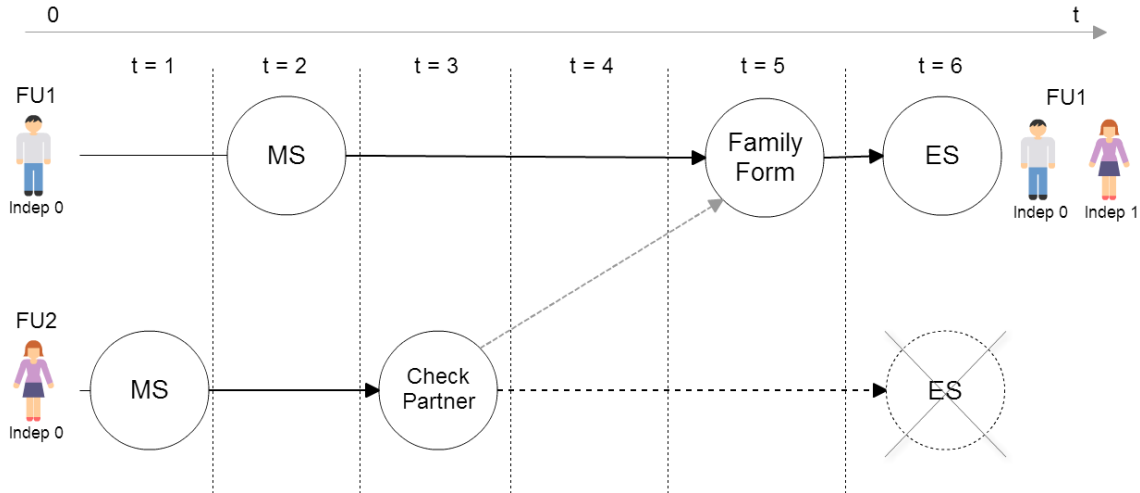


Figura 31: Conflicto de inconsistencia de estados en el MS para el caso que un evento posterior al matrimonio afecte a uno de los miembros

simulación y se ejecuta la condición de finalización previamente definida para cada LP. En el caso de la nueva implementación de Yades, esta condición es un número determinado de años, definido por el modelador en el código o que puede pasar por línea de comandos a la aplicación. En la función *bool OnGVT(...)* se compara el tiempo actual del sistema con el tiempo de finalización, y en caso de que sea el momento de terminar, se le notifica a cada LP.

8.4. Reportes

Los reportes para conocer el estado de simulación pueden ser finales o periódicos. En la versión de Yades original, estos reportes se producían mediante un evento `EV_REPORT` que se le enviaba a la región y aquí se realizaban las acciones de reporte específicas.

Sin embargo, esta aproximación no es la más indicada en una simulación optimista para realizar reportes de estado, si se imprime un estado no confirmado en un archivo de texto y es necesario realizar un rollback, la acción de imprimir en un archivo no podrá ser deshecha. Por esta razón, se ha decidido realizar esta acción en la función *bool OnGVT(...)* donde se trabaja con estados de simulación ya confirmados.

Los reportes se pueden realizar cada intervalo de tiempo definido por el modelador, además del reporte final y el inicial. A través de la línea de comandos el modelador tiene la opción de especificar cada cuantos años, o meses (tiempo de simulación) desea obtener un reporte.

8.5. Migraciones

Para implementar las migraciones en la aplicación de Yades con ROOT-Sim, se ha modificado el proceso de la versión original. En ésta, tal como se vió en la sección 6.1.2 se usaban 3 eventos para realizar todos los pasos de las migraciones domésticas: `EV_REQ_FU`, `EV_FU_READY` y `EV_DOM_MIGR`. En el evento `EV_REQ_FU` se envía la petición de migrar una familia al LP destino, si éste tiene espacio libre para recibirla envía una confirmación (`EV_FU_READY`) y finalmente se envía la migración de la familia (`EV_DOM_MIGR`). En este esquema de migraciones, se envía todo el LP durante la migración, esto significa que no solo se envían los datos de la familia que está migrando sino toda la información que caracteriza a un LP en el modelo original de Yades con μ sik.

Para la nueva versión de Yades, y debido a la arquitectura particular empleada en ROOT-Sim, no es necesario realizar los pasos intermedios de `EV_REQ_FU` y `EV_FU_READY`. Antes de enviar la migración se realiza una copia de los datos de la familia que emigrará, debido a que cada LP tiene su propio ámbito de visión de su memoria, y se envían los datos al LP destino a través del evento `EV_DOM_MIGR`. A la vez, se envía un evento de tipo `EV_CLEAR_FU` con la misma marca temporal que para el `EV_DOM_MIGR`, que se encarga de liberar la estructura de datos empleada por la familia que ya no habita en una región sino en otra.

En esta nueva definición de las migraciones, las familias no constituyen un LP sino que son representadas por datos, por esta razón en el momento de hacer una migración solo se envían los datos de la familia a migrar y no todos los datos y procesos del LP que realiza la migración. Este cambio representa un importante avance para el rendimiento de la aplicación, ya que no es necesario migrar el LP entero sino solo unos datos representados por tipos básicos. Cuando una familia migra, en la región destino se escoge una familia disponible (que no forme parte de la población activa del sistema) y se copian todos los datos de la familia que está migrando allí. En la región origen, por su parte, se deja libre la familia que estaba usando la familia que emigra para que sea reaprovechada por futuras familias. En el caso que no haya más familias disponibles, se podrá crear una familia nueva para alojar a las familias que lleguen.

En la figura 32 se muestran dos diagramas que representan las migraciones en Yades, antes y después de su implementación con ROOT-Sim.

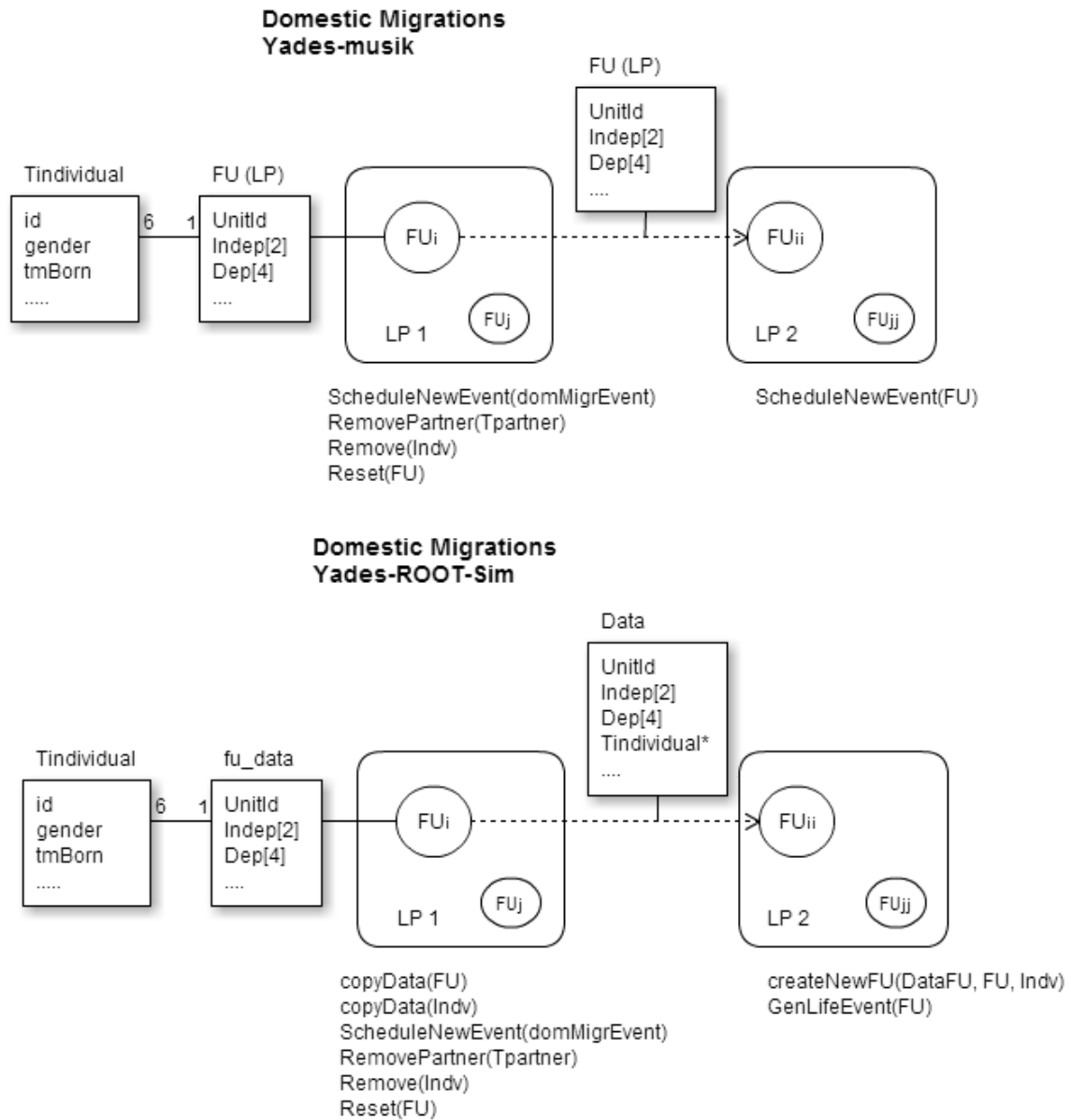


Figura 32: Diagrama de migraciones domésticas de Yades.

9. Implementación de la solución

Para la implementación de la solución a la problemática descrita en la sección 1.2, en primer lugar, será necesario integrar Yades con ROOT-Sim.

Como se explicó en las secciones 6.1.2 y 8.1, Yades está implementado usando la librería de simulación paralela *μsik*, y todas sus clases heredan de esta librería. Por lo tanto, es necesario desvincular las clases de Yades de las clases de *μsik* manteniendo las mismas funcionalidades que tenían en su implementación inicial. Para realizar la integración es necesario traducir todo el código de Yades de C++ a C, para hacerlo compatible con ROOT-Sim. Esta decisión implica sacrificar la orientación a objetos que C++ proporciona en pos de aumentar el rendimiento, para hacer más escalable la aplicación.

9.1. Instalación previa y despliegue

Para empezar a trabajar con ROOT-Sim, el primer paso es descargarse la librería del repositorio de github: <https://github.com/HPDCS/ROOT-Sim.git>

Es necesario instalarlo en todos los equipos que se usan para la realización del proyecto. Para este TFG la aplicación se ha probado en cuatro tipos de equipo diferentes, con arquitecturas diferentes entre ellos con el fin de asegurar que el simulador es compatible en diferentes escenarios. Por tanto, el proceso de instalación varía en cada caso. Los equipos en los que se ha probado y desarrollado la aplicación son:

- Ordenador portátil con procesador Intel Core i3 con 4GB de RAM, de 64 bits.
- Ordenador de sobremesa con procesador Intel Core i5 con 8GB de RAM, de 64 bits.
- Marenostrom 3
- Capitano

En el caso de la instalación en Marenostrom no se tiene acceso al repositorio de GIT por lo que se descarga el paquete y se descomprime manualmente a través del comando: *unzip paquete*.

El proceso estándar de configuración e instalación de ROOT-Sim es el siguiente:

1. Ejecutar el script *autogen.sh*, que ejecuta todos los comandos para configurar las autotools y chequea que todos los parámetros estén correctamente definidos. Además chequea que se disponga de la librería de simulación paralela *mpicc*.

Por motivos de seguridad, en MN3 no se puede ejecutar como super usuario, por lo que antes del primer paso es necesario hacer algunas modificaciones en el *.bashrc* para poder instalar sin permisos de administrador. Por esta razón, se edita el fichero *.bashrc* con el editor



vim, añadiendo la siguiente línea al final: `~$: export PATH=$PATH:/home/user/rootSIM/bin/`.
A continuación, se vuelve a cargar el contenido del `bashrc` a través del comando: `~$: source .bashrc`.

2. Ejecutar el script `./configure` que se genera automáticamente gracias a las autotools. En el caso de la instalación en las máquinas locales se añade la opción `-disable-ult`. En el caso de la instalación en MN3 es necesario añadir la siguiente información para poder decirle dónde se instalará la aplicación: `-prefix=$HOME/rootSIM`
3. Ejecutar el comando `make` que ejecuta el Makefile generado por el script anterior.
4. Ejecutar el comando `sudo make install` en modo súper usuario para instalar ROOT-Sim. En MN3 no se usa el super usuario.

Para compilar la aplicación se ejecuta: `~$: rootsim-cc -pthread *.c -o ejecutable`. En MN no es necesario indicarle que debe añadir la librería `pthread` como en local. Para ejecutar la aplicación, se utiliza el siguiente comando: `~$: ./ejecutable -np NUMBER_OF_PROCESSORS -nprc NUMBER_OF_LPS` (`NUMBER_OF_LPS` tiene que ser un cuadrado perfecto). En caso que se desee ejecutar en secuencial, se debe cambiar la opción `-np` por `-sequential`.

9.2. Estructuras de datos

Es necesario definir nuevas estructuras de datos que puedan realizar las mismas funcionalidades que las antiguas clases de Yades. Además, debido a las particularidades de ROOT-Sim será necesario definir algunas estructuras adicionales que permitan llevar a cabo todo el proceso de simulación.

En la figura 33 se presenta el diagrama de estructuras de datos de Yades una vez redefinidas para ser usadas con la librería ROOT-Sim.

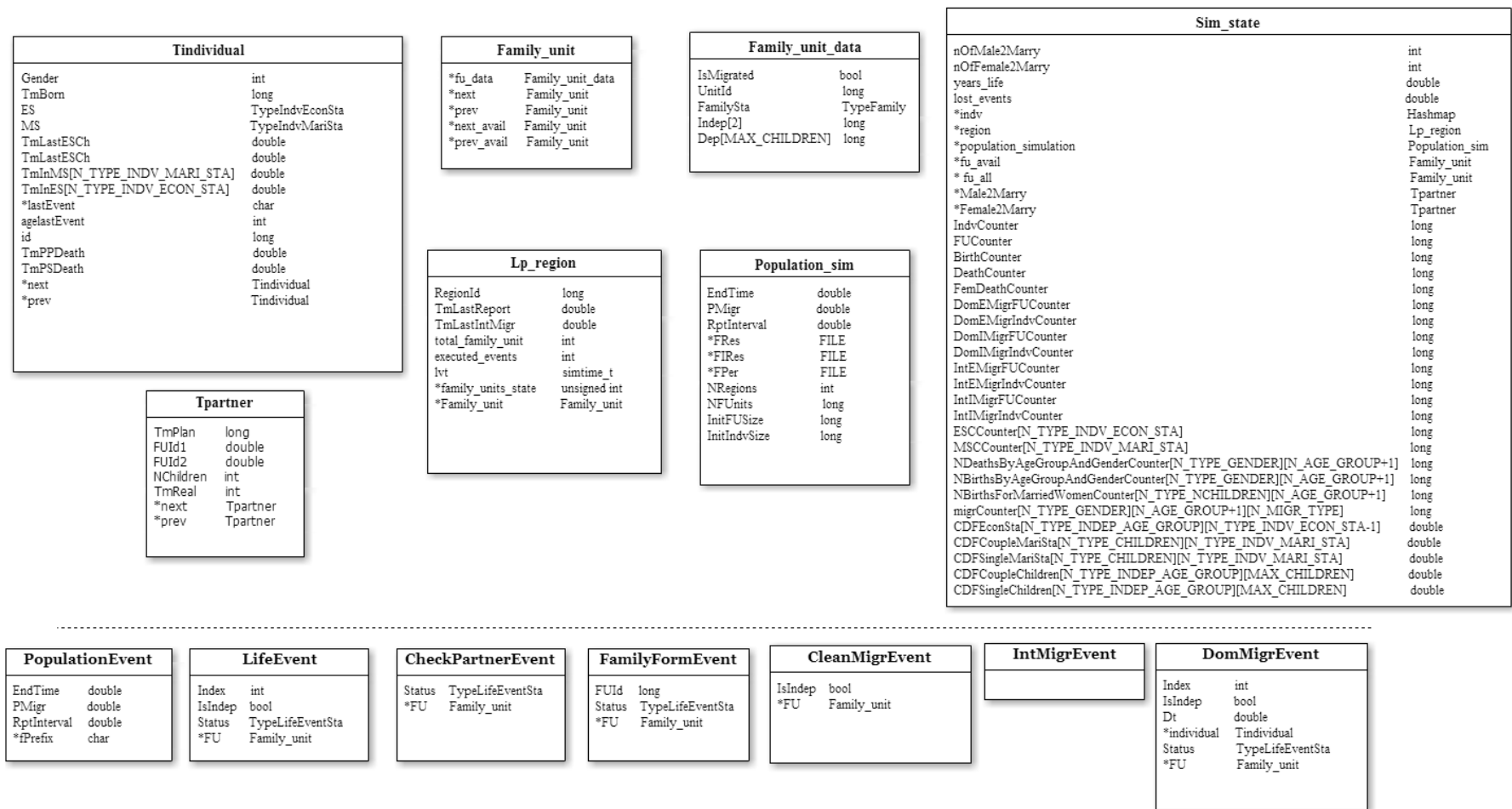


Figura 33: Diagrama de estructuras de datos de Yades con ROOT-Sim.

En la imagen se muestra, por encima de la línea punteada, todas las estructuras de datos necesarias para almacenar los datos de la simulación y representar los conceptos del dominio del proyecto.

Por debajo de la línea punteada, se observan los diferentes eventos que tienen lugar durante la ejecución de la simulación y que representan un cambio en el estado de cada unidad familiar.

Se han cambiado todas las variables globales que empleaba Yades, sustituyéndolas por una estructura que engloba todo el estado de simulación para un mismo LP. En la sección de código siguiente se presenta la estructura `Sim_state` definida para poder almacenar el estado de simulación de un LP y poder almacenar sus datos durante todo el proceso de simulación. Esta estructura no solo ha sido creada para adaptarse al modelo de ROOT-Sim sino que además contribuye a aumentar la modularización del código. De esta manera se asegura que los LP no están accediendo a estructuras de datos que no les corresponden y no están alterando el proceso de simulación.

```
typedef struct _sim_state {
    int nOfMale2Marry;
    int nOfFemale2Marry;
    int years_life;

    Hashmap *indv;           //Hashmap of all individual in the region
    Lp_region *region;
    Population_sim *population_simulation;
    Family_unit *fu_avail;  //list of family unit available
    Family_unit * fu_all;   //list of all family unit
    Tpartner *Male2Marry;  // list of potential husbands
    Tpartner *Female2Marry; // list of potential wives
    //-----global statistics (per region) -----
    long IndvCounter; // number of created individuals
    long FUCounter;   // number of created FU
    long BirthCounter; // # births
    long DeathCounter; // # deaths
    long FemDeathCounter; // # female deaths (for F.R.)
    long DomEMigrFUCounter; // # domestic emmig. (in FU)
    long DomEMigrIndvCounter; // # domestic emmig. (in persons)
    long DomIMigrFUCounter; // # domestic immig. (in FU)
    long DomIMigrIndvCounter; // # domestic immig. (in persons)
    long IntEMigrFUCounter; // # international emmig. (in FU)
    long IntEMigrIndvCounter; // # international emmig. (in persons)
    long IntIMigrFUCounter; // # international immig. (in FU)
    long IntIMigrIndvCounter; // # international immig. (in persons)
    long ESCCounter[N_TYPE_INDV_ECON_STA]; // # economic statuses
    long MSCCounter[N_TYPE_INDV_MARI_STA]; // # marital statuses
    long NDeathsByAgeGroupAndGenderCounter[N_TYPE_GENDER][N_AGE_GROUP
        +1]; // # deaths in MALE(0) & FEMALE(1)
    long NBirthsByAgeGroupAndGenderCounter[N_TYPE_GENDER][N_AGE_GROUP
        +1]; // # individuals in MALE(0) & FEMALE(1)
    long NBirthsForMarriedWomenCounter[N_TYPE_NCHILDREN][N_AGE_GROUP
```

```

+1]; ///  
long migrCounter[N_TYPE_GENDER][N_AGE_GROUP+1][N_MIGR_TYPE];  
///  
// -----cummulative distribution functions  
double CDFEconSta[N_TYPE_INDEP_AGE_GROUP][N_TYPE_INDV_ECON_STA-1];  
double CDFCoupleMariSta[N_TYPE_CHILDREN][N_TYPE_INDV_MARI_STA];  
double CDFSinglMariSta[N_TYPE_CHILDREN][N_TYPE_INDV_MARI_STA];  
double CDFCoupleChildren[N_TYPE_INDEP_AGE_GROUP][MAX_CHILDREN];  
double CDFSinglChildren[N_TYPE_INDEP_AGE_GROUP][MAX_CHILDREN];  
} Sim_state;

```

Debido a que C no es orientado a objetos, se ha decidido implementar las clases Región, Unidad Familiar y Simulador de Población como estructuras de datos (struct).

Como en C tampoco se cuenta con contenedores de datos como HashMaps o listas, se ha decidido implementar las estructuras más importantes de simulación con unas implementaciones propias de lista y HashMap. La implementación del HashMap en C se ha hecho a partir de una implementación inicial opensource disponible a través del proyecto alojado en github: https://github.com/petewarden/c_hashmap/blob/master/hashmap.h.

En la figura 34 se presenta un diagrama de cómo se han definido las listas encadenadas dobles de Yades una vez ha sido integrado con la librería ROOT-Sim.

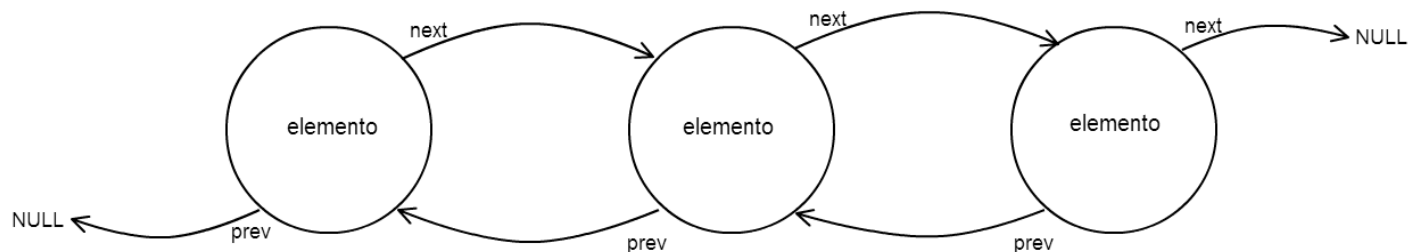


Figura 34: Lista doblemente encadenada.

La lista de unidades familiares (`Family_unit *fu`) es una lista encadenada doble con doble apuntador. Se decidió que, en lugar de tener dos listas de individuos (el total de individuos que hay en el sistema y los individuos que están libres para ser usados) con la información replicada y consumiendo más recursos innecesariamente, tener en la misma lista apuntadores distintos para recorrer ambas listas desde los mismos elementos de la lista. Por esta razón, en la figura 35 se observa una lista encadenada doble con doble apuntador `fu->next` y `fu->next_avail`. El primero, apunta al siguiente elemento (unidad familiar) de la lista y el segundo apunta al siguiente elemento que se encuentra disponible para ser usado.

Para el caso de los individuos se ha decidido emplear una implementación propia de estructura HashMap que permite realizar búsquedas rápidas y eficientes por toda la estructura. Esta estructura presenta una clara ventaja frente a la lista doblemente encadenada, en el caso que no

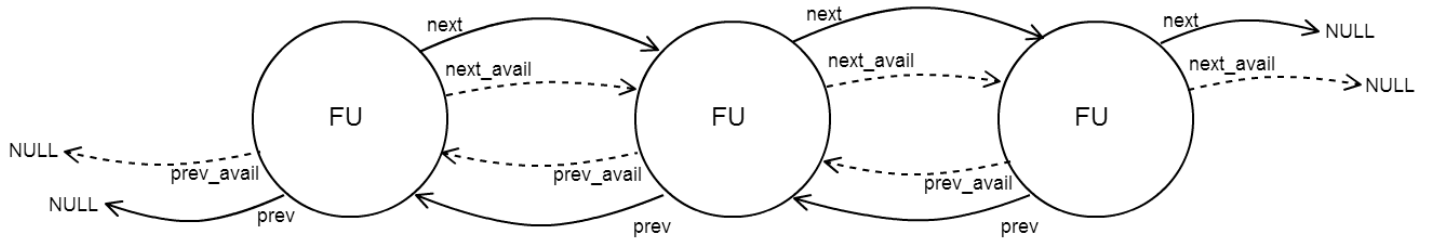


Figura 35: Lista de unidades familiares.

se quiera recorrer toda la lista de individuos sino que se quiera buscar un individuo específico que cumpla con determinadas características.

Para las estructuras que representan las personas que buscan pareja en el sistema también se han empleado listas encadenadas.

