Actas de las Jenui, vol. 3. 2018.
Páginas: 79-86

Asignatura de sistemas embebidos basada en la programación orientada a estados

Lluís Ribas Xirgo
Departament de Microelectrònica i Sistemes Electrònics
Universitat Autònoma de Barcelona
08193 Bellaterra
Lluis.Ribas@UAB.cat

Resumen

La mayoría de cursos de sistemas embebidos se ocupan de todo aquello necesario para el desarrollo del software a ejecutarse en plataformas hardware. Sin embargo, no enseñan ninguna metodología de programación específica. Dado que estos sistemas son, generalmente, de control, es conveniente emplear modelos de cálculo orientados a estado. La asignatura que se presenta cambia la estrategia de la enseñanza de los sistemas embebidos apostando por la programación orientada a estados. Los resultados que se presentan muestran que el grado de satisfacción de las personas que cursan esta asignatura es muy elevado, así como su percepción de la capacidad de diseñar sistemas embebidos.

Abstract

Most embedded systems courses cover everything necessary for software development to run on hardware platforms. However, they do not teach any specific programming methodology. Since these systems are generally controllers, it is advisable to use state-oriented computational models. The course that is presented changes the conventional strategy of teaching embedded systems' programming to state-oriented programming. Results show that the degree of satisfaction of the students with the course is very high, as well as their perception of the ability to design embedded systems.

Palabras clave

Programación orientada a estados, sistemas embebidos, sistemas empotrados.

1. Motivación

En la actualidad es difícil encontrar algún producto que contenga electrónica y no lleve software embebido. En este sentido, la mayoría de carreras universitarias en ingeniería informática o electrónica contemplan una o varias asignaturas que tratan sobre el diseño y desarrollo de sistemas embebidos.

Aprovechando la reforma del currículo en Ingeniería Informática de nuestra universidad, se decidió incorporar un perfil orientado a la Ingeniería de Computadores que incluyera la materia de los sistemas embebidos o empotrados.

Esta materia se organizaría en cuatro asignaturas distintas: Sistemas embebidos, Microprocesadores y periféricos, Integración hardware/software y Prototipado de sistemas embebidos. Así pues, la primera habría de convertirse en una asignatura básica de introducción a estos sistemas y a su diseño, pues las etapas de desarrollo e implementación se cubren con las demás. Por esto, Sistemas embebidos se situó en el primer semestre y las demás, en el segundo.

Todas son obligatorias en el tercer curso del grado en Ingeniería Informática para quienes quieran obtener la mención en Ingeniería de Computadores y optativas en el cuarto curso para los demás. Esto implica que el número de estudiantes pueda ser muy variable y, además, que tengan distinto perfil y estén cursando asignaturas de distintos años.

La asignatura de Sistemas embebidos se planteó de manera que las personas que la cursan pudieran aprender no solo los aspectos teóricos en cuanto a la elaboración y ejecución de proyectos de desarrollo de estos sistemas sino también aquellos aspectos que tienen que ver con su programación, obviando intencionadamente aspectos ligados a su implementación.

Existe abundante material que enseña a programar sistemas embebidos de manera que se aprenda a adecuar el perfil de ejecución de los programas a los requerimientos del sistema (memoria, tiempos, consumo, etcétera). Sin embargo, se olvida que la propia programación puede ser compleja.

En esta asignatura se enseña a utilizar la programación orientada a estados como una forma de simplificar el proceso de síntesis del software embebido. Además, con ello se facilita la verificación posterior del mismo y, sobre todo, su concepción inicial.

Para ofrecer una perspectiva aplicada del proceso de creación y desarrollo de los sistemas embebidos, se incluyen prácticas en las que se diseñan y programan las máquinas de estado que gobiernan el comportamiento de unos robots móviles simples.

En encuestas anónimas se pregunta a *las y los*[1] estudiantes sobre la percepción que tienen de su nivel de aprendizaje y su capacidad de enfrentarse al reto de diseñar un sistema embebido al final del curso. Las respuestas revelan un grado de satisfacción más que notable.

2. Contexto

2.1. Las asignaturas de sistemas embebidos en el mundo

La mayoría de universidades ofrecen grados en los que se incluyen asignaturas que cubren la temática de los sistemas embebidos. Se han tomado como referencia una selección de aquéllas que tienen una posición preferente en las clasificaciones de las universidades en el mundo.

Aprovechando el auge de los ordenadores de placa única (o SBPC, de las siglas de *single-board PC*) como los de las familias Arduino, BeagleBone o Raspberry, por citar algunas, muchas asignaturas incluyen un desarrollo práctico de algún caso de ejemplo.

