UNIVERSIDAD COMPLUTENSE DE MADRID

FACULTAD DE INFORMÁTICA
Departamento de Sistemas Informáticos y Computación

MARCOS TEMPORALES Y PROBABILÍSTICOS PARA TESTING FORMAL.

MEMORIA PARA OPTAR AL GRADO DE DOCTOR

PRESENTADA POR

Mercedes García Merayo

Bajo la dirección de los doctores

Manuel Núñez García
Roberto M. Hierons

Madrid, 2010

- **ISBN: 978-84-692-7622-8**

Marcos temporales y probabilísticos para testing formal

TESIS DOCTORAL

Mercedes García Merayo
Departamento de Sistemas Informáticos y Computación
Universidad Complutense de Madrid

Directores: Manuel Núñez García y Robert M. Hierons

Publicaciones

Esta tesis doctoral se presenta en *formato publicaciones*, de acuerdo con el apartado 4.4 de la Normativa de desarrollo de los artículos 11, 12, 13 y 14 del real decreto 56/2005, de 21 de enero, por el que se regulan los estudios universitarios oficiales de postgrado de la Universidad Complutense (Aprobado en Consejo de Gobierno con fecha 13 de junio de 2005, BOUC, 5 de julio de 2005). Dichas publicaciones recogen todos los resultados que han sido obtenidos en los diferentes trabajos de investigación desarrollados con el fin de alcanzar el objetivo fijado para la realización de la tesis. A continuación se relacionan los artículos que integran la tesis agrupados en tres bloques, teniendo en cuenta sus diferentes contenidos temáticos: Testing de sistemas temporales, Testing de sistemas estocásticos, y Testing de sistemas distribuidos.

I) Testing de Sistemas Temporales

a) M.G. Merayo, M. Núñez, and I. Rodríguez. Formal testing from timed finite state machines. *Computer Networks*, 52(2):432–460, 2008.

b) M.G. Merayo, M. Núñez, and I. Rodríguez. Formal testing of systems presenting soft and hard deadlines. In *2nd IPM Int. Symposium on Fundamentals of Software Engineering, FSEN'07, LNCS 4767*, pages 160–174. Springer, 2007.

c) M.G. Merayo, M. Núñez, and I. Rodríguez. Extending EFSMs to specify and test timed systems with action durations and timeouts. *IEEE Transactions on Computers*, 57(6):835–848, 2008.

d) M.G. Merayo, M. Núñez, and I. Rodríguez. Generation of optimal finite test suites for timed systems. In *1st IEEE & IFIP Int. Symposium on Theoretical Aspects of Software Engineering, TASE'07*, pages 149–158. IEEE Computer Society Pres, 2007.

e) M.G. Merayo, M. Núñez, and I. Rodríguez. A brief introduction to $\mathcal{THOTL}$. In *5th Int. Symposium on Automated Technology for Verification and Analysis, ATVA'07, LNCS 4762*, pages 501–510. Springer, 2007.

f) M.G. Merayo, M. Núñez, and I. Rodríguez. $\mathcal{THOTL}$: A timed extension of $\mathcal{HOTL}$. In *Joint 20th IFIP TC6/WG6.1 Int. Conf. on Testing of Software and Communicating Systems, TestCom'08, and 8th Int. Workshop on Formal Approaches to Software Testing, FATES'08, LNCS 5047*, pages 86–102. Springer, 2008.

g) M.G. Merayo, R.M. Hierons, and M. Núñez. Extending stream X-machines to specify and test systems with timeouts. In *6th IEEE Int. Conf. on Software Engineering and Formal Methods, SEFM'08*, pages 201–210. IEEE Computer Society Press, 2008.

II) Testing de Sistemas Estocásticos

a) M.G. Merayo, M. Núñez, and I. Rodríguez. Implementation relations for stochastic finite state machines. In *3rd European Performance Engineering Workshop, EPEW'06, LNCS 3964*, pages 123–137. Springer, 2006.

b) M.G. Merayo, M. Núñez, and I. Rodríguez. Testing finite state machines presenting stochastic time and timeouts. In *4th European Performance Engineering Workshop, EPEW'07, LNCS 4748*, pages 97–111. Springer, 2007.

c) M.G. Merayo and M. Núñez. Testing conformance on stochastic stream X-machines. In *5th IEEE Int. Conf. on Software Engineering and Formal Methods, SEFM'07*, pages 227–236. IEEE Computer Society Press, 2007.

d) R.M. Hierons, M.G. Merayo, and M. Núñez. Testing from a stochastic timed system with a fault model. *Journal of Logic and Algebraic Programming*, 78:98–115, 2009.

e) R.M. Hierons and M.G. Merayo. Mutation testing from probabilistic and stochastic finite state machines. *Journal of Systems and Software*, 82(11):1804–1818, 2009.

III) Testing de Arquitectura Distribuida

a) R.M. Hierons, M.G. Merayo, and M. Núñez. Implementation relations for the distributed test architecture. In *Joint 20th IFIP TC6/WG6.1 Int. Conf. on Testing of Software and Communicating Systems, TestCom'08, and 8th Int. Workshop on Formal Approaches to Software Testing, FATES'08, LNCS 5047*, pages 200–215. Springer, 2008.

b) R.M. Hierons, M.G. Merayo, and M. Núñez. Controllable test cases for the distributed test architecture. In *6th Int. Symposium on Automated Technology for Verification and Analysis, ATVA'08, LNCS 5311)*, pages 201–215. Springer, 2008.

Resumen

Los métodos formales son técnicas con base matemática que se utilizan tanto para el diseño y análisis de sistemas como para la evaluación de su corrección. El uso de métodos formales es especialmente relevante en sistemas en los que es importante asegurar que durante el proceso de desarrollo no se han producido errores. La representación formal de estos sistemas permite un análisis riguroso de sus propiedades. En particular, permite establecer la corrección del sistema con respecto a su especificación y, por tanto, asegurar su calidad. El método analítico de mayor aplicación en entornos industriales orientado a dicho objetivo es el *testing*.

Inicialmente, los métodos de testing estaban orientados al chequeo de aspectos cualitativos del sistema. Sin embargo, una gran parte de los sistemas desarrollados requieren considerar no sólo las acciones que se pueden ejecutar sino también los aspectos cuantitativos asociados con dichas acciones. Entre estas cabe destacar tanto condiciones temporales como probabilísticas, que juegan un papel decisivo a la hora de establecer la corrección de los sistemas. Un primer paso para poder realizar testing formal de sistemas que presentan este tipo de restricciones es la extensión de los lenguajes de especificación con elementos que permitan expresar dichas propiedades. Así mismo, las nociones de corrección deben adecuarse para tener en cuenta la dimensión temporal y/o probabilística. De un modo similar, los algoritmos de generación de tests deben ser adaptados para tratar con dichos requerimientos.

El principal objetivo de esta tesis es la extensión de los métodos formales utilizados en las metodologías para el *testing* de sistemas, de modo que estos puedan ser aplicados a sistemas con restricciones temporales referentes al tiempo consumido por las acciones, el tiempo de espera del sistema para recibir una reacción del entorno, la probabilidad de que una acción tenga lugar, o la probabilidad de que una acción consuma una determinada cantidad de tiempo en su ejecución. Junto a ello, la aplicación de tests para la comprobación de dichas propiedades y la obtención de diagnósticos respecto a la corrección de los sistemas, es un objetivo prioritario de nuestro trabajo.

Agradecimientos

Gracias Fernando, por animarme a iniciar una nueva etapa en mi vida, tanto profesional como personal. Sin ti, esto no hubiera sido posible.

Mi más profundo agradecimiento a los profesores Manuel Núñez García y Robert M. Hierons por su ayuda y apoyo, así como por la confianza que depositaron en mi. También quisiera mostrar mi gratitud a los compañeros que me acompañaron, ayudaron y aconsejaron durante todo este tiempo, especialmente a Ismael con el que tantos *deadlines* y discusiones he compartido.

Cuando quieres realmente una cosa, todo el universo conspira para ayudarte a conseguirla.

Paulo Coelho

Thank you Fernando, for encouraging me to initiate a new stage in my life, both professional and personal. Without you, this would have not been possible.

My deepest gratefulness to Professors Manuel Núñez García and Robert M. Hierons for their help and support, as well as for the trust they put in me. Also, I want to show my gratitude to my colleagues that helped and advised me during all this time, especially Ismael with whom I have shared so many deadlines and discussions.

When you want something, all the universe conspires to help you to achieve it.

Paulo Coelho

Índice general

Capítulo 1

Introducción

Los *métodos formales* hacen referencia a técnicas matemáticas para la especificación, desarrollo y verificación de sistemas software y hardware. La aplicación de los métodos formales es especialmente relevante en sistemas en los que, debido a razones de seguridad, es necesario establecer que durante el proceso de desarrollo no se han producido errores. De hecho, su uso en las etapas iniciales del proceso de desarrollo, durante el establecimiento de los requerimientos y su especificación, incrementa su efectividad. No obstante, su aplicabilidad no está restringida a estos niveles, pudiendo ser usados a lo largo de todo el ciclo de desarrollo.

La representación formal de los sistemas permite un análisis riguroso de sus propiedades. En particular, permite establecer la corrección del sistema final respecto a la especificación, el cumplimiento de las condiciones requeridas para el mismo, la equivalencia semántica con otros sistemas, el nivel de preferencia de un sistema respecto a otro en base a un determinado criterio, la existencia de posibles comportamientos incorrectos, etc.

En lo referente a las técnicas formales para el establecimiento de la corrección de un sistema respecto a una especificación, las metodologías de *testing formal* han jugado un papel muy relevante. La aplicación de dichas técnicas requiere la identificación de los aspectos críticos del sistema, esto es, aquellos aspectos que marcan la diferencia entre los comportamientos correctos e incorrectos. Inicialmente, las técnicas de testing estaban enfocadas al análisis del comportamiento funcional de los sistemas, es decir, a determinar, por una parte, si el sistema testeado realizaba las acciones requeridas, y por otra a garantizar que no realizaba las acciones no permitidas. Sin embargo, en el caso de los sistemas de tiempo real adquieren tanta relevancia los aspectos cuantitativos como los cualitativos: es tan importante verificar que los sistemas *hacen lo que deben hacer*, como que *lo hacen cuando y como deben hacerlo*. Debido a ello, durante los últimos años, las técnicas de testing formal se han

orientado también hacia el tratamiento de las propiedades no funcionales de los sistemas, como por ejemplo la probabilidad de que se realice una acción o el tiempo consumido por el sistema para realizarla, ya que estos aspectos pueden ser críticos en dichos sistemas.

Un primer paso para poder realizar testing formal de sistemas que presentan este tipo de restricciones es la extensión de los lenguajes de especificación con elementos que permitan expresar dichas propiedades. Así mismo, las nociones de corrección de las implementaciones, expresadas mediante relaciones de conformidad o equivalencia, deben adecuarse para tener en cuenta la dimensión temporal y/o probabilística. También los algoritmos diseñados para realizar la generación de tests deben ser adaptados para tratar con dichos requerimientos.

El principal objetivo de esta tesis es la extensión de los métodos formales utilizados en las metodologías para el testing de sistemas, de modo que estos puedan ser aplicados cuando dichos sistemas presenten restricciones referentes al tiempo consumido por las acciones, el tiempo de espera del sistema para recibir una reacción del entorno, la probabilidad de que una acción tenga lugar o la probabilidad de que una acción consuma una cantidad de tiempo determinada para su ejecución. Otro objetivo prioritario es la aplicación de tests para el chequeo de dichas propiedades y la obtención de diagnósticos respecto a la corrección de los sistemas.

Esta tesis doctoral se presenta en *formato publicaciones*, de acuerdo con el apartado 4.4 de la Normativa de desarrollo de los artículos 11, 12, 13 y 14 del real decreto 56/2005, de 21 de enero, por el que se regulan los estudios universitarios oficiales de postgrado de la Universidad Complutense (Aprobado en Consejo de Gobierno con fecha 13 de junio de 2005, BOUC, 5 de julio de 2005). Dichas publicaciones recogen todos los resultados que han sido obtenidos en los diferentes trabajos de investigación desarrollados con el fin de alcanzar el objetivo fijado para la realización de la tesis. Todos ellos han sido refrendados con su publicación en diferentes congresos y revistas de ámbito internacional. Contando por tanto con la subsiguiente *"evaluación inter pares"* y habiendo salido victoriosos de los correspondientes procesos de selección, e general bastante exigentes.

Los artículos que integran la tesis se presentan agrupados en tres grandes bloques, teniendo en cuenta sus diferentes contenidos temáticos: Testing de sistemas temporales, Testing de sistemas estocásticos, y Testing de sistemas distribuidos. A pesar de que haya sido posible la citada clasificación, no quiere decir que no exista una innegable linea común integradora, que se corresponde con la propuesta de métodos noveles a nuevas extensiones hasta el momento no exploradas en este sentido para aplicar las técnicas de testing formal.

1.1. Discusión integradora y objetivos

En esta sección se presenta una *discusión integradora* de las publicaciones que constituyen esta tesis.

El primer bloque, correspondiente a testing de sistemas temporales, integra las siguientes contribuciones: *Formal testing from timed finite state machines* [MNR08c], *Formal Testing of Systems Presenting Soft and Hard Deadlines* [MNR07b], *Extending EFSMs to Specify and Test Timed Systems with Action Duration and Time-Outs* [MNR08b], *Generation of optimal finite test suites for timed systems* [MNR07c], *A Brief Introduction to $\mathcal{THOTL}$* [MNR07a], *$\mathcal{THOTL}$: A Timed Extension of $\mathcal{HOTL}$* [MNR08a] y *Extending Stream X-Machines to specify and test systems with timeouts* [MHN08].

Estas publicaciones tienen como objetivo común la integración de requerimientos temporales en diferentes formalismos de especificación y la generación de conjuntos de tests completos que permitan establecer la corrección de las correspondientes implementaciones con respecto a sus especificaciones.

Durante los últimos años se han propuesto numerosas técnicas de testing formal para ser aplicadas en sistemas que presenten restricciones temporales (e.g. [CL97, HNTC99, UFSA99, SVD01, EDK02, ED03, KT05, BB05]). Usualmente, estas metodologías presentan una noción de tiempo *determinista*, es decir, los requerimientos temporales expresan condiciones de la forma "después/antes de t unidades de tiempo". Aunque la inclusión de tiempo en los formalismos permite dar una descripción más precisa del sistema que debe ser implementado, en muchas ocasiones, una noción determinista de tiempo no es adecuada para describir sistemas donde las restricciones temporales deben expresarse mediante condiciones como "la probabilidad de que esta acción se ejecute antes de t unidades de tiempo es p." Esta limitación ha dado lugar al desarrollo de técnicas que se ajustan a las diferentes nociones temporales que se pueden considerar ya que, desgraciadamente, en la mayoría de los casos las técnicas que consideran restricciones temporales en un dominio especifico no pueden ser adaptadas de un modo sencillo para ser aplicadas a sistemas donde se utilizan otros dominios temporales. Por tanto, sería deseable disponer de un marco global donde el diseñador pudiera elegir, con pequeñas variaciones, la noción temporal que desea utilizar para especificar su sistema.

En la primera publicación incluida en este bloque [MNR08c], se presenta un marco unificado, basado en máquinas de estados finitos [Moo56], para especificar y testear sistemas donde los requerimientos temporales pueden ser expresados en tres dominios diferentes: *Tiempos fijos, variables aleatorias e intervalos de tiempo.* Intuitivamente, las transiciones en las máquinas de estados finitos clásicas indican que si la máquina está en un estado s y

recibe una entrada i producirá una salida o y cambiará al estado s'. Una notación adecuada para representar dicha transición es $s \xrightarrow{i/o} s'$. Si consideramos una extensión temporal de las máquinas de estados finitos, transiciones de la forma $s \xrightarrow{i/o}_d s'$ indican que el tiempo transcurrido desde que se recibe la entrada i y se produce la salida o viene dada por d, donde d pertenece a un determinado dominio temporal.

Inicialmente el trabajo presentado en [MNR08c] se restringe a sistemas observables no-deterministas. Un sistema es *observable no-determinista* si no presenta dos transiciones tales como $s \xrightarrow{i/o}_{d_1} s_1$ y $s \xrightarrow{i/o}_{d_2} s_2$. Sin embargo, si se permite la coexistencia de transiciones de la forma $s \xrightarrow{i/o_1}_{d_1} s_1$ y $s \xrightarrow{i/o_2}_{d_2} s_2$, siempre que $o_1 \neq o_2$. En la última parte del trabajo, se elimina la restricción de la observabilidad y el marco es extendido para tratar sistemas no-deterministas.

Para definir apropiadamente como testear una implementación es necesario establecer *que significa que una implementación sea correcta con respecto a una especificación.* En el caso de modelos deterministas, la noción estándar de corrección es la *equivalencia.* Sin embargo, en el caso de que la especificación sea no determinista es más apropiado utilizar una noción de corrección más débil, usualmente denominada *conformidad.* En este caso, la corrección de una implementación respecto a una especificación se establece en términos de relaciones entre los comportamientos de ambas. En este trabajo, se presentan diferentes *relaciones de conformidad* relativas tanto a los aspectos funcionales como a los temporales de los sistemas. Inicialmente se propone una relación donde el tiempo no se considera, relativa tan solo al comportamiento funcional. La idea intuitiva es que el comportamiento de la implementación está restringido al comportamiento establecido en la especificación sólo para aquellas entradas consideradas en la misma; en el resto de los casos, el comportamiento de la implementación no está condicionado. Adicionalmente, se proponen varias relaciones de conformidad temporales de acuerdo con la interpretación de lo que es una *buena* implementación para una especificación dada y los diferentes dominios de tiempo considerados. Todas ellas exigen que la implementación sea correcta respecto a la especificación desde el punto de vista funcional y requieren el cumplimiento de diferentes condiciones temporales para cada una de ellas, permitiendo que se pueda elegir la más apropiada en cada caso. Por ejemplo, en un caso se establece que los tiempos consumidos por la implementación para realizar las acciones sean siempre menores que los indicados en la especificación, mientras en otro caso se exige que sean exactamente los mismos.

La metodología propuesta en este trabajo para el testing de los sistemas temporales genera un conjunto de tests a partir de la especificación. Concretamente, el algoritmo de

derivación de tests está diseñado de modo que, para todos los comportamientos de la implementación que deban chequearse antes de establecer la corrección, el algoritmo genera un test. Debido a que no se establece ningún criterio que reduzca el conjunto de tests derivados, éste será (posiblemente) infinito. Necesitaríamos un criterio de cobertura adecuado para poder generar un conjunto finito. En este sentido, el algoritmo es *completo*, es decir, proporciona una cobertura completa de fallos respecto a la relación considerada, *en el límite*. Si se impusiese alguna restricción como "generar n tests" o "generar todos los tests con m entradas" la completitud se alcanzaría para el criterio establecido.