9.3. Redefinición de métodos

En la figura 11 de la sección 6.1.2 se mostró el diagrama de actividades de Yades con la librería μ sik. A continuación, en la figura 36 se presenta el diagrama de actividades de Yades una vez ha sido integrado con la librería ROOT-Sim.

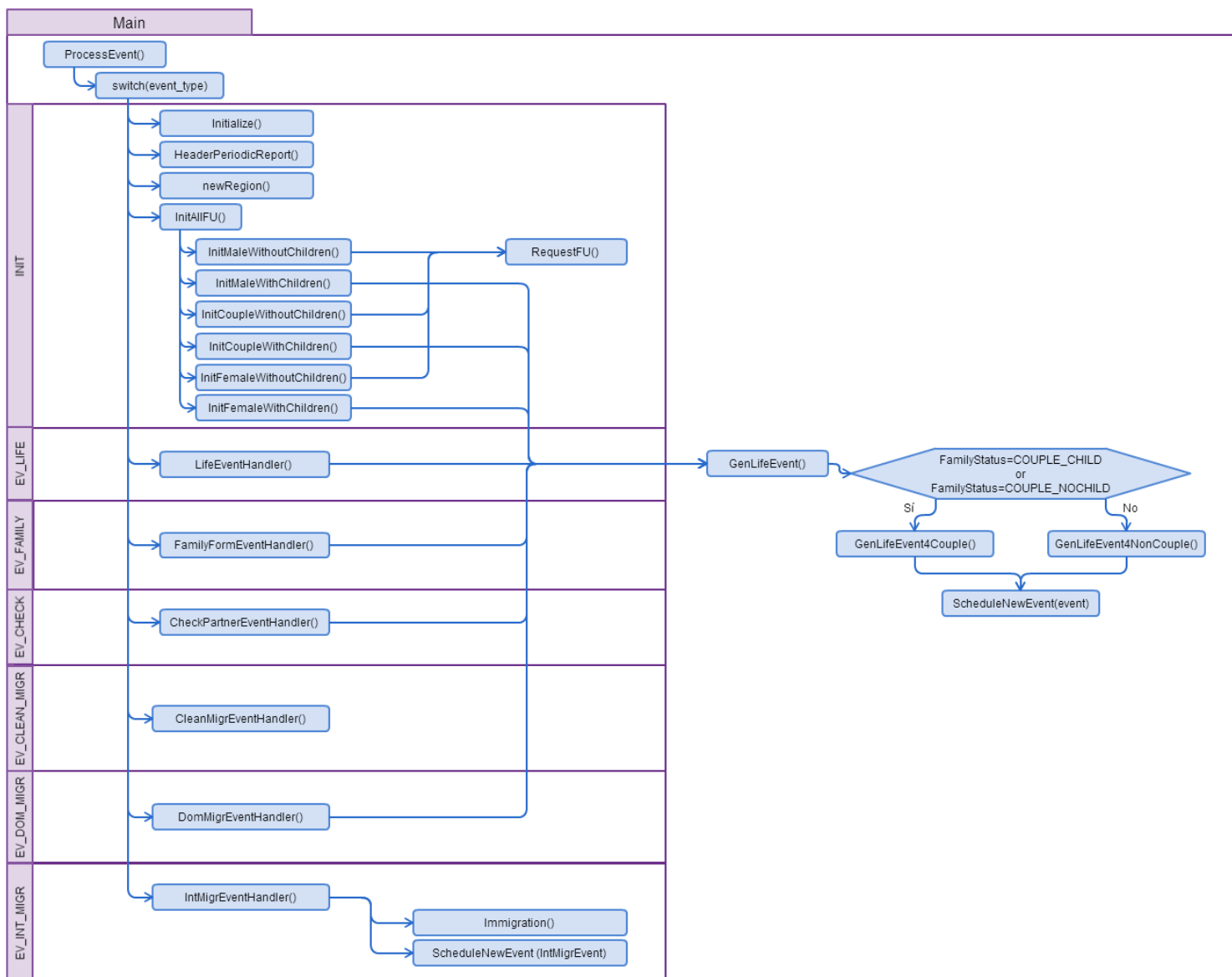


Figura 36: Diagrama de actividad del flujo de vida de Yades con ROOT-Sim.

Los LP representan las regiones en las que viven las familias. En el modelo que se ha seguido en este trabajo, todas las regiones contienen las mismas características por lo que todas están formadas por las mismas estructuras de datos y contienen los mismos atributos. Las regiones están representadas por una lista doblemente enlazada de punteros.

Los LP se comunican entre ellas mediante eventos, en el modelo que se ha seguido en este trabajo en el caso que una familia emigre de una región a otra, en la región origen se tiene una unidad familiar representada por una estructura FU', se envía un evento a esa familia anunciándole que debe migrar, se realiza una copia de la estructura y se envía un evento a la región destino con los datos de esa familia. También se envía un evento a esta misma región con el mismo tiempo que el evento de migración, anunciando que esta familia debe ser eliminada de la lista de familias actuales en la región origen. En la región destino se crea una familia nueva con las características de la familia inmigrante y se planifica un nuevo evento para los miembros de esta familia. De esta manera, las unidades familiares ya no se migran enteras, como en el caso de la versión de Yades con μ sik, sino que su contenido se envía a otro LP. Este cambio aumenta el rendimiento de la aplicación porque ya no es necesario enviar todo el LP, con todos sus datos y procesos internos, sino que se envían y reciben solo los datos que representan las familias.

Las constantes se han modificado por `#define` y se han redefinido los identificadores de todos los eventos para que no pueda haber ninguna confusión con los eventos del `ProcesEvent(...)`.

Debido a las características técnicas de la versión de ROOT-Sim que estamos usando, para controlar el uso de memoria solo se pueden usar las instrucciones: `malloc()`, `calloc()`, `realloc()` y `free()`. Las instrucciones `new()` y `delete()` todavía no están implementadas en esta versión.

La inicialización del estado de cada región se define en la función `void ProcessEvent(...)`, en el evento INIT que implementa el comportamiento de inicialización del Yades original en su función `PopulationSimulator::init()`.

Posteriormente en esta misma función se definen los demás eventos que influyen en la evolución de la población y su simulación: `EV_LIFE`, `EV_FAMILY`, `EV_CHECK`, `EV_CLEAN_MIGR`, `EV_DOM_MIGR` y `EV_INT_MIGR`.

En las tablas que se presentan a continuación se muestran, para cada iteración de desarrollo, las épicas e historias de usuario que se han realizado junto con su contenido.

9.4. Correspondencia diseño-implementación

| Iteración: 3 | | | |
|--|---|--|---|
| Épica: 1 | | | |
| H1.1 | Sim_state PopulationEvent Family_unit Family_unit_data Population_sim Tindividual | | |
| H1.2 | void Reset(...) void InitGlobalVariables(...) void newPopulationSim(...) void endPopulationSim(...) void newFamilyUnitByID(...) void newFamilyUnit(...) void newRegion(...) | | |
| H1.3 | Scripts en perl y bash para cargar datos iniciales | | |
| H1.4 | <table style="width: 100%; border: none;"> <tr> <td style="width: 50%; border: none;"> int InitNChildren4Couple(...) int InitNChildren4Single(...) void InitCoupleWithChildren(...) void InitCoupleWithoutChildren(...) void InitMaleWithChildren(...) void InitFemaleWithChildren(...) void InitMaleWithoutChildren(...) void InitFemaleWithoutChildren(...) void InitFamilyUnits(...) </td> <td style="width: 50%; border: none;"> void SetFamilyStatus(...) void SetIndep(...) void SetDep(...) void SetIndividuals(...) void GetAgeGroup(...) double InitAge(...) inline double IndependentAge(...) void newIndividual(...) </td> </tr> </table> | int InitNChildren4Couple(...) int InitNChildren4Single(...) void InitCoupleWithChildren(...) void InitCoupleWithoutChildren(...) void InitMaleWithChildren(...) void InitFemaleWithChildren(...) void InitMaleWithoutChildren(...) void InitFemaleWithoutChildren(...) void InitFamilyUnits(...) | void SetFamilyStatus(...) void SetIndep(...) void SetDep(...) void SetIndividuals(...) void GetAgeGroup(...) double InitAge(...) inline double IndependentAge(...) void newIndividual(...) |
| int InitNChildren4Couple(...) int InitNChildren4Single(...) void InitCoupleWithChildren(...) void InitCoupleWithoutChildren(...) void InitMaleWithChildren(...) void InitFemaleWithChildren(...) void InitMaleWithoutChildren(...) void InitFemaleWithoutChildren(...) void InitFamilyUnits(...) | void SetFamilyStatus(...) void SetIndep(...) void SetDep(...) void SetIndividuals(...) void GetAgeGroup(...) double InitAge(...) inline double IndependentAge(...) void newIndividual(...) | | |
| H1.5 | void HeaderIndividualReport(...) extern void HeaderPeriodicReport(...) void IndividualReport(...) extern void HeaderPeriodicReport(...) void PeriodicReport(...) Scripts en perl y bash para procesar datos, y hojas de cálculo para crear gráficas | | |
| H1.6 | int ProcessEvent(...) void GenLifeEvents(...) void GenLifeEvents4Couple(...) void GenLifeEvents4NonCouple(...) bool OnGVT(...) | | |

Tabla 56: Tabla con correspondencia de implementación-diseño 1

| Iteración: 3 | |
|---------------------|--|
| Épica: 2 | |
| H2.1 | double Fertility(...) |
| H2.2 | bool Birth(...) |
| H2.3 | void GenLifeEvents(...) void GenLifeEvents4Couple(...) void GenLifeEvents4NonCouple(...) LifeEvent |
| H2.4 | void LifeEventHandler(...) Family_unit *getFUByIndex(...) Family_unit *RequestFU(...) |
| H2.5 | void HeaderIndividualReport(...) extern void HeaderPeriodicReport(...) void IndividualReport(...) extern void HeaderPeriodicReport(...) void PeriodicReport(...) Scripts en perl y bash para procesar datos, y hojas de cálculo para crear gráficas |

Tabla 57: Tabla con correspondencia de implementación-diseño 2

| Iteración: 3 | |
|---------------------|--|
| Épica: 3 | |
| H3.1 | double Mortality(...) |
| H3.2 | double Mortality(...) |
| H3.3 | void GenLifeEvents(...) void GenLifeEvents4Couple(...) void GenLifeEvents4NonCouple(...) LifeEvent |
| H3.4 | void LifeEventHandler(...) Family_unit *getFUByIndex(...) Family_unit *RequestFU(...) |
| H3.5 | void countDeathByAgeGroup(...) void HeaderIndividualReport(...) extern void HeaderPeriodicReport(...) void IndividualReport(...) extern void HeaderPeriodicReport(...) void PeriodicReport(...) Scripts en perl y bash para procesar datos, y hojas de cálculo para crear gráficas |

Tabla 58: Tabla con correspondencia de implementación-diseño 3

| Iteración: 4 | |
|---------------------|--|
| Épica: 4 | |
| H4.1 | void NewEconSta(...) TypeIndvEconSta GenIndepEconSta(...) |
| H4.2 | void NewES(...) |
| H4.3 | void GenLifeEvents(...) void GenLifeEvents4Couple(...) void GenLifeEvents4NonCouple(...) LifeEvent |
| H4.4 | void LifeEventHandler(...) Family_unit *getFUByIndex(...) Family_unit *RequestFU(...) |
| H4.5 | void HeaderIndividualReport(...) extern void HeaderPeriodicReport(...) void IndividualReport(...) extern void HeaderPeriodicReport(...) void PeriodicReport(...) Scripts en perl y bash para procesar datos, y hojas de cálculo para crear gráficas |

Tabla 59: Tabla con correspondencia de implementación-diseño 4

| Iteración: 4 | |
|--------------|--|
| Épica: 5 | |
| H5.1 | long SeekPartner(...) void orderedPartnerInsert(...) TypeIndvMariSta InitCoupleMS(...) TypeIndvMariSta InitSingleMS(...) void NewMariSta(...) Tpartner |
| H5.2 | void NewMS(...) |
| H5.3 | void GenLifeEvents(...) void GenLifeEvents4Couple(...) void GenLifeEvents4NonCouple(...) bool MatchMaker(...) FamilyFormEvent CheckPartnerEvent |
| H5.4 | void CheckPartnerEventHandler(...) void FamilyFormEventHandler(...) void Join(...) Family_unit *getFUByIndex(...) Family_unit *RequestFU(...) void deleteAPartner(...) void AddFUToFUAvail(...) |
| H5.5 | void HeaderIndividualReport(...) extern void HeaderPeriodicReport(...) void IndividualReport(...) extern void HeaderPeriodicReport(...) void PeriodicReport(...) Scripts en perl y bash para procesar datos, y hojas de cálculo para crear gráficas |

Tabla 60: Tabla con correspondencia de implementación-diseño 5

| Iteración: 4 | |
|--------------|--|
| Épica: 6 | |
| H5.1 | bool IsMigrate(...) |
| H6.2 | void NewES(...) |
| H5.3 | void GenLifeEvents(...) void GenLifeEvents4Couple(...) void GenLifeEvents4NonCouple(...) void ErrorNegativeDT(...) void RequestFUEventHandler(...) |
| H6.4 | void copyFUFromAnotherFU(...) void DomMigrEventHandler(...) Family_unit *getFUByIndex(...) Family_unit *RequestFU(...) void deleteAPartner(...) void AddFUToFUAvail(...) ClearEvent DomMigrEvent |
| H6.5 | void HeaderIndividualReport(...) extern void HeaderPeriodicReport(...) void IndividualReport(...) extern void HeaderPeriodicReport(...) void PeriodicReport(...) Scripts en perl y bash para procesar datos, y hojas de cálculo para crear gráficas |
| H6.6 | void IntMigrEventHandler(...) void Immigration(...) IntMigrEvent |
| H6.7 | void HeaderIndividualReport(...) extern void HeaderPeriodicReport(...) void IndividualReport(...) extern void HeaderPeriodicReport(...) void PeriodicReport(...) Scripts en perl y bash para procesar datos, y hojas de cálculo para crear gráficas |

Tabla 61: Tabla con correspondencia de implementación-diseño 6

10. Diseño de experimentos

El objetivo de un estudio de simulación es encontrar respuestas a preguntas sobre el sistema y poder realizar una observación y estudio detallado del mismo. Las respuestas son obtenidas a través de la información que proporcionan las diversas réplicas de experimentos con el modelo del sistema, elaborando preguntas como “¿qué pasaría con la variable respuesta si modifico una variable experimental?”.

Las respuestas obtenidas deben servir de soporte a tomas de decisiones racionales con el conocimiento de cómo afectan las variaciones de unos determinados factores sobre el sistema. Por esta razón, es importante que estas respuestas sean expresadas numéricamente para todos los escenarios posibles.

En este caso de estudio, la variable respuesta a estudiar es el tiempo de ejecución de la simulación, y las variables experimentales o factores son:

- Número de unidades familiares por región de simulación.
- Probabilidad de migración
- Número de regiones de simulación.

Para poder desarrollar un buen análisis de experimentos es necesario seguir las siguientes directrices [Fonseca i Casas, 2013-2014a]:

- Todos los factores que no pueden ser controlados por el experimentador deben tener un valor aleatorio.
- Disminuir la variabilidad entre las respuestas obtenidas a través de la repetición del experimento (diferentes réplicas). Esto implica calcular el número de replicas necesarias para obtener este resultado a través de los pasos:

1. Para cada una de las variables de interés, el primer paso es calcular la *media muestral* μ_1 de todas las réplicas, a partir de la siguiente fórmula:

$$\bar{X} = \frac{\sum_{i=1}^n x_i}{n} \quad (6)$$

Dónde:

- x_i : valor de cada una de las réplicas de la variable de interés.
- n : número de réplicas.

A priori, estos experimentos se realizan asignando un número de réplicas aleatorio. El número de réplicas inicial se ha definido como $n = 10$.

2. Calcular la *varianza muestral* σ_2 para cada variable de interés, para conocer cuánto se están distanciando los elementos, a través de la fórmula:

$$S^2 = \frac{\sum_{i=1}^n (x_i - \bar{X})^2}{n - 1} \quad (7)$$

3. Se calcula el *intervalo de confianza* para saber cuán aproximado está μ de \bar{X} y se utiliza la T-Student de $n-1$ grados de libertad y un 95% de región de aceptación, para saber si la media muestral entre las muestras es estadísticamente significativa. Se calcula a partir de la fórmula:

$$\bar{X} \pm t_{1-\frac{\alpha}{2}, n-1} \sqrt{\frac{S^2}{n}} \quad (8)$$

Dónde:

- $\sqrt{\frac{S^2}{n}}$: Corresponde al error estándar.
 - $\frac{S}{\sqrt{n}}$: Corresponde a la desviación estándar.
4. Se calcula el *semi intervalo de confianza*:

$$h = t_{1-\frac{\alpha}{2}, n-1} \frac{S}{\sqrt{n}} \quad (9)$$

5. Se evalúa si el valor real de la media μ está contenido en el intervalo definido o no:

$$\mu \in (\bar{X} - h, \bar{X} + h) \quad (10)$$

6. Se evalúa si el valor de la media muestral calculada, \bar{X} , con la anchura del intervalo de confianza definido, α' , es $\leq h$. Si esta condición no se cumple, es necesario realizar más réplicas del experimento para estar dentro del intervalo de confianza deseado.

$$\bar{X} * \alpha' \leq h \quad (11)$$

Este proceso se realiza mediante el cálculo del número de réplicas necesarias, (n^*). Es iterativo y también se puede obtener a través de la siguiente fórmula:

$$n^* = n \left(\frac{h}{h^*} \right)^2 \quad (12)$$

Dónde:

- n^* : número total de réplicas necesarias.
- n : número de réplicas de la prueba piloto.
- h : semi intervalo de confianza de la prueba piloto.
- h^* : semi intervalo de confianza para todas las réplicas (semi intervalos deseados).

- Realizar todos los experimentos bajo condiciones homogéneas para comparar diferentes alternativas derivadas de los resultados.

Este proceso se realiza para cada una de las variables de interés para poder definir el número de réplicas necesarias para realizar los experimentos.

La tabla 62 muestra el diseño de experimentos que se ha definido para cada una de las variables de interés definidas en el apartado anterior y que se usarán para el apartado 12.

| Nº FU | P. migr | Nº regiones |
|-------|---------|-------------|
| - | - | - |
| - | - | + |
| - | + | - |
| - | + | + |
| + | - | - |
| + | - | + |
| + | + | - |
| + | + | + |

Tabla 62: Diseño de experimentos para el análisis de las versiones de Yades con μ sik y con ROOT-Sim

Para estas pruebas, el signo - representa que el componente está en su estado inicial que es activado y el signo + representa que el componente se encuentra desactivado. El estado inicial de cada una es:

- Nº regiones = 4
- Nº FU = 2642, 3895, 1470 y 1032 respectivamente para cada región.
- P.migr = 0.1

Para ambas versiones de Yades, con μ sik y con ROOT-Sim, se ha considerado el mismo estado inicial.

En la tabla 63 se muestra el diseño de experimentos que se ha definido para las pruebas de validación que se realizarán en el apartado 11.

Después de realizar los cálculos, se obtiene que el número de réplicas mínimo para los experimentos que se desean realizar es 10. Por lo tanto se realizarán 10 réplicas de cada experimento para obtener valores representativos. En la sección Anexos se puede consultar el procedimiento seguido para la variable de interés “tiempo de ejecución” y la TStudent empleada.

| Mortalidad | Fertilidad | ES | MS | Migr. |
|------------|------------|----|----|-------|
| - | - | - | - | - |
| - | - | - | - | + |
| - | - | - | + | - |
| - | - | - | + | + |
| - | - | + | - | - |
| - | - | + | - | + |
| - | - | + | + | - |
| - | - | + | + | + |
| - | + | - | - | - |
| - | + | - | - | + |
| - | + | - | + | - |
| - | + | - | + | + |
| - | + | + | - | - |
| - | + | + | - | + |
| - | + | + | + | - |
| - | + | + | + | + |
| + | - | - | - | - |
| + | - | - | - | + |
| + | - | - | + | - |
| + | - | - | + | + |
| + | - | + | - | - |
| + | - | + | - | + |
| + | - | + | + | - |
| + | - | + | + | + |
| + | + | - | - | - |
| + | + | - | - | + |
| + | + | - | + | - |
| + | + | - | + | + |
| + | + | + | - | - |
| + | + | + | - | + |
| + | + | + | + | - |
| + | + | + | + | + |

Tabla 63: Diseño de experimentos de validación

11. Validación

Uno de los problemas más importantes en el área de la simulación es demostrar que un modelo es una representación fidedigna del sistema que se pretende estudiar [Kelton and Law, 2000], es decir, la demostración de si un modelo es *válido*.

Antes de definir las técnicas de validación que se han empleado en este trabajo, se introducen algunos conceptos útiles para este capítulo.

La *Verificación* de un sistema es el proceso mediante el cual se determina si el modelo conceptual del sistema que se pretende definir ha sido correctamente definido, es decir, si ha sido correctamente traducido al sistema software [Kelton and Law, 2000]. Este proceso se suele realizar mediante la depuración (debug) del sistema software.

La *Validación* de un sistema es el proceso mediante el cual se determina si el modelo de simulación es una correcta representación del sistema a modelar, para el caso de estudio específico [Kelton and Law, 2000]. Si un modelo de simulación es válido, puede ser usado para tomar determinadas decisiones sobre el sistema real. Sin embargo, la facilidad o dificultad de este proceso dependerá de la dificultad propia del modelo a representar, para este caso de estudio, del sistema social a representar.

La *Acreditación* de un sistema es el proceso mediante el cual se certifica de manera oficial [Kelton and Law, 2000] que un modelo de simulación es aceptable para un propósito específico.

En la figura 37 se puede observar como está ligado el proceso de desarrollo de un software con el proceso de verificación, validación y acreditación [Fonseca i Casas, 2013-2014b].

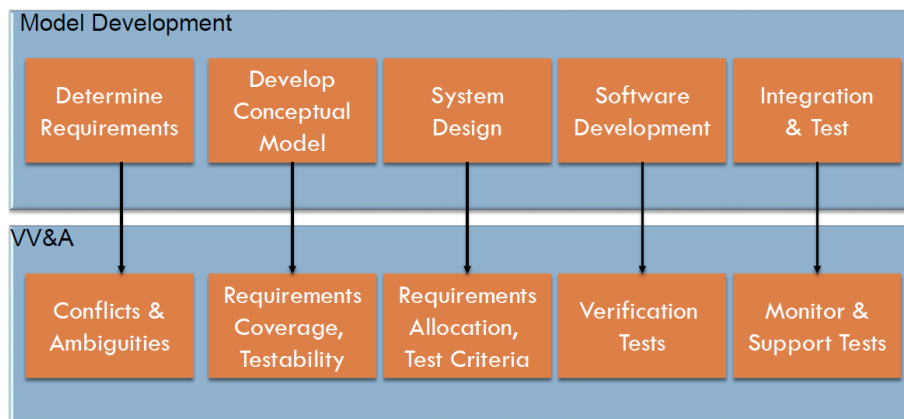


Figura 37: Correspondencias del proceso de desarrollo del software con la verificación, validación y acreditación del mismo [Fonseca i Casas, 2013-2014b].

Existen muchas técnicas que permiten realizar una *verificación* del sistema. En este trabajo se parte de un modelo conceptual de movimientos demográficos que ya ha sido verificado en

[Montañola Sales et al., 2015].

Sin embargo, con el objetivo de que el producto resultante de este sistema sea fiable, se realiza una verificación a través de una de las técnicas descritas en [Kelton and Law, 2000] que consiste en ejecutar el modelo con una variedad de parámetros de entrada (cantidad de población inicial por regiones, número de regiones, diferentes probabilidades de migración, diferentes probabilidades de fertilidad y mortalidad, entre otros) y comprobar si la salida del programa es correcta.