Así pues, en Carnegie Mellon [17], la enseñanza de los sistemas embebidos está dirigida por proyectos, con unas pocas clases teóricas para facilitar el desarrollo de los productos de cada proyecto, desde el diseño de las placas hasta las cuestiones de seguridad y ética. Sin embargo, no abordan el tema del desarrollo del propio software, que suele ser de gran importancia. Aunque hay que tener en cuenta que resulta complicado compaginar, en un único semestre, la realización de un proyecto de asignatura y el aprendizaje de todos los aspectos que se requieren para el desarrollo de un proyecto.

En la Universidad de Ciencia y Tecnología de Hong Kong tienen varios cursos del tema [2], desde uno introductorio en que se desarrolla un pequeño sistema como ejemplo hasta uno que se centra más en el software embebido.

En la Universitad de Princeton se ofrecen cursos de sistemas embebidos de características similares a los descritos previamente, pero con unos títulos sugerentes: “Diseño de sistemas reales” y “Construcción de sistemas reales” [4]. De hecho, que sean embebidos implica que tengan que relacionarse con el mundo físico que les envuelve y, por lo tanto, que tengan que comportarse de acuerdo con unas restricciones fijadas no sólo por la aplicación sino también por la realidad.

Los cursos abiertos, como el de la Universidad de Texas en Austin [18], suelen estar vinculados a cursos de algún programa de estudios y siguen un esquema parecido al de los cursos ya descritos: mostrar el proceso de creación de un controlador a partir de un SBPC. Normalmente, el software acaba respondiendo al modelo de una máquina de estados.

En la universidad de California en Berkeley, sin embargo, la asignatura de Introducción a los sistemas embebidos [11, 12] tiene una orientación distinta, un poco menos práctica y más centrada en los aspectos del modelo computacional. La asignatura se inspira en los sistemas ciberfísicos en tanto que suelen emplearse para modelar controladores de entidades físicas como vehículos o plantas industriales. En este sentido, emplean los modelos orientados a estados (máquinas de estado) para mostrar una metodología de diseño completa, que incluye verificación formal de los modelos.

También hay cursos que se centran en el software, pero solo abordan unos pocos temas para poder llegar a la implementación. Los hay que tratan especialmente del mapeo entre programa y modelo de memoria de la plataforma de ejecución [8, 9], que muestran cómo adaptar la ingeniería del software a este campo [3] o cómo intercomunicar las tareas de manera que los sistemas de control distribuido funcionen correctamente [1], entre otras.

En cualquier caso, todos los cursos comparten el hecho de centrarse en unos pocos aspectos de los sistemas embebidos para que se puedan seguir adecuadamente en un semestre académico.

No hay, según nuestro conocimiento, cursos que muestren el proceso de creación de sistemas embebidos desde la perspectiva de la síntesis del software a partir de modelos de máquinas de estado.

2.2. Modelos de cálculo

El proceso de diseño de un sistema embebido empieza por definir su comportamiento, es decir, cuál es su funcionalidad y qué especificaciones no funcionales debe cumplir. Definirlo implica, primero, concebir qué tareas tiene que hacer y cómo se relacionan entre sí y, después, describirlas en un lenguaje de especificación apropiado.

En esta primera fase es importante organizar las tareas de acuerdo con un modelo de cálculo adecuado, bien orientado a flujo de datos o bien a estados. En ambos casos, se pueden representar como tuplas de nodos y arcos que forman un grafo.

En asignaturas precedentes del plan de estudios del grado en Ingeniería Informática de la UAB, los estudiantes se han familiarizado con los conceptos de grafos de máquinas de estados finitos (FSM) y de

[1] A lo largo del texto se empleará la expresión “los estudiantes” como referencia a un colectivo que incluye personas de ambos sexos. El porcentaje de mujeres en la asignatura está en el rango del 7 al 12% de su alumnado desde el curso 12/13 hasta el actual.

grafos de flujos de datos (DFG), aunque empleados para fines distintos. Sin embargo, no lo están con las máquinas de estado ampliadas o EFSM (del inglés, *extended finite-state machines*), que son FSM que, además de las expresiones lógicas, pueden contener expresiones aritméticas. En este sentido, también se amplía la noción de estado al contenido de todas las variables de la máquina, incluida la del propio estado.

Uno de los ejemplos que se emplea para introducir el modelo de las EFSM es el de un contador programable como el que se muestra en la figura 1.*a*: cada vez que recibe un pulso de inicio de cuenta (señal de entrada *begin*) empieza a contar de manera que la señal de salida *end* muestre un pulso a cero que dure *M* ciclos. (La variable *B* se encarga de guardar el valor de la señal de entrada *M*.) Al final de la cuenta, regresa al estado de espera. En este caso, es necesaria, además de la variable de estado, una variable, *C*, para contener el valor de la cuenta en cada ciclo.

Es importante hacer hincapié en la distinción entre *señales* y *variables*: una *señal* es una entrada o salida del módulo y una *variable* es un contenedor de datos que actúa como señal de entrada y salida, aunque no necesariamente sea accesible desde el exterior. De hecho, la variable de estado no lo tiene que ser, aunque puede determinar el valor de algunas señales de salida.