El interés de esta metodología reside, por una parte, en que las hipótesis requeridas son muy débiles y, por otra, en que proporciona un método para encontrar y construir cualquier test, condición necesaria si se quiere *seleccionar* un conjunto finito de acuerdo a algún criterio.

Uno de los problemas que se presenta cuando se especifican sistemas con información temporal es la dificultad de establecer con precisión los tiempos asociados con las acciones que realiza el sistema. Esta falta de precisión induce cierto nivel de indeterminación a la hora de establecer la corrección temporal de la implementación. En esta línea, los modelos estocásticos permiten especificar restricciones tales como "con probabilidad p esta acción debería realizarse antes de t unidades de tiempo". Ahora el diseñador no necesita indicar un tiempo preciso para una acción determinada, sino una estimación probabilística. Sin embargo, hay situaciones en las que el diseñador no dispone de esta información probabilística, o bien considera que dándola complicaría innecesariamente el modelo. En este caso, la forma más apropiada para especificar restricciones temporales es el uso de *intervalos de tiempo*, mediante los cuales el especificador proporciona un conjunto de posibles valores, en lugar de un único valor, omitiendo la probabilidad asociada a cada uno de ellos. Es más, es posible que durante la etapa de testing el testeador permita alguna imprecisión en el comportamiento temporal del sistema. En algunas ocasiones, podría aceptarse que la ejecución de una tarea tarde más/menos tiempo del esperado: Si la ejecución de la tarea se ejecuta en el tiempo especificado en la mayoría de los casos, alguna desviación puede ser admisible. Otra razón para que el especificador permita imprecisiones es que los aparatos de medida del tiempo cuando se realiza el testing del sistema podrían no ser tan precisos como sería deseable. En tal caso, debido al mal funcionamiento del mecanismo de medida podría no detectarse un comportamiento temporalmente incorrecto.

Teniendo en cuenta estas consideraciones se ha desarrollado un marco formal de testing donde se contempla la posibilidad de que se produzcan algunas imprecisiones en el comportamiento temporal del sistema [MNR07b]. En este caso el tiempo se incluye en las es-

pecificaciones mediante la utilización de una extensión de las máquinas de estado finitos con intervalos de tiempo asociados a la ejecución de la transiciones. Al igual que ocurría en [MNR08c], una transición $s \xrightarrow{i/o}_{[t_1,t_2]} s'$ indica que si la máquina está en un estado s y recibe una entrada i, emitirá una salida o y pasará a un estado s' en un tiempo comprendido entre t_1 y t_2.

El proceso de testing, así como la definición de las relaciones de conformidad, dependerá de la medida de los tiempos de ejecución y del *nivel de error aceptable* en el sistema. Las posibles interpretaciones de *aceptable* dan lugar a diferentes alternativas a la hora de establecer las relaciones de conformidad y de definir la noción de *pasar un test*. Sin embargo, hay algo más que se debe tener en cuenta cuando se trabaja con sistemas en los que los requerimientos temporales se expresan mediante intervalos. Dado que este trabajo considera un *marco de testing de caja negra*, no es posible establecer mediante la ejecución de un único test que el comportamiento de la implementación es correcto respecto a la especificación, desde un punto de vista temporal. De hecho, la ejecución de un test indicará tan sólo el tiempo empleado por la implementación en la ejecución del mismo, no el intervalo asociado. Como consecuencia, y teniendo en cuenta que se consideran intervalos de números reales positivos, necesitaríamos un número infinito de observaciones de la ejecución de cada transición para asegurar que el intervalo de tiempo asociado (desconocido) es correcto con respecto al establecido en la transición correspondiente de la especificación (conocido). La estrategia planteada para superar esta limitación consiste, básicamente, en el registro de los tiempos que la implementación invierte en ejecutar las secuencias de acciones proporcionadas por los tests. Las relaciones de conformidad definidas se basan en dichos valores, requiriendo que cumplan, en cada caso, ciertas condiciones con respecto a los intervalos especificados. Como ya se ha mencionado antes, en este trabajo se considera que el comportamiento temporal de la implementación no tiene porqué corresponder exactamente con lo esperado, admitiéndose una cierta desviación, por lo que es necesario tener en cuenta la misma. Para ello se determina si la cantidad de valores incorrectos entre los tiempos registrados es *relevante*, teniendo en cuenta el grado de desviación, para asegurar la conformidad de la implementación respecto a la especificación.

Los modelos mencionados anteriormente incluyen sólo una noción limitada de paso del tiempo: *El tiempo asociado a la ejecución de acciones por parte del sistema.* Sin embargo, existen otras situaciones en las que es conveniente tener en cuenta el paso del tiempo. Esta consideración motivó la apertura de una variante en esta línea de investigación en la que se consideraba un nuevo concepto temporal: la noción de *time-out*. Esencialmente, un time-out

es un periodo de tiempo específico durante el cual el sistema puede permanecer a la espera de recibir un estímulo del entorno. Si este tiempo se rebasa y no se ha producido ninguna interacción con el sistema, éste cambia de estado, por lo que sus reacciones a los estímulos recibidos pueden ser diferentes a los que se hubieran producido con anterioridad a dicho límite de tiempo.

Las extensiones temporales de las metodologías clásicas de testing están usualmente enfocadas tan sólo a una de las variantes previamente indicadas: El tiempo está asociado o a las acciones o representa time-outs. Con el fin de unificar en un único marco ambas propiedades temporales se desarrolló un nuevo trabajo que culminó con la publicación [MNR06a]. Su versión extendida [MNR08b] ha aparecido muy recientemente.

El formalismo utilizado en estos trabajos es una variante de las máquinas de estados finitos extendidas que permite integrar de un modo natural ambos aspectos temporales. El marco teórico se complica debido, por una parte, a la consideración de variables que pueden afectar a los comportamientos temporales del sistema y, por otra, a la naturaleza no determinista del mismo. Ello implica que la misma secuencia de acciones pueda consumir diferentes tiempos en diferentes ejecuciones.

Asumiendo que tanto la especificación como la implementación pueden representarse en el formalismo mencionado anteriormente, se propone una metodología de testing formal que permita testear un sistema con respecto a una especificación. Nuevamente, el proceso de testing requiere la definición de una noción de corrección. Al igual que en casos anteriores, el tratamiento de sistemas de tiempo real implica la definición de diferentes niveles de conformidad (funcional y temporal). La conformidad funcional requiere, como es habitual, que las secuencias de entradas/salidas del sistema testeado sean permitidas por la especificación. Sin embargo, cuando se define en este marco la conformidad funcional, es necesario tener en cuenta los posibles time-outs; una secuencia puede ser aceptada sólo antes/después de que diferentes time-outs se hayan producido. Por tanto, los aspectos temporales del sistema afectan parcialmente a su comportamiento funcional.

En este trabajo se proponen diferentes relaciones de conformidad temporales para tener en cuenta el funcionamiento no-determinista que puede surgir. Respecto a la aplicación de tests a la implementación, también se ve afectada por el no-determinismo de las especificaciones y/o las implementaciones. En particular, el diagnóstico de la incorrección de un sistema requiere considerar el resultado de la aplicación de todos los tests, ya que un solo test podría ser insuficiente. Por ejemplo, si para establecer la conformidad de una implementación se requiere que el tiempo de ejecución de una de las distintas formas de realizar

una tarea concreta sea menor que el especificado, el hecho de que una de ellas consuma más tiempo no indica que la implementación no sea correcta, ya que otras ejecuciones podrían tardar menos.

La siguiente publicación recogida en este bloque de la tesis [MHN08], realiza el estudio y análisis de los time-outs en un formalismo basado en stream X-machines [Lay92]. Las principales ventajas del uso de este modelo para representar sistemas son su expresividad y la integración del proceso de control y de las estructuras de datos de los mismos. Este formalismo ha sido utilizado para especificar sistemas en diferentes áreas, y varias metodologías de testing han sido desarrolladas para este tipo de máquinas [IH97, HI98, BCG+99, HH00, IH00, HH04]. Intuitivamente, podemos pensar en una stream X-machine como un diagrama de transición de estados en el que los arcos están etiquetados con funciones. Cada función recibe una entrada y los valores actuales de la memoria, y produce una salida y los valores, posiblemente modificados, de la memoria.

En este trabajo dicho formalismo ha sido extendido para permitir al diseñador representar explícitamente los time-outs de un sistema. Asimismo, se propone una metodología de testing formal que permite testear sistemáticamente una implementación respecto a una especificación. La noción de corrección considerada es la equivalencia de trazas, es decir, las secuencias de entradas/salidas generadas por la implementación testeada deben coincidir con las de la especificación. Debido a la inclusión de time-outs en el sistema estos deben ser tenidos en cuenta a la hora de establecer dicha equivalencia, ya que puede haber secuencias que sólo estén permitidas en determinados intervalos de tiempo, no pudiendo ser aceptadas ni con anterioridad ni con posterioridad.

El algoritmo de generación de tests clásico para sistemas especificados mediante stream X-Machines deterministas está basado en el método propuesto por [Cho78] en el contexto de máquinas de estados finitos. Este método genera un conjunto de tests derivado de la especificación que permite determinar la corrección funcional de la implementación. La aplicación de este método requiere que las funciones asociadas a las transiciones estén correctamente implementadas y exige una serie de restricciones técnicas. Bajo estas hipótesis, el conjunto de tests generado por este método garantiza la determinación de la corrección del sistema. En [MNR06a] se eliminan algunas de estas restricciones: no se exige una cota en el número de estados de la implementación y no se requiere que las funciones no puedan producir la misma salida para una misma entrada y valores de la memoria. Asimismo, se demuestra que el algoritmo propuesto para la generación de tests es correcto y completo *en el límite*.

Desgraciadamente, testear completamente en la práctica un sistema complejo es imposi-

ble. El principal problema, como ya se ha mencionado, es que el tiempo del que disponemos para hacerlo es *finito*, mientras que, en general, un sistema suele presentar *infinitos* comportamientos. Usualmente, se trata de obtener un conjunto finito de tests, en base a diferentes hipótesis sobre el sistema, que permitan establecer, con un alto grado de fiabilidad, la corrección del mismo. Teniendo como principal objetivo tanto la finitud del conjunto de tests como la obtención de un alto grado de fiabilidad, se desarrolló una aproximación formal que permitía seleccionar aquellos tests que proporcionan una mayor cobertura de errores [MNR07c].

Como ya se ha mencionado anteriormente, con el fin de reducir la cantidad de comportamientos que se deben chequear para asegurar la corrección de un sistema, las metodologías de testing formal suelen asumir ciertas *hipótesis*. Entre estas hipótesis podemos encontrar que "el número de estados que describen la implementación es finito y conocido", que "el sistema es determinista" o, en el caso de sistemas no-deterministas, que "si probamos suficientes veces las posibilidades no deterministas, todas se ejecutaran al menos una vez". Sin embargo, en el caso de sistemas con restricciones temporales no es posible alcanzar la testeabilidad finita, ni siquiera en el límite.

Aunque el testeo de sistemas que presentan requerimientos temporales comparte metodologías con el testeo de los sistemas no temporales, estos presentan sus propias dificultades. En el caso de los sistemas temporales, acotar el análisis de la implementación requiere no sólo limitar la cantidad de ejecuciones del mismo, sino también el tiempo consumido en las mismas. Si quisiéramos testear el comportamiento de la implementación durante un máximo de t unidades de tiempo, deberíamos comprobar su funcionamiento para cada posible interacción ejecutada en cada valor perteneciente al intervalo de tiempo $[0, t]$, ya que como hemos indicado previamente, las respuestas del sistema pueden ser diferentes si una misma entrada se aplica en diferentes instantes. Si se asume tiempo continuo, el número de tests que deberíamos aplicar sería inmediatamente infinito. Aún en el caso de que consideremos un dominio de tiempo discreto, cuanto menor sea el *quantum* considerado, con el fin de aproximarnos a la realidad, mayor será el número de posibilidades, hasta llegar a alcanzar un número astronómico. Por tanto, el número de tests necesarios para chequear un sistema con restricciones temporales en un intervalo de tiempo finito es o infinito o muy alto. Así, en general, no es posible en la práctica testear de una forma completa y apropiada estos sistemas no es posible en general.

Una aproximación para superar estas limitaciones consiste en abandonar la posibilidad de un análisis riguroso y tratar de *elegir bien* un conjunto de tests, de modo que permita

detectar una amplia variedad de errores de acuerdo a un criterio establecido [Car00, KT05]. Desafortunadamente, estos criterios suelen estar basados en la experiencia del testeador o en heurísticas. Por el contrario, en [MNR07c] se propone un criterio riguroso para la selección de este conjunto de tests. En concreto, la metodología presentada permite la generación de los tests eligiendo los tiempos *más representativos* en el intervalo de tiempo considerado. La elección de estos tiempos se basa en la probabilidad de que la máquina no haya modificado su estado. De este modo, la aplicación de este conjunto de tests permite inferir no sólo el comportamiento de la implementación en un instante preciso de tiempo, sino también, en términos probabilísticos, en el periodo de tiempo anterior.

La estrategia más usada para reducir el conjunto de tests que permiten establecer la corrección de un sistema consiste en que los testeadores asuman hipótesis razonables sobre la estructura de las implementaciones. En esta línea, se han propuesto muchas metodologías que, para un conjunto específico de hipótesis, garantizan que el conjunto de tests obtenido con su aplicación es *correcto y completo*. Sin embargo, todo marco de hipótesis establecido a priori resulta muy estricto y limita la aplicabilidad de una técnica específica. Con el fin de eliminar esta limitación se propuso el marco $\mathcal{HOTL}$ [RMN06, RMN08] (*Hypotheses and Observations Testing Logic*), que permite evaluar si la aplicación de un conjunto de *observaciones* implica la corrección de una implementación para *un conjunto de hipótesis determinado*. Este conjunto de hipótesis es elegido por el testeador entre las disponibles en el repertorio proporcionado por la lógica. Estas hipótesis pueden estar asociadas a partes específicas de la implementación, como por ejemplo que *un determinado estado es determinista*, o al comportamiento global de la misma. Con estas últimas, el testeador puede asumir, por ejemplo, que la "implementación es determinista", que "tiene como mucho n estados", o que "el estado inicial es único", entre otras.

El estudio realizado con $\mathcal{HOTL}$ se extendió y adaptó para dar lugar a $\mathcal{THOTL}$ un marco para tratar con sistemas que presentan restricciones temporales. El trabajo realizado con $\mathcal{THOTL}$ culminó con las dos publicaciones [MNR07a, MNR08a] que cierran este primer bloque de la tesis. La adaptación de $\mathcal{HOTL}$ requirió la modificación de las reglas de la lógica de modo que los tiempos pudieran incluirse y tratarse con propiedad. Así mismo, el repertorio de hipótesis disponible se extendió de forma que se pudieran tener en cuenta suposiciones acerca de los intervalos de tiempo asociados a las transiciones.

El segundo bloque de la tesis corresponde al testing de sistemas con tiempo estocástico. Esta parte está constituido por las siguientes publicaciones: *Implementation Relations for Stochastic Finite State Machines* [MNR06b], *Testing Finite State Machines Present-*

ing Stochastic Time and Timeouts [MNR07d], *Testing conformance on Stochastic Stream X-Machines* [MN07], *Testing from a Stochastic Timed System with a Fault Model* [?] y *Mutation Testing from Probabilistic and Stochastic Finite State Machines* [HM09].

Como continuación de la línea de investigación enfocada al estudio del testing de sistemas con información temporal se decidió integrar en un mismo marco, por una parte, las limitaciones derivadas de las posibles indecisiones en la especificación de los requerimientos temporales de las acciones de los sistemas y, por otro, los comportamientos temporales correspondientes a los tiempos de espera de los mismos. Para ello consideramos el uso de modelos estocásticos que permiten la especificación de los primeros mediante una estimación probabilística.

Como ya se ha mencionado previamente, aunque la extensión de los formalismos para incluir restricciones temporales permite al especificador dar una descripción más precisa del sistema, hay situaciones en las que una noción de tiempo determinista no es adecuada. Por ejemplo, si se desea especificar un sistema donde se espera que un mensaje se reciba con probabilidad $\frac{1}{2}$ en el intervalo de tiempo $(0, 1]$, con probabilidad $\frac{1}{4}$ en el intervalo $(1, 2]$, y así sucesivamente, un formalismo en el que los tiempos asociados a las acciones se representen mediante tiempos fijos no permitiría describirlo con precisión. Sería necesario utilizar un dominio temporal *estocástico*. La idea subyacente es que la información temporal está definida en términos probabilísticos. En los *modelos estocásticos* la duración de las acciones se expresa mediante *variables aleatorias*. Ello permite tener expresiones de la forma "el tiempo de ejecución de la acción a está determinado por la distribución de la variable aleatoria ξ" en lugar de "la ejecución de la acción a consume t unidades de tiempo". Intuitivamente, la interpretación de una transición estocástica en dicho formalismo, que podemos denotar como $s \xrightarrow{i/o}_{\xi} s'$, indica que si la máquina está en un estado s y recibe una entrada i generará una salida o antes de t unidades de tiempo con probabilidad $P(\xi \leq t)$ y cambiará al estado s'.

En un marco de testing de caja negra, la detección de fallos temporales en sistemas donde el tiempo se rige de forma estocástica es todavía más compleja que en los sistemas temporales que tratan con otros dominios más sencillos. Ello es debido a que el testeador no tiene acceso a las funciones de probabilidad asociadas con las variables aleatorias. Debido a la naturaleza probabilística de los tiempos en los sistemas estocásticos, el tiempo consumido por el sistema para realizar una acción puede variar entre diferentes ejecuciones, incluso en sistemas funcionalmente deterministas. Una primera aproximación al testing formal de sistemas con tiempo estocástico aparece en [MNR08c], trabajo ya comentado y contenido en la primera parte de la tesis.

Dentro de esta línea de investigación se desarrolló un marco de testing formal para sistemas representados mediante una extensión de máquinas de estados finitos que permitan la descripción de requerimientos temporales, relativos tanto al tiempo consumido por el sistema en la ejecución de las acciones como al tiempo que el sistema puede permanecer en un estado sin recibir un estímulo del entorno, es decir, time-outs. Los primeros llevaban asociadas restricciones temporales de naturaleza estocástica, mientras que los time-outs se establecen mediante tiempos fijos.

Así mismo, se propusieron diferentes relaciones de conformidad, distinguiéndose entre los aspectos funcionales y temporales. Como ya se ha mencionado anteriormente, la inclusión de time-outs requiere su consideración a la hora de establecer la conformidad funcional de los sistemas, ya que pueden influir en las secuencias de entradas/salidas que el sistema puede ejecutar en cada instante.