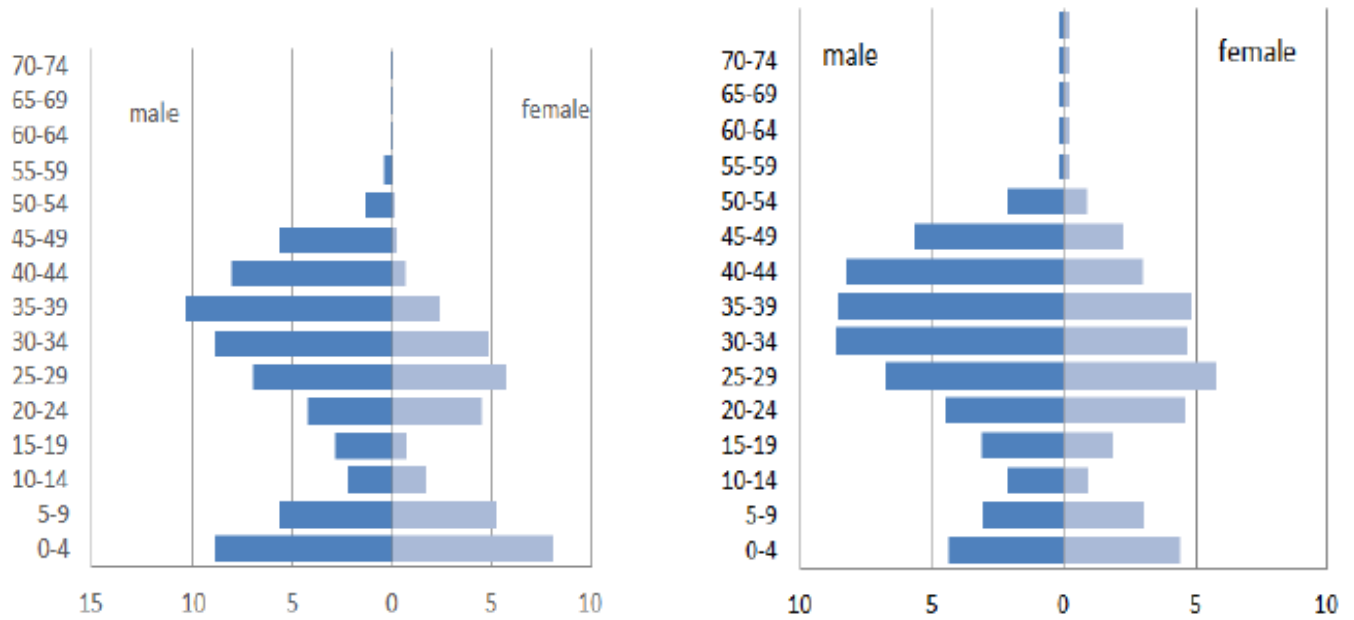
En la figura 38 se pueden observar las gráficas que corresponden a la inicialización de la población en Yades, en la gráfica 38a se observan los datos iniciales de los cuales parte el sistema, en la gráfica 38b se muestran los datos obtenidos a partir de ejecutar la simulación con los mismos datos en el simulador Yades con su implementación original con μsik y en la gráfica 38c se muestra la salida del simulador Yades con su implementación con ROOT-Sim. Como se puede apreciar, la población mantiene las mismas tendencias en ambos casos.

Otra de las técnicas empleadas en este trabajo para realizar la verificación ha sido el análisis de trazas y el uso de un depurador de código para analizar, paso a paso, la ejecución del simulador.

Para realizar la validación del sistema software se ha empleado una *validación por componentes* con la que se ha realizado una *validación de los resultados*. En estas pruebas se han probado cada uno de los componentes de simulación y se han contrastado con datos del sistema real y con el simulador Yades, ya validado en [Montañola-Sales, 2014], haciendo una *comparación con un sistema ya existente*.

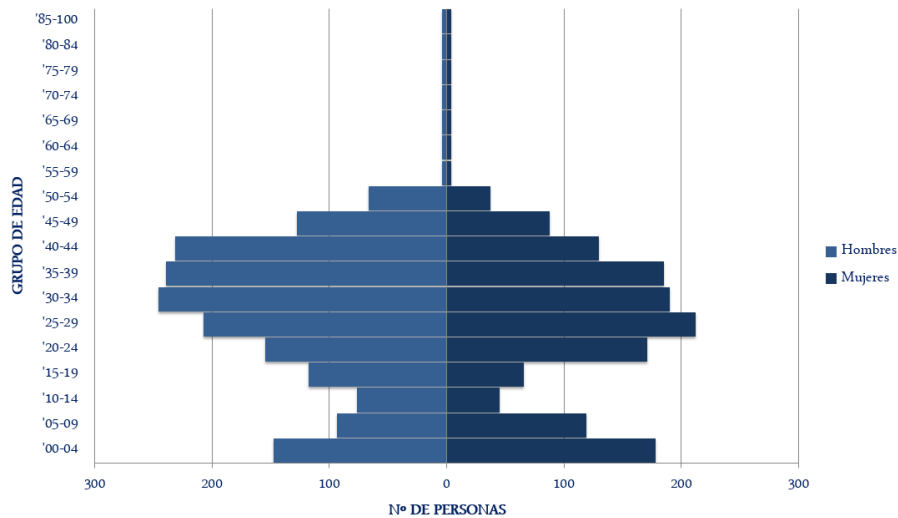
11.1. Validación por componentes

En la *validación por componentes* de la solución software se han aislado cada uno de los 5 componentes del modelo para ejecutarlos individualmente y ver como se comporta el simulador en cada uno de los casos. Además se ha probado como se comportan los componentes al interactuar entre ellos con diferentes configuraciones. Los resultados de este estudio se han contrastado con los resultados obtenidos en [Montañola-Sales, 2014] para el simulador Yades original y con datos del sistema real, las figuras 39 y 40 son un extracto de este análisis, representan el componente de mortalidad y el de estados maritales para Yades con μsik .



(a) Probabilidades para la población inicial, datos iniciales [Montañola-Sales, 2014].

(b) Validación por caja blanca: inicialización de la población Yades- μ sik [Montañola-Sales, 2014].



(c) Validación por caja blanca: inicialización de la población Yades-ROOT-Sim.

Figura 38: Inicialización de la población en Yades.

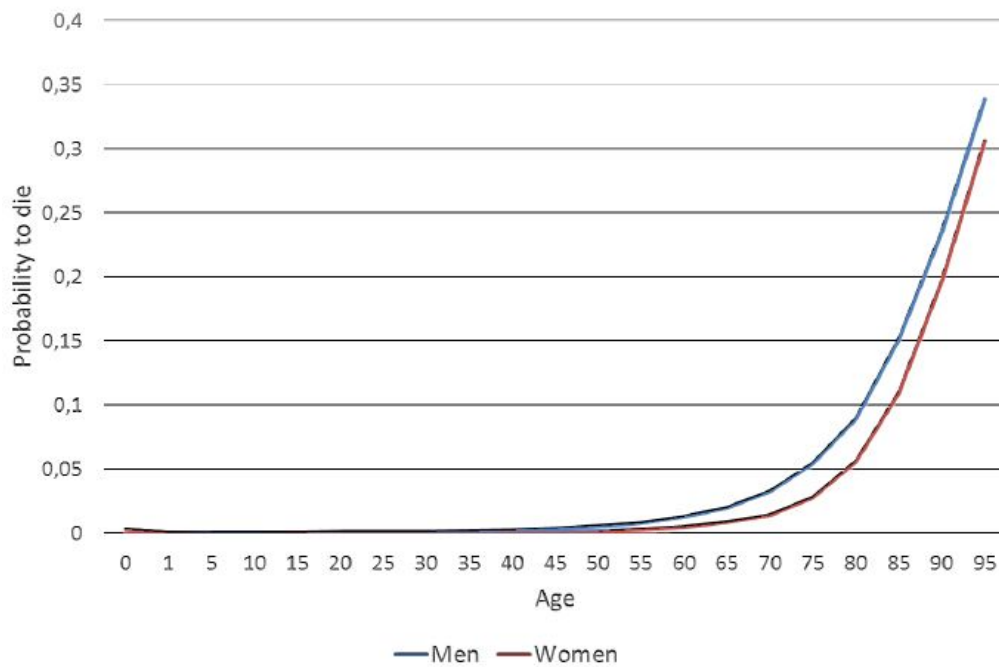


Figura 39: Validación por caja blanca: mortalidad [Montañola-Sales, 2014].

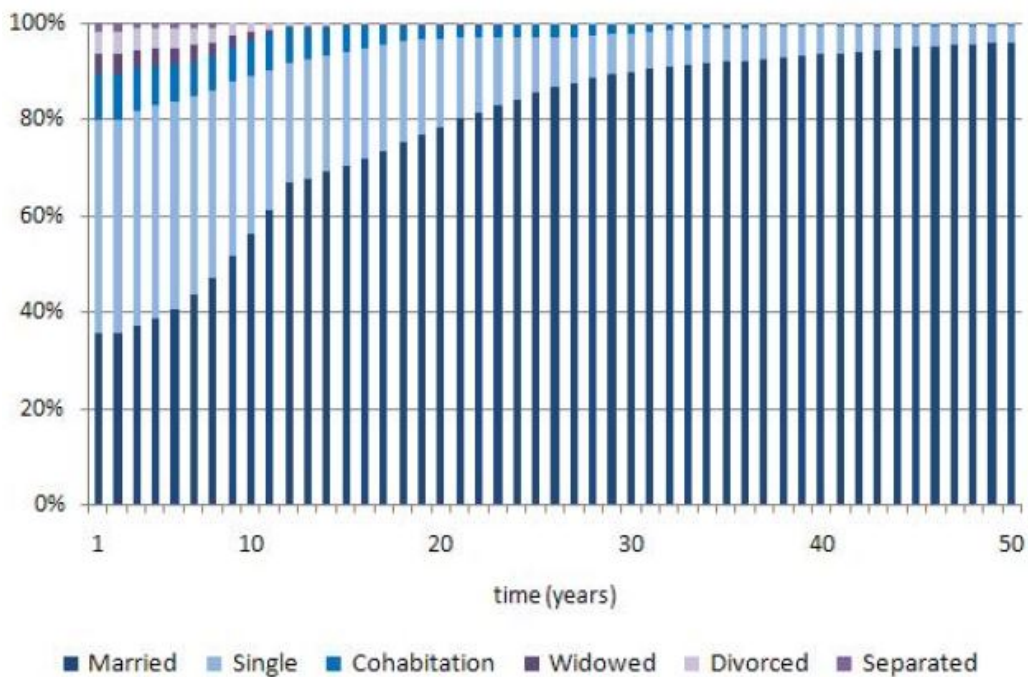


Figura 40: Validación por caja blanca: estados maritales [Montañola-Sales, 2014].

A continuación se presenta cada una de las pruebas realizadas. En la sección 10 se presentó la tabla 63 en la que se muestra el diseño de experimentos realizado para esta validación.

Todas estas pruebas han sido realizadas tanto en los equipos locales de desarrollo como en Marenostrium 3.

En Marenostrium 3 ha sido probado en un nodo de computación, empleando 4 cores. Se han simulado 4 regiones físicas (4 LP), 50 años de simulación, con un intervalo de reporte de 1 año y se ha empleado una población inicial (número máximo de unidades familiares para cada una de las regiones de simulación) de: 2642, 3895, 1470 y 1032 respectivamente para cada región. Estos datos iniciales han sido extraídos de [Montañola-Sales, 2014].

11.1.1. Mortalidad

En esta prueba de validación se aísla el componente de mortalidad del resto de componentes del modelo y se prueba su efecto en la población.

En la figura 41 se puede observar la gráfica resultante de realizar esta prueba. En el eje de abscisas se observan los diferentes grupos de edad en los que se ubica la población. En el eje de ordenadas se encuentra el número de muertes.

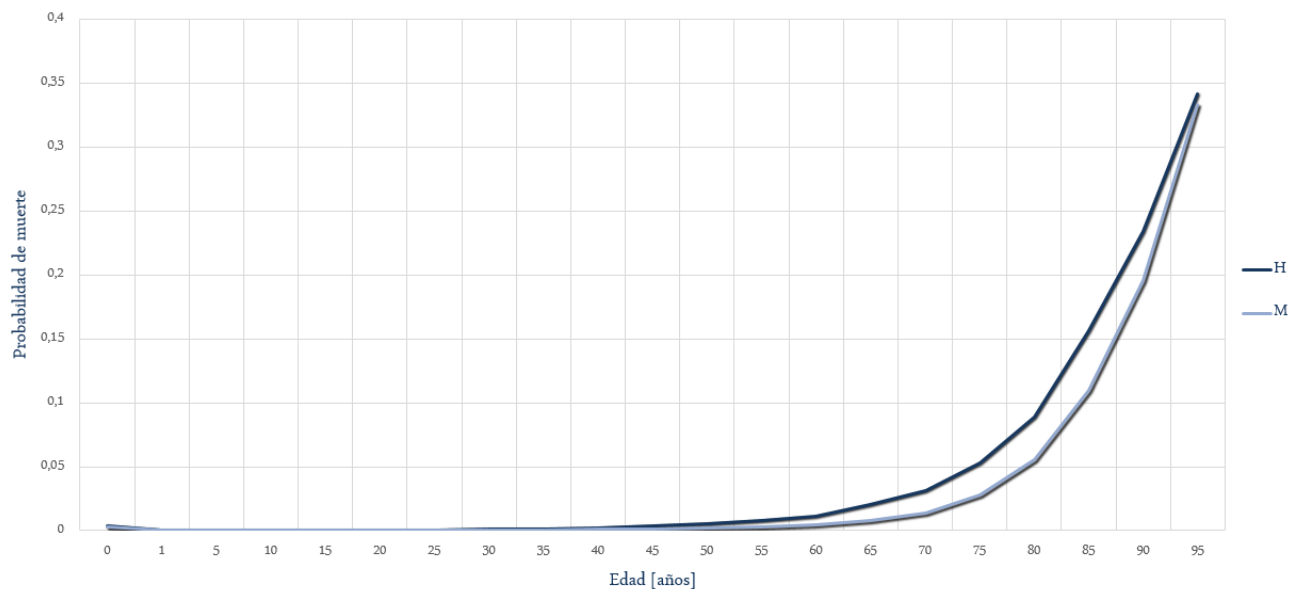


Figura 41: Validación por caja blanca: componente de mortalidad.

Tal como sucede en el modelo original de Yades, al aislar la mortalidad del resto de componentes se observa que la población va evolucionando con el paso del tiempo y en función de la tabla de supervivencia asignada. Como es de esperarse, la tasa de mortalidad es mayor para

los grupos de población de edad más avanzada, a partir de los 70 años. En esta prueba están desactivados los demás componentes demográficos, por lo que la población inicial entra en el sistema y va envejecimiento progresivamente hasta que les llega un evento de muerte y fallecen. También se observa un ligero incremento de la mortalidad en los primeros años de vida de los individuos.

Tal como sucede en la sociedad actual, en la gráfica se observa a partir de los 55 años aproximadamente, la tasa de mortalidad es mayor para los hombres que para las mujeres.

11.1.2. Fertilidad

En esta prueba de validación se aísla el componente de fertilidad del resto de componentes del modelo y se prueba su efecto en la población. En esta gráfica la población afectada es la población femenina, ya que el modelo está definido de manera que solo las mujeres, ya sean solteras o casadas, pueden tener hijos. El número máximo de hijos por persona está fijado a 4.

En la figura 42 se puede observar la gráfica resultante de realizar esta prueba. En el eje de abscisas se observan los diferentes grupos de edad en los que se aplica la fertilidad. En el eje de ordenadas se encuentra el porcentaje de nacimientos.

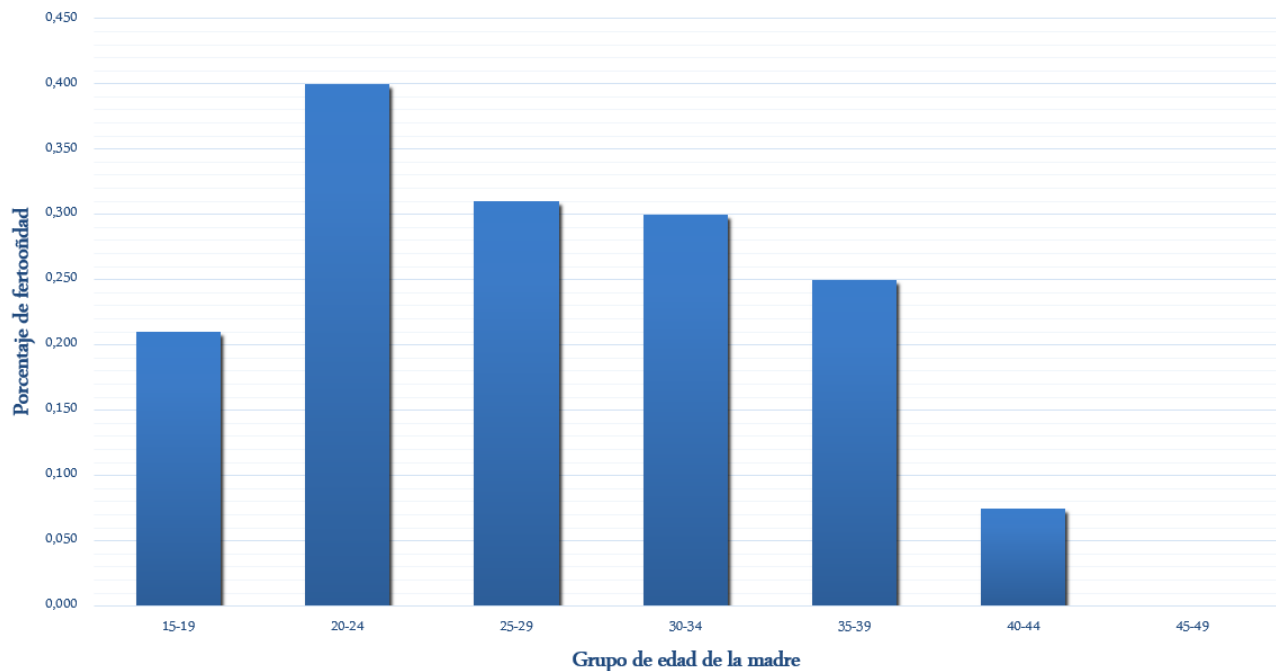


Figura 42: Validación por caja blanca: componente de fertilidad.

Tal como sucede en el modelo original de Yades, al aislar el componentes de la fertilidad del resto de componentes se observa que la población va evolucionando con el paso del tiempo.

po, en función de la tasa de fertilidad asignada. En este experimento, los demás componentes demográficos están desactivados por lo que se observa un envejecimiento progresivo de la población. El modelo gambiano para la fertilidad es diferente al empleado en las sociedades actuales europeas, y las mujeres tienen hijos a edades tempranas. Ésto se puede observar en la gráfica, donde el rango de fertilidad es mayor para el grupo de edad entre los 15 a 19 años, que para el de 40 a 45. La tasa de fertilidad más alta se observa en el grupo de edad comprendido entre los 20-25 años.

11.1.3. Estados económicos

En esta prueba de validación se aísla el componente de estados económicos del resto de componentes del modelo y se prueba su efecto en la población. En esta gráfica se tienen en cuenta tanto el sexo femenino como el masculino, ya que los estados económicos afectan a ambos. En este modelo se definen 5 estados económicos: con empleo, desempleado, retirado, estudiando y dependiente.

En la figura 43 se puede observar la gráfica resultante de realizar esta prueba. En el eje de abscisas se observan los diferentes grupos de edad en los que se aplican los estados económicos. En el eje de ordenadas se encuentra el número de personas en cada estado económico.

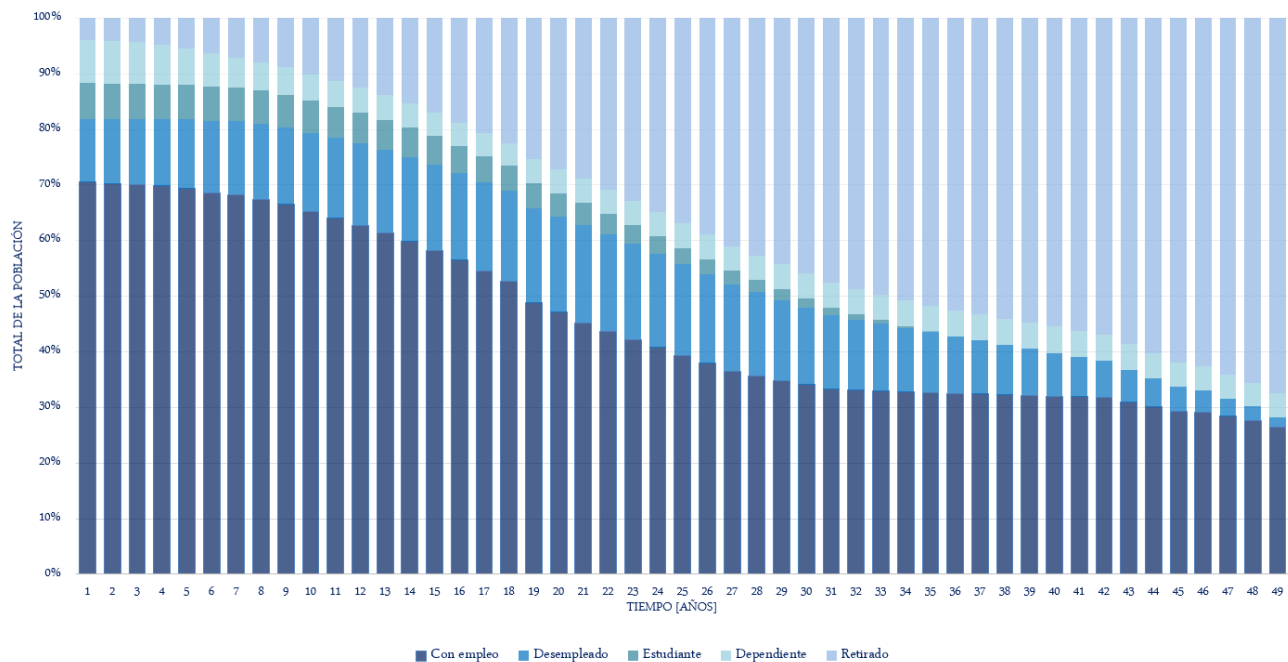


Figura 43: Validación por caja blanca: componente de estados económicos.

Tal como sucede en el modelo original de Yades, al aislar el componente de estados económicos del resto de componentes se observa que la población va evolucionando con el paso del tiempo y

se van moviendo de estados económicos. Estos estados económicos marcan también la posibilidad de migración, sin embargo, para esta prueba se ha puesto la probabilidad de migración a 0, para observar el efecto de los estados económicos en una única región de simulación. Al comienzo de la simulación, la mayor parte de la población se sitúa en estados económicos activos. Sin embargo, a medida que va evolucionando la simulación, se puede observar que la población va cambiando hacia el estado “Retirado”, ya que la población va envejeciendo progresivamente debido a la falta de fertilidad y mortalidad. Esto a su vez, se traduce en una disminución de la población en edad de estudiar y de los desempleados.

11.1.4. Estados maritales

En esta prueba de validación se aísla el componente de estados maritales del resto de componentes del modelo y se prueba su efecto en la población. En esta gráfica se tienen en cuenta tanto el sexo femenino como el masculino.

Los estados maritales definidos para el caso de prueba de Gambia permiten que los emparejamientos de la población puedan ser entre personas del mismo sexo o de sexos diferentes, pero siempre serán relaciones monógamas. En este modelo se definen 5 estados maritales: solteros, casados, viudos, separados y divorciados.

En la figura 44 se puede observar la gráfica resultante de realizar esta prueba. En el eje de abscisas se observan los diferentes grupos de edad en los que se aplican los estados maritales. Estos grupos solo corresponden a la población que está definida como independiente. En el eje de ordenadas se encuentra el porcentaje de personas en cada estado marital para toda la simulación.

Tal como sucede en el modelo original de Yades, al aislar el componentes de estados maritales del resto de componentes, se observa que la población va evolucionando con el paso del tiempo y se van moviendo de estados maritales. Al comienzo de la simulación, la población se sitúa en los estados maritales de “Casado” o “Soltero”. Sin embargo a medida que va avanzando el tiempo de simulación, la población va evolucionando y los solteros van encontrando pareja. Como el componente de mortalidad se encuentra desactivado, se observa que la población en estado de viudedad va disminuyendo progresivamente.

11.1.5. Migraciones

En esta prueba de validación se aísla el componente de migración del resto de componentes del modelo y se prueba su efecto en la población. En esta gráfica se tienen en cuenta tanto el sexo femenino como el masculino.