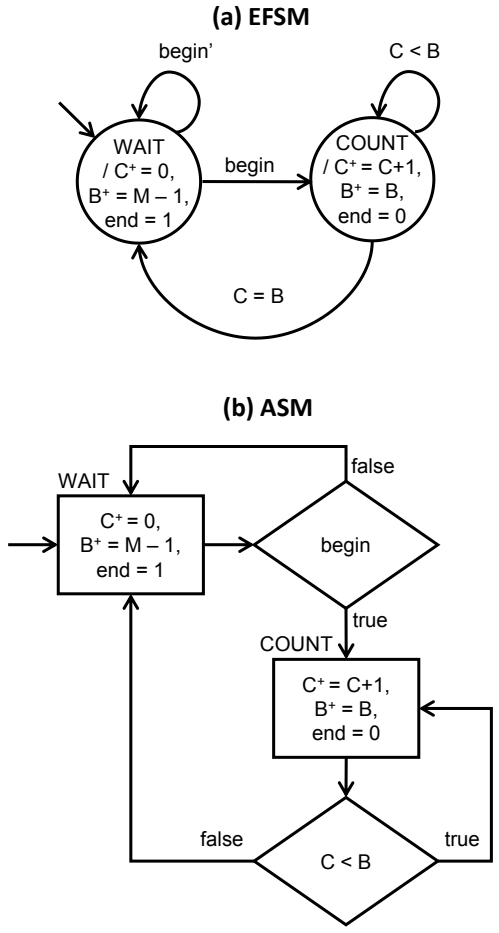

Figura 1: Grafos para representar máquinas de estado: (*a*) ampliadas con cálculos aritméticos y (*b*) en forma de algoritmo.

De la misma manera, cabe incidir en el funcionamiento de este modelo en cuanto a los valores que se emplean en los cálculos, que son los que se hallan en el estado presente, y respecto a las asignaciones, que pueden ser inmediatas (para señales de salida dependientes del estado actual) o retardadas, para variables. Así pues, en el ejemplo, la notación C^+ se refiere al valor de la variable en el ciclo siguiente.

Además del modelo de EFSM visto, en la asignatura también se trabaja con el de máquinas de estados algorítmicas (ASM), que son grafos de control de flujo de datos o CDFG (del inglés, *control data flow graphs*) y que facilitan la representación de sistemas de proceso de datos (véase la figura 1.*b*).

Para completar el espectro de modelos de cálculo, se presentan versiones jerárquicas y concurrentes de los anteriores.

Sea cuál sea el modelo de cálculo que se emplea para pensar un sistema, tiene que traducirse a algún lenguaje de especificación que permita avanzar en su proceso de desarrollo.

Los lenguajes de especificación pueden ser gráficos y también textuales, entre los que se incluyen los de programación.

Entre las ventajas de emplear un lenguaje de programación se hallan la facilidad de simular el sistema para su verificación y que la etapa de síntesis del software final es directa.

2.3. La programación orientada a estados

Se trata de un estilo de organización del código de la programación imperativa en la que la ejecución del programa se regula como una máquina de estados[2,3]. A diferencia de la programación habitual, hay que incluir variables de estado cuyo valor se utiliza para determinar qué código hay que ejecutar en cada período de tiempo o ciclo.

De hecho, podría verse al revés: se trata de reducir al máximo la parte implícita del estado de ejecución de un programa (el contenido de sus variables, el contador de programa y todo el estado de la unidad central de procesamiento) para facilitar tanto la síntesis a partir de modelos de máquinas de estado como el análisis de su comportamiento.

Un caso sencillo que ilustra este estilo de programación es el del controlador de una "máquina inútil[4]", que se apaga cuando se enciende [16].

[2] https://en.wikipedia.org/wiki/Automata-based_programming
[3] https://en.wikipedia.org/(…)ing_(Shalyto%27s_approach)
[4] https://en.wikipedia.org/wiki/Useless_machine

(a) Procedural

```
main = function()
  while true do
    while get_switch_pos()==0 do
      move_finger( 0 )
    end -- while
    while get_switch_pos()==1 do
      move_finger( 1 )
    end -- while
  end -- while
end -- main()
```

(b) Orientado a estados

```
main = function()
  state = "APAGADO" -- inicialización
  while true do -- bucle principal
    if state == "APAGADO" then
      move_finger( 0 )
      if get_switch_pos()==1 then state = "ENCENDIDO" end
    else -- state == "ENCENDIDO"
      move_finger( 1 )
      if get_switch_pos()==0 then state = "APAGADO" end
    end -- if
  end -- while
end -- main()
```

Figura 2: Estilo de programación procedural (*a*) y orientada a estados (*b*).