En lo referente a la conformidad estocástico-temporal de los sistemas, el hecho de asumir un marco de testing de caja negra impide determinar si las acciones de la implementación tienen asociada una variable aleatoria (desconocida) idénticamente distribuida a la correspondiente en la especificación (conocida). En realidad, sería necesario un número infinito de observaciones de la implementación para determinar esta equivalencia. Por tanto, en este marco, se debe proponer una relación de conformidad más *realista*, basada en un conjunto finito de observaciones. La idea consiste en comprobar que para cualquier transición de la implementación, permitida por la especificación, los tiempos de ejecución observados son *compatibles* con la correspondiente variable aleatoria de la especificación. Esta noción de *compatibilidad* se establece mediante un *contraste de hipótesis.*

Adicionalmente, se propuso un algoritmo para derivar conjuntos de tests de las especificaciones considerando las diferentes relaciones de conformidad. La aplicación de este conjunto de tests permite establecer al testeador la corrección de la implementación respecto a la especificación cuando los tests se pasan satisfactoriamente. En términos más técnicos, el algoritmo propuesto es completo y correcto.

Estos resultados dieron lugar a las publicaciones [MNR06b, MNR07d] recogidas en esta tesis.

También en el ámbito de tiempos estocásticos se propuso una metodología de testing para especificaciones representadas mediante stream X-Machines [MN07]. En esta ocasión se realizó una adaptación de dicho formalismo que permite la representación de restricciones temporales estocásticas y se definió la conformidad funcional de una implementación respecto a una especificación mediante la equivalencia de las funciones computadas por ambas. Sin

embargo, la conformidad estocástico-temporal no podía establecerse como la equivalencia de las variables aleatorias asociadas a cada cómputo debido a las consideraciones planteadas previamente. Por tanto, se consideró nuevamente la aplicación de un contraste de hipótesis para decidir, para un determinado nivel de confianza, si una muestra de tiempos de ejecución de un cómputo puede generarse por la correspondiente variable aleatoria.

La metodología de testing propuesta en este marco es una adaptación de la técnica desarrollada por [IH97] para stream X-machines deterministas ya comentada anteriormente. La adaptación de esta técnica exigió la revisión completa de la misma, de modo que permitiera comparar las variables aleatorias con los tiempos observados durante la interacción con la implementación.

La noción de conformidad estocástico-temporal propuesta en estos trabajos requiere la compatibilidad de los tiempos de ejecución, observados al testear la implementación, con la distribución de las variables aleatorias correspondientes de la especificación. Sin embargo, se podrían aplicar nociones de conformidad menos estrictas. Por ejemplo, podría ser suficiente que la variable aleatoria de la implementación y la correspondiente de la especificación tuviesen la misma media. Con el objetivo de capturar diferentes nociones de conformidad estocástico-temporales y proponer un marco de testing más general, y por tanto de mayor aplicabilidad, se desarrolló el trabajo [**?**]. La adaptabilidad de la metodología propuesta se resuelve mediante la definición de una *relación de conformidad parametrizada* que podrá instanciarse en cada caso con la noción de conformidad más adecuada.

La detección de errores en esta metodología se basa en una técnica de generación de tests a partir máquinas de estados finitos no deterministas usando la noción de *maquina producto* [PYB96, Hie04, PY05]. La máquina producto puede verse como la composición en paralelo de la especificación y el modelo de la implementación. Intuitivamente, un estado de la máquina producto es un par (s, t) que representa estados de la implementación y la especificación. Su comportamiento coincide con el de la implementación si éste es consistente con el de la especificación; en otro caso, lleva a un estado de *fallo*. Por tanto, el proceso de testing consiste en detectar si el estado de fallo es alcanzable a partir del estado inicial de la máquina producto. En [**?**] el concepto de máquina producto se extiende para tratar la situación en que las máquinas presentan información estocástico-temporal, teniendo ésta adecuadamente en cuenta. En lo referente a la técnica de generación de tests, el método *state counting*, que proporciona cobertura total de fallos en implementaciones deterministas, se extendió al nuevo marco bajo la hipótesis de que se conoce una cota superior del número de estados de la implementación. El método fue adaptado para producir un conjunto de tests

que permita detectar si puede alcanzarse el estado de fallo de la máquina producto.

Muchos sistemas son de naturaleza probabilística debido bien al uso de comunicaciones en un medio con un grado de fiabilidad bajo, a la compartición de recursos o, como en el caso de los protocolos de comunicación, a la presencia de requerimientos probabilísticos. Por tanto, la descripción de estos sistemas incluye la *probabilidad* de que las acciones se ejecuten. Este componente probabilístico permite cuantificar el no determinismo de los sistemas.

[HM07] presenta una nueva línea de trabajo en el área de testing donde se aplica la técnica de mutación para la detección de errores en sistemas que presentan información probabilística. El formalismo propuesto para la representación de sistemas con restricciones probabilísticas es una extensión de las máquinas de estados finitos cuyas transiciones tienen asociada la probabilidad de que se ejecute la acción con la que está etiquetada. En el citado trabajo se considera una variante de la *interpretación reactiva de las probabilidades* [LS91]. Intuitivamente, una interpretación reactiva impone una relación probabilística entre transiciones con el mismo estado de origen y que llevan asociada la misma entrada (la salida puede ser diferente), sin embargo, la selección entre diferentes entradas en un estado no está cuantificada.

En el trabajo las mutaciones aplicadas consisten en el cambio del estado inicial o del estado alcanzado por una transición, la creación de una nueva transición y el cambio de la probabilidad asociada a una de ellas. Naturalmente, si se modifica la probabilidad de una transición, la probabilidad asociada al menos a una de las transiciones con el mismo estado de origen y la misma entrada debe ser modificada también, de modo que la suma de probabilidades asociadas a las mismas siga siendo 1. Lo mismo ocurre cuando se incluye una nueva transición y ya existe una transición con el mismo estado de origen y la misma entrada. Con el fin de encontrar secuencias que permitan distinguir la especificación y los mutantes, se han adaptado y aplicado resultados de autómatas probabilísticos que permiten establecer en tiempo polinomial la equivalencia de éstos, o generar, en caso contrario, dichas secuencias.

Al considerarse un marco de testing de caja negra, si se aplica una entrada a la implementación que se está testeando podremos observar la salida que produce pero, al igual que en los modelos estocástico-temporales en los que no podíamos ver la distribución de la variable aleatoria asociada, no obtendremos información de las probabilidades asignadas a las diferentes transiciones. Por tanto, aunque estas probabilidades sean fijas, no se puede determinar sus valores mediante el proceso de testeo. En este caso, la estrategia consiste en *estimar* las probabilidades aplicando varias veces los tests. Mediante el uso de *resultados*

estadísticos se establece el número de veces que se deben aplicar los tests para obtener un determinado nivel de confianza.

Una vez establecido este marco de testing para sistemas con información probabilística, y siguiendo con la principal linea de investigación de la tesis, se realizó una extensión para tratar con modelos estocástico-temporales [HM09]. La contribución más importante de este trabajo es el tratamiento simultaneo de dos tipos de comportamientos no-funcionales. Ello requirió la generalización del resultado de autómatas probabilísticos, adaptado en la publicación anterior, a este nuevo marco. Así mismo, se añadió un nuevo operador de mutación para modificar la variable aleatoria asociada a una transición.

El último bloque de esta tesis incluye dos publicaciones, *Implementation Relations for the Distributed Test Architecture* [HMN08b] y *Controllable test cases for the distributed test architecture* [HMN08a], que exploran el testing de sistemas que interactúan con el entorno mediante varios *interface*. Estos trabajos estudian los problemas de controlabilidad y observabilidad que presentan este tipo de sistemas en la fase de testing.

Algunos sistemas interactúan con su entorno mediante varios interfaces distribuidos denominados *puertos*. Cuando se testean dichos sistemas es habitual situar un testeador en cada puerto. Si los testeadores no pueden comunicarse y no existe un reloj global, entonces estamos ante una *arquitectura distribuida de testing*. Usualmente, los trabajos de testing en arquitecturas distribuidas se han centrado en máquinas de estados finitos deterministas, detectándose la posible presencia de tipos de dos problemas en la fase de aplicación de los tests. En primer lugar, pueden surgir problemas de *controlabilidad* en los que un testeador desconoce cuando debe aplicar una entrada. Por ejemplo, supongamos que un test comienza con la aplicación de una entrada x_p en el puerto p, que produce una salida y_p en el mismo puerto, y entonces, el testeador de otro puerto q debe aplicar la entrada x_q. El testeador del puerto q no sabe cuando se ha aplicado la entrada x_p ya que no puede observar las interacciones en el puerto p y por tanto no sabe cuando debe aplicar la entrada x_q. El segundo problema tiene relación con la *observabilidad* que puede llevar a un fallo enmascarado. Supongamos que, como en el ejemplo anterior, un test comienza con la aplicación de la entrada x_p y la emisión de una salida y_p en el puerto p, a continuación se debe aplicar nuevamente x_p y observar la salida y_p en el mismo puerto p, así como la salida y_q en el puerto q. La secuencia del test en el puerto p es $x_p y_p x_p y_p$ y y_q en el puerto q. Las secuencias observadas en ambos puertos serían las mismas si las salidas y_p e y_q se produjesen como respuesta a la primera aplicación de la entrada x_p y tan sólo y_p como respuesta a la segunda entrada. En tal caso, dos fallos simultaneos se enmascaran mutuamente. El trabajo en este tipo de

arquitecturas ha estado orientado a encontrar secuencias que no presentaran problemas de controlabilidad y observabilidad.

Aunque las máquinas de estados finitos deterministas son apropiadas para modelar muchas clases de sistemas, los sistemas distribuidos son frecuentemente no deterministas. Ello llevó al trabajo publicado en [HMN08b] que estudia el testing en una arquitectura distribuida para sistemas no deterministas.

El formalismo elegido en este caso fue *Input Output Transition Systems*, que aceptan el no determinismo y no exigen un alternancia estricta entre entradas y salidas. Se definió una nueva relación de implementación, *dioco*, basada en la conocida *ioco* [Tre96, Tre99]. Aunque esta relación de conformidad ha sido adaptada en numerosos casos, ésta fue la primera vez que se aplicó en un marco de testing distribuido. Es interesante señalar que las relaciones de conformidad *ioco* y *dioco* son incomparables, a excepción del caso en el que se considere que la especificación sea completa, es decir, se especifique el comportamiento de la máquina para todas las posibles entradas. En este caso, *dioco* es más débil que *ioco*. En este trabajo también se proporciona un método para obtener tests locales, para cada puerto, mediante la proyección de un test global.

La segunda publicación [HMN08a] extiende este primer trabajo mediante la consideración exclusiva de los tests *controlables*. Un test es controlable cuando no puede generar una situación en la que un testeador local tenga que aplicar una entrada o esperar una salida, dependiendo de lo que haya ocurrido en otro puerto. Además de caracterizar formalmente los tests controlables, se muestra como se puede decidir en tiempo polinomial si un test tiene la propiedad de ser controlable y se presenta una nueva relación de conformidad. Finalmente, se propone un algoritmo para la generación de tests controlables.

Por último, en el apéndice se recogen publicaciones que complementan el contenido de la tesis, no formando parte integral de la misma. La primera publicación [MNR06a] es una versión preliminar de [MNR08b] que aparece en el primer bloque. Del mismo modo, el trabajo recogido en [HM07] representa la base de [HM09] que se incluye en el segundo bloque. Finalmente, [RMN08] presenta el marco $\mathcal{HOTL}$ cuya extensión y adaptación para tratar sistemas con restricciones temporales [MNR07a, MNR08a] aparece en el primer bloque.

Capítulo 2

Estado del arte: métodos formales y testing

Con el avance de la tecnología informática, las técnicas de apoyo a la producción de software han aumentado notablemente su relevancia en este área. Entre ellas, los *métodos formales* y el *testing* han adquirido especial significación.

El testing de los sistemas que se desarrollan en la actualidad supone un alto coste en el proceso de producción, llegando a destinarse más del 50 % del presupuesto total del desarrollo de un sistema a la detección de errores. Uno de los aspectos más costosos asociado con las tareas de testing es el trabajo que debe desarrollarse manualmente, por lo que la mejora de dicho proceso se ha orientado a la automatización del mismo. Ha habido ya numerosas propuestas, basadas en la combinación de los métodos formales y técnicas de testing, enfocadas a dicha finalidad. Tradicionalmente, los métodos formales y el testing no estaban muy vinculados, siendo la interacción entre ellos muy escasa. Sin embargo, durante los últimos años han llegado a ser complementarios, existiendo muchas metodologías de testing en las que la generación de tests se basa en la especificación formal de los sistemas.

Por otra parte, existen muchas técnicas de testing que tratan de aumentar la calidad del producto final mediante el incremento de su nivel de *efectividad* y *eficiencia*. La efectividad se optimiza mediante el uso de conjuntos de tests que proporcionen una alta probabilidad de detectar fallos, mientras que la eficiencia se incrementa reduciendo el número de tests que deben ser ejecutados para alcanzar dicho objetivo.

Este capítulo presenta una revisión de los lenguajes utilizados para especificar formalmente sistemas para su testeo, así como las principales contribuciones en el área de testing formal para dichos lenguajes.

2.1. Lenguajes formales de especificación

Los lenguajes de especificación *"informal"* aplican una combinación de gramáticas semiformales, texto libre y representaciones gráficas para describir los requerimientos de los sistemas, por lo que la precisión de las especificaciones se ve afectada por diferentes factores, entre ellos la experiencia del especificador. Sin embargo, los requerimientos de un sistema pueden ser representados mediante el uso de *lenguajes de especificación formales* que permiten evitar las ambigüedades usualmente asociadas con las especificaciones informales.

Los lenguajes formales tienen como propósito facilitar una descripción rigurosa de los sistemas que se desea desarrollar, así como el análisis de su comportamiento. En los últimos treinta años los métodos formales han alcanzado un nivel de madurez que permite que sean aplicados alta frecuencia a lo largo de todo el ciclo de vida del desarrollo de un sistema, en especial, cuando los sistemas requieran un alto nivel de seguridad.

Un lenguaje formal utiliza una *sintaxis* que permite describir de forma precisa la especificación de los sistemas, textual o gráficamente. También presenta una *semántica* que proporciona un significado preciso de cada descripción de los sistemas para las que el lenguaje se usa. La especificación de un sistema suele contemplar diferentes aspectos del mismo, incluyendo su comportamiento funcional, su estructura o arquitectura e incluso aspectos no funcionales, como propiedades temporales o de rendimiento, por lo que la elección de un lenguaje u otro vendrá determinado por la adecuación del mismo a las características de cada sistema.

A continuación se revisan los principales lenguajes de especificación formales usados en marcos formales para realizar testing y su aplicabilidad a los diferentes tipos de sistemas.

2.1.1. Lenguajes basados en modelos

Hay diferentes modos de describir la especificación de un sistema, siendo uno de ellos la construcción de un modelo del comportamiento previsto. Lenguajes como Z [Spi88, Spi92], VDM [Jon91] y B [Abr96] permiten describir los estados del sistema junto con las operaciones que provocan el cambio de estado. En estos lenguajes los estados de un sistema suelen describirse mediante conjuntos, secuencias, relaciones y funciones, mientras que las operaciones lo hacen mediante predicados en términos de condiciones pre-post.

2.1.2. Lenguajes basados en estados finitos

Los lenguajes basados en modelos permiten describir sistemas generales, en particular, sistemas que potencialmente pueden presentar infinitos estados. Este carácter general tiene como inconveniente que el razonamiento sobre los mismos sea menos susceptible a la automatización. Los lenguajes de especificación basados en estados finitos, como su nombre sugiere, permiten la definición de un conjunto finito de estados, que suelen ser representados gráficamente mediante transiciones que indican los cambios entre dichos estados. Entre estos lenguajes cabe destacar las *máquinas de estados finitos* (FSM) [LY96], SDL [CCI88, ITU92], *Statecharts* [Har87, HG97] y *X-machines* [HI98]. La mayor parte del trabajo desarrollado con FSMs en el área de testing ha sido motivado por el testing de protocolos de comunicación, ya que las FSMs son muy apropiadas para la especificación de la estructura de control de los mismos. Con posterioridad, el testing basado en FSMs ha sido aplicado en el área de *testing basado en modelos*, en el que las especificaciones se utilizan para dirigir el proceso de testing.

En muchos casos, estos lenguajes de especificación permiten la representación de datos internos adicionales. Las operaciones representadas por las transiciones pueden acceder a estos datos y modificarlos, pudiéndose establecer también condiciones sobre los mismos. Las maquinas de estados finitos que presentan estas características son conocidas como *máquinas de estados finitos extendidas* (EFSM).

Otro formalismo de especificación que permite la integración de la estructura de control y los datos de un sistema son las X-machines. Estas máquinas disponen de una *memoria interna* y un conjunto de *funciones de proceso* que etiquetan las transiciones entre estados. Dichas funciones están definidas sobre el conjunto de entradas posibles y los valores de la memoria, produciendo en cada caso la salida esperada. Este formalismo fue introducido en [Eil74], y posteriormente propuesto por Holcombe en [Hol88] como lenguaje de especificación. Se han estudiado y desarrollado diferentes tipos de X-machines basados en restricciones sobre el conjunto de funciones y datos, siendo las stream X-machines [Lay92] las que han recibido una mayor atención.

2.1.3. Lenguajes basados en álgebras de procesos

Las *álgebras de procesos*, véase [BPS01] para tener una visión panorámica del campo, permiten describir sistemas con procesos concurrentes, donde varios procesos interactúan comunicándose entre ellos. Entre estos lenguajes se incluyen CSP [Hoa85], CCS [Mil80, Mil89], ACP [BW90], π-*calculus* [MPW92, Mil99, AG99] y LOTOS [LOT88].

Los *sistemas de transiciones etiquetadas* suelen utilizarse para describir el comportamien-

to de una especificación definida mediante un álgebra de procesos. Ello ha llevado a que el testing de sistemas descritos mediante estos lenguajes se haya centrado en el testing de este tipo de sistemas.

2.1.4. Lenguajes algebraicos

Aunque las álgebras de proceso son adecuadas para la manipulación algebraica, hay lenguajes que describen los sistemas en términos de sus propiedades algebraicas, mediante axiomas y reglas que caracterizan completamente las propiedades deseadas. Ejemplos de estos lenguajes son OBJ [GT79] y CASL [BM04, Mos04].

En términos matemáticos, un álgebra consiste en un conjunto de símbolos que denotan valores de algún tipo y un conjunto de operaciones sobre los mismos. Para describir las reglas que gobiernan el comportamiento de las operaciones es necesario especificar la sintaxis y la semántica de las operaciones. En este tipo de lenguajes, la primera usa una signatura para cada operación, indicando su dominio y rango. En cuanto a la semántica, ésta viene dada mediante ecuaciones que de modo implícito describen las propiedades requeridas.