En el modelo están definidas tanto las migraciones internacionales como las domésticas (migraciones que se realizan en el mismo territorio nacional pero a regiones diferentes). En esta

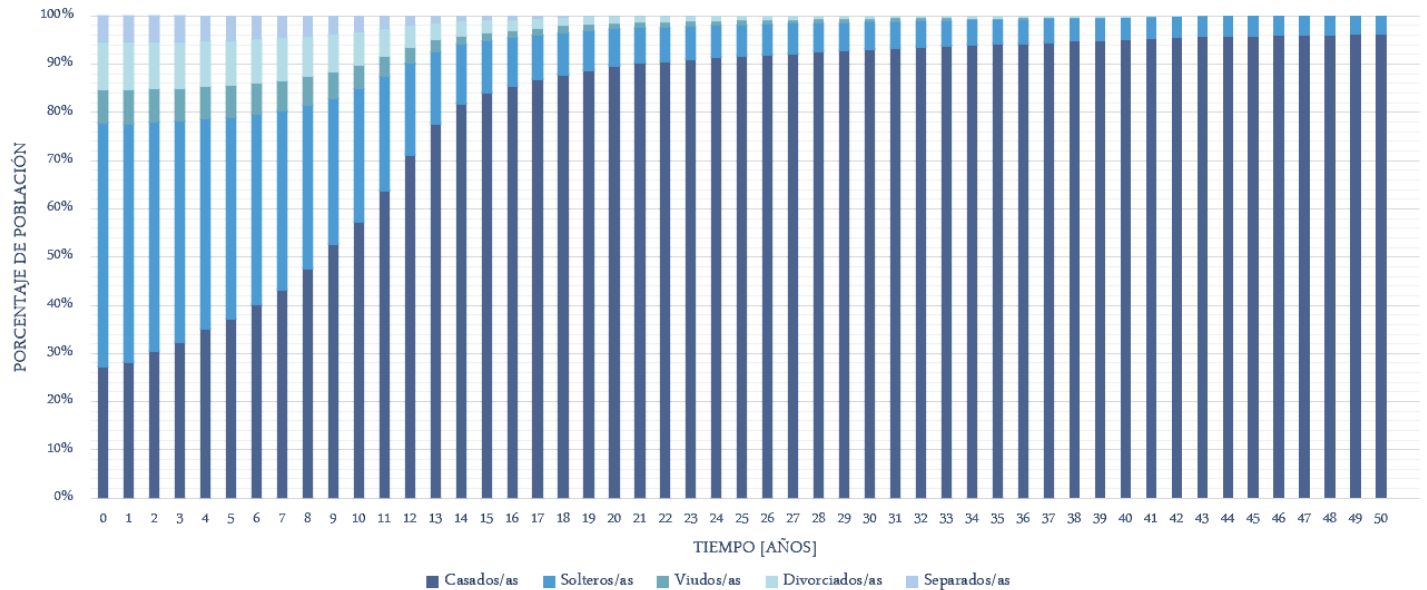


Figura 44: Validación por caja blanca: componente de estados maritales.

prueba se ha comprobado el funcionamiento de ambos tipos de migración.

Para aislar el componente de migración, en el caso de las migraciones domésticas, se ha aislado el componente de estados económicos, ya que las migraciones definidas en el modelo de Gambia se deben a factores económicos de la población. Las pruebas se han realizado con una probabilidad de migración del 20 %.

En la figura 45 se puede observar la gráfica de las emigraciones domésticas para las 4 regiones de simulación. En el eje de abscisas se observan los años de simulación. En el eje de ordenadas se encuentra el número de personas que realizan emigraciones domésticas.

Tal como sucede en el modelo original de Yades, las emigraciones van aumentando con el paso del tiempo hasta situarse en el máximo nivel al final de los años de simulación. La diferencia en el número de individuos que emigran por región, se debe al número de población inicial con el que están inicializadas cada región. Además, a medida que va llegando más población a una región, aumenta el número de migraciones ya que aumenta la población total.

En la figura 46 se puede observar la gráfica de las inmigraciones domésticas para las 4 regiones de simulación. En el eje de abscisas se observan los años de simulación. En el eje de ordenadas se encuentra el número de personas que realizan inmigraciones domésticas.

Asimismo, para el caso del componente demográfico de inmigración, al aumentar la cantidad de población por región aumenta el número de emigraciones y, por tanto, el número de inmigraciones para cada región. Como se está empleando una topología MESH (véase sección 6.2) para determinar las regiones destino para las migraciones, el número de personas que llegan a cada región es equivalente para todas ellas.

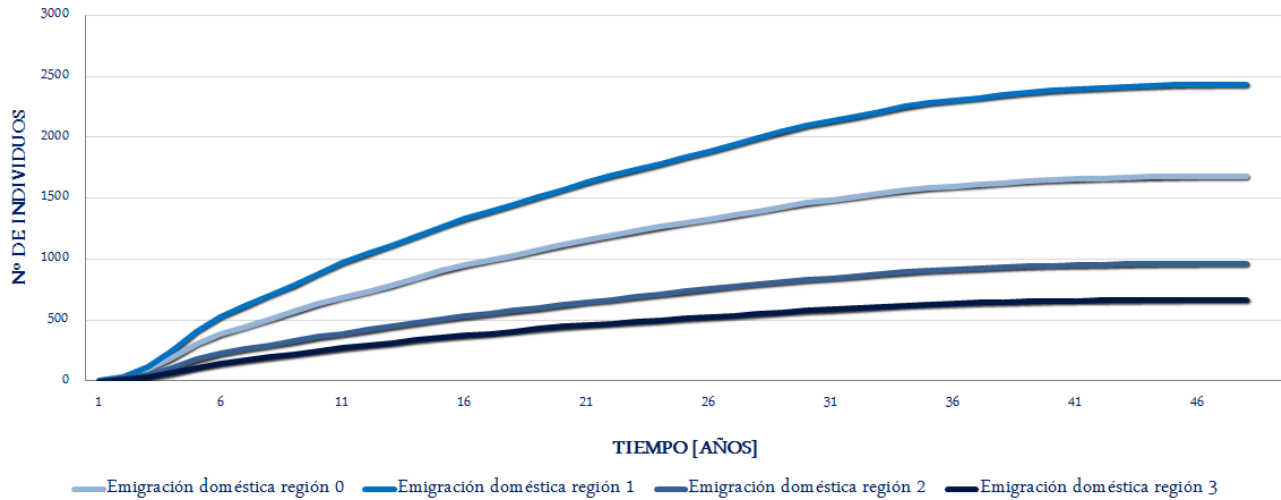


Figura 45: Validación por caja blanca: componente de emigración doméstica para 4 regiones.

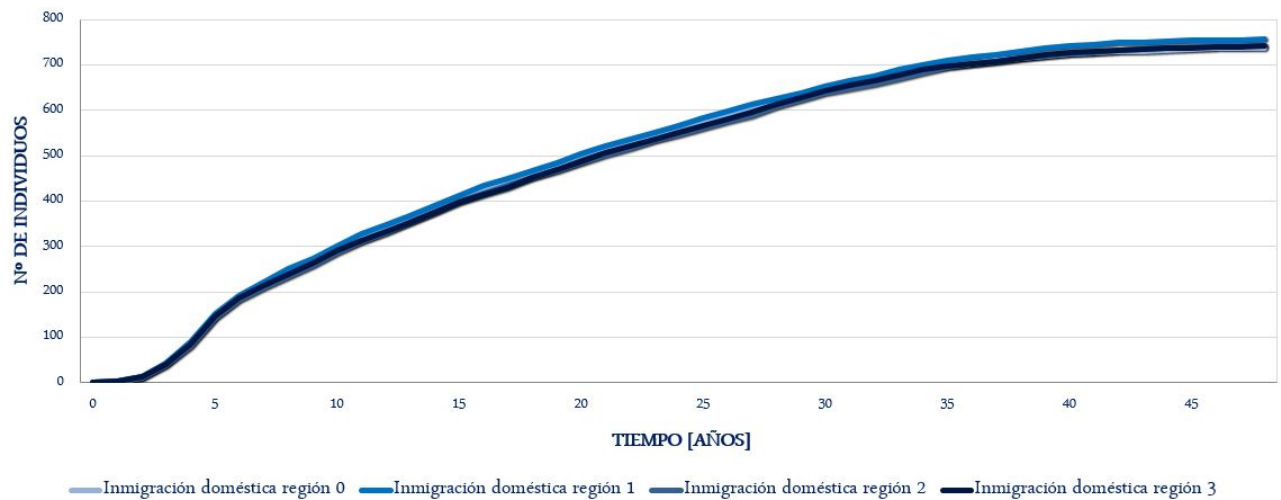


Figura 46: Validación por caja blanca: componente de inmigración doméstica para 4 regiones.

En la figura 47 se puede observar la gráfica de las migraciones internacionales para las 4 regiones de simulación. En el eje de abscisas se observan los años de simulación. En el eje de ordenadas se encuentra el número de personas que realizan migraciones internacionales.

Para el caso de las migraciones internacionales también se puede observar como, a medida que aumenta la población total en cada región, aumentan las migraciones internacionales. En el modelo que Yades tiene actualmente definido, las migraciones internacionales están definidas con unas probabilidades fijas para cada grupo de edad que se corresponden con los datos obtenidos para el caso de estudio de Gambia que se presenta en [Montañola-Sales, 2014].

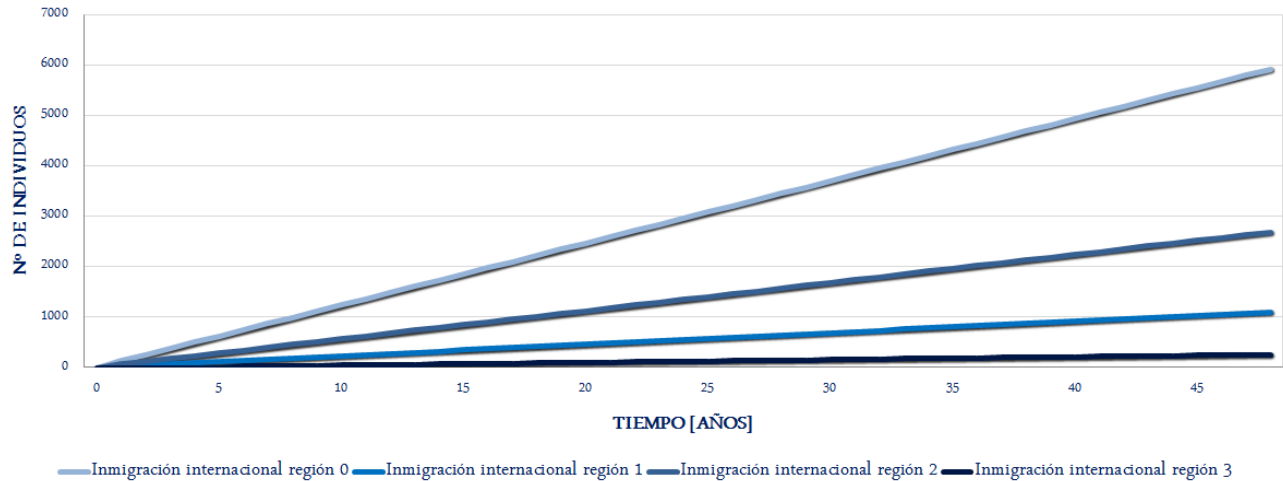


Figura 47: Validación por caja blanca: componente de migración internacional para 4 regiones.

11.2. Testing de aplicaciones paralelas

Como se vió en la figura 37, la fase de test de un sistema de simulación es de vital importancia para asegurar la verificación y la validez de un modelo.

En este proyecto se trabaja con un sistema de simulación paralelo, por lo que se deben tener en cuenta algunas consideraciones especiales a la hora de realizar los test.

La fase de testing de un sistema paralelo tiene como principales objetivos:

- Detectar posibles problemas de comunicación
- Detectar posibles cuellos de botella
- Depurar la aplicación
- Localizar oportunidades de refactorizar o reestructuración del código

Los métodos para buscar errores en sistemas paralelos se basan en los mismos que en sistemas secuenciales [Karpov], y se clasifican en:

- Análisis dinámicos: se lanza el programa y se ejecutan varias réplicas con distintos componentes del sistema, con el objetivo de detectar comportamientos erróneos.
- Análisis estáticos: se examina el código de la aplicación sin lanzarla, en busca de posibles errores.
- Test basados en el modelo: se generan test automáticos de acuerdo a reglas predefinidas para comprobar diferentes funcionalidades del código.

- Pruebas de validación del programa específicas del área de trabajo.

Asimismo, se pueden aplicar algunas estrategias empleadas en testing secuencial para probar software paralelo, como por ejemplo:

- Test de caja negra (black box validation).
- Test de caja blanca (white box validation).
- Mezcla de ambos.

Sin embargo, realizar testing de una aplicación paralela no es una tarea sencilla. Existen muchos artículos sobre como realizar debugging y testing en ésta área y aún queda mucho por investigar. En sistemas paralelos muchas veces es difícil localizar y replicar los errores, ya que en muchos casos dependen del orden en que se ejecuten los eventos o de la impredecibilidad de los planificadores (*scheduler*) de eventos (Non-repeatability property), del compilador usado, de la cantidad de procesadores usados, incluso en muchos casos estos sistemas presentan un comportamiento no determinista que los hacen más difícil de analizar.

Mientras más interacción entre los diferentes componentes paralelos haya, más posibilidades habrá de que se produzcan fallos o problemas de concurrencia como: condición de carrera (race condition), problemas de inanición (starvation), abrazo mortal (deadlock) o bloqueo activo (livelock). Incluso cuando se logra detectar un error, es complicado reproducirlo. Este tipo de errores reciben el nombre de *Heisenbugs* y se refieren a los errores que desaparecen o cambian su comportamiento en el momento de buscarlos. En sistemas paralelos estos errores se pueden presentar en las sincronizaciones, cuando se introducen elementos para retrasar las sincronizaciones (Probe Effect) o cuando se detecta un bug y se intenta aislar para solucionarlo.

En el caso concreto del simulador Yades, si hay dos eventos que se han lanzado con exactamente la misma marca temporal, al ejecutarse en paralelo, el compilador tomará la decisión sobre cual ejecutar primero en función del orden de llegada del evento y otros parámetros propios del planificador de eventos usado, en este caso de ROOT-Sim. Esta decisión no es la misma para todas las ejecuciones ni en todas las máquinas, por lo que pueden aparecer errores que no se habían detectado con anterioridad. Por esta razón, este proyecto se ha probado en diferentes equipos con configuraciones muy diferentes entre ellas, para reducir la posibilidad de tener errores de este tipo.

Además de las técnicas de testing para sistemas generales mencionadas anteriormente, existen otras que se basan en el estudio en profundidad del sistema y que pueden ayudar a realizar testing de sistemas paralelos y encontrar errores. Entre estas técnicas destacan:

- *Instrumentación del código*: a través de programas que permiten obtener información sobre las ejecuciones del programa y que permiten ejecutarlo paso a paso, mientras se van

comprobando los valores generados, así como la comprobación del uso que se realiza de la memoria del sistema, entre otros.

- *Realización de experimentos*: emplear diferentes semillas y realizar varias ejecuciones para evaluar el sistema bajo diferentes escenarios.
- *Análisis de los resultados*: analizar las ejecuciones tanto a nivel de la traza generada en las ejecuciones como a nivel de resultados, para obtener información sobre el comportamiento de la aplicación.
- Documentar bien el proceso para poder repetir los resultados en caso de que sea necesario. Este caso solo se aplica para sistemas deterministas, en los que usando la misma semilla es posible obtener los mismos resultados.

Además, se pueden aplicar técnicas específicas de testing para aplicaciones paralelas que pueden contribuir a detectar y solucionar algunos tipos de errores (como los Heisenbugs) con mayor rapidez y fiabilidad. Entre estas técnicas destacan [Ball et al., 2011]:

- *Usar un planificador determinista* (Deterministic Scheduling): consiste en usar un planificador externo con el fin de evitar errores provenientes de esta fuente. En muchos casos los planificadores funcionan a través de propiedades específicas de los equipos como: los ticks de reloj o intervalos de tiempo empleados, el estado de la caché, entre otros. Por lo tanto, se puede instrumentar el programa a testear para que realice llamadas a este planificador controlado, al principio y al final de cada thread y antes de las sincronizaciones. Este proceso se basa en volver secuencial la ejecución y, por tanto, repetible.
- *Testing dirigido por Feedback*: debido al principio de localidad, tanto temporal como espacial [AC, 2014] para los datos e instrucciones de código, es probable que una vez que se encuentre un error, se encuentren más en esa misma posición de memoria o en posiciones cercanas. Por lo tanto, acompañado de la herramienta del planificador, se puede testear por pasos el software de manera que, en cada uno, se detecta una zona potencialmente susceptible de tener errores y se reporta. Esta técnica es útil para detectar errores de race condition.
- *Testing relacionado con el tiempo* [Yang and Pollock, 1997]: esta estrategia se emplea para asegurarse que la aplicación no tiene problemas de Probe Effect, y consiste en examinar posibles retrasos introducidos en la aplicación, para poder detectar problemas temporales.
- Pruebas de estrés: esta estrategia es usada también en testing paralelo y consiste en llevar el sistema hasta unas condiciones extremas, para evaluar su fiabilidad y como responde ante situaciones de cálculo intensivo.

Para verificar la salida de las ejecuciones de las aplicaciones, o de una parte de ellas, se pueden usar varias estrategias como por ejemplo [Yang and Pollock, 1997] [Booth and Henry]:

- *Test calculado*: se le da una entrada al sistema para que produzca un resultado ya conocido, el resultado que da el sistema se compara con el esperado y se determina si es el correcto o no. Este método sólo permite determinar si hay un error o no.
- Testing implementado en paralelo: este tipo de test se suele usar para aplicaciones que emplean memoria compartida. Permite aislar errores de cálculo al realizar operaciones en diferentes procesadores.
- *Test de simetría*: esta estrategia es usada en aplicaciones científicas en las que se emplean aplicaciones paralelas. En muchos casos, no se puede predecir la salida ya que se trata de aplicaciones no deterministas en las que no se pueden obtener los mismos resultados en diferentes ejecuciones con la misma semilla. Sin embargo como estas aplicaciones se basan en leyes científicas se comparan con patrones establecidos y en caso de que no haya coincidencia se busca la posible anomalía.

Durante la realización de las fases de testing de este trabajo, se ha empleado el test de simetría junto con el testing de caja blanca, para poder comparar y validar los resultados obtenidos en las diferentes ejecuciones del simulador. También en algunas fases se han empleado tests calculados para comprobar si algunas funcionalidades del código producían el resultado esperado.

Además se han realizado diversos test de estrés para los componentes principales de simulación y en cada uno de los casos se han descrito los escenarios empleados con el fin de que se puedan reproducir estas pruebas.

12. Comparativa de resultados

Para realizar la comparativa de resultados de la nueva implementación de Yades con ROOT-Sim con la versión original Yades con la librería μ sik, se ha tomado el caso de estudio de la simulación demográfica de Gambia, junto con todos sus datos y sus reglas demográficas específicas. Estos datos han sido extraídos de [Montañola-Sales, 2014].

12.1. Arquitectura técnica del sistema

12.1.1. Marenostrium 3

Actualmente Yades se ejecuta en el súper computador Marenostrium 3. En este proyecto se ha empleado este supercomputador para realizar pruebas con la aplicación Yades ROOT-Sim, además de para evaluar las aplicaciones iniciales Yades con μ sik y ROOT-Sim.

Este supercomputador cuenta con 3.056 nodos IBM iDataPlex dx360 M4 y un total de 103,5 TB de memoria principal (RAM) y 2 PB de almacenamiento. Cada nodo tiene dos procesadores con 8 cores Intel SandyBridge-EP E5-2670 a 2.6 GHz cada uno, que hacen un total de 16 para cada nodo, en total 48.896 cores. Cada core cuenta con dos hilos de ejecución que compiten por el acceso a cache. La cache cuenta con tres niveles L1, L2 y L3, que se comprueban en el momento de hacer una petición de un dato y antes de acceder a memoria.

La red de comunicación es una combinación de Infiniband FDR10 con Gigabit Ethernet para el acceso a disco.

Cuenta con un sistema operativo de 64 bits Linux, distribución SuSe 11 SP3.

En la figura 48 se muestra la arquitectura de un nodo de Marenostrium 3.

En la figura 49 se muestra la arquitectura de un core de Marenostrium 3.

12.1.2. Capitano

ROOT-Sim fue desarrollado en la Università di Sapienza, Roma, y ha sido probado en el cluster de dicha universidad. En este proyecto se ha empleado este cluster para realizar pruebas con la aplicación Yades ROOT-Sim para comprobar su funcionamiento en diferentes arquitecturas.

Capitano cuenta con un servidor HP ProLiant DL580 Gen7, equipado con cuatro procesadores AMD Opteron 6128 a 2GHz y 64 GB de RAM. Cada procesador tiene 8 cores, que hacen un total de 32. Los cores de cada procesador comparten una cache con tres niveles L1, L2 y L3 de 12MB (6 MB para cada grupo de cuatro cores). Cada core además tiene una cache privada de dos niveles de 512KB. En la figura 50 se muestra la arquitectura de Capitano.

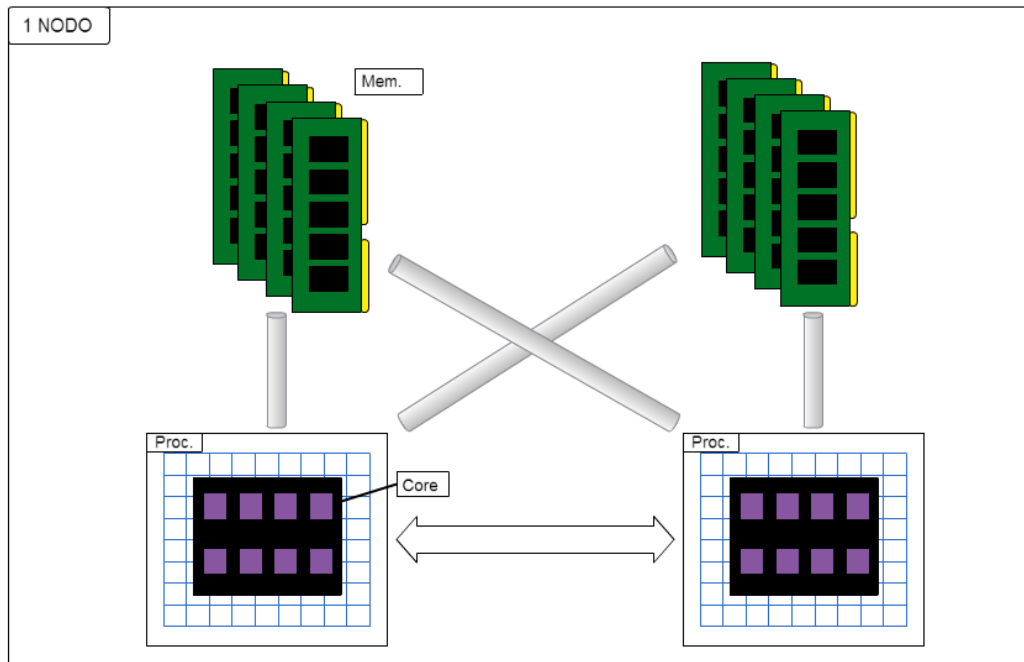


Figura 48: Arquitectura de un nodo de Marenostrium.

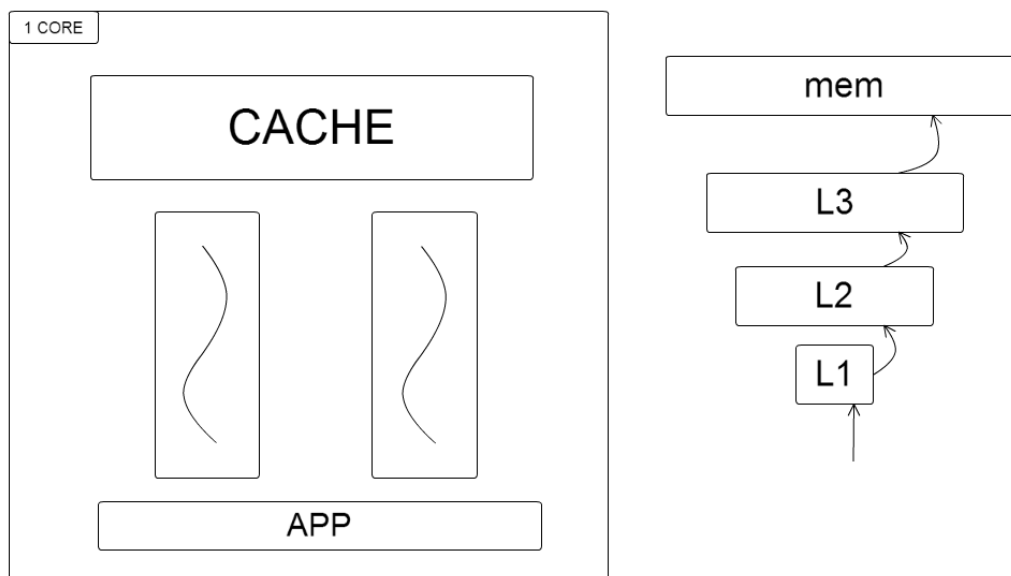


Figura 49: Arquitectura de un core de Marenostrium.

Cuenta con un sistema operativo de 64 bits, Debian 6, con un kernel Linux versión 2.6.32.5. Tiene un compilador gcc 4.9.2 y emplea la librería binutils 2.25.