Esta máquina dispone de un conmutador externo para ser encendida y de un dedo mecánico que se mueve para accionar el conmutador externo en sentido contrario.

La programación convencional de su controlador incluye un bucle principal [14] que repite indefinidamente la espera de que se la encienda seguida de una acción para apagarse que se mantiene hasta conseguirlo (figura 2.*a*). Para ello, el programa consulta el estado del conmutador con `get_switch_pos()` y activa o desactiva el movimiento del dedo llamando a `move_finger()` con el parámetro adecuado.

En la programación orientada a estados (figura 2.*b*), el controlador tiene una inicialización y un bucle para simular la evolución de la máquina de estados correspondiente a lo largo del tiempo. Tiene la ventaja de tener un perfil de ejecución más simple y, por ejemplo, poder calcular en tiempo de ejecución máximo por iteración sin mayor dificultad.

De hecho, hay lenguajes específicos que simplifican la programación orientada a estados, tanto textuales, como Esterel [5, 13] o Lustre/SCADE [6, 7], como gráficos, como Ptolemy [15]. Sin embargo, se descartó su uso porque la experiencia previa con ellos en otras asignaturas mostró que los estudiantes tenían que invertir demasiado tiempo en su aprendizaje.

Por este motivo, se optó por emplear lenguajes de programación imperativos en los que los estudiantes ya tuvieran una cierta experiencia como C, C++, Java o Python. Esto facilita que aprendan cómo se sintetiza de código a partir de los modelos de cálculo. Para ello, se les proporciona un patrón de programación.

En este patrón, se separan las máquinas de estados del *motor de ejecución*, es decir, del bucle principal que las ejecuta.

Cada máquina es un objeto con su espacio para datos y métodos para su inicialización, la lectura de datos de entrada, el cálculo de valores del estado siguiente, la actualización de los valores del estado actual y el acceso a los datos de salida, entre otros.

En la asignatura se enseña cómo, a partir de este patrón básico, se pueden construir simuladores y controladores reales. Y también cómo evaluar el rendimiento de los sistemas y estrategias para su mejora como, por ejemplo, transformar el motor de ejecución para que la misma sea dirigida por sucesos (*event-driven*) o apoyarse en el sistema operativo de la plataforma en la que se embeban.

Desde el curso 12/13 (primera edición de la asignatura) hasta el curso 16/17 se trabajó con un patrón de programación en C. En la primera edición de la asignatura, los problemas se podían resolver en cualquier lenguaje de programación imperativo. En las ediciones siguientes, solo podía usarse C para agilizar el proceso de evaluación y, con ello, que la retroalimentación pudiera efectuarse en pocos días. (El número de estudiantes pasó de 18 a 49 en el curso 13/14, y a 75, el siguiente.)

En este último curso, se ha utilizado Lua[5] para facilitar la integración de contenidos entre la parte teórica y de problemas y la práctica, en que se emplea el simulador V-Rep[6].

Las prácticas de esta asignatura consisten en el desarrollo del software de control de unos pequeños robots móviles que se construyeron para este fin en el departamento de Microelectrónica y Sistemas Electrónicos de la UAB. Se trata de robots que se mueven con tracción diferencial y que tienen un sensor de distancia montado en un rotor para poder detectar obstáculos en ángulos frontales de 180º.

Para que los estudiantes pudieran trabajar fuera del laboratorio, se diseñó un modelo virtual del robot en este simulador, que utiliza Lua como lenguaje de programación de los elementos de las "escenas" que simula.

[5] http://www.lua.org
[6] http://www.coppeliarobotics.com

Además, los robots reales pueden comunicarse remotamente con V-Rep, con lo que no es necesario efectuar ninguna traducción de código, en tanto que los controladores se ejecutan en el simulador y la entrada/salida se hace mediante comunicación con un pequeño módulo que sí se ejecuta en los robots.

3. Organización de la asignatura

El objetivo principal de la asignatura es que las personas que la estudien adquieran una experiencia básica en el diseño de sistemas embebidos. Para ello, tienen que conocer el proceso de desarrollo, desde su concepción inicial hasta su comercialización, y comprender cómo se puede representar su comportamiento mediante máquinas de estado y los mecanismos para programarlo.

Como se trata de una asignatura de 6 créditos, se supone una dedicación por estudiante de 150 horas, 50 de las cuales, presenciales. Éstas se dividen en 13 clases de teoría (2 horas a la semana), 12 de resolución de problemas (1 hora a la semana) y 6 sesiones de prácticas (2 horas cada quince días).

Las clases de teoría se dividen en dos partes de una hora cada una. La primera parte se aprovecha para exponer tanto el proceso de desarrollo de los sistemas embebidos como diversos ejemplos de dominios de aplicación, desde la automoción hasta los *wearables*.