2.2. Técnicas de testing

El desarrollo de software ha pasado de ser un proceso constituido por pocas tareas en las que intervienen pocas personas, a ser un proceso muy complejo en el que participan grandes equipos. Ello ha provocado que el proceso de testing haya pasado en muchos casos de ser realizado por un solo ingeniero a involucrar a todos los participantes en cada una de las etapas de desarrollo. Por tanto, se ha convertido en una necesidad la buena planificación de esta tarea en el ciclo de vida de la producción de un sistema.

Las tareas de testing están asociadas principalmente con el chequeo empírico de la corrección de los sistemas. Por el contrario, los métodos formales han estado relacionados con la verificación formal de la corrección de los mismos. El uso conjunto de métodos formales y testing, desde una etapa inicial de la producción del sistema, puede reducir el coste de su desarrollo.

Cuando se utiliza un lenguaje de especificación formal, se puede aplicar un análisis formal en las fases de especificación, diseño y codificación. Ello puede suponer tan sólo la comprobación de ciertas propiedades, o el establecimiento formal de la conformidad de un sistema con respecto a su especificación. Estas demostraciones se pueden realizar, en algunos casos, de un modo automático reduciendose así la probabilidad de incurrir en errores humanos.

El proceso de testing implica la ejecución de la implementación que debe ser chequeada: se aplica una secuencia de entradas al sistema y se observan las salidas recibidas. Esta relación entre entradas y salidas permite establecer la adecuación del sistema testeado a la especificación de la que partimos. Las secuencias de entradas aplicadas a la implementación se llaman *tests*. Entre las diferentes técnicas de testing se puede distinguir entre el testing de *caja negra* y el testing de *caja blanca*. En el *testing de caja negra* un sistema se chequea sin que se conozca su estructura interna. Los tests se generan a partir de la especificación y sólo se dispone de información acerca de las salidas esperadas para las entradas aplicadas. Ya que no se va a contar con ninguna información sobre cómo ha sido desarrollado el sistema, los tests se pueden generar a partir del momento en que la especificación está disponible. Otra categoría es el *testing de caja blanca*, que utiliza información de la estructura interna de la implementación para la generación de tests. En este tipo de testing, se pueden generar tests con el fin de chequear características específicas del sistema.

Además de poder argumentar si una técnica de testing puede establecer la conformidad de una implementación respecto a una especificación mediante un determinado conjunto de tests, hay otras propiedades de dichas implementaciones que se pueden establecer. Esta idea se formaliza en [GG75] mediante las nociones de *corrección y completitud*.

Un conjunto de tests es correcto si es capaz de detectar la incorrección de cualquier sistema erróneo. Por otra parte, un conjunto de tests es completo si el hecho de que un sistema supere la aplicación de todos los tests del mismo permite deducir la corrección del sistema. Por tanto, si el conjunto de tests derivado mediante una técnica específica de testing es correcto y completo, su aplicación determinará la corrección o incorrección de una implementación. En general, para que una técnica de testing tenga estas dos propiedades se requiere la generación de un conjunto infinito de tests, lo que no permite su aplicación práctica y hace necesario aplicar un criterio de selección que permita obtener un conjunto finito. En la mayoría de los casos, los criterios de selección se basan en *hipótesis* como "si el sistema es correcto para un subconjunto de los valores de entrada permitidos entonces es correcto para todos" o "si un sistema es correcto para un valor de todos los que pueden ser aplicados en una determinada secuencia, entonces es correcto para todos los valores posibles". Aunque hay muchas hipótesis posibles para la selección de tests [Gau95] se pueden identificar dos tipos principales: *hipótesis de uniformidad e hipótesis de regularidad*. El primer tipo hace referencia a la uniformidad del comportamiento de un sistema sobre un rango de datos. El segundo tipo se relaciona con la regularidad del comportamiento del sistema cuando el tamaño de los datos aumenta. Por ejemplo, se podría asumir que si un sistema funciona

correctamente para un buffer de tamaño 0, 1, y 2, entonces lo hará sea cual sea el tamaño del mismo. La idea de hipótesis está muy relacionada con la noción de *modelo de fallos* [IT97]: un conjunto de modelos entre los cuales se encuentra uno (desconocido) funcionalmente equivalente a la implementación testeada.

Una vez establecido el conjunto de hipótesis o el modelo de fallos más apropiado, la técnica de testing debe generar un conjunto de tests que permita establecer la corrección de los sistemas respecto a las especificaciones, asumiendo que se cumplen dichas hipótesis.

A continuación se revisan diversas metodologías de testing propuestas para los diferentes formalismos previamente presentados.

2.2.1. Testing para especificaciones formales basadas en modelos

Hay muchos trabajos de testing orientados a especificaciones Z, B, y VDM. Las técnicas de generación de tests para ellos están basadas en hipótesis de uniformidad. El dominio de los valores de entrada es particionado en subdominios en los que el comportamiento del sistema se asume uniforme. Si la hipótesis de uniformidad se cumple, basta con aplicar un dato de cada clase para testear el sistema. Muchas de las técnicas de derivación de tests propuestas en este ámbito han sido automatizadas; en ellas son aplicadas diferentes heurísticas para realizar la partición en clases de equivalencia y la selección de los datos que se utilizarán durante el proceso de testeo.

[Hal89] propone un método de testing para especificaciones Z basado en una clasificación en dominios de tests. La idea se basa en considerar diferentes combinaciones de subdominios obtenidos mediante la partición de los conjuntos de entradas, salidas y estados y las condiciones contenidas en los predicados de la especificación. [DF93] demuestra que, mediante la reescritura de la precondición y la postcondición de una especificación en forma normal disyuntiva, una gran parte del proceso de análisis de partición puede ser automatizado. Así mismo, los autores describen una técnica para generar a partir de la especificación, un autómata finito que puede ser usado para dirigir el proceso de generación de tests. La combinación de partición de categorías y forma normal disyuntiva fueron aplicadas para las generación de tests en [SCS97].

Una propuesta completamente diferente es la *mutación de especificaciones* [CS94]. La idea está inspirada en la técnica de testing mediante *mutación* de programas [How82, BM99]. La aplicación de *operadores de mutación* a la especificación genera mutantes. Para cada mutante obtenido se genera un test que distinga el comportamiento de éste y la especificación. La hipótesis en este caso es que si un conjunto de tests obtenido mediante este método permite

distinguir entre un mutante y la especificación, también distinguirá entre la especificación y una implementación incorrecta.

2.2.2. Testing basado en máquinas de estados finitos

Muchos sistemas tienen una estructura de estados finitos y por ello las FSMs son un formalismo adecuado para su representación. Por ello, el testing de FSMs ha recibido mucha atención. Moore planteó el marco conceptual de testing basado en FSMs en su trabajo [Moo56] referido a un concepto muy utilizado en la física, los *experimentos gedanken*. Los principios fundamentales del chequeo de transiciones fueron enunciados en [Hen64], trabajo motivado por el testing de circuitos secuenciales, que más tarde fue aplicado al testing de protocolos de comunicación [LY96].

Las FSMs definen un lenguaje relativamente pobre en cuanto a expresividad y técnicas de abstracción. Sin embargo, la falta de expresividad tiene grandes ventajas cuando se analizan las FSMs. Muchos problemas que no son decidibles en marcos más generales lo son para FSMs, y frecuentemente además con complejidad polinomial, lo que facilita la automatización de la generación de tests. Como se dijo anteriormente, muchas propuestas para testing basado en FSMs consideran o *hipótesis* o un *modelo de fallos*. En ambos casos, el conjunto de tests garantiza la determinación de la corrección de los sistemas siempre que se cumplan las hipótesis consideradas. En el testing basado en una especificación representada mediante una FSM, es normal asumir que la implementación es equivalente a una FSM desconocida, y que el conjunto de tests generado permite chequear si la implementación es *conforme* a la especificación. Si la especificación es determinista la noción de conformidad coincide con la equivalencia de especificación e implementación. Si la especificación es no determinista hay nociones alternativas de conformidad, como considerar que el comportamiento de la implementación es un *subconjunto* del comportamiento de la especificación.

FSMs completamente especificadas y deterministas

En primer lugar trataremos el caso de las FSMs completamente especificadas, mínimas y deterministas, esto es, máquinas donde para cada entrada disponible y cada estado existe una única transición etiquetada con dicha entrada, no tienen estados equivalentes y toda entrada tiene una transición asociada en cada estado. En este caso la relación de conformidad entre la especificación y la implementación es la *equivalencia*, es decir, se producen las mismas secuencia de salidas para las mismas secuencias de entradas. El método de recorrido de transiciones [NT81], abreviado como método TT, produce un conjunto de tests que ejecuta

todas las transiciones de la especificación. Este método sólo está orientado a la detección de fallos de salida. La variante de este método presentada en [SMIM90] tiene menos capacidad de detección de fallos: cubre todos los estados, pero no necesariamente todas las transiciones.

El método D [Hen64, Gon70, Koh78] asume como hipótesis que la implementación no tiene más estados que la especificación. El método chequea todas las transiciones con el fin de localizar tanto fallos de salida como de transición de estados. Este método ha sido optimizado para reducir el número de tests necesarios [UWZ97, HU02].

Otra propuesta está basada en secuencias únicas de entrada/salida (UIO) [SD88]. En este caso las hipótesis son, por una parte, que la implementación no tiene más estados que la especificación y, por otra, que existe un *reset* que lleva la implementación al estado inicial. Las secuencias UIO se usan para verificar que tras una serie de interacciones, la máquina se encuentra en el estado esperado.

Cuando no se cumple la hipótesis de que el número de estados de la implementación no supera al de la especificación, se puede aplicar el método W [Vas73, Cho78]. En este caso se considera que existe una cota superior en el número de estados de la implementación y un reset correctamente implementado. Bajo estas condiciones el método proporciona cobertura total de fallos. Este método genera el conjunto de tests basándose en un *conjunto de caracterización* y un *conjunto de cobertura de estados*.

FSMs parciales

La mayoría de las especificaciones reales son *parciales*, por lo que no cubren todas las posibles combinaciones de estado/entrada. Debido a las diferentes interpretaciones de las *transiciones no definidas* han sido consideradas diversas semánticas [BP94]. Puede considerarse que estas transiciones están implícitamente definidas, es decir, se sustituyen por transiciones con el mismo estado de origen y destino con salidas nulas, o bien por transiciones que llevan a un estado de error y producen una salida de error. De este modo se obtiene una FSM completamente especificada y se pueden aplicar las estrategias presentadas anteriormente. Otra posible interpretación considera que son transiciones prohibidas, por lo que no deberían ser ejecutadas por la implementación. En consecuencia, el conjunto de tests no debe considerar estas transiciones.

FSMs No Deterministas

Hay diferentes motivos que pueden ocasionar la presencia de indeterminismo en una especificación. El no determinismo puede representar un requerimiento para el sistema es-

pecificado, en cuyo caso la conformidad de la implementación corresponderá a la equivalencia de la misma respecto a la especificación. Sin embargo, la presencia de no determinismo puede representar un abanico de opciones, por lo que será suficiente que los comportamientos de la implementación sean un subconjunto de los especificados.

El no determinismo presenta una cuestión a tener en cuenta: La observabilidad de cada posible repuesta asociada con una secuencia de entradas en particular. Con el fin de abordar este problema es frecuente asumir la hipótesis de que existe un número máximo n, de modo que si una misma secuencia de entradas ha sido aplicada n veces en un estado, entonces está garantizado que todas las posibles salidas asociadas con dicho estado y secuencia de entradas han sido observadas. Esta hipótesis se conoce como *fairness hypothesis*.

FSMs extendidas

Una máquina de estados finitos extendida (EFSM) es una FSM con parámetros de entrada y salida, variables auxiliares y operaciones y predicados definidos sobre las variables y los parámetros de entrada. Si se aplica una hipótesis de uniformidad a los datos es posible generar un conjunto de tests abstrayendo los citados parámetros de la EFSM. En consecuencia, el proceso de testing estará orientado a la estructura de control de la implementación [DU04]. Si no se asume la hipótesis de uniformidad entonces el proceso de testing afectará tanto a la estructura de control como a la estructura de datos. Estos dos componentes se testean independientemente, aplicando métodos basados en FSMs para la parte de control y métodos de testing para el flujo de datos a la parte correspondiente a los mismos [USW00]. Bajo hipótesis técnicas precisas una EFSM se puede transformar en una FSM equivalente. Aunque estas transformaciones frecuentemente conllevan un problema de explosión de estados, en algunos casos hay métodos efectivos para realizar la conversión [PBG04].

Una estrategia alternativa está basada en el uso de conjuntos de *propósitos de test*. Un propósito de test es una descripción de un requerimiento para un test, como por ejemplo, la ejecución de una transición o una secuencia de transiciones en particular, y puede representarse mediante un autómata de estados finitos. Para cada propósito de test se genera un test que lo satisface. El test puede construirse automáticamente aplicando análisis de alcanzabilidad al producto del propósito de test y la especificación. Aunque el análisis de alcanzabilidad sufre del problema de explosión de estados, hay herramientas que aplican una estrategia *on-the-fly*, como TGV, que son efectivas en la práctica [KJG99]. Usualmente los propósitos de test son proporcionados por el testeador, pero pueden obtenerse también automáticamente a partir de la EFSM que representa a la especificación, teniendo en cuenta un cierto criterio

de test establecido, como podría ser la ejecución de todas sus transiciones. Otro tipo de propósito de test, más restringido, consiste en requerir una secuencia de interacciones que puede describirse mediante un *message sequence chart* (MSC). Existen herramientas como AUTOLINK que aplican un análisis de alcanzabilidad al producto de la especificación y un MSC para producir un test apropiado [SEK+98].

Finalmente, se ha abordado también el problema de testing mediante la conversión en FSM equivalentes de especificaciones realizadas en el contexto de otros modelos formales, con el fin de aplicar los métodos de testing basados en dicho formalismo. Entre ellos se incluyen métodos de conversión de variantes de especificaciones Z [DF93, Hie97, DB99], Statecharts [HSS01] y SDL [BPBM97].

Stream X-Machines

El primer método de testing basado en Stream X-Machines se propuso en [IH97, HI98] para sistemas deterministas. Esta técnica de testing genera un conjunto de tests a partir de una especificación, asumiendo que las funciones de proceso asociadas a las transiciones están correctamente implementadas. Además, se requiere que el conjunto de funciones de la especificación y la implementación coincidan, y que se satisfagan dos condiciones, habitualmente denominadas *design for test conditions*. Estas condiciones consisten en la distinguibilidad de las funciones de proceso mediante las salidas emitidas y la seguridad de que mediante las entradas apropiadas dichas funciones podrán ser chequeadas en la implementación. Este método ha sido generalizado en [Ipa04] para suprimir la hipótesis de la corrección de la implementación de las funciones de proceso. Así mismo, se relaja la restricción de que el conjunto de funciones deba coincidir en la especificación y la implementación.

También existen propuestas para máquinas no deterministas. En [IH00] se extiende el método para cubrir máquinas no deterministas y se presenta una técnica de testing en la que la noción de corrección es la equivalencia de máquinas. Otra metodología alternativa ha sido propuesta en [HH00, HH04], en ella se hace uso del método de testing conocido como *state counting* y se considera una noción de conformidad basada en que los comportamientos de la implementación son un subconjunto de los de la especificación.

2.2.3. Testing para álgebras de procesos

El principal estudio de testing en este área fue desarrollado por de Nicola y Hennessy [dNH84, Hen85, Hen88]. Ellos introdujeron diferentes *preordenes y equivalencias* para relacionar procesos mediante su interacción con conjuntos de tests. Esencialmente se distinguen

dos familias de relaciones: *may* y *must*. En las primeras se permite que el proceso pase el test con éxito en alguna ejecución; sin embargo en la segunda categoría ello debe ser así en todas las ejecuciones posibles. El marco original ha sido extendido para tratar con propiedades no funcionales como tiempo [HR95, LdF99], probabilidades [LS89, Chr90, NdF95, Núñ03], y una combinación de ambas [GLNP97].

Como ya se ha mencionado, los sistemas de transiciones etiquetadas se usan frecuentemente para describir la semántica de las álgebras de proceso. Las técnicas de testing para LTSs se basan en relaciones de conformidad y existen numerosos algoritmos de generación de conjuntos de tests basados en diferentes relaciones de conformidad. Entre ellos se pueden citar [Led91, Pha94, Tre96].

2.2.4. Testing para especificaciones algebraicas

El primer trabajo que empleó especificaciones algebraicas en una metodología de testing fue el desarrollado para el sistema DAISTS [GMH81]. Sin embargo, las propuestas más significativas se presentan en [GJ98, Gau01].

Es evidente que la aplicación de un conjunto de tests exhaustivo que no muestra fallos garantiza la corrección del sistema. No obstante, la naturaleza infinita de este conjunto de tests lo hace impracticable, por lo que se requiere limitar este conjunto mediante hipótesis de regularidad y uniformidad. Las hipótesis de regularidad, en el contexto de las especificaciones algebraicas, se basan en el número de constantes y constructores que aparecen en un término. La hipótesis permite establecer la corrección del sistema si éste funciona correctamente para tests de hasta un cierto tamaño n. Las hipótesis de uniformidad permiten de nuevo establecer si el sistema funciona correctamente para una selección de valores dentro de los diferentes subdominios establecidos mediante partición. El método utilizado más frecuentemente para la selección de los valores es *Boundary Value Analysis* [WC80, CHR82, LPU02].

Hay dos posibilidades para generar tests a partir de especificaciones algebraicas: usando la sintaxis de las operaciones o los axiomas. El primer método se presento inicialmente en [Jal83] y posteriormente se ha aplicado en diferentes experimentos [JC88, Woo93, AW96]. El segundo método se introdujo en [GMH81] y ha sido utilizado en muchas propuestas posteriores [Cho86, DF94, CTCC98, CTC01].

Capítulo 3

Estado del arte: testing y extensiones temporales

En el capítulo anterior se ha presentado una visión panorámica del estado del arte de las metodologías de testing formal. Dada la amplia selección de trabajos en este área, ha sido inevitable limitarnos a la presentación de ideas generales, sin prestar especial atención a los desarrollos específicos. En el presente capítulo nos restringimos a las técnicas de testing formal para el análisis del comportamiento *temporal* de los sistemas, lo que nos permitirá ser más precisos. Inicialmente se revisan algunos formalismos propuestos para la representación de sistemas de tiempo real y sistemas en los que el tiempo juega un papel relevante en las restricciones que los mismos presentan. A continuación, se presentan las técnicas propuestas para el establecimiento de la corrección de las implementaciones con respecto a las especificaciones descritas utilizando los formalismos anteriores.