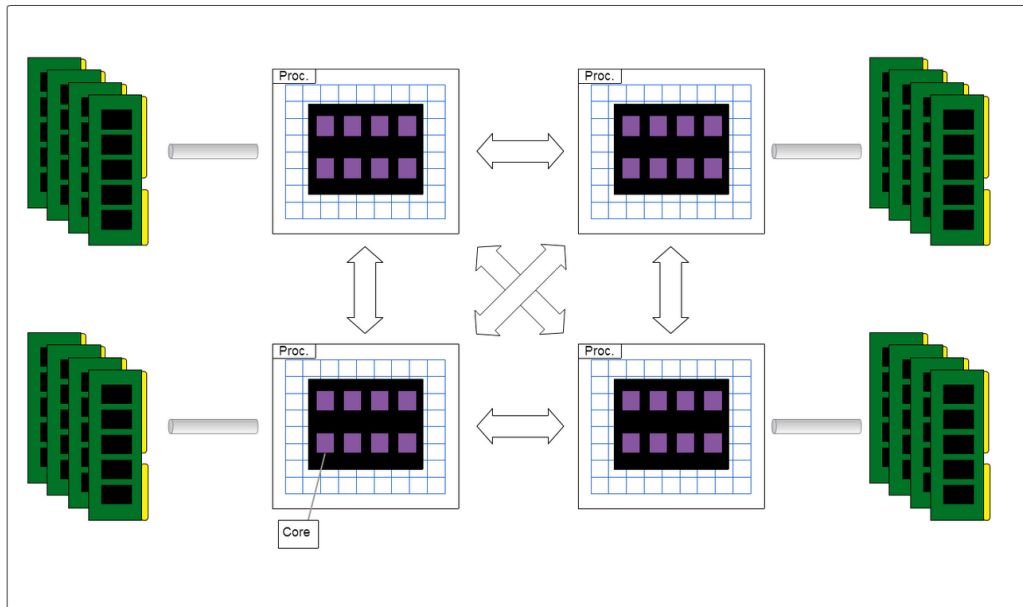


Figura 50: Arquitectura de Capitano.

12.2. Comparativa

En la sección 10 se presentó la tabla 62 en la que se muestra el diseño de experimentos realizado para esta comparativa.

Los diferentes experimentos tienen como finalidad realizar comparativas a varios niveles secuencial y paralelo para evaluar el rendimiento (ρ) de la aplicación.

12.2.1. Análisis de ρ en secuencial

A diferencia de la versión de Yades con μ sik, la versión con ROOT-Sim permite ejecutar la aplicación en modo secuencial incluyendo varios LP, regiones en este caso de estudio concreto, y migraciones. Este modo de ejecución, está especialmente optimizado para obtener el mayor rendimiento posible de la aplicación, entre las técnicas que se emplean para realizar las ejecuciones se encuentran el hecho que no se salva el estado de la aplicación, es decir el estado de cada LP, como en el caso paralelo. Esta diferencia se debe a que en secuencial no es posible realizar un rollback ya que las ejecuciones se realizarán en orden y se mantendrá la consistencia temporal para cada LP. Por esta razón se han incluido los resultados secuenciales en este análisis ya que representan una ganancia para la aplicación, al permitirle ejecutar un gran número de regiones en secuencial en un tiempo de ejecución aceptable.

Las dos pruebas que se presentan a continuación se han realizado en secuencial, es decir con un thread, en Marenostrium 3, con los datos iniciales siguientes: 4 regiones, población inicial de

2642, 3895, 1470 y 1032, respectivamente para cada región, probabilidad de migración del 0.1 %.

En la figura 51 se muestra una traza de Yades con ROOT-Sim, esta ejecución tardó 0.969 segundos. En el caso de musik, no es posible generar una traza ya que en secuencial no se lanzan comunicaciones MPI.

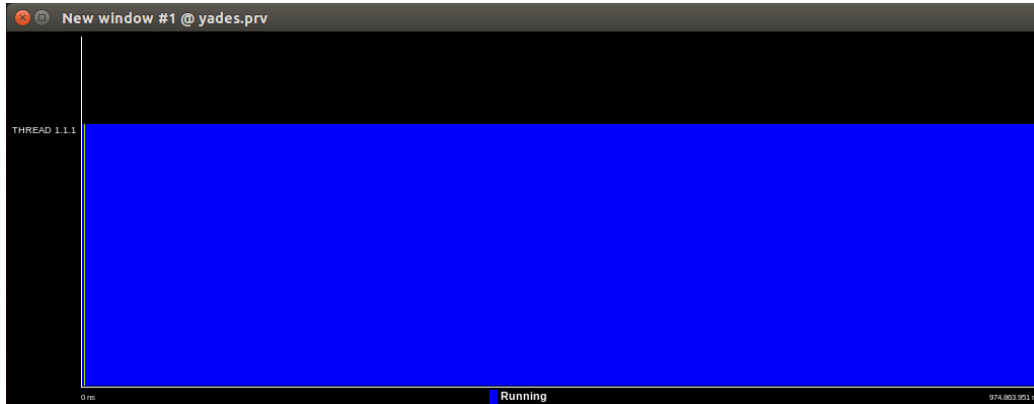


Figura 51: Traza Yades-ROOT-Sim en secuencial.

Se puede observar como a medida que aumenta la población, aumenta el tiempo de ejecución de la aplicación. Para esa gráfica, se ha simulado la población que aparece en el eje de abscisas para cada región de simulación.

En la figura 52 se muestra el impacto en el tiempo de ejecución en la versión de Yades con ROOT-Sim a medida que se aumenta la población inicial por regiones. Esta gráfica ha sido extraída en MN3, en secuencial, con 4 regiones y una probabilidad de migración del 0%.

En la figura 53 se muestra el impacto en el tiempo de ejecución en Yades con ROOT-Sim a medida que se aumenta la población inicial por regiones y se aumenta la probabilidad de migración. Esta gráfica ha sido extraída en Marenostrom 3, en secuencial, con 4 regiones. No se ha podido obtener valores más allá de 40 % de migraciones para los casos de población de 20 y 40 mil familias por región debido a restricciones de memoria de la arquitectura. Como se puede ver, el tiempo de ejecución aumenta de forma no lineal a medida que la población aumenta.

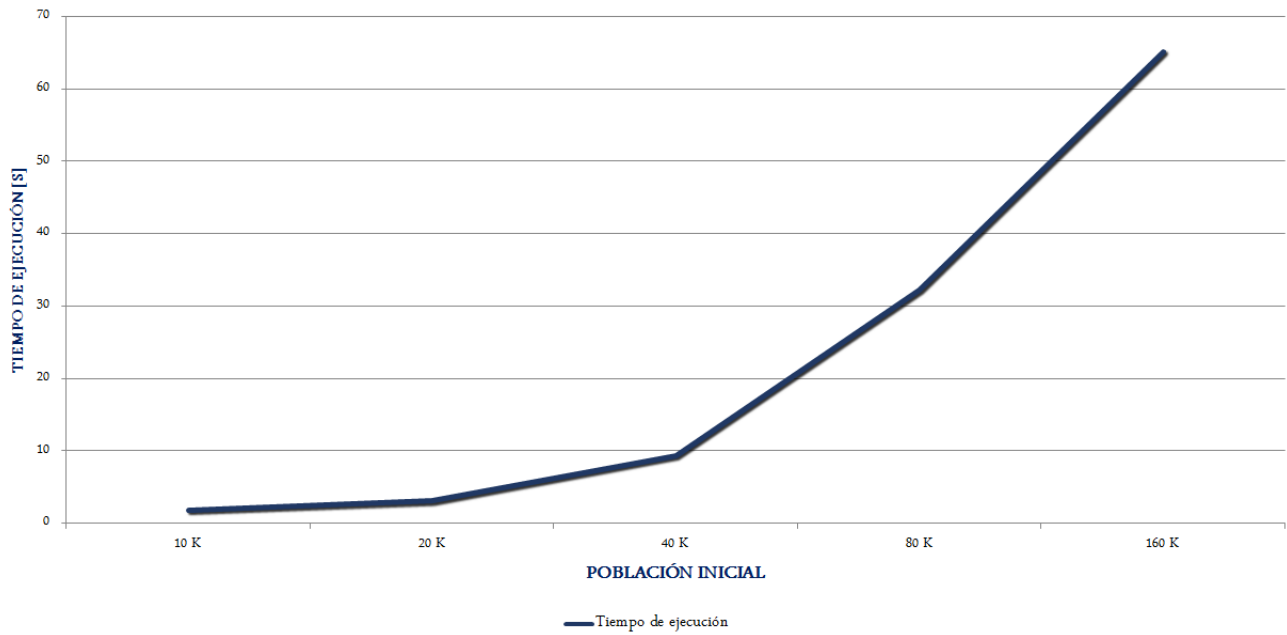


Figura 52: Impacto en el tiempo de ejecución de Yades con ROOT-Sim en secuencial en MN3 a medida que se aumenta la población.

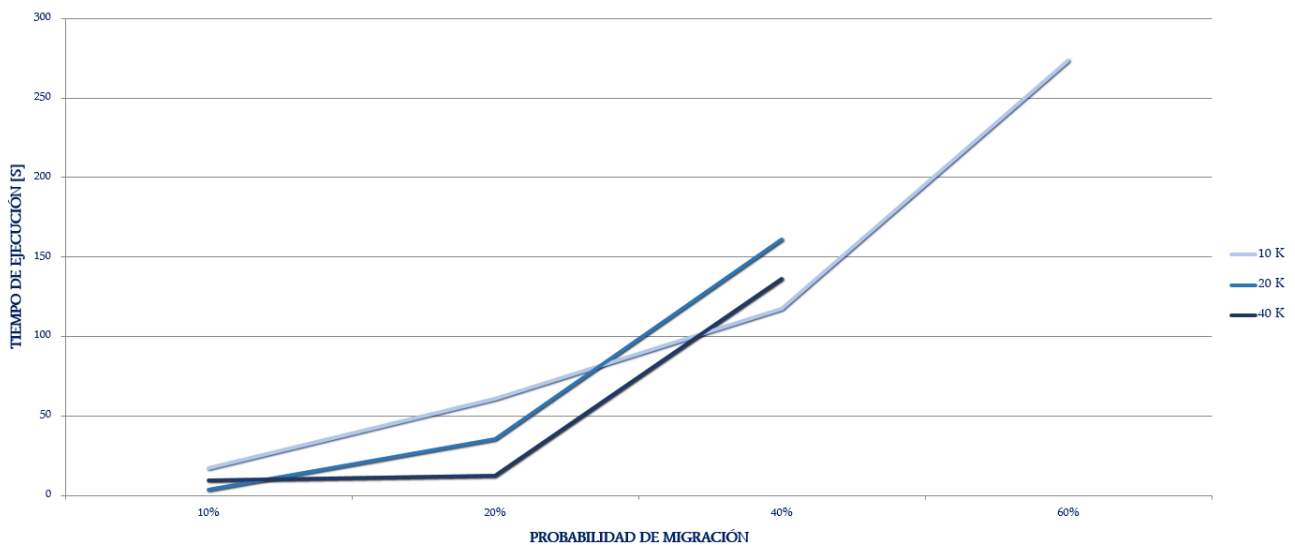


Figura 53: Tiempo de ejecución de Yades con ROOT-Sim en secuencial en MN3 a medida que aumenta la población y la probabilidad de migración.

A continuación se presenta el análisis de Yades con μsik en secuencial. Como características especiales, este análisis solo se ha realizado con una región, o LP, y con una probabilidad de migración del 0%, ya que es lo máximo que permite este modo de ejecución. Este análisis se presenta en la figura 54.

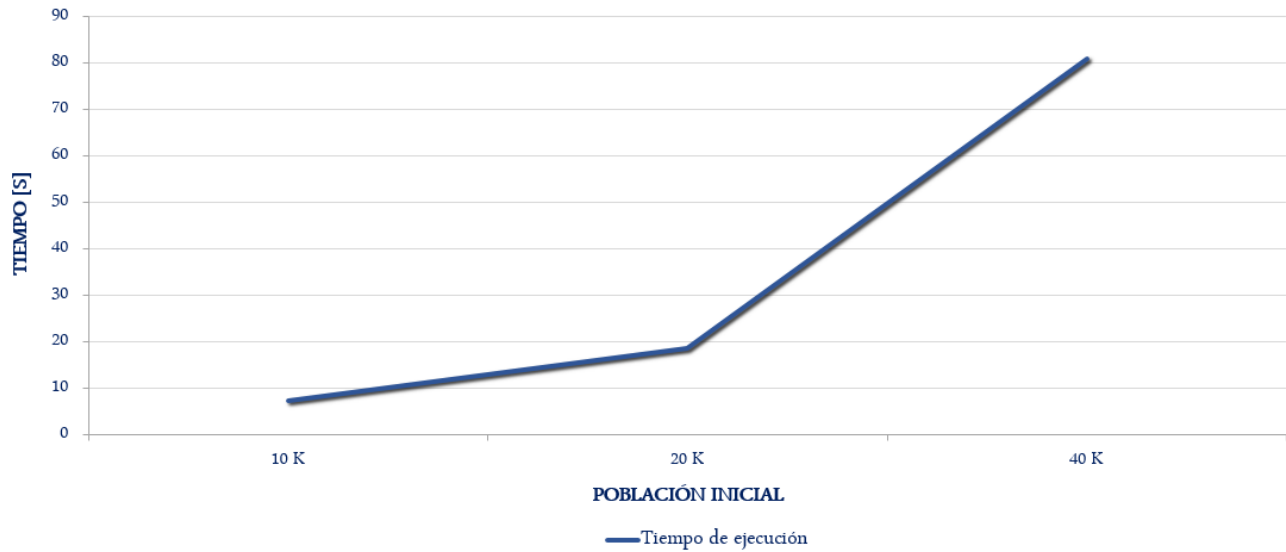


Figura 54: Tiempo de ejecución Yades μsik , en secuencial, con población creciente en Marenstrum 3.

En la figura 55 se muestra una comparativa en el tiempo de ejecución de ambas aplicaciones ejecutándose bajo las mismas condiciones: en secuencial, 1 región (LP o Thread) población creciente, y probabilidad de migración del 0%, en Marenostrom 3. Como se puede observar, en la ejecución secuencial la implementación de Yades con ROOT-Sim es claramente mejor respecto a la anterior con μsik .

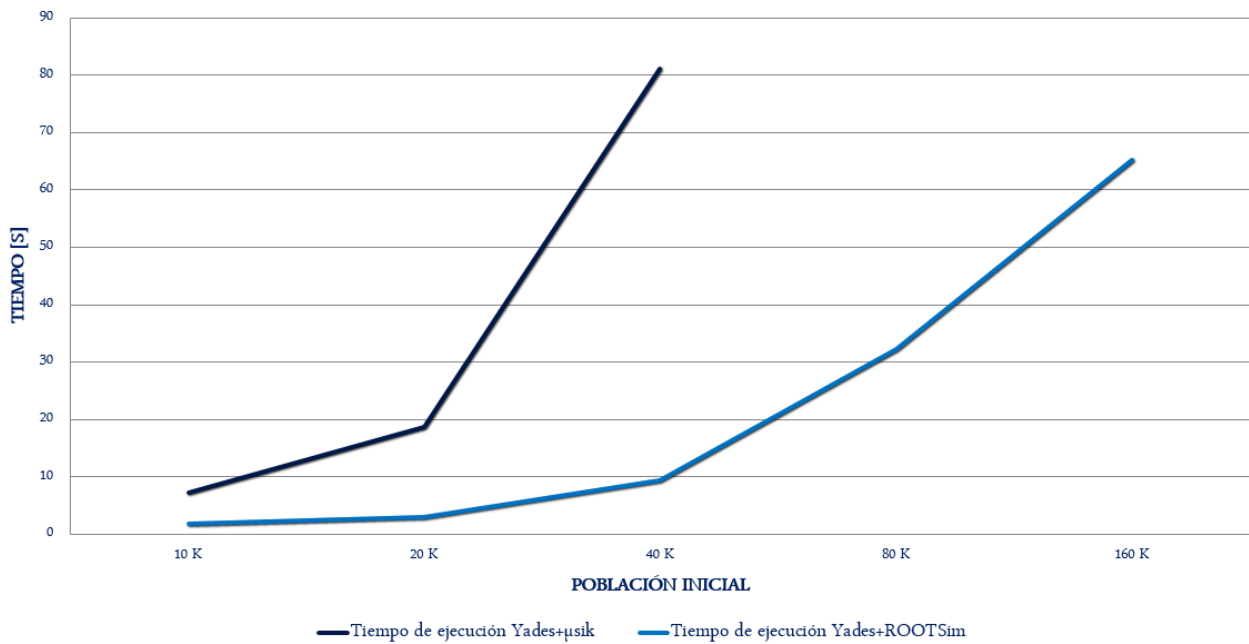


Figura 55: Comparativa secuencial Yades μsik - Yades ROOT-Sim con población creciente y probabilidad de migración 0.

12.2.2. Análisis del paralelismo

A continuación se presenta el análisis en paralelo de Yades con ROOT-Sim y se realiza una comparativa con la versión de Yades con μ sik. Estos análisis han sido realizados en Marenostrum3 y en Capitano. En la figura 56 se muestra una traza de Yades con μ sik, esta ejecución tardó 67.3746 segundos.

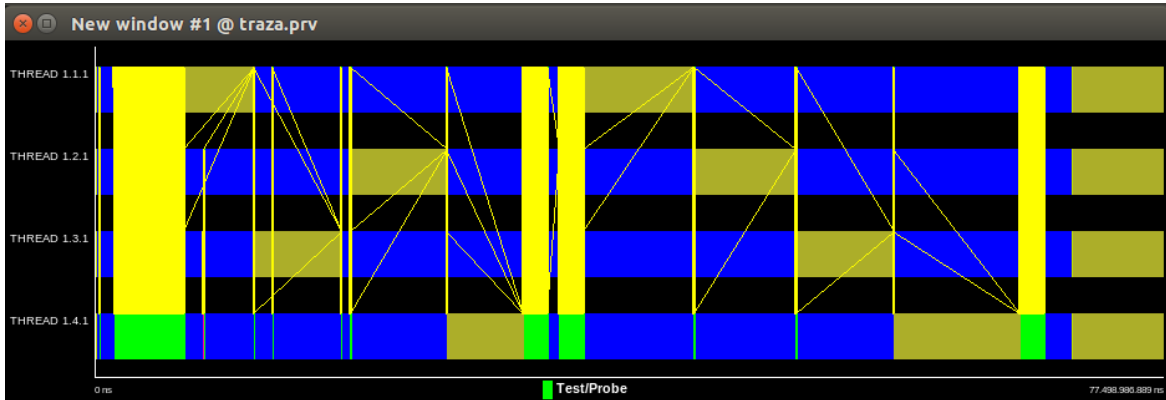


Figura 56: Traza Yades- μ sik con 4 procesadores.

En la figura 59 se muestra el impacto en el tiempo de ejecución en la versión original de Yades con μ sik a medida que se aumenta la población inicial por regiones. Esta gráfica ha sido extraída en MN3, con 4 procesadores, 4 regiones y una probabilidad de migración del 0.1%.

En la figura 58 se muestra una traza de Yades con ROOT-Sim, esta ejecución tardó 6.029 segundos con una probabilidad de migración del 0.2%. En la figura no se observa ninguna zona amarilla que represente comunicaciones, esto es debido a que ROOT-Sim realiza las comunicaciones a nivel de memoria y representa un gran avance a nivel de rendimiento. La zona azul, por tanto, corresponde a trabajo de ejecución de la aplicación. En cambio, la zona blanca que se observa al inicio pertenece a la creación y sincronización de threads.

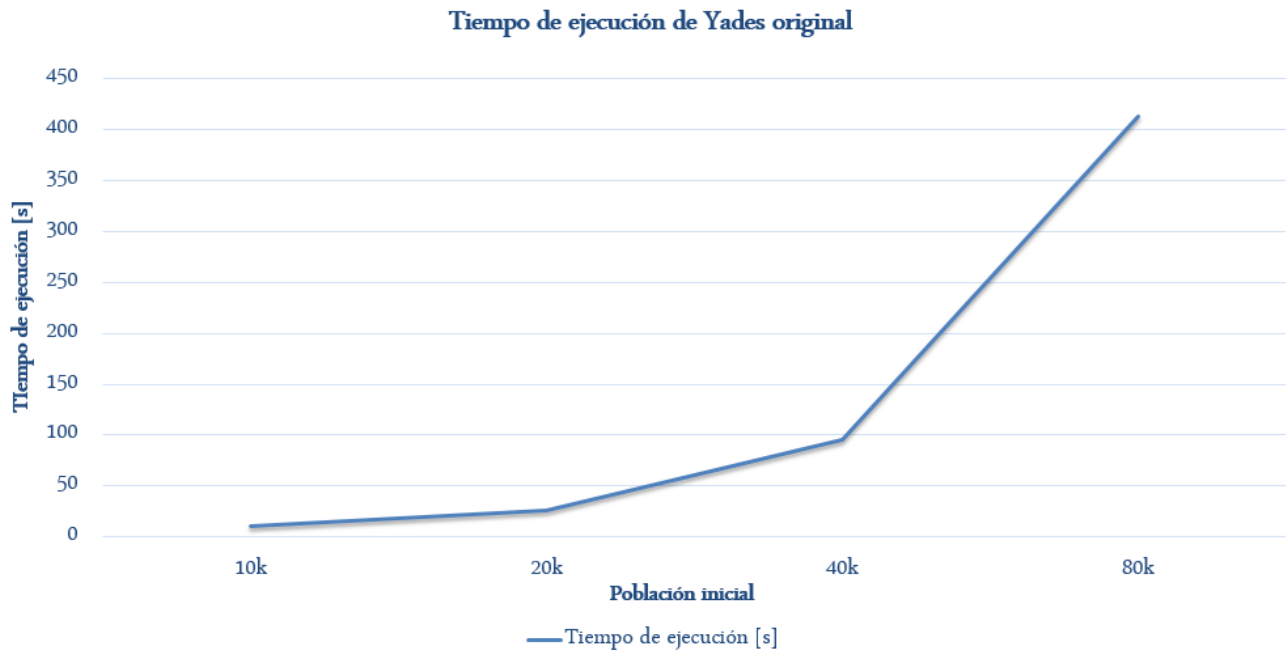


Figura 57: Tiempo de ejecución de Yades con μsik en paralelo en MN3 a medida que aumenta la población.

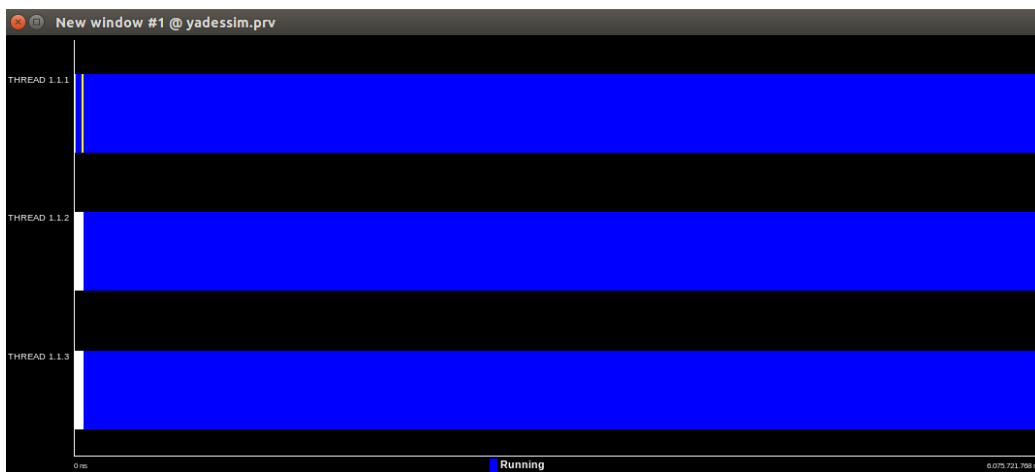


Figura 58: Traza Yades-ROOT-Sim con 3 procesadores.

En este punto, se desea evaluar el rendimiento del trabajo realizado en un entorno paralelo. Para ello, se ha decidido hacer 3 tipos de pruebas distintas:

- Medir el impacto de simular distintos tamaños de población.
- Medir el rendimiento al variar la probabilidad de migración.
- Observar el efecto de aumentar el número de recursos sobre el tiempo de ejecución.

Las siguientes secciones presentan los resultados para cada una de las perspectivas de estudio.

Efecto de variar la población sobre el rendimiento

Para observar el impacto de aumentar la población sobre el rendimiento de Yades con ROOT-Sim se ha realizado un experimento con una población sintética de 2642, 3895, 1470, 1032, 1071, 1344, 1055 y 1550 familias para cada región respectivamente. En este caso, el número de threads (que equivalen al número de procesadores) debe ser menor o igual que el número de regiones a simular debido a la configuración que se está empleando. Por lo tanto, en el caso de simular 4 regiones sólo se podrá ejecutar la configuración de 1 a 4 procesadores. En este caso, las pruebas han sido realizadas en Capitano, con una probabilidad de migración de 10 %. Los resultados de este experimento se pueden ver en la figura 59, donde se puede observar el efecto de variar diferentes números de LP (regiones) a medida que se aumenta el número de threads/procesadores. Como se puede observar, a medida que se simulan poblaciones más grandes a la vez que se aumenta el número de recursos, el tiempo de ejecución disminuye. Cabe notar que en ROOT-Sim el rendimiento cae cuando el número de threads se iguala al número de procesadores. Esto explica, por ejemplo, que en el caso de 32 regiones el tiempo de ejecución es mayor que con 16.