Por ejemplo, en el curso 17/18, las primeras horas de las clases de teoría trataron los temas siguientes:

- Introducción al diseño de sistemas embebidos.
- Modelos de cálculo.
- Arquitecturas.
- Flujo de diseño.
- Sistemas de verificación formal.
- Verificación mediante simulación.
- Sistemas operativos en tiempo real.
- Sistemas embebidos en el automóvil.
- Sistemas embebidos en el transporte ferroviario.
- Sistemas industriales.
- Sistemas domóticos.
- Sistemas portables o personales (*wearables*).

La segunda hora de las clases magistrales se dedica a plantear un problema concreto que los estudiantes habrán de acabar de solucionar, por equipos, en la siguiente clase de problemas, en la que tienen una hora para resolver dudas y entregar una propuesta de solución. El listado de las clases de problemas fue, en el curso 17/18 el siguiente:

- Programación de un simulador de una máquina inútil (FSM).
- Diseño de una FSM para detectar marcas en el entorno (etiquetas en cajas).
- Programación de una EFSM para el control de la locomoción de un robot movido por par motriz diferencial, vinculado a las prácticas.
- Programación una EFSM para la simulación de un velocímetro y rediseño de la misma para otros requerimientos.
- Diseño y programación parcial de una red de EFSM para la simulación de un contador de agua.
- Programación de un controlador adaptativo de velocidad.
- Programación de una ASM de control de una tostadora.
- Programación de un DFG para el cálculo del centro de masas.
- Diseño de una ASM para el cálculo de puntos de destino para la exploración del entorno.
- Programación de una ASM jerárquica de un controlador de un robot explorador, vinculado a las prácticas.

Este ritmo de trabajo semanal se completa con un trabajo práctico en el laboratorio cada quince días y que guarda relación con los problemas que se van tratando.

El trabajo de laboratorio consiste en desarrollar la pila de controladores para que un robot sencillo que sea capaz de explorar un área desconocida[7]. Para ello, disponen de seis sesiones:

- Introducción al simulador V-Rep y al modelo del robot de prácticas.
- Programación del controlador de locomoción del robot, de manera que pueda responder a órdenes de ir a una determinada posición expresada en coordenadas polares relativas al mismo.
- Programación del controlador del sonar del robot para que responda a órdenes de empezar un barrido con el sensor de ultrasonidos y de suministro de datos de distancias a objetos detectados.
- Programación de la interfaz de usuario con el robot real.
- Desarrollo del controlador de seguimiento de rutas. Este controlador comparte el mismo nivel en la pila de controladores que la interfaz de usuario anterior.
- Desarrollo del controlador de exploración del entorno aprovechando el anterior.

[7] Resumen en vídeo de las prácticas realizadas en el curso 17/18: https://youtu.be/zLvf2PPwnKE

En esta asignatura se intenta fomentar al máximo el trabajo en equipo y también el uso del inglés, puesto que es parte de las competencias transversales que tienen que adquirir para poder desarrollar una vida profesional satisfactoria.

En este sentido, la mayor parte del material de trabajo de que disponen está en inglés y, además, se les valora positivamente que entreguen los informes en este idioma.

En cuanto al trabajo en equipo, se les dan unas pautas para que puedan coordinarse convenientemente y se valora su seguimiento.

En las clases de problemas, las propuestas de solución tienen que presentarse manuscritas por todos los autores a partes iguales y se penalizan las entregas con un único autor.

Las entregas manuscritas permiten dejar constancia de que todos los miembros del equipo han trabajado en la resolución del problema. Por otra parte, facilitan una corrección más rápida que con el modelo aplicado en los años anteriores, en que las entregas se hacían a través del campus virtual y se corregían por ordenador, con lo que había que elaborar un documento de evaluación para cada trabajo.

La experiencia de este último curso ha sido muy positiva: al principio de cada clase de problemas, se entregaban los ya evaluados de la clase anterior, cosa que fomentaba la discusión –en sentido literal– entre estudiantes y profesor. Por otra parte, la exigencia de entregar el problema resuelto al final de la clase también promovió una actitud más activa entre los estudiantes.

El trabajo de equipo en el laboratorio tiene que realizarse de manera que cada miembro asuma un rol concreto (programador, observador y responsable) para cada sesión. Los equipos de prácticas se deciden a principio del semestre y no pueden modificarse a lo largo del mismo: deben estar formados por tres personas (o dos, como mínimo).

Desde que se empezó a impartir esta asignatura en el curso 12/13, la organización docente ha cambiado poco, a excepción de los equipos. El primer año, con un grupo pequeño de estudiantes (18), los problemas eran individuales y los equipos de prácticas, de dos personas. El notable incremento de alumnos en los posteriores años, hasta situarse en el rango entre 73 y 81 desde el curso 14/15, hizo inviable mantener este esquema si no se pasaba a equipos de tres personas.