3.1. Formalismos para representar sistemas temporales

Aunque el tiempo afecta a todos los sistemas, el interés de los investigadores para incluirlo de forma explícita en los modelos formales es relativamente reciente. El desarrollo de sistemas de tiempo real ha llevado a un nuevo escenario en el que las restricciones temporales se han convertido en un tema de interés. Esencialmente, el paso del tiempo afecta al comportamiento de los sistemas de dos formas:

- Después de que un usuario solicita un cómputo, el sistema necesitará una cantidad de tiempo perceptible para realizarlo.

- En algunos casos el sistema requiere un periodo de espera para poder realizar una operación, o bien puede ocurrir que el paso del mismo haga que cambie su estado si al no haber recibido ninguna reacción del entorno.

En ambos casos estas restricciones temporales deben estar determinadas en la especificación, lo que permitirá establecer la diferencia entre un sistema aceptable y uno que no lo es, es decir, entre considerar un sistema correcto o incorrecto.

La inclusión explícita de esta información en los formalismos de especificación no es sencilla. Por una parte, la sintaxis del lenguaje debe permitir la representación de los requerimientos de una forma expresiva y, por otra, la semántica debe ser capaz de denotar el paso del tiempo de un modo manejable. En general, dada la configuración de un sistema, la cantidad de valores temporales que son relevantes para la descripción del mismo puede ser significativamente alta, pudiendo llegar a ser incontable. Por tanto, se necesita disponer de una representación compacta que permita que la especificación denote *explícitamente* todos los valores temporales aceptables.

Además, hay muchas formas de interpretar los requerimientos temporales. Por una parte, se puede considerar que las restricciones temporales son *estrictas* e indican de forma precisa los requerimientos temporales, como por ejemplo que "la acción a ocurrirá exactamente en t unidades de tiempo". Otra alternativa es una interpretación más relajada de los condicionamientos temporales, expresando los mismos mediante intervalos, o incluso plantear la opción de utilizar términos probabilísticos para definir las restricciones temporales, por ejemplo, "la acción a ocurrirá antes de t unidades de tiempo con probabilidad p". Las diferencias entre estos planteamientos afectan tanto a la definición de las especificaciones como a los comportamientos que se derivan de ellas.

Los formalismos utilizados para describir sistemas temporales son, habitualmente, extensiones o adaptaciones de otros formalismos previamente propuestos para representar sistemas donde el tiempo no se considera explícitamente. A continuación se describen brevemente algunos de estos formalismos. En primer lugar se consideran aquellos donde el tiempo no se define en términos probabilísticos, lo que no significa necesariamente que se tengan que utilizar tiempos fijos. Como veremos más adelante, las especificaciones permitirán denotar cualquier tiempo que cumpla una condición establecida. Posteriormente se revisarán los sistemas con restricciones estocástico-temporales.

3.1.1. Representación de tiempo no probabilístico

Durante los últimos 30 años se han propuesto muchos modelos para el análisis y especificación de sistemas temporales, entre los que cabe destacar las *redes de Petri* [Sif77, Zub80], el *calculus duration* [CHR91], y diferentes extensiones de autómatas con información temporal [LV96, SGSL98, LSV03]. Sin embargo, el modelo más aceptado ha sido el de los *autómatas temporales* [AD90, AD94], que permiten describir los infinitos comportamientos inducidos por dominios de tiempo continuo de un modo simbólico. Ello hace posible disponer de una representación finita del sistema, facilitando el análisis de sus propiedades.

Un autómata temporal representa el comportamiento de un sistema mediante un conjunto de *relojes* que registran el paso del tiempo. Durante la ejecución, los relojes tienen asociados valores que se incrementan de modo síncrono. Estos relojes pueden ser actualizados o consultados para comprobar el tiempo transcurrido desde su inicialización, permitiendo medir y comparar los tiempos en que se producen los diferentes eventos. Un autómata temporal es básicamente un sistema de transiciones que se ejecutan instantáneamente, considerándose que no consumen tiempo. Las transiciones dependen de condiciones definidas sobre los relojes, conocidas como *guardas*, y pueden reiniciar los mismos. Los estados pueden tener asociados *invariantes* que deben cumplirse mientras el sistema permanece en dichos estados.

La representación del paso del tiempo en los autómatas temporales es *simbólica*, en el sentido de que el comportamiento temporal del sistema se expresa mediante la actualización de los relojes y condiciones definidas sobre éstos, en lugar de utilizar transiciones con valores de tiempo *concretos*.

Los autómatas temporales han sido aplicados con éxito en la verificación automática de sistemas de tiempo real, particularmente por medio de model checking [ACD93, HNSY94, YPD95, ACH94, DY95] y herramientas que implementan dichos algoritmos como Kronos [DOTY96, BDM+98], UPPAAL [BLL+96, LPY97, BLL+98] y HyTech [HHWT95].

También se han propuesto varias aproximaciones de las álgebras de procesos que sirven para especificar sistemas temporales. Originalmente, el tiempo incluido era discreto [MT90, Gro91, Han91, QdFA93, NS91, BB96], es decir, el tiempo se representaba mediante una acción *tick* que representa el paso de una unidad de tiempo. Estas álgebras de procesos son adecuadas para modelar sistemas digitales, pero no para describir sistemas de tiempo real de un modo *natural* [1]. Por esta razón se desarrollaron álgebras de procesos para

[1] La discusión sobre lo inapropiado del uso del tiempo discreto para la descripción de los sistemas temporales frente al tiempo continuo ha generado una gran controversia. Por una parte, el tiempo en los sistemas temporales computacionales y electrónicos suele ser digital. Debido a que el diseño de los sistemas no tienen una precisión temporal infinita, se puede argumentar que no es posible construir sistemas en los que deba

tiempo continuo [Yi90, BB91, SDJ+92, LL97]. Algunas de ellas han sido relacionadas formalmente con los autómatas temporales [NSY92, BHKR95], sin embargo, estas relaciones solo proporcionan una conexión semántica y ninguna es completa en el sentido de ser biyectivas. Finalmente, lenguajes que representan completamente autómatas temporales han sido definidos en [AH94, YPD95, LV96].

3.1.2. Representación de tiempo estocástico

El análisis de sistemas estocástico-temporales ha recibido mucha atención pero usualmente fuera del marco de los métodos formales. Inicialmente, en el campo de las Matemáticas se definieron modelos para el análisis de procesos estocásticos. Estos modelos, que incluyen *continuous time Markov chains*, abreviado CTMC, se usaron para analizar el comportamiento de los sistemas. Pero los sistemas se fueron haciendo más complejos y se hizo necesaria una notación más sofisticada para la descripción de sus modelos. Surgiendo así nuevos formalismos como las *queueing networks* [Kle75, HP92] y diferentes *redes de Petri estocásticas* [ACB84, ABC+95].

En los modelos basados en CTMCs el tiempo asociado a las transiciones viene dado por una variable aleatoria con distribución exponencial. Esta restricción es la clave de una teoría numérica y analítica muy amplia sobre CTMCs. La restricción a distribuciones exponenciales proporciona permite tratar con actividades que tienen la propiedad de la *falta de memoria*. Esta propiedad dice, básicamente, que en cada instante de tiempo a partir de aquel en el que la actividad ha comenzado, si la misma aun no ha concluido, el tiempo residual de duración sigue estando distribuido como su propia duración total. La falta de memoria hace posible representar el comportamiento temporal de los sistemas mediante una CTMC, esto es, un proceso continuo en el que en cada instante de tiempo, el comportamiento futuro del proceso es completamente independiente del comportamiento pasado, dependiendo tan solo de su estado actual (propiedad de Markov). De hecho, la falta de memoria en el marco Markoviano permite evitar la representación explícita del paso del tiempo en las descripciones de los sistemas.

Desgraciadamente, la restricción a distribuciones exponenciales no es siempre realista y puede llevar a resultados poco útiles. Otros modelos más generales, como *generalised semi-Markov processes* [Whi80, Gly89, Cas93, She93], permiten que la descripción de los tiempos

ser considerado tiempo continuo. Además, las observaciones y el testeo de los dispositivos no permiten tal precisión. Por otra parte, suponer la existencia de un *tick* mínimo lleva a la idea errónea de que las técnicas de análisis puedan basarse en la consideración sistemática de que todos los valores temporales son posibles. En estos casos la cantidad de valores temporales que habría que considerar es astronómico.

pueda corresponder a una distribución cualquiera. Desgraciadamente, ninguno de estos modelos proporciona un marco adecuado para la composición de sistemas distribuidos.

En el ámbito de los métodos formales la descripción y análisis de los sistemas estocásticos se abordo en primer término mediante la definición de las *álgebras de proceso probabilísticas* [GJS90, GSS95, BBS95, CCVP01, Núñ03, CCV+03]. Inicialmente estos marcos algebraicos se orientaron a la verificación más que al análisis del rendimiento, dado que tan sólo trataban con distribuciones de probabilidad discretas, y en la mayoría de los casos no tenían carácter temporal. Fue en [Her90] donde se introdujeron las *álgebras de proceso estocásticas*, aprovechando el marco analítico proporcionado por las *continuous time Markov chains*. Entre las más conocidas cabe mencionar TIPP [HR94], PEPA [Hil96], EMPA [BG98], IMC [Her98] y NMSPA [LN00].

3.2. Testing de sistemas temporales

Como ya se mencionó en el capítulo anterior, el testing no es una tarea fácil. En general, hay infinitas formas de interactuar con un sistema, por lo que la corrección de una implementación sólo puede establecerse después de comprobar *todos* sus comportamientos. Hay que considerar que el tiempo disponible para testear un sistema es finito. Por tanto, el número de tests que pueden ser aplicados, así como el tamaño de los mismos, también debe ser finito. La dificultad del testing reside pues en la imposibilidad de que los tests puedan alcanzar y chequear *cualquier* comportamiento posible de una implementación. No obstante, se puede testear parcialmente un sistema hasta alcanzar un *criterio de cobertura*, por ejemplo aplicando sólo tests de un tamaño menor o igual que un valor determinado.

En los sistemas temporales deben tenerse en cuenta además las restricciones temporales a la hora de testear su comportamiento. Ello requeriría chequear el sistema en todos los tiempos posibles, por lo que el número de tests necesarios se incrementaría dramáticamente. Tanto si se considera tiempo discreto como si este es continuo, la generación y aplicación de un conjunto de tests para comprobar el comportamiento del sistema en cualquier instante es simplemente inviable, aún en el caso de establecer una cota para el tiempo.

Como en el caso de las metodologías para testear sistemas no temporales, hay diferentes propuestas para abordar el problema en el caso de los sistemas temporales. Una posibilidad es aceptar la imposibilidad de abarcar todos los comportamientos de la implementación y testear sólo aquéllos que son especialmente *relevantes* o *representativos*. Otra alternativa consiste en considerar alguna *hipótesis* acerca del comportamiento de la implementación, de modo que la aplicación de un conjunto finito de tests sea suficiente para garantizar la

corrección del sistema. Aunque es imposible, en general, derivar y aplicar el conjunto de tests para comprobar a partir de estos la corrección de un sistema por completo, resulta muy conveniente disponer de una técnica para seleccionar tests relevantes de acuerdo a un criterio determinado. La mayoría de las propuestas realizadas se centran en esta tarea: La generación de conjuntos de tests completos en una fase inicial del método para posteriormente seleccionar sólo aquéllos que serán aplicados.

Respecto a sistemas temporales definidos mediante un dominio temporal probabilístico apenas se encuentran propuestas en la literatura. Tan sólo en [NR03] se presenta un método para testear sistemas estocástico-temporales.

A continuación se revisan metodologías de testing en las que el tiempo no se define probabilísticamente. En primer lugar se consideran técnicas que derivan conjuntos de tests completos e infinitos a partir de las especificaciones. Después se comentarán otros métodos que generan conjuntos de tests finitos.

En [BB04] se presenta una propuesta para testing de sistemas temporales basada en *ioco* [Tre96]. El concepto de *quiescence*, esto es, la existencia de estados que no pueden producir salidas o realizar acciones internas, también es considerado, lo que permite asumir que los tests tienen la capacidad de detectar esta situación. Básicamente, se establece una cota que representa el tiempo que el estado permanece inactivo hasta que concluye el estado de *quiescence*. El tratamiento de sistemas que presentan esta característica da lugar a una familia de relaciones de implementación parametrizadas por la duración de la observación de la *quiescence*. En este marco se utilizan sistemas de transiciones etiquetadas temporales con inputs y outputs para describir las especificaciones, y se presenta un algoritmo de derivación de un conjunto de tests correcto y completo. El algoritmo, al igual que en [Tre96], es no determinista, representando cada opción un recorrido diferente en la especificación. Al igual que en el caso no temporal, el conjunto de tests obtenido es, en general, infinito.

En [BB05] se presenta una extensión del trabajo anterior para tratar sistemas temporales que pueden comunicarse con el entorno mediante múltiples canales. En el formalismo propuesto los canales se representan como una partición de los conjuntos de acciones de entrada y salida, en los que cada clase de la partición define las entradas y salidas correspondientes a un canal. La hipótesis de que todas las entradas pueden ser aplicadas en cualquier estado se limita a conjuntos de acciones que están o no permitidas, y la cota utilizada en los sistemas temporales para detectar *quiescence* también se determina para diferentes conjuntos de salidas permitidas. Se presenta una nueva relación de conformidad parametrizada por estos factores y se propone un procedimiento parametrizado para la derivación de tests. El

conjunto de tests obtenido es correcto y completo respecto a la nueva relación de conformidad.

Otras metodologías [MMM95, PS97] generan un conjunto de tests completo e infinito. Como es usual sólo los conjuntos de tests finitos tienen utilidad práctica. Por tanto, estas técnicas tan sólo proporcionan la base teórica para otros métodos en los que se generan y aplican conjuntos finitos de tests. En estos casos hay dos estrategias posibles: Considerar hipótesis muy restrictivas acerca del comportamiento de la implementación, generándose en este caso conjuntos completos de tests, o bien restringirse a la búsqueda de tests con una alta capacidad de detección de errores, obteniéndose inevitablemente conjuntos de tests incompletos.

En [SVD01] se propone una generalización de la teoría clásica de testing a un marco de sistemas de tiempo continuo. En concreto se presenta un modelo de *autómatas temporales I/O* inspirado en el propuesto en [AD90, AD94]. La principal contribución de este trabajo es un algoritmo de testing de caja negra para sistemas representados mediante dicho formalismo. Aunque dicho algoritmo tiene complejidad exponencial y por tanto no puede aplicarse en la práctica, es el primer algoritmo que produce un conjunto de tests *finito* y completo para sistemas temporales en un dominio de tiempo continuo. El algoritmo requiere que se asuman algunas hipótesis muy restrictivas sobre las salidas: éstas deben emitirse en instantes de tiempo precisos y en cada estado solo puede producirse una salida. Los conceptos y técnicas presentadas en este trabajo han sido ciertamente muy útiles para algoritmos más prácticos propuestos posteriormente.

En [FPS01] se propone otra técnica para testing de sistemas temporales mediante la derivación de tests a partir de especificaciones modeladas como autómatas temporales. La mayor peculiaridad de este trabajo es la forma en la que los autores buscan que la técnica de testing sea viable. Mientras que otros estudios se centran en reducir el formalismo de especificación para ser capaces de derivar tests de forma práctica, en este trabajo los tests se orientan a *propósitos de tests* específicos establecidos por el usuario. Aunque los autómatas temporales se usan para la descripción de especificaciones, el estudio utiliza una representación equivalente de los mismos, los llamados *Clock region graphs*. Éstos se extraen del autómata temporal considerando todas las posibles evaluaciones de los relojes que son equivalentes en términos del cumplimiento (o no) de los requerimientos impuestos por el autómata en cada guarda. En concreto, una *clock region* es una clase de equivalencia inducida por dicha relación.

En [CG98] se presenta un método formal para la generación de un conjunto finito de

tests mediante un algoritmo que, bajo el conjunto de hipótesis considerado, permite concluir que el sistema testeado es bisimilar a su especificación si pasa todos los tests. En contraste con [SVD01], donde los autores reconocen la imposibilidad de su aplicación práctica, el conjunto de tests obtenido mediante el algoritmo propuesto en [CG98] puede ser utilizado para buscar redundancias, reduciéndose el número de tests de forma que se pueden construir conjuntos de tests de aplicabilidad práctica manteniendo la completitud del análisis. Desgraciadamente, aunque estas ideas se presentan como una metodología, no se proporciona un procedimiento automático para llevar a cabo la citada reducción. El modelo temporal propuesto está basado en reglas de la forma "Si G entonces A entre L y U", donde G es una guarda sobre variables y relojes, A es una acción sobre estos, y L y U son los límites superior e inferior, respectivamente, del tiempo que puede ser invertido en la transición. Las hipótesis que se establecen para obtener conjuntos de tests completos requieren que si dos sistemas no son equivalentes sus comportamientos difieran en al menos una unidad de tiempo, que la implementación sea determinista y que ésta pueda reinicializarse.

En [Car99] el trabajo anterior es adaptado al lenguaje usado en UPPAAL para representar autómatas temporales. El método aborda el problema de la *intesteabilidad* de las computaciones de autómatas temporales y trata de proporcionar un método práctico para el testing de la conformidad de sistemas de tiempo real. Se pone de manifiesto que aunque el método propuesto en [SVD01] trata con una clase muy general de autómatas temporales, genera un número astronómico de tests, y por ello se propone un método para reducir el número de tests. Este método se basa en propósitos de tests que se representan mediante autómatas temporales que pueden presentar variaciones respecto a la especificación original. Se espera que cada modelo generado sea más simple que la especificación original y genere un número de tests abordable. Básicamente, la técnica consiste en dividir la especificación en diferentes propósitos de tests. En base a cada uno de ellos se generan nuevas versiones de la especificación, denominadas *test views*, cada una de las cuales sólo refleja un aspecto específico del sistema. Los tests se derivan a partir de cada *test view* mediante una adaptación del método propuesto en [Cho78]. El conjunto de tests generados para cada propósito de test es completo respecto a dicho propósito. El uso de esta aproximación de testing reduce parcialmente el número de tests. Sin embargo, la desventaja es que el testing aislado de cada propósito de test no permite, en general, detectar errores que pueden surgir debido a la interrelación de las funcionalidades asociadas a cada propósito.

En [Car00] se refina el trabajo anterior introduciendo un lenguaje intermedio en el proceso. Los autómatas temporales UPPAAL se transforman en *sistemas de transiciones tem-*

porales testeables mediante un *test view*. Además se propone un algoritmo de derivación de tests a partir de dichas representaciones. Estos conjuntos de tests son completos respecto al *test view* elegido.