Asimismo, es interesante medir el speedup que se consigue con la optimización de Yades que se plantea en este proyecto. Para ello, se ha realizado un experimento en Capitano con la misma población sintética del experimento anterior y la misma probabilidad de migración (al 10 %). En este caso, se ha variado el número de procesadores (threads) y se ha estudiado el efecto de variar el número de regiones (LP). Por lo tanto, se han obtenido resultados para diferentes tamaño de población. Los resultados de este experimento se muestran en la gráfica 60 donde se puede observar como aumenta la escalabilidad de la aplicación a medida que se simulan más regiones por cada thread/procesadores. Esto es un indicador de que ROOT-Sim espera un número de LP elevado por thread, es decir, es un librería diseñada para gestionar un gran número de regiones por procesador. Esta característica es de vital importancia para evaluar el rendimiento de futuras aplicaciones prácticas de simulación social con Yades-ROOT-Sim.

Efecto de variar las migraciones sobre el rendimiento

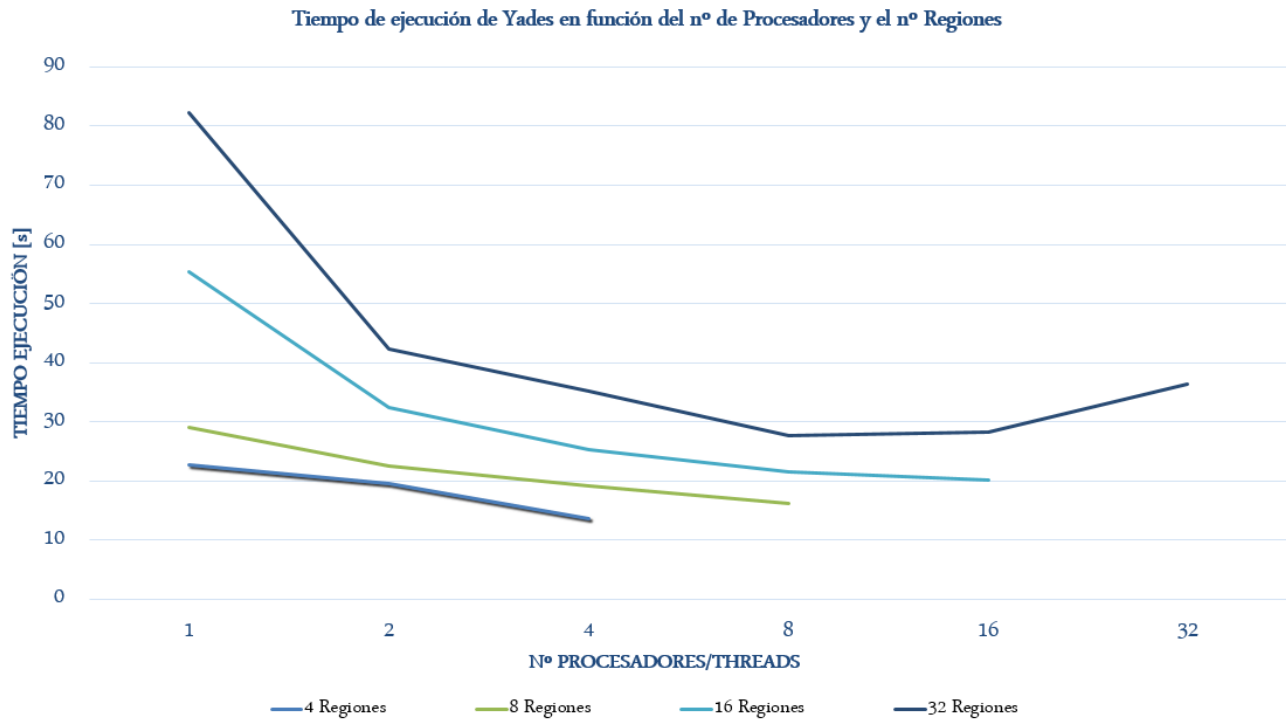


Figura 59: Tiempo de ejecución de Yades con ROOT-Sim en paralelo en MN3 a medida que aumenta la población.

En esta sección, se desea medir el impacto que tiene variar las migraciones sobre el rendimiento de la aplicación. Para ello se ha realizado un experimento en Capitano con una población sintética de 2642, 3895, 1470, 1032 familias respectivamente para cada región, con 4 procesadores y 4 threads. En la gráfica 61 se pueden ver los resultados de este experimento. Como se puede observar, a medida que la probabilidad de migración se incrementa (y por tanto el número de migraciones que tienen lugar), el rendimiento del simulador se reduce. Esto es debido a que las migraciones son comunicaciones entre procesos lógicos que tienen un overhead o coste añadido.

Asimismo, es interesante medir el speedup que se consigue con la optimización de Yades cuando varía el número de migraciones. Para ello, se ha realizado un experimento en Capitano con la misma población sintética del experimento anterior y probabilidad de migración variable de 0% al 40%. En este caso, se ha variado el número de procesadores (threads) y se ha estudiado el efecto de variar las migraciones. Los resultados de este experimento se muestran en la gráfica 62 donde se puede observar como aumenta la escalabilidad de la aplicación a medida que las migraciones disminuyen. Esto es un resultado esperado ya que en ROOT-Sim, mientras más alta es la probabilidad de migración, más población se ve desplazada hacia otras regiones (LP) con el consiguiente coste computacional asociado.

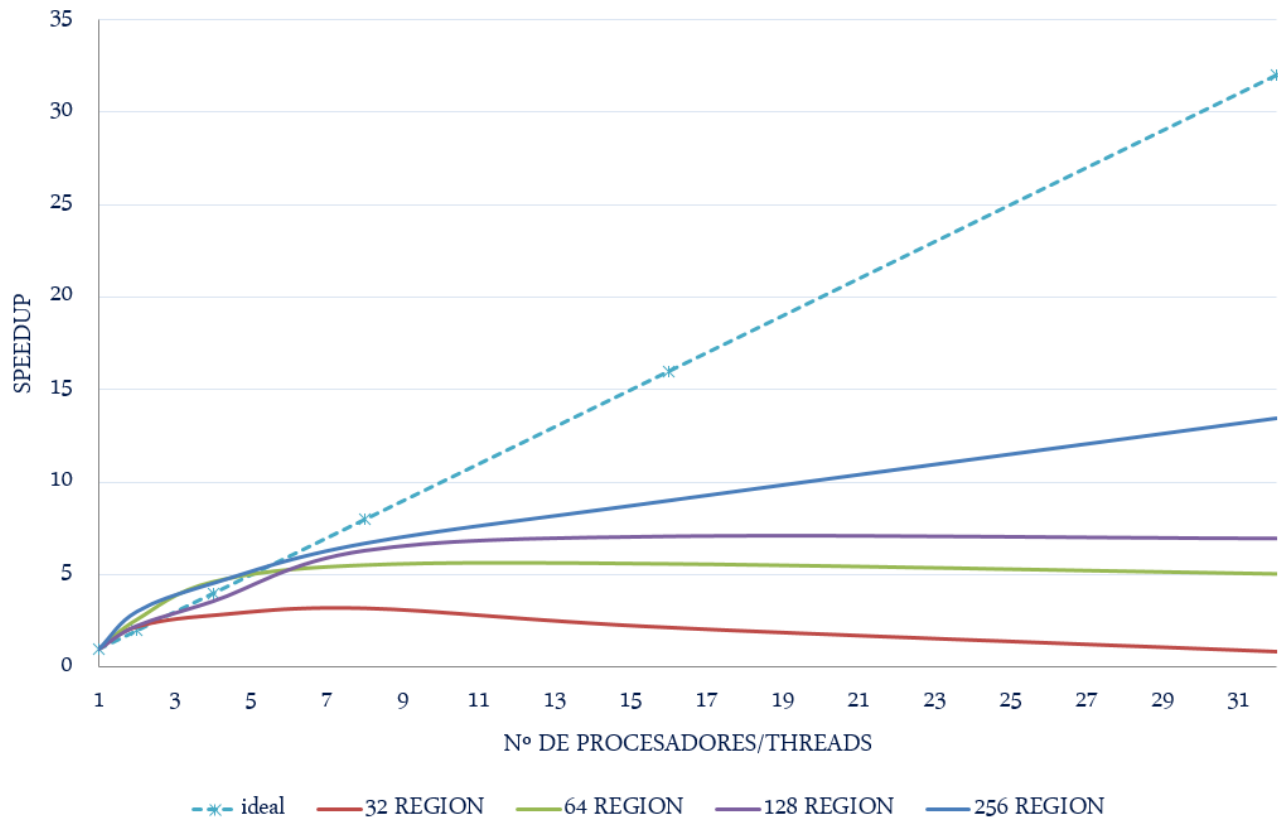


Figura 60: Análisis del speedup de Yades con ROOT-Sim en paralelo, en Capitano.

Efecto de variar los recursos sobre el rendimiento

En esta sección se desea medir el impacto de, dada una población determinada, variar el número de recursos (threads o procesadores) sobre su tiempo de ejecución. Para ello, se ha realizado un experimento en el clúster Capitano donde se simulan tres tamaños de población diferentes (de 16 mil familias a 64 mil familias en total), sin migraciones y con un tamaño de threads (que en este caso es igual al número de LP o regiones) predefinido de 32. En la gráfica 63 se presentan los resultados de aumentar los recursos (cores) cuando se están simulando esos tres tamaños de población diferentes con el número de threads predeterminado a 32.

En esta gráfica se observa que para una población de 16 mil familias, aumentar los recursos no afecta al tiempo de ejecución; en cambio para una población de 32 o 64 mil familias, el tiempo de ejecución se ve reducido a medida que se pone a disposición de la simulación más procesadores. Esto es especialmente significativo para el caso de la población mayor. Sin embargo, se observa que a partir de 8 cores, el rendimiento de la simulación de 64 mil familias se reduce. Se puede intuir que este comportamiento se debe a que existe un número ideal de procesadores por cantidad de threads a usar [Aiguade and Jimenez, 2012-2013] y probablemente, este número

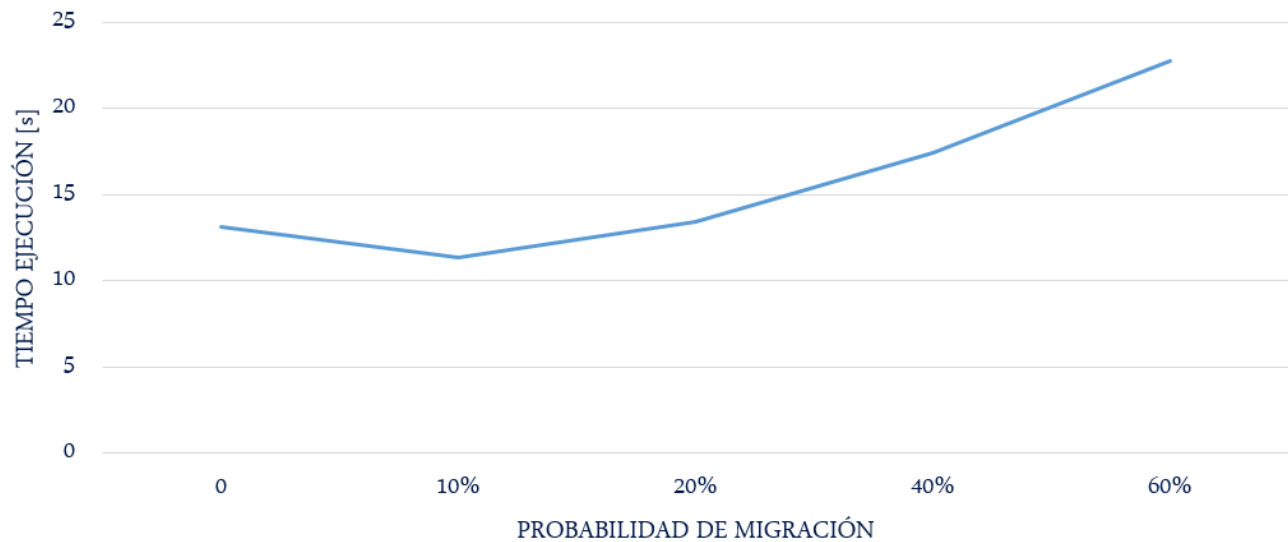


Figura 61: Efecto de variar la cantidad de migraciones sobre Yades con ROOT-Sim con una población sintética y 4 procesadores/threads.

esté entre 8 y 16 procesadores, para 64 mil familias y 32 threads.

En la gráfica 64 se presentan los resultados de aumentar los recursos, en este caso haciendo la equivalencia un thread una región y dejando estable el número de procesadores (cores) a 4. Se simula un tamaño de población total estable de 32 mil familias, repartidas entre las regiones de simulación. En la gráfica se observa que a medida que se aumenta el número de regiones y por lo tanto, la población total está más repartida entre ellas, el tiempo de ejecución disminuye. Este resultado es muy significativo y está acorde con el esquema que sigue ROOT-Sim y que se ha observado en otros experimentos de utilizar un alto número de LP (regiones o threads) para la simulación.

Como conclusión, se puede ver que tanto el tamaño de la población como las migraciones del modelo son cruciales a la hora de determinar el rendimiento de la solución que se ha propuesto. Sin embargo, en líneas generales el tiempo de ejecución disminuye a medida que se paraleliza más la población. De forma contraria, el tiempo de ejecución se ve perjudicado cuando las migraciones se incrementan. Finalmente, la escalabilidad de la solución propuesta en este trabajo muestra un buen potencial, sobre todo cuando se explota el hecho de que ROOT-Sim se comporta mejor en un contexto en el que gestione un número elevado de procesos lógicos por procesador.

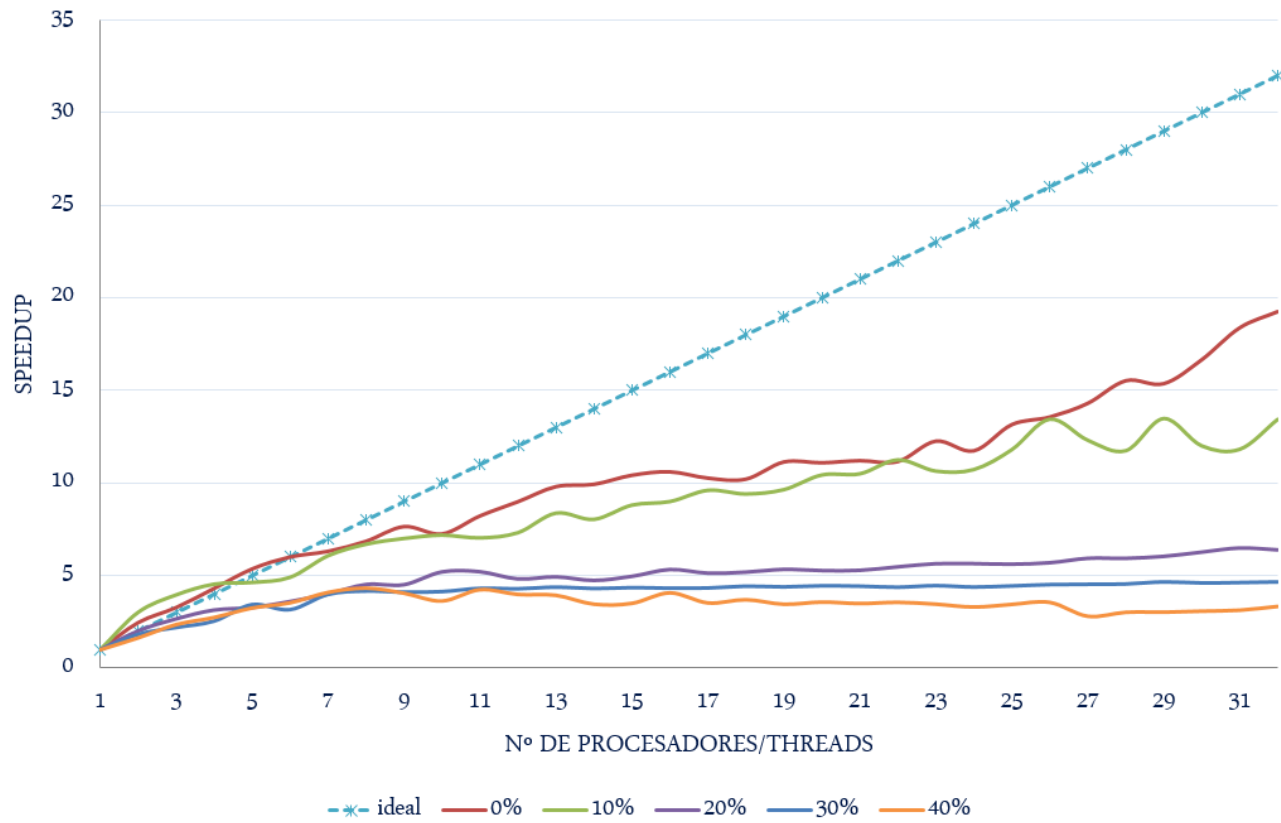


Figura 62: Análisis del speedup con migración variable de Yades con ROOT-Sim en paralelo, en Capitano.

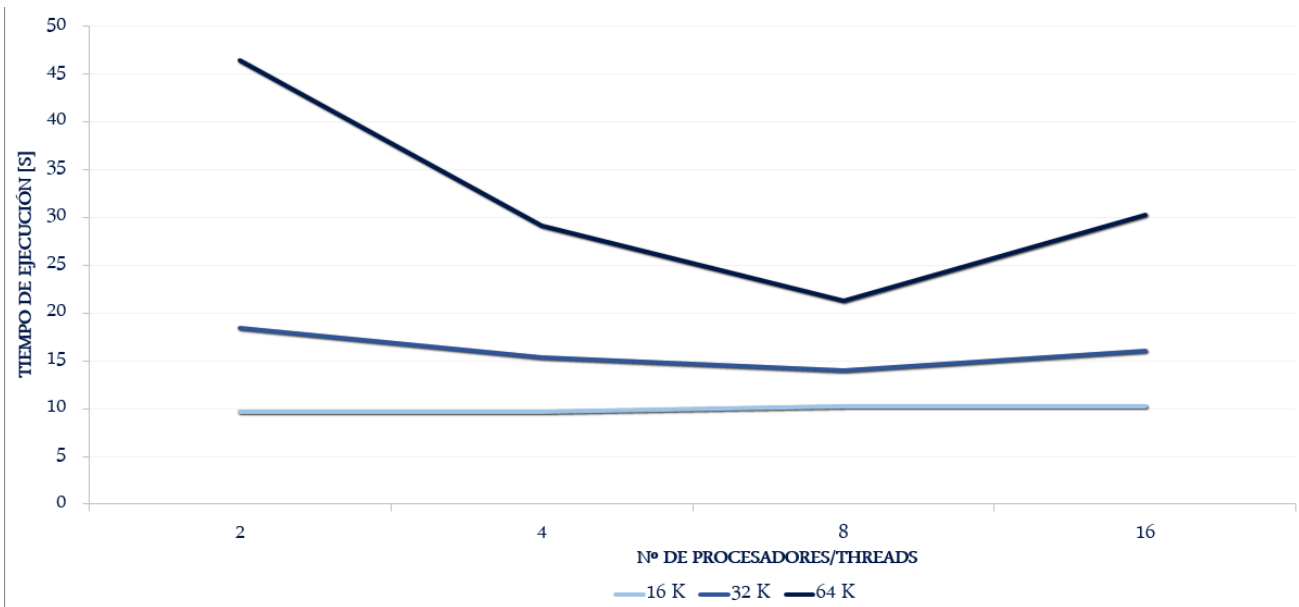


Figura 63: Análisis de aumento de recursos con 3 poblaciones diferentes, en 32 regiones para Yades con ROOT-Sim, en Capitano.

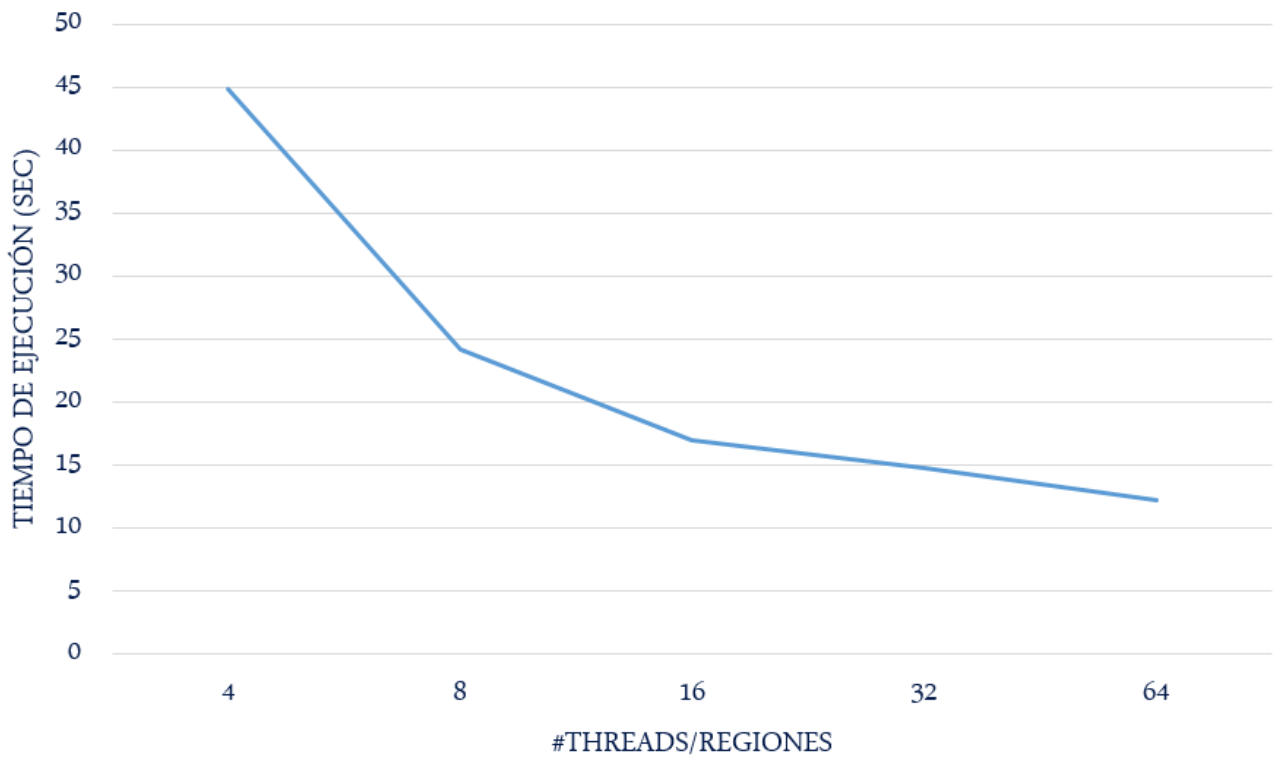


Figura 64: Análisis de aumento de recursos con población estable para Yades con ROOT-Sim, en Capitano.

13. Presupuesto y sostenibilidad

Es necesario realizar un estudio económico del proyecto para evaluar si es viable económicamente y hacer un estudio del impacto ambiental y social de éste.

Para determinar si el proyecto es económicamente viable se deben identificar y estimar los costes para saber si se adecuan al presupuesto establecido.

13.1. Identificación de costes

En la tabla 64 se presentan los recursos humanos que son necesarios durante la realización del proyecto, con sus costes asociados. Los precios que se presentan a continuación, se basan en la política de precios seguida en la empresa.

| Rol | Coste por hora € |
|---------------------|------------------|
| Jefe de proyecto | 35,00€ |
| Analista | 29,00€ |
| Diseñador | 29,00€ |
| Técnico programador | 24,00€ |
| Tester | 24,00€ |
| Documentador | 24,00€ |

Tabla 64: Análisis de costes por recursos humanos del proyecto

En la tabla 65 se presentan los costes asociados al software empleado para la realización del proyecto.

En la tabla 66 se presentan los costes asociados por la infraestructura del proyecto.

A pesar que al comienzo del proyecto ya se cuenta con el hardware para su realización, se debe tener en cuenta su coste. En este caso, se trata de un ordenador de sobremesa HP Compaq 8100, con procesador Intel(R) Core(TM) i5 CPU y 8GB de memoria RAM. Se calcula el coste del equipo utilizando el coeficiente lineal mínimo de amortización del Gobierno de España, en el documento de “Manual práctico Sociedades 2013” [Varios, 2013]. En este documento se especifica que este coeficiente es de un 25 % durante un periodo de 4 años de amortización. Teniendo en cuenta que el coste del equipo ascendió a 750€, que la jornada de trabajo es de 4h al día y que la duración estimada del proyecto es de 107 días de trabajo, el cálculo de la amortización del hardware es el siguiente: $(750 * 0,25 * 107 * 4) / (8*365) = 27,48€$.