Con esta organización, cada grupo de laboratorio acoge hasta 8 equipos, lo que da una atención media de profesor por equipo de 15 minutos.

La evaluación es continuada y se procura que los estudiantes reciban una retroalimentación de todo aquello que entregan. Esto es especialmente cierto para las entregas de problemas y de la parte práctica, pero no tanto en los exámenes teóricos que, por cuestiones pragmáticas, solo disponen de una oportunidad para aprobar por exámenes parciales y de una recuperación al final del semestre.

La nota final se obtiene calculando la media ponderada de las notas de teoría (40%), problemas (30%) y prácticas (30%).

La nota de teoría es individual y consiste en el promedio de las notas parciales o de la parte correspondiente en el examen de recuperación.

La nota de problemas es de carácter grupal y se obtiene haciendo un promedio de las notas más altas que se hayan obtenido (las ausencias computan como ceros). El número de notas que se tienen en cuenta depende de cada curso.

La nota de prácticas es una combinación entre la nota de equipo y evaluaciones individuales del trabajo en laboratorio y de las *door pitches* (presentaciones en la puerta del laboratorio en las que un miembro del equipo tiene que exponer, en un minuto, lo que ha preparado para esa sesión).

Se les exige que las notas que obtengan de cada aspecto de la asignatura sean iguales o superiores a 5 para aprobar. No hay ningún mínimo para notas parciales dentro de cada aspecto y, por ejemplo, una persona puede suspender el primer parcial de teoría y compensarlo con la nota que obtenga en el segundo.

4. Resultados

Los resultados académicos de los estudiantes en esta asignatura (figura 3) son muy satisfactorios, pues la gran mayoría aprueba: a excepción del primer año, en que se obtuvo un rendimiento académico del 80%, en el resto el porcentaje sube a más del 90%.

Sin embargo, la evaluación continuada hace que sea difícil obtener un sobresaliente (siempre por debajo del 8%), pero también más fácil superar alguna evaluación puntual desfavorable.

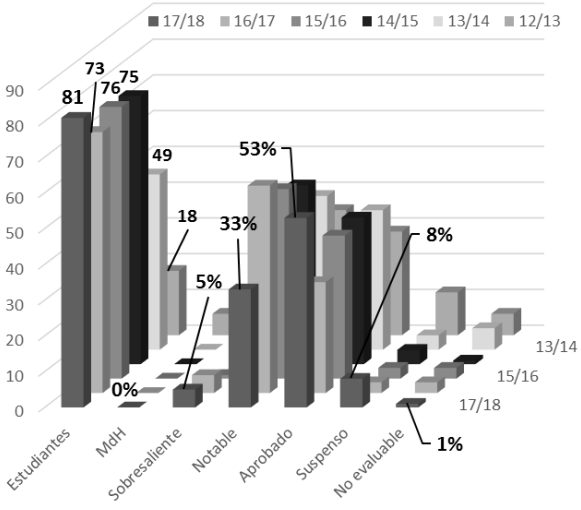

Figura 3: Evolución del número de estudiantes y de los resultados académicos.

Aunque la Universidad pasa encuestas tanto en cuanto a la asignatura como al profesorado, se echa en falta alguna cuestión más específica en cuanto a su aprendizaje. Por este motivo, cada curso se pasa una encuesta propia en la que se pide a los estudiantes que den su opinión de la asignatura (texto libre) y también que contesten a unas preguntas (escala de Likert):

- ¿Ha respondido la asignatura a las expectativas que tenías al principio de curso?
- ¿Te ha parecido que los temas tratados en teoría, prácticas y problemas estaban bien relacionados?
- ¿Tienes claros los conceptos tratados en la teoría (dominios de aplicación, modelos, arquitecturas y flujo de diseño)?
- ¿Te ha ayudado la resolución de problemas a entender cómo se construye el modelo de un sistema y cómo se simula?
- ¿Crees que las prácticas te han ayudado a ver cómo se implementan los sistemas embebidos?
- ¿Te sientes capaz de diseñar un pequeño sistema embebido?
- Finalmente, ¿te ha gustado la asignatura?

Las primeras preguntas sirven para evaluar el efecto de los cambios introducidos respecto de la edición anterior, la mayoría, de acuerdo con los comentarios de las repuestas abiertas de las encuestas de los años previos. Por otra parte, las dos últimas preguntas se refieren a la percepción de los estudiantes de lo que ha aprendido en la asignatura (*competencia*) y de su satisfacción por cursarla (*satisfacción*).

En la figura 4 se muestra la evolución de los resultados de esta encuesta a lo largo de las distintas ediciones de la asignatura. Las columnas del fondo representan el porcentaje de participación, que pasó del 50% el primer año a rondar el 25% en los años sucesivos, coincidiendo con el incremento de estudiantes (figura 3). En este sentido, hay que tener presente que el número de estudiantes que hacen la asignatura como obligatoria dentro de la mención de Ingeniería de Computadores se ha mantenido más o menos constante, alrededor de 20, mientras que los que la hacen como optativa son unos 60.