En [CL97, CKL97] se propone un marco para el testeo de restricciones temporales de sistemas. Aunque en este marco de trabajo aparecen conceptos usuales de testing, los tests se aplican a un *modelo* en lugar de a una implementación. En particular, la técnica se presenta como un método para *validar* propiedades en el modelo del sistema. Los tests se derivan automáticamente a partir de las especificaciones, considerándose tan sólo los tiempos máximos y mínimos permitidos entre entradas y salidas durante la ejecución del sistema. Al contrario de las propuestas anteriores, las restricciones temporales se definen mediante un formalismo diferente al utilizado para describir las especificaciones. En concreto, el esquema de derivación de tests utiliza un formalismo de especificación gráfico para los requerimientos temporales y el álgebra de procesos temporal ACSR [BLG93, BL97] para representar los tests y los modelos. ASCR está basado en el modelo de sincronización de CCS que incluye características para representar tiempo, recursos, sincronización y prioridades. Los autores argumentan que el uso de un lenguaje expresivo, como ACSR, que proporciona precisión semántica para describir los tests aporta dos ventajas. Primero, los tests pueden aplicarse a un modelo ACSR del sistema dentro del marco semántico con el propósito de validar el modelo. En segundo lugar, ACSR tiene una notación concisa y una semántica precisa que facilitan dicho propósito.

Los autores proponen su método de validación como un medio para analizar sistemas complejos. De hecho, el número de tests es elegido por el testeador, por lo que puede usarse para validar un diseño que tenga muchos estados, evitando realizar un análisis exhaustivo del espacio de estados. El algoritmo propuesto considera todos los tests necesarios para alcanzar el criterio de cobertura establecido y selecciona aquellos que parecen ser más representativos.

En [HLN+03] los tests se generan automáticamente a partir de especificaciones representadas mediante autómatas temporales. Este trabajo está enfocado a la generación de tests con un *tiempo óptimo de ejecución*. La técnica permite que los tests se generen manualmente, mediante propósitos de tests establecidos, o automáticamente, mediante la aplicación de diferentes criterios de cobertura para el modelo. Para justificar la propuesta, los autores asumen que, en el contexto de sistemas de tiempo real, los tests más rápidos tienen más probabilidad de detectar errores. Además, argumentan que los conjuntos de tests óptimos en tiempo de ejecución reducen el tiempo total de su aplicación, lo que permite que se testeen más comportamientos del sistema en el tiempo limitado del que se dispone para esta tarea.

Por otra parte, los autores exponen que siempre es deseable que los tests se ejecuten lo más rápido posible, para mejorar así el tiempo de respuesta entre las diferentes revisiones del sistema.

La relación de conformidad aplicada en esta metodología es la inclusión de trazas. Cabe destacar que el método propuesto en este trabajo se basa en la existencia de técnicas eficientes de análisis simbólico de autómatas temporales. La principal contribución es su enfoque a la optimización del tiempo requerido en el proceso de testing así como la aplicación de criterios de cobertura con este fin. La mayoría de los trabajos de optimización de conjuntos de tests se centran en minimizar su tamaño, lo que no tiene por qué estar relacionado con la optimización del tiempo de ejecución.

En [KT04] se propone un marco para testing de caja negra en el que las especificaciones se representan como autómatas temporales, aunque en este caso se permite no determinismo y observabilidad parcial. La relación de conformidad considerada, *tioco*, es una extensión temporal de *ioco*. Esta relación establece que una implementación es conforme a una especificación si para cada comportamiento observable especificado, el conjunto de salidas en la implementación es un subconjunto de los que aparecen en la especificación. Esta relación se define mediante la asociación de valores temporales a los elementos del conjunto de salidas observables. Ello permite capturar la no conformidad de las implementaciones que emiten las salidas antes o después de lo especificado. Los autores comparan esta noción de conformidad con otras previamente consideradas, argumentando que la que ellos proponen es superior ya que da más libertad de diseño para las posibles implementaciones. Se proporcionan también algoritmos para la generación de dos tipos diferentes de tests: tests *analógicos*, para tiempo continuo, y tests *digitales*, para tiempo discreto.

El objetivo de los autores es superar algunas limitaciones de las metodologías previas. En primer lugar, plantean que dichas propuestas restringen las condiciones temporales que pueden ser descritas en las especificaciones. Por ejemplo, en [SVD01, HLN+03] no podría representarse una especificación que estableciera "cuando una entrada a es recibida, puede producirse o la salida b o la c". Tampoco sería posible expresar situaciones de la forma "cuando una entrada a es recibida, se produce la salida b antes de que transcurran t unidades de tiempo". Otras restricciones se plantean cuando las especificaciones deben ser deterministas u observables. Por el contrario, [KT04] permite no determinismo y observación parcial en las especificaciones.

Otra limitación se refiere a la *implementabilidad* de los tests. En referencia a la clásica controversia entre tiempo discreto y tiempo continuo, los autores manifiestan que, en la

práctica, solo los tests digitales pueden generarse y ser aplicados. Sin embargo, en los trabajos previos solo se consideran tests analógicos. Estos tests se limitan a la aplicación de las entradas en tiempos precisos y a la emisión de salidas en instantes de tiempo concretos. Por ejemplo, un test como "emite la salida a en tiempo 1; si en tiempo 5 se recibe la entrada b, emite el veredicto `pass` y para; en otro caso, emite el veredicto `fail`", es un test analógico. Desafortunadamente, los tests analógicos presentan el problema de que es difícil, si no imposible, su implementación con relojes de precisión finita. El testeador que implementa el test del ejemplo debe ser capaz de emitir a precisamente en tiempo 1 y comprobar que b ocurre exactamente en tiempo 5. Sin embargo, normalmente se comprueban las salidas de manera periódica, por ejemplo, cada 0,1 unidades de tiempo. Por tanto, no se puede determinar el instante concreto del intervalo (4,9, 5,1) en el que b se ha producido.

En [KT05] se extendió el trabajo anterior para permitir al testeador establecer hipótesis sobre el entorno de la implementación. Básicamente, estas hipótesis se expresan mediante un autómata temporal. Dicho autómata se compone con el autómata temporal que modela los requerimientos de la especificación. Además, para definir el interface entre los tests y la implementación se pueden usar autómatas temporales adicionales. Por ejemplo, pueden aplicarse para definir los lapsos de tiempo entre un estímulo del test y la recepción de una señal de la implementación. También se proponen algoritmos para la derivación de tests respecto a diferentes criterios de cobertura.

En [HNTC99] se proponen los *autómatas temporales I/O* extendidos con *datos* para modelar protocolos de tiempo real. Para realizar la derivación de los tests, los autómatas se transforman en una clase de máquinas de estados finitos que permiten la aplicación de técnicas clásicas de generación de conjuntos de tests.

Los autores consideran inadecuados los autómatas temporales clásicos [AD90, AD94] por no considerar datos, dado que éstos suelen ser muy relevantes en los protocolos de comunicación. Por ejemplo, en algunas ocasiones puede ser necesario especificar intervalos de tiempo que dependan del tamaño de los datos transmitidos. Por tanto, es necesario disponer de modelos que manejen datos además de tiempo, y de métodos de testing eficientes para los mismos. Con este objetivo los autores proponen una combinación de autómatas temporales con EFSMs.

A la hora de testear EFSMs cabe destacar que un test no es siempre ejecutable, siendo necesario encontrar los valores de los datos de entrada que satisfagan las condiciones de las transiciones que queremos ejecutar. En los sistemas de tiempo real el testeador puede decidir el instante en el que se aplican las entradas. Sin embargo, por lo general, no es posible

controlar el momento en el que se producen las salidas, el cual viene determinado por cada implementación. Es más, el tiempo de ejecución de algunas acciones puede depender de los tiempos de ejecución de otras acciones precedentes. Es deseable que independientemente de cuando se hayan producido las salidas previas, exista siempre un instante de tiempo que permita la aplicación de las entradas. De hecho, los tiempos de ejecución de las acciones de entrada pueden especificarse mediante una función dependiente de los tiempos de ejecución de las acciones previas. Los autores proponen un algoritmo para decidir si un test puede ser ejecutado y un método para derivar dicha automáticamente función a partir de los tests ejecutables.

En el modelo propuesto en [HNTC99] cada transición corresponde o a una entrada o a una salida. Para describir las restricciones temporales se utilizan variables que almacenan valores temporales o expresiones definidas sobre valores temporales y datos de entrada. Además, se dispone de una variable global especial que representa el reloj del sistema. Las condiciones de las transiciones se pueden especificar mediante una conjunción de desigualdades de todas estas variables.

La relación de conformidad considerada es un criterio de testing may-must. Para distinguir las secuencias que pueden ser ejecutadas siempre, independientemente de los tiempos en los que se producen las salidas, y aquellas otras que podrían llegar a su terminación, se definen dos tipos de secuencias de transiciones: Una secuencia *must-traceable* siempre puede ejecutarse si se especifican los tiempos adecuados para las acciones de entrada, independientemente de cuando se produzcan las salidas; Una secuencia *may-traceable* puede ejecutarse sólo cuando los tiempos de ejecución de las salidas lo hacen posible.

En el trabajo se presenta un algoritmo para chequear el carácter de las secuencias de test y obtener las cotas superior e inferior de los tiempos de aplicación de las entradas en función de los tiempos de ejecución de los acciones que las preceden. Basado en el método UIOv [VCI90], se propone un método de chequeo de conformidad para los modelos. El método se aplica a una FSM derivada del autómata mediante la eliminación de las condiciones temporales de las transiciones.

A continuación, se comentan brevemente otras propuestas para testear sistemas temporales.

- [EDKE98, EDK02] presenta una adaptación del método Wp [FBK+91] para sistemas temporales. Como en [Car00, SVD01], los tests se generan a partir de un autómata temporal mediante la aplicación de variantes de las técnicas disponibles para máquinas de estados finitos a una discretización del espacio de estados. Consecuentemente, esta

propuesta sufre el problema de la explosión de estados y produce un número muy elevado de tests.

- [NS01] propone un método completamente automático para la generación de tests a partir de una subclase de autómatas para tiempo continuo conocidos como *event-recording automata* [AFH94]. Esta aproximación esta basada en la teoría de testing de Nicola & Hennessy [dNH84] mediante el análisis simbólico de una amplia partición en clases de equivalencia del espacio de estados. La relación de conformidad aplicada es un preorden may-must.

- [Kho02] considera un modelo de autómatas temporales restringido, donde todas las transiciones que producen la misma observación reinician el mismo conjunto de relojes. El autómata temporal es inicialmente transformado en un autómata alternativo donde las condiciones de los relojes se representan mediante dos eventos : *set-timer* y *expire-timer*. Basado en esta representación se utiliza una generalización del método Wp para producir las secuencias de chequeo.

- [CKL98] presenta una aproximación diferente para la generación y selección de tests. Al igual que en algunas de las propuestas previas, se utiliza un propósito de test para determinar las secuencias de la especificación que se quieren chequear. Una composición del propósito de test y del autómata temporal del modelo se utiliza para obtener una secuencia simbólica con restricciones temporales que permite alcanzar el objetivo perseguido con el propósito de test. Esta traza simbólica puede interpretarse en tiempo de ejecución para dar un veredicto.

- [RNHW98] plantea un método particular para la derivación de las secuencias de entradas del sistema mas relevantes.

- [PF99] propone una técnica para transformar un grafo de regiones en un grafo en el que las restricciones temporales se expresan mediante etiquetas específicas usando *clock zones*.

Capítulo 4

Conclusiones y trabajo futuro

El principal objetivo de esta tesis, tal como se expuso en la introducción, es la extensión de los métodos formales utilizados en las metodologías para el *testing* de sistemas, de modo que estos puedan aplicarse a sistemas que presenten restricciones temporales y/o probabilísticas referentes a la ejecución de las acciones que en ellos pueda tener lugar.

La tesis recoge diferentes extensiones de formalismos de modo que éstos puedan describir restricciones temporales referentes a el tiempo consumido por las acciones, el tiempo de espera del sistema para recibir una reacción del entorno, la probabilidad de que una acción tenga lugar y la probabilidad de que una acción consuma una cantidad de tiempo determinada en su ejecución. Asimismo, la tesis aborda la aplicación de tests para la comprobación de dichas propiedades y la obtención de diagnósticos respecto a la corrección de los sistemas.

Los trabajos que se recogen en esta tesis consideran diferentes restricciones de aplicabilidad que están presentes en las distintas metodologías de testing y se proponen posibles soluciones para la superación de las mismas. Algunas de estas restricciones están exclusivamente relacionadas con los requerimientos temporales de los sistemas, mientras otras son de ámbito general, afectando a técnicas que se aplican a sistemas en los que las restricciones temporales no están presentes.

Cabe destacar que una limitación muy severa que presentan las metodologías de testing que consideran sistemas con requerimientos temporales es la consideración de un dominio temporal específico para la representación de las mismas, que en la mayoría de los casos corresponde a tiempos fijos. Ello impide la aplicación de dichas técnicas a sistemas que utilizan otras nociones temporales. Con el fin de proporcionar una metodología que pueda adaptarse con facilidad a sistemas que consideren diferentes dominios temporales, se ha propuesto un marco de testing integrado que puede ser utilizado para tratar con tiempos

fijos, intervalos temporales o variables aleatorias, en función de la especificación requerida por el sistema. El uso de esta nueva metodología de testing proporciona a los diseñadores de los sistemas flexibilidad a la hora de seleccionar el dominio temporal más adecuado en cada caso, así como la posibilidad de aplicarlo en un amplio número de especificaciones en los que las nociones temporales requeridas presentan diferentes grados de precisión.

Otra restricción a tener en cuenta a la hora de aplicar una técnica de testing que proporcione un diagnóstico de corrección de nuestro sistema es la precisión exigida por las mismas, tanto en la especificación de los requerimientos temporales como en los comportamientos observados durante su ejecución. Sin embargo, no siempre es posible especificar con exactitud los límites temporales asociados con las acciones que el sistema puede llevar a cabo. La superación de esta limitación ha motivado la propuesta de un marco formal de testing donde se permite por una parte, cierta indeterminación en la especificación de las restricciones temporales del sistema sin necesidad de hacer uso de información probabilística, y por otra, considerar cierto nivel de imprecision a la hora de establecer la corrección temporal de las implementaciones.

Continuando con el objetivo de reducir las restricciones asociadas a la aplicación de una metodología de testing especifica, se ha abordado el inconveniente que supone habitualmente la necesidad de que la estructura de las implementaciones a testear cumplan ciertas hipótesis. Obviamente, un marco de hipótesis establecido a priori es muy estricto y limita la utilidad de la técnica que los exige. En este caso, esta desventaja está presente tanto en metodologías orientadas al testing de sistemas con restricciones temporales, como en aquellas en las que los aspectos temporales no son críticos. Con el fin de eliminar esta desventaja en el ámbito de los sistemas temporales se ha extendido el marco $\mathcal{HOTL}$: *Hypotheses and Observations Testing Logic*, para trabajar con este tipo de sistemas.

Además de los requerimientos temporales asociados al tiempo de ejecución de acciones por parte del sistema, existen otras situaciones en las que es conveniente tener en cuenta el paso del tiempo: la necesidad de incluir time-outs en los modelos. Las extensiones temporales de las metodologías clásicas de testing están usualmente enfocadas tan sólo a una de las variantes previamente indicadas: el tiempo está asociado o bien con acciones o bien con time-outs. Con el fin de permitir la inclusión en los modelos de ambas restricciones temporales simultáneamente se ha propuesto un nuevo marco de testing en el que los sistemas que las presentan pueden ser fácilmente representados. Hay que indicar que la presencia de time-outs debe tenerse en cuenta a la hora de establecer la conformidad funcional de las implementaciones, ya que los comportamientos correctos están condicionados por los posibles

tiempos de espera establecidos, lo que complica considerablemente la combinación de ambos en una misma técnica.

Como continuación de esta línea de integración en un mismo marco de los tiempos de ejecución de las acciones y de los tiempos de espera, se ha desarrollo una nueva metodología en la que se consideran las posibles indecisiones en la especificación de los requerimientos temporales de las acciones de los sistemas. Para ello consideramos el uso de modelos estocásticos que permiten la especificación de los primeros mediante una estimación probabilística. En lo referente a la conformidad estocástico-temporal de los sistemas, el hecho de asumir un marco de testing de caja negra impide determinar si las acciones de la implementación tienen asociada una variable aleatoria idénticamente distribuida a la correspondiente en la especificación. Por tanto, en este marco se ha propuesto una relación de conformidad novedosa basada en la compatibilidad de un conjunto finito de tiempos de ejecución observados con la correspondiente variable aleatoria de la especificación. Esta compatibilidad se establece mediante un contraste de hipótesis. También en el ámbito de tiempos estocásticos se ha propuesto una metodología de testing para especificaciones representadas mediante stream X-Machines, formalismo para el que no se había propuesto hasta el momento ninguna técnica de testing que considere los aspectos temporales de los sistemas.

Con el objetivo de capturar diferentes nociones de conformidad estocástico-temporales y proponer un marco de testing más general, y por tanto de mayor aplicabilidad, se ha desarrollado una metodología, cuya adaptabilidad se resuelve mediante la definición de una relación de conformidad parametrizada que puede instanciarse con la noción de conformidad más adecuada en cada caso. Asimismo se propone un algoritmo que genera un conjunto de tests mínimo que permite determinar si una implementación es conforme a la especificación para un nivel de confianza establecido.

Además de considerar aspectos puramente temporales, esta tesis también representa una importante contribución al estado del arte en la especificación y testing de sistemas con componentes probabilísticos. Esta componente probabilística permite cuantificar el no determinismo de los sistemas, por lo que la descripción de estos sistemas incluye la probabilidad de que las acciones se ejecuten. Ello ha llevado al desarrollo de una metodología de testing basada en la técnica de mutación para la detección de errores en sistemas que presentan información probabilística. Una vez establecido este marco se ha propuesto una extensión para tratar con modelos estocástico-temporales. La contribución más relevante de esta tesis es el tratamiento simultáneo de dos tipos de comportamientos no-funcionales, así como la determinación en tiempo polinómico de la equivalencia de mutantes, lo que presenta dificultades

muy significativas en la técnica de testing por mutación.

Esta tesis también ha realizado una contribución modesta, aunque relevante, en un área en el que todavía la investigación usando métodos formales se encuentra en una etapa muy preliminar. Me refiero al estudio de testing en una arquitectura distribuida para sistemas no deterministas. Los trabajos que se habían desarrollado en este área consideran máquinas de estados finitos deterministas. Sin embargo, la mayoría de los sistemas que se enmarcan en este tipo de arquitectura suelen ser no deterministas. Por ello se ha propuesto su estudio considerando un formalismo más adecuado para su representación, *I/O transition systems*. Con el fin de evitar problemas de controlabilidad, de modo que los tests no puedan generar una situación en la que un testeador local tenga que aplicar una entrada o esperar una salida dependiendo de lo que ha ocurrido en otro puerto, se ha definido una caracterización formal de los tests controlables.