Otro de los equipos que se ha empleado en el desarrollo es un portátil SVT1112M1E, con procesador Intel(R) Core(TM) i3 CPU y 4GB de memoria RAM. Su coste se calcula utilizando el mismo coeficiente anterior, del 25 % durante un periodo de 4 años de amortización. Teniendo

| Software | Coste € |
|------------------------------|---------|
| Microsoft Office Project | 0€ |
| Microsoft Office Power Point | 0€ |
| Microsoft Office Excel | 0€ |
| Texmaker | 0€ |
| ShareLatex online | 0€ |
| Cacoo | 0€ |
| Thunderbird | 0€ |
| Skype | 0€ |
| Gdb | 0€ |
| Valgrind | 0€ |
| Paraver-Extrac | 0€ |
| Vim | 0€ |
| Geany | 0€ |
| RootSim | 0€ |
| Yades | 0€ |
| μ sik | 0€ |
| Trello | 0€ |
| Git | 0€ |

Tabla 65: Análisis coste por software del proyecto

en cuenta que el coste del equipo ascendió a 730 €, que este equipo se empleará como soporte, documentación y testing, durante una jornada de trabajo de 4h al día y durante 20 días para todo el proyecto, aproximadamente, el cálculo de la amortización del hardware es el siguiente: $(730 * 0,25 * 20 * 4) / (8 * 365) = 5€$.

Los otros costes derivados del desarrollo del proyecto son los costes indirectos, en este caso el consumo eléctrico de los ordenadores de trabajo y el coste del acceso a internet. La estimación del coste de un equipo de sobremesa es de 1,76kWh [Varios, b], multiplicado por las 540 horas de trabajo previstas. Para el equipo portátil la estimación del coste es de 0,88kWh [Varios, b], multiplicado por 80 horas de trabajo. De esta manera se determina que se consumen 950,4 Kw con el equipo de sobremesa y para el equipo portátil 70,4 Kw, para desarrollar el proyecto. Con la tarifa eléctrica contratada en la oficina donde se desarrolla el proyecto el precio del kWh es de 0,126122€. El precio de la energía consumida por el equipo de sobremesa es de 119,86€ y de 8,88€ para el equipo portátil, de manera que el precio final de la energía consumida es de 128,74€. Por otro lado, la cuota de acceso a internet contratada es de 45€/mes, y teniendo en cuenta que la jornada de trabajo es de 4 horas al día y que la duración estimada es de 107 días, el cálculo del acceso a internet para el equipo de sobremesa es el siguiente: $(45 * 12 * 107 * 4) / (8 * 365) = 79,15€$ y para el equipo portátil: $(45 * 12 * 20 * 4) / (8 * 365) = 14,80€$. El total del acceso a internet para ambos equipos es de 93,95€.

El coste de los recursos asociados a MN3 (Marenostrum 3) se ha cuantificado como cero, debido a que actualmente el uso de estos recursos proviene de la red de supercomputación europea y, en este momento, no tiene coste para el equipo de investigación. Asimismo el coste de los recursos asociados a Capitano también se ha cuantificado como cero, debido a que es proporcionado por uno de los desarrolladores de ROOT-Sim y en este momento, no tiene coste para el equipo de investigación.

| Infraestructura | Coste € |
|--|----------------|
| Ordenador Intel(R) Core(TM) i5 CPU con 8GB RAM | 27,48€ |
| Ordenador Intel(R) Core(TM) i3 CPU con 4GB RAM | 5€ |
| Máquina virtual con Ubuntu 14.04 | 0€ |
| Cuenta en MN3 (Marenostrum 3) | 0€ |
| Cuenta en Capitano | 0€ |
| GIT scholar edition | 0€ |
| Bitbucket free edition | 0€ |
| Electricidad | 128,74€ |
| ADSL | 93,95€ |

Tabla 66: Análisis de costes por infraestructura del proyecto

El coste total de la infraestructura del proyecto es de 255,17€.

13.2. Estimación de costes

Para poder realizar la estimación de costes del proyecto en relación con los recursos humanos, se ha tenido en cuenta que cada tarea la realiza un rol determinado dentro del equipo y le dedica cierta cantidad de horas. En la tabla 67 se presenta el análisis de costes del proyecto, desglosado en las diferentes fases.

Los costes totales por recursos humanos del proyecto son de 10.868€.

El coste total del proyecto es la suma del coste de recursos humanos más el coste de la infraestructura, que supone un total de 11.123,17€ sin IVA (Impuesto sobre el Valor Añadido). El IVA se corresponde con el 21 % del coste total del proyecto, es decir, 2.335,87€. El coste final una vez añadido el IVA es de 13.459,04€.

13.3. Viabilidad económica

El presupuesto disponible para realizar el proyecto es de 17.000€. Como el coste del proyecto es inferior al límite presupuestario, se constata que el proyecto es viable económicamente.



| Nombre de la tarea | Horas de trabajo [h] | Recurso | Coste[€] |
|---|----------------------|----------------------------|-------------|
| Análisis de requisitos inicial | 52h | - | - |
| Estudio de la problemática | 12h | Analista | 348€ |
| Análisis de la solución actual | 8h | Analista | 232€ |
| Definición de la solución | 8h | Jefe de proyecto | 280€ |
| Estudio de la viabilidad económica | 8h | Jefe de proyecto | 280€ |
| Instalación del entorno de trabajo en local | 16h | Técnico programador | 384€ |
| <i>Posibles desviaciones</i> | <i>8h</i> | <i>Técnico programador</i> | <i>192€</i> |
| Familiarización con el entorno, aprendizaje autónomo e instalación en MN3 | 60h | Técnico programador | 1440€ |
| <i>Posibles desviaciones</i> | <i>8h</i> | <i>Técnico programador</i> | <i>192€</i> |
| Modelo del sistema | 60h | - | - |
| Definición de estructuras y métodos primarios | 10h | Diseñador | 290€ |
| Implementación de estructuras y métodos primarios | 50h | Técnico programador | 1.200€ |
| <i>Posibles desviaciones</i> | <i>8h</i> | <i>Técnico programador</i> | <i>192€</i> |
| Modelo del sistema II | 60h | - | - |
| Definición de métodos secundarios | 10h | Diseñador | 290€ |
| Implementación de métodos secundarios | 20h | Técnico programador | 480€ |
| Modelo Gambia I | 30h | Técnico programador | 720€ |
| Modelo de Gambia II | 56h | Técnico programador | 1.344€ |
| Comparativa y análisis resultados | 60h | - | - |
| Comparativa resultados | 28h | Técnico programador | 672€ |
| Test integrado final | 32h | Tester | 768€ |
| <i>Posibles desviaciones</i> | <i>8h</i> | <i>Técnico programador</i> | <i>192€</i> |
| Memoria del proyecto | 60h | Documentador | 1.440€ |
| Preparación presentación | 20h | Jefe de proyecto | 700€ |

Tabla 67: Análisis coste de proyecto

Una vez realizada la planificación del proyecto, se pueden producir desviaciones que se corregirán según el plan de contingencia presentado anteriormente. Sin embargo, es importante

disponer de mecanismos que permitan detectar desviaciones respecto al coste real del proyecto en relación con el presupuesto establecido inicialmente para su realización.

Con el fin de minimizar estos riesgos, se ha calculado un plan de desviaciones en las fases en las que es más probable que estas ocurran y se ha estimado su coste en horas, para poder confrontarlo con la planificación inicial, el presupuesto del proyecto y poder buscar el motivo de estas y las posibles soluciones.

Sin embargo, gracias a la planificación inicial del proyecto, al plan de contingencia realizado para éste y a las metodologías ágiles empleadas, se espera que estas desviaciones no representen un alto impacto para el coste final de la realización del proyecto.

Los costes asociados a las posibles desviaciones asociadas al proyecto se contabilizan en 768€ sin IVA y 921,96€ con IVA.

Si se suman las posibles desviaciones a la estimación total de las horas de trabajo del proyecto, el coste final del proyecto sería de 11.891,17€ sin IVA y 14.381€ con IVA. Como este importe sigue siendo inferior que el límite presupuestario establecido, 17.000€, el proyecto es viable económicamente incluso en el peor escenario contemplado.

13.4. Sostenibilidad

Para valorar el impacto en la componente de sostenibilidad de este proyecto, se desglosa en 3 dimensiones de estudio: económica, social y ambiental, aplicadas a tres fases del proyecto: planificación, resultados y riesgos.

La valoración de la sostenibilidad de este proyecto, se ha realizado a través de asignar una puntuación del impacto de una serie de preguntas provenientes del documento de la asignatura GEP “El informe de sostenibilidad del proyecto”, y sus correspondientes respuestas para cada dimensión. En la tabla 68 se presenta el resultado de este análisis de manera resumida.

| ¿Sostenible? | Económica | Social | Ambiental | Total |
|-------------------------|-----------------------------------|---------------------------|----------------------|-------|
| Planificación | Viabilidad económica | Mejora en calidad de vida | Análisis de recursos | |
| Valoración | 10 | 8 | 6 | 24 |
| Resultados | Coste final vs previsión | Impacto en entorno social | Consumo de recursos | |
| Valoración | 10 | 10 | 7 | 27 |
| Riesgos | Adaptación a cambios de escenario | Daños sociales | Daños ambientales | |
| Valoración | 0 | 0 | 0 | 0 |
| Valoración Total | 51 | | | |

Tabla 68: Matriz de sostenibilidad del proyecto

A continuación se presenta una explicación detallada sobre el análisis de cada dimensión de la matriz de sostenibilidad del proyecto.

13.4.1. Dimensión económica

Después de haber realizado una valoración exhaustiva de los recursos necesarios para la realización del proyecto, se ha demostrado que es económicamente viable, sostenible y competitivo, incluso en el peor de los escenarios contemplados. Este último factor representa un valor añadido importante en el caso que, posteriormente, se quiera continuar el proyecto.

Como valor añadido de este proyecto, se destaca que contribuye con un proyecto realizado en el BSC (Barcelona Super Computing Center) en colaboración con el InLab FIB, permitiéndole mejorar sus necesidades. También contribuye con un proyecto realizado en la Università di Sapienza.

Gracias a los resultados obtenidos en este proyecto, está previsto que se continúe desarrollando y que siga la colaboración entre los grupos de investigación implicados, para continuar mejorando las herramientas.

13.4.2. Dimensión social

El proyecto tiene una fuerte componente social: su finalidad es realizar un análisis en profundidad de un simulador socio-demográfico e intentar aumentar su rendimiento permitiendo, entre otras cosas, poder mejorar las planificaciones sociales y poder estimar mejor y más rápidamente, los recursos de una región determinada.

Debido al aumento de la velocidad y a la mejora de rendimiento del simulador, se prevé que las personas que pueden tomar decisiones de carácter social en el ámbito de políticas migratorias y sociales, se vean beneficiadas con su desarrollo y tengan más y mejor información a su disposición de la previsión de los cambios sociales y migratorios de la gente en determinadas regiones, pudiendo tomar mejores decisiones para ayudar a la gente.

Este proyecto se sitúa en el ámbito de la investigación universitaria, un sector importante para el desarrollo social y muy necesario para impulsar el desarrollo del país. Este proyecto surge debido a una necesidad real de los principales equipos de investigación implicados.

Además, la contribución con otros grupos de investigación de otras universidades europeas ha abierto una línea de investigación que representa una importante contribución a la investigación y al conocimiento.

13.4.3. Dimensión ambiental

En esta dimensión, el proyecto no tiene un fuerte impacto, debido a que se trata de un proyecto de computación paralela, un sector que en estos momentos representa un importante impacto medioambiental. Los principales costes que tiene el proyecto en este sector, corresponden



a costes indirectos asociados al consumo de luz. Sin embargo, debido a que este proyecto surge por la necesidad de optimizar el simulador socio-demográfico paralelo Yades, reduciendo las horas de cómputo que realiza, obtiene alguna puntuación extra en esta dimensión.

Además, la realización de este proyecto ha permitido que el simulador Yades pueda ser empleado en arquitectura multicore, dando la posibilidad de utilizarlo no solo en supercomputadores y clústeres, sino también en equipos de sobremesa que cuenten con esta tecnología. Además, se ha mejorado el rendimiento de la aplicación en secuencial, haciendo posible realizar simulaciones con un único hilo de proceso. Estas mejoras representan un importante avance en la componente medioambiental del simulador, ya que le permiten funcionar en arquitecturas que representan un menor impacto ambiental.

Además este proyecto se basa en software de código abierto, por lo que podrá ser reaprovechado en otros proyectos que se quieran realizar a posteriori.

14. Leyes y regulación

En este proyecto se trabaja con datos sobre la población gambiana residente en España. Estos datos han sido proporcionados por el Instituto Nacional de Estadística (INE) para la tesis doctoral [Montañola-Sales, 2014], y otros han sido extraídos de algunas líneas de investigación sobre la migración de estos grupos de población al territorio español, también para esta tesis. Estos datos están publicados en revistas científicas y bases de datos gubernamentales por lo que son de libre acceso para proyectos educativos y de investigación.

Los datos que constan en la tesis doctoral [Montañola-Sales, 2014], así como todo el procedimiento para la construcción y especificación de Yades, pueden ser consultados y aplicados a investigaciones posteriores de manera gratuita, siempre que se cite su procedencia y con consentimiento del autor y no se saque provecho económico.

El código del simulador Yades ha sido empleado y modificado para la realización de este proyecto con el consentimiento explícito del autor, Cristina Montañola Sales [Montañola-Sales, 2014]. Asimismo, el código de la herramienta de simulación paralela ROOT-Sim, ha sido empleado y modificado con el consentimiento explícito del autor, Alessandro Pellegrini [Pellegrini, 2013-2014].

15. Conclusión

En este proyecto se ha realizado un análisis de en profundidad del simulador paralelo para simulaciones de dinámicas socio-demográficas, Yades, con el objetivo de buscar solución a los problemas de rendimiento que presenta actualmente debido a la librería de simulación paralela que emplea, μsik (ver sección 1.2 para más detalle). Gracias a este análisis, se ha realizado una búsqueda de herramientas de simulación paralela por eventos discretos (ver sección 2.2.1) que proporcionen las mismas funcionalidades que la librería actual, con el propósito que su uso pueda representar una mejora en la escalabilidad de Yades. La herramienta escogida fue la librería de simulación paralela por eventos discretos de propósito general, ROOT-Sim.

A continuación, se decidió integrar Yades con ROOT-Sim para intentar resolver los problemas de escalabilidad que presentaba el simulador con su implementación original con la librería μsik . Esta integración constó de varias fases en las que fue necesario redefinir algunos componentes de Yades, pero asegurando que conservara su comportamiento inicial. La solución software final pasó por un proceso de validación y experimentación en el que se pudo constatar que se realizó una optimización de la versión inicial de Yades y que aporta nuevas vías a la actual línea de investigación que representa.

Esta optimización del simulador Yades tiene como consecuencia directa la mejora del tiempo de ejecución del simulador y la reducción de las comunicaciones. Como se vio en el apartado 12 esta versión permite ejecutar el simulador en escenarios más grandes, con una mayor cantidad de población. Además permite simular condiciones extremas para el caso de las migraciones, como por ejemplo en un escenario de guerra o desastres naturales, en los que más del 50% de la población de una región se puede ver obligada a migrar hacia otras regiones cercanas. También permite ejecutar la aplicación en modo secuencial (ver sección 12.2.1) obteniendo buenos resultados de rendimiento.

Este trabajo no sólo aporta valor por sí mismo al cliente, sino que además ha permitido hacer un estudio en profundidad del simulador Yades, y se han detectado y solucionado algunos puntos claves para su correcto funcionamiento. Así, gracias al test llevado a cabo en la aplicación se han podido solucionar algunos problemas que presentaba el simulador antes de realizar este proyecto. Se corrigieron algunos errores importantes que han permitido que los resultados se adecuen más a los modelos de los que se tienen datos empíricos.

La realización de este proyecto también ha representado un valor añadido para el grupo de investigación de ROOT-Sim ya que ha permitido realizar una prueba de concepto a gran escala de la librería y mejorar algunos errores de desarrollo que no se habían presentado en pruebas anteriores. Incluso ha permitido plantear nuevas funcionalidades para la librería y mejorar la capacidad de explotación de sistemas de memoria compartida para aumentar el rendimiento de las aplicaciones.

Además, con la realización de este proyecto se han desarrollado aptitudes para el uso de



código de terceros, una aptitud difícil de adquirir y que suele tener una problemática asociada.

A nivel personal este proyecto ha representado un importante reto, ya que se sitúa en el campo de la investigación universitaria y ha representado una introducción a este mundo. Esta investigación tiene carácter interdisciplinar y ha requerido el uso de nuevas metodologías y técnicas de trabajo nuevas para mí.

Así mismo, una dificultad añadida de este trabajo ha sido el realizarlo de forma conjunta con otro grupo de investigación en Italia.

Como se mencionó anteriormente, el software resultante de este proyecto, abre nuevas líneas de investigación en esta área y posibles colaboraciones futuras entre los grupos de investigación implicados. Además aporta un valor añadido al InLab ya que fue desarrollado aquí y se podrá usar, tanto el proyecto en si como todo el conocimiento adquirido, para futuros proyectos.

16. Trabajo futuro

Como se vio en el capítulo 12, la escalabilidad de la integración de Yades con ROOT-Sim es prometedora aunque no es ideal. Sin embargo, la contribución de este trabajo es dotar a Yades de los mecanismos para aprovechar el esquema de memoria compartida que proporciona ROOT-Sim del que hasta ahora carecía. Por lo tanto, el trabajo futuro principal se plantea alrededor de mejorar el rendimiento que se ha obtenido hasta ahora.

Se ha observado que el tiempo de ejecución es menor a medida que se aumenta el número de procesos lógicos que hay que procesar en cada thread o procesador. Sin embargo, originalmente en Yades con su implementación con la librería *μsik*, se conceptualizaba tanto regiones como familias como procesos lógicos (LP). Esto se ha corregido con ROOT-Sim y los procesos lógicos han pasado a ser sólo regiones y las familias estructuras de datos dentro de cada región. El rendimiento que este cambio implica es notable pero es necesario un cambio de esquema en la implementación.

En Yades-*μsik* las regiones tenían una gran cantidad de población porque se usaba todo un procesador para cada región. Por el contrario, la nueva implementación permite simular muchas regiones en un solo procesador. Como consecuencia, el esquema empleado en la distribución del espacio debe cambiar para aprovechar esta característica. Una posibilidad es gestionar el espacio como un grid, cuyas celdas serían las regiones. Hasta ahora, esta conceptualización del espacio no era posible por las restricciones de la implementación. Como consecuencia directa de este proyecto, en los próximos meses se estudiará la forma de implementar estos cambios que pueden ser de mucha utilidad para un proyecto de simulación de la transmisión de la tuberculosis que se están desarrollando actualmente en el InLab-FIB.

Además, ROOT-Sim cuenta con un módulo para hacer balanceo de carga de sus aplicaciones. De cara al futuro sería interesante poder aplicar este módulo a las diferentes LP y realizar un estudio y comparación del rendimiento entre la nueva implementación de Yades con ROOT-Sim y la antigua implementación con *μsik*.

A pesar de que la actual implementación de Yades con ROOT-Sim aprovecha una arquitectura de memoria compartida, no es capaz de explotar la de un cluster o supercomputador. Esta es una limitación impuesta por ROOT-Sim que sólo permite ejecutar la aplicación en paralelo en un mismo nodo (o máquina) y aprovechar la potencia de los cores que ésta tiene. Como trabajo futuro se podría ampliar la versión de ROOT-Sim introduciendo comunicaciones MPI entre nodos, para permitir que se pueda ejecutar en una arquitectura como la de Marenostrum 3 que cuenta con 3.056 nodos.

También se deja como trabajo futuro aplicar algunas mejoras al simulador Yades, como puede ser desarrollar algunas técnicas de optimización para reutilizar o redirigir los eventos que se pierden en el momento que una familia migra a otra región. Para esto sería necesario trabajar con la cola de eventos de ROOT-Sim para poder redireccionar los eventos de familias que han



migrado de una región (LP) a otra.

Actualmente las migraciones de las unidades familiares se implementan usando un modelo simple en el que la decisión de hacia donde migrar es tomada siguiendo un muestreo aleatorio, proporcionado por la topología Mesh de ROOT-Sim. Sin embargo, sería interesante desarrollar un modelo complejo en el que se implemente una topología basada en factores específicos de la vida del individuo, de las regiones origen y destino, de su familia, económicos, etc. que se traduzca a probabilidades de migrar hacia un sitio u otro, para aplicar a las migraciones de las unidades familiares, y cambiar la actual topología Random.

Dentro de la segunda fase de desarrollo de este proyecto, se encuentra la posibilidad de aplicar patrones al simulador con la finalidad de hacer el código más cambiabile e independiente de las plataformas con las que se trabaja actualmente. Para esta fase futura se podría realizar una versión de Yades aplicando el patrón Adaptador [Gómez Seoane and Franch, 2013] para aportar mayor cambiabilidad e independencia al código del simulador.

17. Competencias técnicas trabajadas

Este proyecto es multidisciplinar y engloba conocimientos de diferentes áreas de investigación. En especial, se centra en el área de la simulación social, empleada para mejorar algunos sectores de la vida cotidiana. La simulación se encuentra dentro de la especialidad de Ingeniería del Software de la FIB.

El proyecto también engloba otros conocimientos previos adquiridos antes de la elección de la especialidad, como por ejemplo conocimientos de paralelismo.

Al inicio del proyecto se determinaron las competencias técnicas de la especialidad de Ingeniería del Software que se trabajarían durante su desarrollo.

A continuación se presentan las competencias técnicas asociadas con este proyecto y la justificación de su cumplimiento.

CES1.1: Desenvolupar, mantenir i avaluar sistemes i serveis software complexos i/o crítics. [En profunditat]

Esta competencia técnica se consigue en este proyecto debido a que se desarrolla un sistema software complejo, sobre todo debido a la dificultad inherente a su contexto, que engloba varios campos de investigación, cada uno de los cuales tiene un alto grado de complejidad por si mismo.

El sistema software desarrollado es complejo ya que la programación para sistemas paralelos representa un alto grado de complejidad por si mismo, debido a todas las particularidades que hay que tener en cuenta y que han sido explicadas en secciones previas. También encierra complejidad el hecho que el sistema sea orientado a eventos, y no una clásica ejecución secuencial.

Por último, el hecho de integrar dos sistemas software ya existentes representa una complejidad añadida.

El trabajo del análisis previo y la definición de requisitos es costoso y es necesaria la colaboración y el trabajo con equipos de investigación externos y otras personas relacionadas y expertas en la materia.

CES1.2: Donar solució a problemes d'integració en funció de les estratègies, dels estàndards i de les tecnologies disponibles. [Bastant]

La solución propuesta integrará dos sistemas de simulación que trabajan en arquitectura paralela, Yades y ROOT-Sim, para mejorar los problemas de rendimiento que presenta Yades actualmente con la librería μ sik.

Por tanto, para desarrollar la solución propuesta es necesario desintegrar dos herramientas actualmente funcionales, e integrar una de ellas con una nueva librería, y que se mantengan todas las funcionalidades previas.

CES1.4: Desenvolupar, mantenir i avaluar serveis i aplicacions distribuïdes amb suport de xarxa. [En profunditat]

La solución propuesta consiste en desarrollar un sistema que se ejecuta en arquitectura paralela, a partir de dos librerías también de arquitectura paralela que funcionan con memoria distribuida.

El sistema en arquitectura paralela tendrá los datos distribuidos en diferentes nodos para que cada uno pueda realizar la tarea asignada, manteniendo comunicaciones entre nodos para poder combinar los resultados obtenidos.

Este proyecto ha permitido optimizar las comunicaciones que realizan las unidades de procesamiento, permitiendo a la aplicación realizar comunicaciones intra-nodo.

CES1.7: Controlar la qualitat i dissenyar proves en la producció de software. [Bastant]

Para la realización de este proyecto se lleva a cabo un control de calidad exhaustivo, mediante diversas fases de test (de compatibilidad, de regresión, de integración, de esfuerzo y de validación) durante y al final del desarrollo, para asegurar que la solución software cumple con los parámetros de calidad establecidos.

Además de que la aplicación pase un control de calidad clásico de software, debe pasar la validación del modelo y de los resultados obtenidos, que deben asemejarse a la realidad para ser considerados válidos.

CES2.1: Definir i gestionar els requisits d'un sistema software. [Bastant]

La definición de los requisitos del sistema software es costosa debido al contexto del proyecto, y a la falta de experiencia en estas diversas áreas al comienzo. A pesar de eso, la definición y la gestión de estos requisitos funcionales y no funcionales se hace con la máxima rigurosidad y precisión, con la colaboración de expertos en el tema.

En este proyecto se ha ampliado la documentación del simulador de dinámicas socio-demográficas para arquitecturas paralelas Yades, que en su documentación inicial no cuenta con una especificación exhaustiva de requisitos del software.