Aunque en los gráficos de la figura 4 se puede observar que los porcentajes de respuestas positivas son más elevados que el de las negativas, se ha optado por calcular una figura de mérito de los niveles subjetivos de consecución de los objetivos de la asignatura (competencia) y de satisfacción por cursarla. Dicho valor se obtiene de la relación en porcentaje entre la suma de las valoraciones de los estudiantes y el producto del número de respuestas por su grado máximo (4). En la figura 5 se muestran estos valores comparados con el rendimiento académico en porcentaje de aprobados respecto de matriculados.

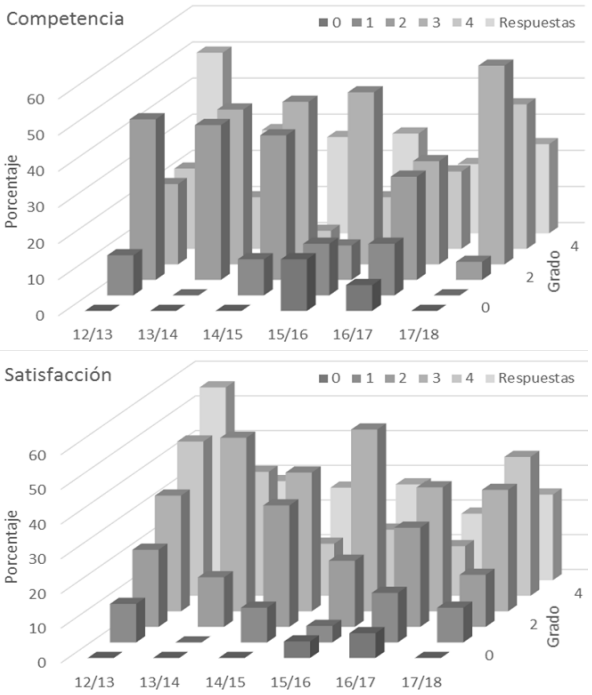

Figura 4: Porcentaje de respuestas de cada valor de la escala 0-4 y porcentaje de encuestas respondidas sobre matriculados al fondo.

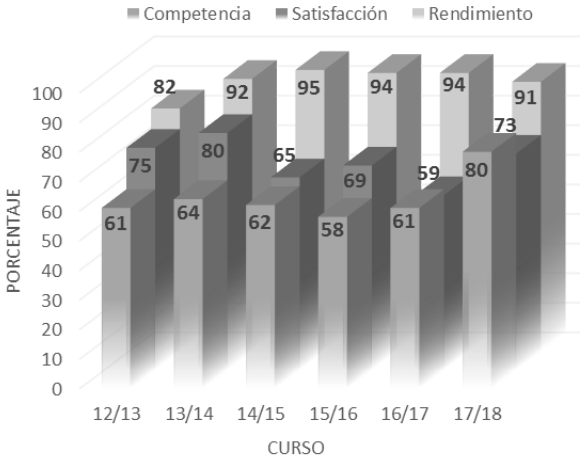

Figura 5: Rendimiento académico (al fondo) y de los grados de satisfacción y competencia (delante) percibidos por el alumnado.

Hasta el curso pasado (16/17), los estudiantes se han sentido capaces de diseñar sistemas embebidos en poco más del 60% de media, cosa que concuerda bastante con la media de las notas finales. El último curso, este valor ha subido hasta el 80%, cosa atribuible al alto grado de integración entre los problemas y las prácticas.

El grado de satisfacción con la asignatura ha aumentado respecto de los años anteriores, pero aún queda por debajo de los valores de las dos primeras ediciones, con menos estudiantes.

5. Conclusiones

El desarrollo de los sistemas embebidos es muy complejo y su aprendizaje requiere más esfuerzo que el que puede dedicar en un curso de un semestre. Por este motivo, en muchas universidades se opta bien por ofrecer un curso práctico que aborde el desarrollo de un pequeño sistema embebido o bien por ofrecer diversos cursos que aborden otros tantos aspectos del desarrollo de estos sistemas. Esta última opción es la que se eligió para los estudios de grado en Ingeniería Informática de nuestra universidad.

La asignatura de Sistemas embebidos se ha focalizado en el modelado de los sistemas y la síntesis del software correspondiente.

Con este enfoque, la asignatura consigue, de una parte, introducir al alumnado en el campo del desarrollo del software embebido y, por otra, complementar los conocimientos y competencias prácticas del resto de asignaturas de la materia, que son obligatorias para aquellas personas que eligen la especialidad en Ingeniería de Computadores.