Respecto a líneas de trabajo futuro, los trabajos presentados en esta tesis han cerrado muchas líneas de investigación en el área del testing formal de sistemas probabilísticos y/o temporales. Por ello, en el futuro cercano no nos planteamos trabajar en el campo de testing *activo* de este tipo de sistemas. Sin embargo, en el área de testing *pasivo* el aspecto temporal de los sistemas no ha sido tratado de forma adecuada hasta el momento. Esta técnica de testing permite abordar el testeo de sistemas en los que es muy difícil o imposible interactuar con la implementación, como en el caso de sistemas cuya ejecución no puede interrumpirse durante un un largo periodo de tiempo. Por todo ello, consideramos que la creación de un marco de testing pasivo que permita expresar y analizar propiedades temporales sobre la duración de las acciones podría ser de gran utilidad para este tipo de sistemas, cuando el tiempo es un aspecto crítico en su comportamiento. Esta es una linea de investigación lo suficientemente amplia para que, sin lugar a dudas, de lugar a una nueva tesis doctoral.

Bibliografía

[ABC+95] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets.* John Wiley & Sons, 1995.

[Abr96] J.R. Abrial. *The B-Book: Assigning Programs to Meanings.* Cambridge University Press, 1996.

[ACB84] M. Ajmone Marsan, G. Conte, and G. Balbo. A class of generalized stochastic petri nets for the performance evaluation of multiprocessor systems. *ACM Transactions on Computer Systems*, 5(2):93–122, 1984.

[ACD93] R. Alur, C. Courcoubetis, and D. Dill. Model checking in dense real time. *Information and Computation*, 104:2–34, 1993.

[ACH94] R. Alur, C. Courcoubetis, and T.A. Henzinger. The observational power of clocks. In *5th Int. Conf. on Concurrency Theory, CONCUR'94, LNCS 836*, pages 162–177. Springer, 1994.

[AD90] R. Alur and D. Dill. Automata for modeling real-time systems. In *17th Int. Colloquium on Automata, Languages and Programming, ICALP'90, LNCS 443*, pages 322–335. Springer, 1990.

[AD94] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[AFH94] R. Alur, L. Fix, and T. Henzinger. A determinizable class of timed automata. In *6th Int. Conf. on Computer Aided Verification, CAV'94, LNCS 818*, pages 1–13. Springer, 1994.

[AG99] M. Abadi and A. Gordon. A calculus for cryptographic protocols: The Spi calculus. *Information and Computation*, 148:1–70, 1999.

[AH94] R. Alur and T.A. Henzinger. Real-time system = discrete time + clock variables. In *Theories and Experiences for Real-Time System Development, 1st AMAST Workshop on Real-Time System Development*, pages 1–29. World Scientific, 1994.

[AW96] S.P. Allen and M.R. Woodward. Assessing the quality of specification-based testing. In *3rd Int. Conf. on Achieving Quality in Software, AQuIS'96*, pages 341–354. Chapman & Hall, 1996.

[BB91] J.C.M. Baeten and J.A. Bergstra. Real time process algebra. *Formal Aspects of Computing*, 3(2):142–188, 1991.

[BB96] J.C.M. Baeten and J.A. Bergstra. Discrete time process algebra. *Formal Aspects of Computing*, 8(2):188–208, 1996.

[BB04] L. Brandán Briones and E. Brinksma. A test generation framework for quiescent real-time systems. In *4th Int. Workshop on Formal Approaches to Testing of Software, FATES'04, LNCS 3395*, pages 64–78. Springer, 2004.

[BB05] L. Brandán Briones and E. Brinksma. Testing real-time multi input-output systems. In *7th Int. Conf. on Formal Engineering Methods, ICFEM'05, LNCS 3785*, pages 264–279. Springer, 2005.

[BBS95] J.C.M. Baeten, J.A. Bergstra, and S.A. Smolka. Axiomatizing probabilistic processes: ACP with generative probabilities. *Information and Computation*, 121(2):234–255, 1995.

[BCG+99] T. Balanescu, A.J. Cowling, H. Georgescu, M. Gheorghe, M. Holcombe, and C. Vertan. Communicating stream X-Machines systems are no more than X-Machines. *Journal of Universal Computer Science*, 5(9):494–507, 1999.

[BDM+98] M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine. KRONOS: A model-checking tool for real-time systems. In *10th Int. Conf. on Computer Aided Verification, CAV'98, LNCS 1427*, pages 546–550. Springer, 1998.

[BG98] M. Bernardo and R. Gorrieri. A tutorial on EMPA: A theory of concurrent processes with nondeterminism, priorities, probabilities and time. *Theoretical Computer Science*, 202(1-2):1–54, 1998.

[BHKR95] S. Bradley, W. Henderson, D. Kendall, and A. Robson. Verification, validation and implementation of timed protocols using AORTA. In *15th WG6.1 Int. Conf. on Protocol Specification, Testing, and Verification, PSTV'95*, pages 205–220. Chapman & Hall, 1995.

[BL97] P. Brémond-Grégoire and I. Lee. A process algebra of communicating shared resources with dense time and priorities. *Theoretical Computer Science*, 189(1-2):179–219, 1997.

[BLG93] P. Brémond-Grégoire, I. Lee, and R. Gerber. ACSR: An algebra of communicating shared resources with dense time and priorities. In *4th Int. Conf. on Concurrency Theory, CONCUR'93, LNCS 715*, pages 417–431. Springer, 1993.

[BLL+96] J. Bengtsson, K.G. Larsen, F. Larsson, P. Pettersson, and W.Yi. UPPAAL - a tool suite for automatic verification of real-time systems. In *Hybrid Systems III, LNCS 1066*, pages 232–243. Springer, 1996.

[BLL+98] J. Bengtsson, K.G. Larsen, F. Larsson, P. Petterson, W. Yi, and C. Weise. New generation of UPPAAL. In *Int. Workshop on Software Tools and Technology Transfer*, 1998.

[BM99] L. Bottaci and E.S. Mresa. Efficiency of mutation operators and selective mutation strategies: An empirical study. *Software Testing, Verification and Reliability*, 9(4):205–232, 1999.

[BM04] M. Bidoit and P. Mosses, editors. *CASL Reference User Manual: Introduction to using the Common Algebraic Specification Language (LNCS 2900)*. Springer, 2004.

[BP94] G. von Bochmann and A. Petrenko. Protocol testing: Review of methods and relevance for software testing. In *ACM Int. Symposium on Software Testing and Analysis, ISSTA'94*, pages 109–123. ACM Press, 1994.

[BPBM97] G. von Bochmann, A. Petrenko, O. Bellal, and S. Maguiraga. Automating the process of test derivation from SDL specifications. In *8th Int. SDL Forum, SDL'97*, pages 261–276. Elsevier, 1997.

[BPS01] J.A. Bergstra, A. Ponse, and S.A. Smolka, editors. *Handbook of Process Algebra*. North Holland, 2001.

[BW90] J.C.M. Baeten and W.P. Weijland. *Process Algebra*. Cambridge Tracts in Computer Science 18. Cambridge University Press, 1990.

[Car99] R. Cardell-Oliver. Conformance testing of real-time systems against timed automata specifications. Technical Report CSM-330, University of Essex, 1999.

[Car00] R. Cardell-Oliver. Conformance tests for real-time systems with timed automata specifications. *Formal Aspects of Computing*, 12(5):350–371, 2000.

[Cas93] C.G. Cassandras. *Discrete Event Systems. Modeling and Performance Analysis*. Aksen Associates - Irwin, 1993.

[CCI88] SDL: Specification and design language, 1988. CCITT Recommendations Z.101-Z.104, *Blue Book Series*. Consultative Committe for International Telegraph and Telephone.

[CCV+03] D. Cazorla, F. Cuartero, V. Valero, F.L. Pelayo, and J.J. Pardo. Algebraic theory of probabilistic and non-deterministic processes. *Journal of Logic and Algebraic Programming*, 55(1–2):57–103, 2003.

[CCVP01] D. Cazorla, F. Cuartero, V. Valero, and F.L. Pelayo. A process algebra for probabilistic and nondeterministic processes. *Information Processing Letters*, 80:15–23, 2001.

[CG98] R. Cardell-Oliver and T. Glover. A practical and complete algorithm for testing real-time systems. In *5th Int. Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems, FTRTFT'98, LNCS 1486*, pages 251–260. Springer, 1998.

[Cho78] T.S. Chow. Testing software design modelled by finite state machines. *IEEE Transactions on Software Engineering*, 4:178–187, 1978.

[Cho86] N. Choquet. Test data generation using a prolog with constraints. In *Workshop on Software Testing*, pages 132–141. IEEE Computer Society Press, 1986.

[CHR82] L.A. Clarke, J. Hassell, and D.J. Richardson. A close look at domain testing. *IEEE Transactions on Software Engineering*, 8(4):380–390, 1982.

[Chr90] I. Christoff. Testing equivalences and fully abstract models for probabilistic processes. In *1st Int. Conf. on Concurrency Theory, CONCUR'90, LNCS 458*, pages 126–140. Springer, 1990.

[CHR91] Z. Chaochen, C.A.R. Hoare, and A.P. Ravn. A calculus of durations. *Information Processing Letters*, 40:269–276, 1991.

[CKL97] D. Clarke, Y.S. Kim, and I. Lee. Automatic test generation for the analysis of a real-time system: Case study. In *3rd IEEE Real Time Technology and Applications Symposium, RTAS'97*, pages 112–124. IEEE Computer Society Press, 1997.

[CKL98] R. Castanet, O. Koné, and P. Laurençot. On the fly test generation for real-time protocols. In *7th Int. Conf. on Computer Communications and Networks, IC3N'98*, pages 378–387. IEEE Computer Society Press, 1998.

[CL97] D. Clarke and I. Lee. Automatic generation of tests for timing constraints from requirements. In *3rd Workshop on Object-Oriented Real-Time Dependable Systems, WORDS'97*, pages 199–206. IEEE Computer Society Press, 1997.

[CS94] D.A. Carrington and P.A: Stocks. A tale of two paradigms: Formal methods and software testing. In *Z User Workshop*, pages 51–68. Springer, Workshops in Computing, 1994.

[CTC01] H.Y. Chen, T.H. Tse, and T.Y. Chen. TACCLE: A methodology for object-oriented software testing at the class and cluster levels. *ACM Transactions on Software Engineering and Methodology*, 10(1):56–109, 2001.

[CTCC98] H.Y. Chen, T.H. Tse, F.T. Chan, and T.Y. Chen. In black and white: An integrated approach to class-level testing of object-oriented programs. *ACM Transactions on Software Engineering and Methodology*, 7(3):250–295, 1998.

[DB99] J. Derrick and E. Boiten. Testing refinements of state-based formal specifications. *Software Testing, Verification and Reliability*, 9(1):27–50, 1999.

[DF93] J. Dick and A. Faivre. Automating the generation and sequencing of test cases from model based specifications. In *1st. Int. Symposium of Formal Methods Europe, FME'96, LNCS 670*, pages 268–284. Springer, 1993.

[DF94] R.-K. Doong and P.G. Frankl. The ASTOOT approach to testing object-oriented programs. *ACM Transactions on Software Engineering and Methodology*, 3(2):101–130, 1994.

[dNH84] R. de Nicola and M.C.B. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.

[DOTY96] C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool Kronos. In *Hybrid Systems III, LNCS 1066*, pages 208–219. Springer, 1996.

[DU04] A.Y. Duale and M.Ü. Uyar. A method enabling feasible conformance test sequence generation for EFSM models. *IEEE Transactions on Computers*, 53(5):614–627, 2004.

[DY95] C. Daws and S. Yovine. Two examples of verification of multirate timed automata with KRONOS. In *16th IEEE Real-Time Systems Symposium, RTSS'95*, pages 66–75. IEEE Computer Society Press, 1995.

[ED03] A. En-Nouaary and R. Dssouli. A guided method for testing timed input output automata. In *15th Int. Conf. on Testing Communicating Systems, TestCom'03, LNCS 2644*, pages 211–225. Springer, 2003.

[EDK02] A. En-Nouaary, R. Dssouli, and F. Khendek. Timed Wp-method: Testing real time systems. *IEEE Transactions on Software Engineering*, 28(11):1024–1039, 2002.

[EDKE98] A. En-Nouaary, R. Dssouli, F. Khendek, and A. Elqortobi. Timed test cases generation based on state characterization technique. In *19th IEEE Real Time Systems Symposium, RTSS'98*, pages 220–231. IEEE Computer Society Press, 1998.

[Eil74] S. Eilenberg. *Automata, languages and machines*, volume A. Academic Press, 1974.

[FBK+91] S. Fujiwara, G. von Bochmann, F. Khendek, M. Amalou, and A. Ghedamsi. Test selection based on finite-state models. *IEEE Transactions on Software Engineering*, 17(6):591–603, 1991.

[FPS01] H. Fouchal, E. Petitjean, and S. Salva. An user-oriented testing of real time systems. In *IEEE Workshop on Real-Time Embedded Systems, RTES'01*. IEEE Computer Society Press, 2001.

[Gau95] M.-C. Gaudel. Testing can be formal, too! In *6th Int. Joint Conf. CAAP/FASE, Theory and Practice of Software Development, TAPSOFT'95, LNCS 915*, pages 82–96. Springer, 1995.

[Gau01] M.-C. Gaudel. Testing from formal specifications, a generic approach. In *6th Int. Conf. Ada-Europe, LNCS 2043*, pages 35–48. Springer, 2001.

[GG75] J.B. Goodenough and S.L. Gerhart. Toward a theory of test data selection. *IEEE Transactions on Software Engineering*, 1(2):156–173, 1975.

[GJ98] M.-C. Gaudel and P.R. James. Testing algebraic data types and processes: A unifying theory. *Formal Aspects of Computing*, 10(5-6):436–451, 1998.

[GJS90] A. Giacalone, C.-C. Jou, and S.A. Smolka. Algebraic reasoning for probabilistic concurrent systems. In *IFIP WG2.3 Conf. on Programming Concepts and Methods*. North Holland, 1990.

[GLNP97] C. Gregorio, L. Llana, M. Núñez, and P. Palao. Testing semantics for a probabilistic-timed process algebra. In *4th International AMAST Workshop on Real-Time Systems, Concurrent, and Distributed Software, LNCS 1231*, pages 353–367. Springer, 1997.

[Gly89] P.W. Glynn. A GSMP formalism for discrete event simulation. *Proceedings of the IEEE*, 77(1):14–23, 1989.

[GMH81] J.D. Gannon, P.R. McMullin, and R.G. Hamlet. Data-abstraction implementation, specification, and testing. *ACM Transactions on Programming Languages and Systems*, 3(3):211–223, 1981.

[Gon70] G. Gonenc. A method for the design of fault detection experiments. *IEEE Transactions on Computers*, 19:551–558, 1970.

[Gro91] J.F. Groote. Specification and verification of real time systems in ACP. In *10th WG6.1 Int. Conf. on Protocol Specification, Testing, and Verification, PSTV'90*, pages 261–274. North-Holland, 1991.

[GSS95] R. van Glabbeek, S.A. Smolka, and B. Steffen. Reactive, generative and stratified models of probabilistic processes. *Information and Computation*, 121(1):59–80, 1995.

[GT79] J.A. Goguen and J.J. Tardo. An introduction to OBJ: A language for writing and testing formal algebraic specifications. In *IEEE Conf. on Specifications of Reliable Software*, pages 170–189. IEEE Computer Society Press, 1979.

[Hal89] P.A.V. Hall. Towards testing with respect to formal specification. In *2nd IEE/BCS Conference on Software Engineering*, pages 159–163, 1989.

[Han91] H. Hansson. *Time and Probability in Formal Design of Distributed Systems.* PhD thesis, Department of Computer Systems. Uppsala University, 1991.

[Har87] D. Harel. Statecharts: A visual formulation for complex systems. *Science of Computer Programming*, 8(3):231–274, 1987.

[Hen64] F.C. Hennie. Fault-detecting experiments for sequential circuits. In *5th Annual Symposium on Switching Circuit Theory and Logical Design*, pages 95–110, 1964.

[Hen85] M. Hennessy. Acceptance trees. *Journal of the ACM*, 32(4):896–928, 1985.

[Hen88] M. Hennessy. *Algebraic Theory of Processes.* MIT Press, 1988.

[Her90] U. Herzog. Formal description, time and performance analysis. a framework. In T. Härder, H. Wedekind, and G. Zimmermann, editors, *Entwurf und Betrieb verteilter Systeme, Fachtagung des Sonderforschungsbereiche 124 und 182*, pages 172–190. Springer, 1990.

[Her98] H. Hermanns. *Interactive Markov Chains.* PhD thesis, Universität Erlangen-Nürnberg, 1998. Also appeared as *LNCS 2428*, Springer, 2002.

[HG97] D. Harel and E. Gery. Executable object modeling with statecharts. *IEEE Computer*, 30(7):31–42, 1997.

[HH00] R.M. Hierons and M. Harman. Testing comformance to a quasi-non-determinstic stream X-machine. *Formal Aspects of Computing*, 12(6):423–442, 2000.

[HH04] R.M. Hierons and M. Harman. Testing conformance of a deterministic implementation to a non-deterministic stream X-machine. *Theoretical Computer Science*, 323(1–3):191–233, 2004.

[HHWT95] T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: The next generation. In *16th IEEE Real-Time Systems Symposium, RTSS'95*, pages 55–65. IEEE Computer Society Press, 1995.

[HI98] M. Holcombe and F. Ipate. *Correct Systems: Building a Business Process Solution.* Springer, 1998.

[Hie97] R.M. Hierons. Testing from a Z specification. *Software Testing, Verification and Reliability*, 7(1):19–33, 1997.

[Hie04] R.M. Hierons. Testing from a non-deterministic finite state machine using adaptive state counting. *IEEE Transactions on Computers*, 53(10):1330–1342, 2004.

[Hil96] J. Hillston. *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.

[HLN+03] A. Hessel, K. Larsen, B. Nielsen, P. Petterson, and A. Skou. Time-optimal real-time test case generation using UPPAAL. In *3rd Int. Workshop on Formal Approaches to Testing of Software, FATES'03, LNCS 2931*, pages 114–130. Springer, 2003.

[HM07] R.M. Hierons and M.G. Merayo. Mutation testing from probabilistic finite state machines. In *3rd Workshop on Mutation Analysis, Mutation'07*, pages 141–150. IEEE Computer Society Press, 2007.

[HM09] R.M. Hierons and M.G. Merayo. Mutation testing from probabilistic and stochastic finite state machines. *Journal of Systems and Software (in press)*, 2009.

[HMN08a] R.M. Hierons, M.G. Merayo, and M. Núñez. Controllable test cases for the distributed test architecture. In *6th Int. Symposium on Automated Technology for Verification and Analysis, ATVA'08, LNCS 5311*, pages 201–215. Springer, 2008.