Glosario

A

Acreditación

Certificación oficial que determina si un modelo de simulación es aceptable para un propósito específico.. 137

API

Conjunto de subrutinas, funciones y procedimientos que ofrece una librería de programación para trabajar con ella.. 60

E

Eficiencia

Fracción de tiempo en la cual un elemento procesado está trabajando.. 56

Ejecución Concurrente

Romper el programa en trozos más pequeños, llamados tareas y organizarlos para que se ejecuten simultáneamente, o den esa impresión.. 56

Ejecución Paralela

Uso simultáneo de varios procesadores (CPUs) para ejecutar las tareas identificadas para la ejecución concurrente. Idealmente cada procesador podría recibir $\frac{1}{p}$ partes del programa, reduciendo el tiempo de ejecución por P.. 56

Escalabilidad

Capacidad del sistema de alcanzar una gran cantidad de trabajo a medida que el hardware crece.. 15

Estocástica

Proceso en el que el comportamiento depende de otros factores, por lo que el resultado no siempre es igual, es decir no es determinista.. 14

G

Granularidad fina

Consiste en el empleo de muchos elementos de procesado de poca potencia, en este caso el grado de paralelismo es máximo.. 62

H



Heisenbugs

Errores que desaparecen o cambian su comportamiento en el momento de buscarlos. Término que proviene del físico Werner Heisenberg que dedujo el efecto de observación de la mecánica cuántica, que sugiere que solo por el hecho de observar un sistema de una manera determinada se puede alterar su estado.. 148, 149

L

Latencia

Delay al transmitir un mensaje de un computador a otro.. 16

Locality constraint

Propiedad que establece que los eventos deben ser producidos en su correspondiente secuencia temporal.. 16

M

Memory Swapping

Proceso de desmapeo de una página de la memoria principal para ubicar otra en su lugar necesaria para la ejecución del programa. Este desmapeo implica almacenar la página en disco, realizando un acceso muy costoso, que tiene un impacto muy fuerte en el tiempo de ejecución del orden una o dos ordenes de magnitud.. 15

Micro Kernel

Núcleo de un SO que proporciona un conjunto de llamadas básicas al sistema para implementar servicios básicos como espacios de direcciones, comunicación entre procesos y planificación básica.. 48

Miss de cache

Error de lectura o escritura de un dato en la memoria cache.. 56

Multiprocesamiento genérico

Estos sistemas también son conocidos como UMA (Uniform Memory Access). Son sistemas en los que varias unidades de procesamiento comparten el acceso a memoria, accediendo en igualdad de condiciones a ella. Se componen de varios microprocesadores independientes que se comunican con la memoria a través de un bus compartido.. 62

O

Overhead

Tiempo que se añade debido a la sincronización.. 56



R

Rollback

Operación que devuelve al sistema a un estado previo consistente.. 17

S

Sincronización

Procesamiento que se añade a la ejecución del programa para asegurarse que se satisfacen las dependencias del grafo de dependencias de la aplicación.. 56

Speedup

Métrica que permite evaluar la reducción temporal en la ejecución de un problema resuelto de forma paralela, usando P procesadores, con respecto a su correspondiente versión en secuencial.. 55

Strong scaling

El tamaño del problema permanece fijo, y se aumenta el número de procesadores. Esta medida se usa en programas que toman una gran cantidad de tiempo en ejecutarse. Se considera que un programa escala linealmente si su speedup es igual al número de procesadores usados.. 55, 56

T

Thread

Un hilo es la unidad mínima independiente de instrucciones dentro de un proceso que duplica sólo los recursos esenciales para funcionar. 18

V

Validación

Proceso mediante el cual se determina si el modelo de simulación es una correcta representación del sistema a modelar, para el caso de estudio específico.. 137

Verificación

Proceso mediante el cual se determina si el modelo conceptual del sistema que se pretende definir ha sido correctamente definido.. 137

W



Weak scaling

el tamaño del problema (workload) por procesador permanece constante y se varían otros parámetros, como el número de procesadores a usar. Este tipo de medidas permiten analizar sistemas que requieren una gran cantidad de memoria. Se considera que un programa escala linealmente cuando el tiempo de ejecución permanece constante mientras que el tamaño del problema aumenta proporcionalmente al número de procesadores.. 55, 56



Acrónimos

A

ABM

Agent Base Modeling.. 14, 15

ABS

Agent Base Simulation.. 15

D

DEV

Discrete Event Simulation.. 14–16

H

HPC

High Performance Computing.. 15, 16

L

LP

Logical Processes. 48

M

MPI

Message Passing Interface. 48

S

SMP

Symmetric multiprocessing. 62

Anexos

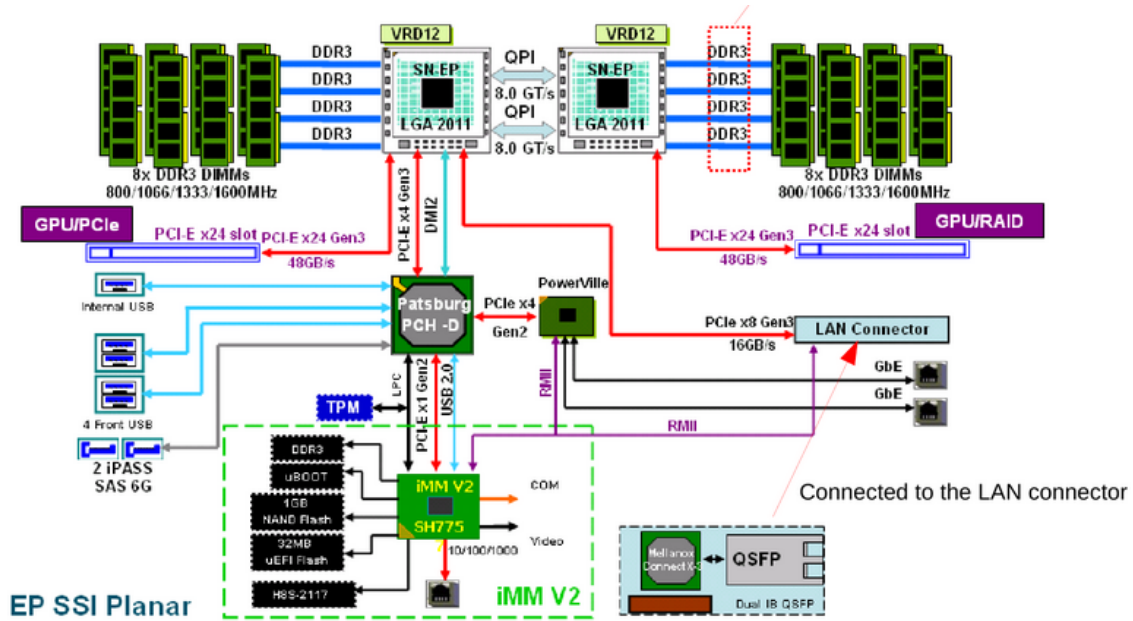


Figura 65: Arquitectura de un nodo de Marenostrum [BSC].

Áreas acumuladas de la distribución T-STUDENT

1. ¿Cómo se usa la tabla de la distribución T-STUDENT para averiguar $t_{1-\alpha/2, \nu}$?

Supongamos un riesgo del 5% (o un nivel de confianza del 95%), $\alpha=0.05$, y grados de libertad $\nu=10$. Utilizaremos $\alpha/2$ ya que dejamos el mismo espacio correspondiente a la región de rechazo por ambos lados. ¿Cuál es el valor, pues, de $t_{0,975,10}^2$? Se busca la intersección y el resultado es **2.228**. Éste es el valor crítico para rechazar la hipótesis alternativa.

| ν | 0,6 | 0,75 | 0,9 | 0,95 | 0,975 | 0,99 | 0,995 | 0,9975 | 0,999 | 0,9995 |
|----------|-------|-------|-------|-------|--------|--------|--------|---------|---------|---------|
| 1 | 0,325 | 1,000 | 3,078 | 6,314 | 12,706 | 31,821 | 63,656 | 127,321 | 318,289 | 636,578 |
| 2 | 0,289 | 0,816 | 1,886 | 2,920 | 4,303 | 6,965 | 9,925 | 14,089 | 22,328 | 31,600 |
| 3 | 0,277 | 0,765 | 1,638 | 2,353 | 3,182 | 4,541 | 5,841 | 7,453 | 10,214 | 12,924 |
| 4 | 0,271 | 0,741 | 1,533 | 2,132 | 2,776 | 3,747 | 4,604 | 5,598 | 7,173 | 8,610 |
| 5 | 0,267 | 0,727 | 1,476 | 2,015 | 2,571 | 3,365 | 4,032 | 4,773 | 5,894 | 6,869 |
| 6 | 0,265 | 0,718 | 1,440 | 1,943 | 2,447 | 3,143 | 3,707 | 4,317 | 5,208 | 5,959 |
| 7 | 0,263 | 0,711 | 1,415 | 1,895 | 2,365 | 2,998 | 3,499 | 4,029 | 4,785 | 5,408 |
| 8 | 0,262 | 0,706 | 1,397 | 1,860 | 2,306 | 2,896 | 3,355 | 3,833 | 4,501 | 5,041 |
| 9 | 0,261 | 0,703 | 1,383 | 1,833 | 2,262 | 2,821 | 3,250 | 3,690 | 4,297 | 4,781 |
| 10 | 0,260 | 0,700 | 1,372 | 1,812 | 2,228 | 2,764 | 3,169 | 3,581 | 4,144 | 4,587 |
| 11 | 0,260 | 0,697 | 1,363 | 1,796 | 2,201 | 2,718 | 3,106 | 3,497 | 4,025 | 4,437 |
| 12 | 0,259 | 0,695 | 1,356 | 1,782 | 2,179 | 2,681 | 3,055 | 3,428 | 3,930 | 4,318 |
| 13 | 0,259 | 0,694 | 1,350 | 1,771 | 2,160 | 2,650 | 3,012 | 3,372 | 3,852 | 4,221 |
| 14 | 0,258 | 0,692 | 1,345 | 1,761 | 2,145 | 2,624 | 2,977 | 3,326 | 3,787 | 4,140 |
| 15 | 0,258 | 0,691 | 1,341 | 1,753 | 2,131 | 2,602 | 2,947 | 3,286 | 3,733 | 4,073 |
| 16 | 0,258 | 0,690 | 1,337 | 1,746 | 2,120 | 2,583 | 2,921 | 3,252 | 3,686 | 4,015 |
| 17 | 0,257 | 0,689 | 1,333 | 1,740 | 2,110 | 2,567 | 2,898 | 3,222 | 3,646 | 3,965 |
| 18 | 0,257 | 0,688 | 1,330 | 1,734 | 2,101 | 2,552 | 2,878 | 3,197 | 3,610 | 3,922 |
| 19 | 0,257 | 0,688 | 1,328 | 1,729 | 2,093 | 2,539 | 2,861 | 3,174 | 3,579 | 3,883 |
| 20 | 0,257 | 0,687 | 1,325 | 1,725 | 2,086 | 2,528 | 2,845 | 3,153 | 3,552 | 3,850 |
| 21 | 0,257 | 0,686 | 1,323 | 1,721 | 2,080 | 2,518 | 2,831 | 3,135 | 3,527 | 3,819 |
| 22 | 0,256 | 0,686 | 1,321 | 1,717 | 2,074 | 2,508 | 2,819 | 3,119 | 3,505 | 3,792 |
| 23 | 0,256 | 0,685 | 1,319 | 1,714 | 2,069 | 2,500 | 2,807 | 3,104 | 3,485 | 3,768 |
| 24 | 0,256 | 0,685 | 1,318 | 1,711 | 2,064 | 2,492 | 2,797 | 3,091 | 3,467 | 3,745 |
| 25 | 0,256 | 0,684 | 1,316 | 1,708 | 2,060 | 2,485 | 2,787 | 3,078 | 3,450 | 3,725 |
| 26 | 0,256 | 0,684 | 1,315 | 1,706 | 2,056 | 2,479 | 2,779 | 3,067 | 3,435 | 3,707 |
| 27 | 0,256 | 0,684 | 1,314 | 1,703 | 2,052 | 2,473 | 2,771 | 3,057 | 3,421 | 3,689 |
| 28 | 0,256 | 0,683 | 1,313 | 1,701 | 2,048 | 2,467 | 2,763 | 3,047 | 3,408 | 3,674 |
| 29 | 0,256 | 0,683 | 1,311 | 1,699 | 2,045 | 2,462 | 2,756 | 3,038 | 3,396 | 3,660 |
| 30 | 0,256 | 0,683 | 1,310 | 1,697 | 2,042 | 2,457 | 2,750 | 3,030 | 3,385 | 3,646 |
| 40 | 0,255 | 0,681 | 1,303 | 1,684 | 2,021 | 2,423 | 2,704 | 2,971 | 3,307 | 3,551 |
| 60 | 0,254 | 0,679 | 1,296 | 1,671 | 2,000 | 2,390 | 2,660 | 2,915 | 3,232 | 3,460 |
| 120 | 0,254 | 0,677 | 1,289 | 1,658 | 1,980 | 2,358 | 2,617 | 2,860 | 3,160 | 3,373 |
| ∞ | 0,253 | 0,674 | 1,282 | 1,645 | 1,960 | 2,326 | 2,576 | 2,807 | 3,090 | 3,290 |

Cálculo del N^o de réplicas para análisis de rendimiento

| Réplica | Tiempo de ejecución (x_i) | $(x_i - \bar{X})^2$ |
|--------------------|-------------------------------|---------------------|
| 1 | 1.046 | 7.29E-6 |
| 2 | 0.971 | 5.22E-3 |
| 3 | 1.057 | 1.87E-4 |
| 4 | 1.097 | 2.88E-3 |
| 5 | 1.119 | 5.73E-3 |
| 6 | 0.846 | 3.89E-2 |
| 7 | 0.949 | 8.89E-3 |
| 8 | 1.098 | 2.99E-3 |
| 9 | 1.189 | 2.12E-2 |
| 10 | 1.061 | 3.13E-4 |
| $\bar{X} = 1.0433$ | | $S^2 = 0.0096$ |

Tabla 69: Valores para cálculo de réplicas

Una vez se han calculado la media muestral (\bar{X}) y la varianza muestral (S^2) en la tabla 69, se calcula la desviación estándar:

$$\frac{S}{\sqrt{n}} = 0.030982092$$

Como se parte de la hipótesis que se tendrán menos de 30 réplicas, se trabaja con la distribución TStudent. Se establece un nivel de confianza del 95 %, $\alpha = 0.05$ y n-1 grados de libertad y se busca la intersección en la tabla $t_{1-\frac{0.05}{2},9} = 2.262$. Con este valor, se calcula el semi intervalo de confianza (h):

$$h = 2.262 * 0.030982092$$

$$h = 0.070081491$$

y se calcula el intervalo:

$$\mu \in (0.973218509, 1.113381491)$$

y se mira si se cumple:

$$1.0433 * 0.05 \leq 0.070081491$$

$$0,052165 \leq 0.070081491$$

Por tanto como esta condición se cumple, se coge como válido el N^o de réplicas = 10.



Referencias

- B. G. Aaby, K. S. Perumalla, and S. K. Seal. Efficient simulation of agent-based models on multi-gpu and multi-core clusters. In *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*, page 29. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2010.
- P. AC. Tema 3: Jerarquía de memorias, 2014. University Lecture.
- M. Ajelli and S. Merler. An individual-based model of hepatitis a transmission. *Journal of theoretical biology*, 259(3):478–488, 2009.
- R. Axtell. Why agents?: on the varied motivations for agent computing in the social sciences. 2000.
- E. Ayguade. Posix threads (pthreads) programming, 2014-2015. University Lecture.
- E. Ayguade and D. Jimenez. Analysis of parallel applications, 2012-2013. University Lecture.
- R. Bagrodia, R. Meyer, M. Takai, Y. Chen, X. Zeng, J. Martin, and H. Y. Song. Parsec: A parallel simulation environment for complex systems. *Computer*, 31(10):77–85, 1998.
- T. Ball, S. Burckhardt, P. de Halleux, M. Musuvathi, and S. Qadeer. Predictable and progressive testing of multithreaded code. *IEEE Software*, 28(3):0075–83, 2011.
- F. C. Billari and A. Prskawetz. *Agent-based computational demography: Using simulation to improve our understanding of demographic behaviour*. Springer Science & Business Media, 2003.
- S. Booth and D. Henry. Verification strategies for high performance computing software.
- F. Bourguignon and A. Spadaro. Microsimulation as a tool for evaluating redistribution policies. *The Journal of Economic Inequality*, 4(1):77–106, 2006.
- B. S. C. C. BSC.
- C. D. Carothers, D. Bauer, and S. Pearce. Ross: A high-performance, low-memory, modular time warp system. *Journal of Parallel and Distributed Computing*, 62(11):1648–1669, 2002.
- M. Cohn. *Agile estimating and planning*. Pearson Education, 2005.
- N. Collier and M. North. *Repast HPC: A platform for large-scale agent-based modeling*. Wiley, 2011.
- N. Collier and M. North. Parallel agent-based simulation with repast for high performance computing. *Simulation*, 2012.



- G. Cordasco, R. De Chiara, V. Scarano, M. Carillo, A. Mancuso, D. Mazzeo, F. Raia, F. Serrapica, C. Spagnuolo, and L. Vicidomini. D-mason: Distributed multi-agent based simulations toolkit. 2011. URL <http://isis.dia.unisa.it/projects/dmason/>.
- S. Das, R. Fujimoto, K. Panesar, D. Allison, and M. Hybinette. Gtw: a time warp system for shared memory multiprocessors. In *Simulation Conference Proceedings, 1994. Winter*, pages 1332–1339. IEEE, 1994.
- P. Fonseca i Casas. Disseny d'experiments en simulació, 2013-2014a. University Lecture.
- P. Fonseca i Casas. Verificació, validació i acreditació, 2013-2014b. University Lecture.
- R. Fujimoto. Distributed simulation systems. In *Simulation Conference, 2003. Proceedings of the 2003 Winter*, volume 1, pages 124–134. IEEE, 2003.
- R. M. Fujimoto. *Parallel and distributed simulation systems*, volume 300. Wiley New York, 2000.
- L. Goes, L. Ramos, and C. Martins. Clustersim: a java-based parallel discrete-event simulation tool for cluster computing. In *Cluster Computing, 2004 IEEE International Conference on*, pages 401–410. IEEE, 2004.
- O. M. Group. Unified modeling language (uml) resource page. <http://www.uml.org/>. [Online; accedido 10-Mayo-2015].
- C. Gómez Seoane and X. Franch. Adapter pattern, 2013. University Lecture.
- V. Jha and R. L. Bagrodia. *A unified framework for conservative and optimistic distributed simulation*, volume 24. ACM, 1994.
- A. Karpov. Testing parallel programs. <http://www.viva64.com/en/a/0031/>. [Online; accedido 27-Mayo-2015].
- W. Kelton and A. Law. *Simulation modeling and analysis*. McGraw Hill Boston, 2000.
- M. Livi Bacci. Introducción a la demografía. *Editorial Ariel. Barcelona (España)*, 1993-2007.
- C. Lozares Colina. La simulación social, ¿una nueva manera de investigar en ciencia social? In *Papers: revista de sociologia*, pages 165–188, 2004.
- D. E. Martin, T. McBrayer, and P. A. Wilsey. Warped: A time warp simulation kernel for analysis and application development. In *2013 46th Hawaii International Conference on System Sciences*, pages 383–383. IEEE Computer Society, 1996.
- H. Merz et al. Measuring parallel scaling performance, sharcnet. https://www.sharcnet.ca/help/index.php/Measuring_Parallel_Scaling_Performance, 2015. [Online; accedido 27-Mayo-2015].



- N. Minar, R. Burkhart, C. Langton, and M. Askenazi. The swarm simulation system: A toolkit for building multi-agent simulations. Institute Santa Fe, 1996.
- C. Montañola Sales, B. Onggo, J. Casanovas-Garcia, J. Cela-Espin, and A. Kaplan-Marcusán. Yades: A parallel simulation tool for demography. *Parallel Computing*, 2015. (submitted).
- C. Montañola-Sales. *Large-scale simulation of population dynamics for socio-demographic analysis*. PhD thesis, Universitat Politècnica de Catalunya - BarcelonaTech, 2014.
- C. Montañola-Sales, X. Rubio-Campillo, J. Casanovas-Garcia, J. Cela-Espín, and A. Kaplan-Marcusán. Large-scale social simulation, dealing with complexity challenges in high performance environments. *Interdisciplinary Applications of Agent-Based Social Simulation and Modeling*, page 106, 2014.
- K. Muller and T. Vignaux. Simpy: Simulating systems in python. *ONLamp.com Python Devcenter*, 2003.
- P. S. Pacheco. *Parallel programming with MPI*. Morgan Kaufmann Publishers Inc., San Francisco, 1997.
- A. Pellegrini. *Techniques for Transparent Parallelization of Discrete Event Simulation Models*. PhD thesis, Sapienza University of Rome, 2013-2014.
- A. Pellegrini and F. Quaglia. The rome optimistic simulator: A tutorial. In *Euro-Par 2013: Parallel Processing Workshops*, pages 501–512. Springer, 2014.
- A. Pellegrini, R. Vitali, and F. Quaglia. The rome optimistic simulator: core internals and programming model. In *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*, pages 96–98. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2011.
- K. S. Perumalla. *μsik—a micro kernel for parallel/distributed simulation*. 2004.
- D. K. Poulsen and P. Yew. Execution-driven tools for parallel simulation of parallel architectures and applications. In *Proceedings of the 1993 ACM/IEEE conference on Supercomputing*, pages 860–869. ACM, 1993.
- R. Reuillon, M. Leclaire, and S. Rey-Coyrehourcq. Openmole, a workflow engine specifically tailored for the distributed exploration of simulation models. *Future Generation Computer Systems*, 29(8):1981–1990, 2013.
- P. Richmond, D. Walker, S. Coakley, and D. Romano. High performance cellular level agent-based simulation with flame for the gpu. *Briefings in bioinformatics*, 11(3):334–347, 2010.
- G. Rincón, M. Álvarez, M. Pérez, and S. Hernández. Modelo de calidad (mosca+) para evaluar software de simulación de eventos discretos. In *Proceedings of the IDEAS*, pages 167–177, 2003.



- X. Rubio-Campillo. Pandora: A versatile agent-based modelling platform for social simulation.
- P. Siebers, C. M. Macal, J. Garnett, D. Buxton, and M. Pidd. Discrete-event simulation is dead, long live agent-based simulation! *Journal of Simulation*, 4(3):204–210, 2010.
- J. Steinman. Speedes: Synchronous parallel environment for emulation and discrete event simulation. 1991.
- J. Steinman. Speedes-a multiple-synchronization environment for parallel discrete-event simulation. *International Journal in Computer Simulation;(United States)*, 2, 1992.
- S. Thulasidasan, L. Kroc, and S. Eidenbenz. Developing parallel discrete event simulations in python: First results and user experiences with the simx library. Technical report, Technical Report LA-UR-12-26739, 2012.
- I. J. Timm and D. Pawlaszczyk. Large scale multiagent simulation on the grid. In *Cluster Computing and the Grid, 2005. CCGrid 2005. IEEE International Symposium on*, volume 1, pages 334–341. IEEE, 2005.
- S. Tisue and U. Wilensky. Netlogo: A simple environment for modeling complexity. In *International conference on complex systems*, pages 16–21, 2004.
- A. Varga. The omnet++ discrete event simulation system. In *Proceedings of the European simulation multiconference (ESM'2001)*, volume 9, page 65. sn, 2001.
- Varios. The agile manifesto. <http://www.agilealliance.org/the-alliance/the-agile-manifesto/>, a. [Online; accedido 28-Mayo-2015].
- Varios. Uso del ordenador portátil vs ordenador de sobremesa. <http://www.rsc2.es/fichas/portatil.htm>, b. [Online; accedido 03-Marzo-2015].
- Varios. Walking skeleton, alistair cockburn. <http://alistair.cockburn.us/Walking+skeleton>, c. [Online; accedido 25-Mayo-2015].
- Varios. Agencia Tributaria. Sociedades. http://www.agenciatributaria.es/AEAT.internet/Inicio_es_ES/_Configuracion/_top_/Ayuda/Manuales__Folletos_y_Videos/Manuales_practicos/Sociedades/Sociedades.shtml, 2013. [Online; accedido 03-Marzo-2015].
- R. Vitali, A. Pellegrini, and F. Quaglia. Towards symmetric multi-threaded optimistic simulation kernels. In *Principles of Advanced and Distributed Simulation (PADS), 2012 ACM/IEEE/SCS 26th Workshop on*, pages 211–220. IEEE, 2012.
- N. Wagner and V. Agrawal. An agent-based simulation system for concert venue crowd evacuation modeling in the presence of a fire disaster. *Expert Systems with Applications*, 41(6): 2807–2815, 2014.



- C. D. Yang and L. Pollock. The challenges in automated testing of multithreaded programs. In *The 14th International Conference on Testing Computer Software*, pages 157–166. Citeseer, 1997.
- G. Yaun, C. D. Carothers, and S. Kalyanaraman. Large-scale tcp models using optimistic parallel simulation. In *Proceedings of the seventeenth workshop on Parallel and distributed simulation*, page 153. IEEE Computer Society, 2003.