Adicionalmente, a pesar de que pudiera verse como una dificultad añadida, que aprendan un nuevo paradigma de programación les puede servir no sólo para desarrollar sistemas embebidos de una manera más robusta sino también en otros campos en que se emplean máquinas de estados para modelar comportamientos de algunos agentes.

El rendimiento académico indica que los contenidos de la asignatura son adecuados para el nivel del alumnado cuando la cursa. Además, su percepción de la competencia en el tema concuerda bastante con la media de las notas finales. Afortunadamente, este último año, se ha situado muy por encima de esta media: 80% de competencia respecto de una nota media de 6,6. Uno de los factores que ha contribuido a ello es el uso de un único lenguaje de programación que facilitó la integración de los problemas con las prácticas y, por lo tanto, su efectividad.

Un elemento que, como profesores, nos gustaría ver siempre bien valorado, es el de la satisfacción con la asignatura. Aun siendo positivo, no llega a los niveles de excelencia que desearíamos. Sin embargo, en el último curso se ha llegado a un valor (73%) cercano al de las dos primeras ediciones, aun con más alumnos, cosa que apunta a que se está yendo en la dirección adecuada. Las claves son dar sentido a todo lo que se enseña y hacer fluida la relación entre profesores y estudiantes, con una retroalimentación lo más continua posible. El reto es conseguirlo con las clases magistrales de teoría.

En cualquier caso, a pesar del riesgo inicial de impartir una asignatura con contenidos poco convencionales, al final se ha conseguido no solo que el alumnado adquiera una competencia suficiente en los mismos sino también que le guste.

Referencias

[1] —, *Embedded Systems Programming*, Caltech. [https://www.cds.caltech.edu/~murray/wiki/index.php/EECI08:_Embedded_Systems_Programming]

[2] —, *Embedded Systems courses*, Electronic and Comp. Eng. Dept., The Hong Kong U. of Science and Technology. [http://prog-crs.ust.hk/ugcourse/2017-18/search?keyword=embedded]

[3] —, *Embedded Software and Systems*, Oxford Univ. [https://www.cs.ox.ac.uk/softeng/subjects/ESS.html]

[4] —, *Department of Electrical Engineering courses*, Princeton Univ. [https://ua.princeton.edu/academic-units/department-electrical-engineering#]

[5] Gérard Berry y Georges Gonthier. "The Esterel Synchronous Programming Language: Design, Semantics, Implementation". En *Science of Computer Programming*, 19 (87-152). 1992.

[6] Amar Bouali. "SCADE: industrial success of a synchronous language and its future challenges." En Anniversary Workshop in honour of Gérard Berry and Jean-Jacques Lévy. Gérardmer (Vosges). Feb. 2011.

[7] Jean-Louis Colaço. "SCADE: Industrial Success of a Synchronous Language and its Future Challenges." En *11th Int'l. Symp. on Theoretical Aspects of Software Engineering* (TASE). Niza (Francia). Sep. 2017.

[8] Alex Fosdick. *Introduction to Embedded Systems Software and Development Environments*. University of Colorado Boulder. [https://www.coursera.org/learn/introduction-embedded-systems#%20]

[9] Catherine Gamboa. *Embedded Systems*. Georgia Tech. 2018. [eu.udacity.com/course/embedded-systems--ud169]

[10] Raj Kamal. *Embedded Systems - Architecture, Programming and Design*. McGraw-Hill, Inc. 2009.

[11] Edward A. Lee y Prabal Dutta. *Introduction to embedded systems*, UC Berkeley, Otoño 2017. [https://bcourses.berkeley.edu/courses/1464526]

[12] Edward A. Lee y Sanjit A. Seshia. *Introduction to Embedded Systems, A Cyber-Physical Systems Approach, Second Edition*. MIT Press, 2017.

[13] Girish Keshav Palshikar. "An Introduction to Esterel". En *Embedded*, 1 de noviembre de 2001. [https://www.embedded.com/design/prototyping-and-development/4024644/An-Introduction-to-Esterel#]

[14] Michael J. Pont. *Embedded C*. Pearson Education Ltd.: Essex, England. 2005.

[15] Claudius Ptolemaeus, Editor. *System Design, Modeling, and Simulation Using Ptolemy II*, Ptolemy.org, 2014.

[16] Lluís Ribas-Xirgo. *How to code finite state machines (FSMs) in C. A systematic approach*. 2014. DOI 10.13140/2.1.4147.9200.

[17] Anthony Rowe. *18-549 Embedded System Design*. Carnegie Mellon University. Primavera de 2017. [https://www.ece.cmu.edu/~ece549/docs/18-549_Syllabus_S17.pdf]

[18] Jonathan Valvano y Ramesh Yerraballi. *Embedded Systems - Shape The World: Microcontroller Input/Output*. UTAustinX. Primavera de 2017. [https://www.edx.org/course/embedded-systems-shape-world-utaustinx-ut-6-10x]