[HMN08b] R.M. Hierons, M.G. Merayo, and M. Núñez. Implementation relations for the distributed test architecture. In *Joint 20th IFIP TC6/WG6.1 Int. Conf. on Testing of Software and Communicating Systems, TestCom'08, and 8th Int. Workshop on Formal Approaches to Software Testing, FATES'08, LNCS 5047*, pages 200–215. Springer, 2008.

[HNSY94] T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111:193–244, 1994.

[HNTC99] T. Higashino, A. Nakata, K. Taniguchi, and A. Cavalli. Generating test cases for a timed I/O automaton model. In *12th Int. Workshop on Testing of Communicating Systems, IWTCS'99*, pages 197–214. Kluwer Academic Publishers, 1999.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice Hall, 1985.

[Hol88] M. Holcombe. X–machines as a basis for dynamic system specification. *Software Engineering Journal*, 3(2):69–76, 1988.

[How82] W.E. Howden. Weak mutation testing and completeness of test sets. *IEEE Transactions on Software Engineering*, 8:371–379, 1982.

[HP92] P.G. Harrison and N.M. Patel. *Performance Modelling of Communication Networks and Computer Architectures.* Addison-Wesley, 1992.

[HR94] H. Hermanns and M. Rettelbach. Syntax, semantics, equivalences, and axioms for MTIPP. In *2nd Workshop on Process Algebra and Performance Modelling*, pages 71–88, 1994.

[HR95] M. Hennessy and T. Regan. A process algebra for timed systems. *Information and Computation*, 117(2):221–239, 1995.

[HSS01] R.M. Hierons, S. Sadeghipour, and H. Singh. Testing a system specified using statecharts and Z. *Information and Software Technology*, 43(2):137–149, 2001.

[HU02] R.M. Hierons and H. Ural. Reduced length checking sequences. *IEEE Transactions on Computers*, 51(9):1111–1117, 2002.

[IH97] F. Ipate and M. Holcombe. An integration testing method that is proved to find all faults. *International Journal of Computer Mathematics*, 63(3-4):159–178, 1997.

[IH00] F. Ipate and M. Holcombe. Generating test sets from non-deterministic stream X-machines. *Formal Aspects of Computing*, 12(6):443–458, 2000.

[Ipa04] F. Ipate. Complete deterministic stream X-machine testing. *Formal Aspects of Computing*, 16(4):374–386, 2004.

[IT97] ITU-T. *Recommendation Z.500 Framework on formal methods in conformance testing.* International Telecommunications Union, Geneva, Switzerland, 1997.

[ITU92] ITU. Recommendation Z.100: CCITT Specification and Description Language (SDL), 1992.

[Jal83] P. Jalote. Specification and testing of abstract data types. In *7th Int. Computer Software and Applications Conference, COMPSAC'83*, pages 508–511. IEEE Computer Society Press, 1983.

[JC88] P. Jalote and M.G. Caballero. Automated testcase generation for data abstraction. In *12th Int. Computer Software and Applications Conference, COMPSAC'88*, pages 205–210. IEEE Computer Society Press, 1988.

[Jon91] C.B. Jones. *Systematic Software Development using VDM.* Prentice Hall International, 2nd edition, 1991.

[Kho02] A. Khoumsi. A method for testing the conformance of real-time systems. In *7th Int. Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems, FTRTFT'02, LNCS 2469*, pages 331–354. Springer, 2002.

[KJG99] A. Kerbrat, T. Jéron, and R. Groz. Automated test generation from SDL specifications. In *9th Int. SDL Forum, SDL'99*, pages 135–152. Elsevier, 1999.

[Kle75] L. Kleinrock. *Queueing Systems.* John Wiley & Sons, 1975.

[Koh78] Z. Kohavi. *Switching and Finite State Automata Theory.* McGraw-Hill, 1978.

[KT04] M. Krichen and S. Tripakis. Black-box conformance testing for real-time systems. In *11th Int. SPIN Workshop on Model Checking of Software, SPIN'04, LNCS 2989*, pages 109–126. Springer, 2004.

[KT05] M. Krichen and S. Tripakis. An expressive and implementable formal framework for testing real-time systems. In *17th Int. Conf. on Testing of Communicating Systems, TestCom'05, LNCS 3502*, pages 209–225. Springer, 2005.

[Lay92] G.T. Laycock. *The Theory and Practice of Specification Based Software Testing.* PhD thesis, University of Sheffield, 1992.

[LdF99] L. Llana and D. de Frutos. Relating may and must testing semantics for discrete timed process algebras. In *5th Asian Computing Science Conference, ASIAN'99, LNCS 1742*, pages 74–86. Springer, 1999.

[Led91] G. Leduc. Conformance relation, associated equivalence, and minimum canonical tester in LOTOS. In *11th WG6.1 Int. Conf. on Protocol Specification, Testing, and Verification, PSTV'91*, pages 249–264. North Holland, 1991.

[LL97] L. Léonard and G. Leduc. An introduction to ET-LOTOS for the description of time-sensitive systems. *Computer Networks and ISDN Systems*, 29:271–292, 1997.

[LN00] N. López and M. Núñez. `NMSPA`: A non-markovian model for stochastic processes. In *International Workshop on Distributed System Validation and Verification (DSVV'2000)*, pages 33–40, 2000.

[LOT88] LOTOS. A formal description technique based on the temporal ordering of observational behaviour. IS 8807, TC97/SC21, 1988.

[LPU02] B. Legeard, F. Peureux, and M. Utting. A comparison of the BTT and TTF test-generation methods. In *2nd Int. Conf. of B and Z Users, LNCS 2272*, pages 309–329. Springer, 2002.

[LPY97] K.G. Larsen, P. Pettersson, and W. Yi. Uppaal in a nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1-2):134–152, 1997.

[LS89] K. Larsen and A. Skou. Bisimulation through probabilistic testing. In *16th ACM Simposium on Principles of Programming Languages, POPL'89*, pages 344–352. ACM Press, 1989.

[LS91] K. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, 1991.

[LSV03] N.A. Lynch, R. Segala, and F.W. Vaandrager. Hybrid I/O automata. *Information and Computation*, 185(1):105–157, 2003.

[LV96] N.A. Lynch and F.W. Vaandrager. Action transducers and timed automata. *Formal Aspects of Computing*, 8(5):499–538, 1996.

[LY96] D. Lee and M. Yannakakis. Principles and methods of testing finite state machines: A survey. *Proceedings of the IEEE*, 84(8):1090–1123, 1996.

[MHN08] M.G. Merayo, R.M. Hierons, and M. Núñez. Extending stream X-machines to specify and test systems with timeouts. In *6th IEEE Int. Conf. on Software Engineering and Formal Methods, SEFM'08*, pages 201–210. IEEE Computer Society Press, 2008.

[Mil80] R. Milner. *A Calculus of Communicating Systems (LNCS 92)*. Springer, 1980.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[Mil99] R. Milner. *Communicating and Mobile Systems: the π-Calculus*. Cambridge University Press, 1999.

[MMM95] D. Mandrioli, S. Morasca, and A. Morzenti. Generating test cases for real time systems from logic specifications. *ACM Transactions on Computer Systems*, 13(4):356–398, 1995.

[MN07] M.G. Merayo and M. Núñez. Testing conformance on stochastic stream X-machines. In *5th IEEE Int. Conf. on Software Engineering and Formal Methods, SEFM'07*, pages 227–236. IEEE Computer Society Press, 2007.

[MNR06a] M.G. Merayo, M. Núñez, and I. Rodríguez. Extending EFSMs to specify and test timed systems with action durations and timeouts. In *26th IFIP WG 6.1 Int. Conf. on Formal Techniques for Networked and Distributed Systems, FORTE'06, LNCS 4229*, pages 372–387. Springer, 2006.

[MNR06b] M.G. Merayo, M. Núñez, and I. Rodríguez. Implementation relations for stochastic finite state machines. In *3rd European Performance Engineering Workshop, EPEW'06, LNCS 3964*, pages 123–137. Springer, 2006.

[MNR07a] M.G. Merayo, M. Núñez, and I. Rodríguez. A brief introduction to $\mathcal{THOTL}$. In *5th Int. Symposium on Automated Technology for Verification and Analysis, ATVA'07, LNCS 4762*, pages 501–510. Springer, 2007.

[MNR07b] M.G. Merayo, M. Núñez, and I. Rodríguez. Formal testing of systems presenting soft and hard deadlines. In *2nd IPM Int. Symposium on Fundamentals of Software Engineering, FSEN'07, LNCS 4767*, pages 160–174. Springer, 2007.

[MNR07c] M.G. Merayo, M. Núñez, and I. Rodríguez. Generation of optimal finite test suites for timed systems. In *1st IEEE & IFIP Int. Symposium on Theoretical Aspects of Software Engineering, TASE'07*, pages 149–158. IEEE Computer Society Pres, 2007.

[MNR07d] M.G. Merayo, M. Núñez, and I. Rodríguez. Testing finite state machines presenting stochastic time and timeouts. In *4th European Performance Engineering Workshop, EPEW'07, LNCS 4748*, pages 97–111. Springer, 2007.

[MNR08a] M.G. Merayo, M. Núñez, and I. Rodríguez. $\mathcal{THOTL}$: A timed extension of $\mathcal{HOTL}$. In *Joint 20th IFIP TC6/WG6.1 Int. Conf. on Testing of Software and Communicating Systems, TestCom'08, and 8th Int. Workshop on Formal Approaches to Software Testing, FATES'08, LNCS 5047*, pages 86–102. Springer, 2008.

[MNR08b] M.G. Merayo, M. Núñez, and I. Rodríguez. Extending EFSMs to specify and test timed systems with action durations and timeouts. *IEEE Transactions on Computers*, 57(6):835–848, 2008.

[MNR08c] M.G. Merayo, M. Núñez, and I. Rodríguez. Formal testing from timed finite state machines. *Computer Networks*, 52(2):432–460, 2008.

[Moo56] E.P. Moore. Gedanken experiments on sequential machines. In C. Shannon and J. McCarthy, editors, *Automata Studies*. Princeton University Press, 1956.

[Mos04] P. Mosses, editor. *CASL Reference Manual: The Complete Documentation of the Common Algebraic Specification Language (LNCS 2960)*. Springer, 2004.

[MPW92] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes I & II. *Information and Computation*, 100:1–77, 1992.

[MT90] F. Moller and C. Tofts. A temporal calculus of communicating systems. In *1st Int. Conf. on Concurrency Theory, CONCUR'90, LNCS 458*, pages 401–415. Springer, 1990.

[NdF95] M. Núñez and D. de Frutos. Testing semantics for probabilistic LOTOS. In *8th IFIP WG6.1 Int. Conf. on Formal Description Techniques, FORTE'95*, pages 365–380. Chapman & Hall, 1995.

[NR03] M. Núñez and I. Rodríguez. Towards testing stochastic timed systems. In *23rd IFIP WG 6.1 Int. Conf. on Formal Techniques for Networked and Distributed Systems, FORTE'03, LNCS 2767*, pages 335–350. Springer, 2003.

[NS91] X. Nicollin and J. Sifakis. An overview and synthesis on timed process algebras. In *3rd Int. Conf. on Computer Aided Verification, CAV'91, LNCS 575*, pages 376–398. Springer, 1991.

[NS01] B. Nielsen and A. Skou. Automated test generation from timed automata. In *7th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems, TACAS'01, LNCS 2031*, pages 343–357. Springer, 2001.

[NSY92] X. Nicollin, J. Sifakis, and S. Yovine. Compiling real-time specifications into extended automata. *IEEE Transactions on Software Engineering*, 18(9):794–804, 1992.

[NT81] S. Naito and M. Tsunoyama. Fault detection for sequential machines. In *IEEE Fault Tolerant Computer Systems*, pages 238–243. IEEE Computer Society Press, 1981.

[Núñ03] M. Núñez. Algebraic theory of probabilistic processes. *Journal of Logic and Algebraic Programming*, 56(1–2):117–177, 2003.

[PBG04] A. Petrenko, S. Boroday, and R. Groz. Confirming configurations in EFSM testing. *IEEE Transactions on Software Engineering*, 30(1):29–42, 2004.

[PF99] E. Petitjean and H. Fouchal. From timed automata to testable untimed automata. In *24th IFAC/IFIP International Workshop on Real-Time Programming, WRTP'99*. Elsevier, 1999.

[Pha94] M. Phalippou. Executable testers. In *6th IFIP Int. Workshop on Protocol Test Systems, IWPTS'93*, pages 35–50. North-Holland, 1994.

[PS97] J. Peleska and M. Siegel. Test automation of safety-critical reactive systems. *South African Computer Journal*, 19:53–77, 1997.

[PY05] A. Petrenko and N. Yevtushenko. Testing from partial deterministic FSM specifications. *IEEE Transactions on Computers*, 54(9):1154–1165, 2005.

[PYB96] A. Petrenko, N. Yevtushenko, and G. von Bochmann. Testing deterministic implementations from their nondeterministic FSM specifications. In *9th IFIP Workshop on Testing of Communicating Systems, IWTCS'96*, pages 125–140. Chapman & Hall, 1996.

[QdFA93] J. Quemada, D. de Frutos, and A. Azcorra. TIC: A TImed Calculus. *Formal Aspects of Computing*, 5:224–252, 1993.

[RMN06] I. Rodríguez, M.G. Merayo, and M. Núñez. A logic for assessing sets of heterogeneous testing hypotheses. In *18th Int. Conf. on Testing Communicating Systems, TestCom'06, LNCS 3964*, pages 39–54. Springer, 2006.

[RMN08] I. Rodríguez, M.G. Merayo, and M. Núñez. $\mathcal{HOTL}$: Hypotheses and observations testing logic. *Journal of Logic and Algebraic Programming*, 74(2):57–93, 2008.

[RNHW98] P. Raymond, X. Nicollin, N. Halbwatchs, and D. Waber. Automatic testing of reactive systems. In *19th IEEE Real Time Systems Symposium, RTSS'98*, pages 200–209. IEEE Computer Society Press, 1998.

[SCS97] H. Singh, M. Conrad, and S. Sadeghipour. Test case design based on Z and the classification-tree method. In *1st IEEE Int. Conf. on Formal Engineering Methods, ICFEM'97*, pages 81–90. IEEE Computer Society Press, 1997.

[SD88] K. Sabnani and A. Dahbura. A protocol test generation procedure. *Computer Networks and ISDN Systems*, 15:285–297, 1988.

[SDJ+92] S. Schneider, J. Davies, D. M. Jackson, G.M. Reed, J.N. Reed, and A.W. Roscoe. Timed CSP: Theory and practice. In *Real-Time: Theory in Practice, REX Workshop, LNCS 600*, pages 640–675. Springer, 1992.

[SEK+98] M. Schmitt, A. Ek, B. Koch, J. Grabowski, and D. Hogrefe. Autolink - putting SDL-based test generation into practice. In *11th IFIP Workshop on Testing of Communicating Systems, IWTCS'98*, pages 227–244. Kluwer Academic Publishers, 1998.

[SGSL98] R. Segala, R. Gawlick, J.F. Søgaard-Andersen, and N. Lynch. Liveness in timed and untimed systems. *Information and Computation*, 141:119–171, 1998.

[She93] G.S. Shedler. *Regenerative Stochastic Simulation.* Academic Press, 1993.

[Sif77] J. Sifakis. Use of Petri nets for performance evaluation. In *3rd Int. Symposium on Measuring, Modelling and Evaluating Computer Systems*, pages 75–93. North-Holland, 1977.

[SMIM90] F. Sato, J. Munemori, T. Ideguchi, and T. Mizuno. Sequence generation tool for communication systems. In *2nd WG6.1 Int. Conf. on Formal Description Techniques, FORTE'89*, pages 1–5. North-Holland, 1990.

[Spi88] J.M. Spivey. *Understanding Z: A Specification Language and its Formal Semantics.* Cambridge University Press, 1988.

[Spi92] J.M. Spivey. *The Z Notation: A Reference Manual.* Prentice Hall International, 2nd edition, 1992.

[SVD01] J. Springintveld, F. Vaandrager, and P.R. D'Argenio. Testing timed automata. *Theoretical Computer Science*, 254(1-2):225–257, 2001. Previously appeared as Technical Report CTIT-97-17, University of Twente, 1997.

[Tre96] J. Tretmans. Test generation with inputs, outputs and repetitive quiescence. *Software – Concepts and Tools*, 17(3):103–120, 1996.

[Tre99] J. Tretmans. Testing concurrent systems: A formal approach. In *10th Int. Conf. on Concurrency Theory, CONCUR'99, LNCS 1664*, pages 46–65. Springer, 1999.

[UFSA99] M.Ü. Uyar, M.A. Fecko, A.S. Sethi, and P.D. Amar. Testing protocols modeled as FSMs with timing parameters. *Computer Networks*, 31(18):1967–1998, 1999.

[USW00] H. Ural, K. Saleh, and A. Williams. Test generation based on control and data dependencies within system specifications in SDL. *Computer Communications*, 23:609–627, 2000.

[UWZ97] H. Ural, X. Wu, and F. Zhang. On minimizing the lengths of checking sequences. *IEEE Transactions on Computers*, 46(1):93–99, 1997.

[Vas73] M.P. Vasilevskii. Failure diagnosis of automata. *Cybernetics*, 4:653–665, 1973.

[VCI90] S.T. Voung, W.L. Chan, and M.R. Ito. The UIOv-method for protocol test sequence generation. In *2nd IFIP TC6 Int. Workshop on Protocol Test Systems, IWPTS'89*, pages 161–175. North-Holland, 1990.

[WC80] L.J. White and E.I. Cohen. A domain strategy for computer program testing. *IEEE Transactions on Software Engineering*, 6(3):247–257, 1980.

[Whi80] W. Whitt. Continuity of generalized semi-markov processes. *Mathematics of Operational Research*, 5:494–501, 1980.

[Woo93] M.R. Woodward. Errors in algebraic specifications and an experimental mutation testing tool. *IEE/BCS Software Engineering Journal*, 8(4):211–224, 1993.

[Yi90] W. Yi. Real-time behavior of asynchronous agents. In *1st Int. Conf. on Concurrency Theory, CONCUR'90, LNCS 458*, pages 502–520. Springer, 1990.

[YPD95] W. Yi, P. Pettersson, and M. Daniels. Automatic verification of real-time communicating systems by constraint-solving. In *7th IFIP WG6.1 Int. Conf. on*

Formal Description Techniques, FORTE'94, pages 243–258. Chapman & Hall, 1995.

[Zub80] W.M. Zuberek. Timed Petri nets and preliminary performance evaluation. In *7th Annual Symposium on Computer Architecture*, pages 88–96. ACM Press, 1980.