**Systemy reaktywne**

W poprzedniej części omówiono ogólne problemy, które pojawiają się, gdy musimy opracować nie tylko algorytmy i programy, ale także duże i złożone systemy. W tym rozdziale skoncentrujemy się na jednym szczególnie problematycznym typie systemu oraz na jego najtrudniejszym i śliskim aspekcie. Rodzaje systemów, o których myślimy, mają głównie charakter reaktywny, a trudnym aspektem jest określenie, przeanalizowanie i wdrożenie ich zachowania w czasie. Niektóre systemy są rzeczywiście złożone, ale mają charakter transformacyjny; są typu wejścia/procesu/wyjścia, a ich zalecona praca jest wykonywana wielokrotnie dla każdego nowego zestawu danych wejściowych. Oznacza to, że swoją złożoność zawdzięczają obliczeniom i przepływowi danych. W takich przypadkach możemy powiedzieć, że nasz przyjaciel, mały Runaround, ma dużo do myślenia lub dużo podnoszenia i przenoszenia. Typowy komponent takiego systemu czeka na przybycie wszystkich danych wejściowych, przetwarza je, wysyła dane wyjściowe do kilku innych komponentów i wraca do stanu uśpienia, dopóki nie zostaną wyświetlone nowe dane wejściowe. Systemy tego rodzaju można w zadowalający sposób opisywać i analizować za pomocą technik przepływu danych, które identyfikują przetwarzanie zachodzące wewnątrz różnych komponentów oraz dane, które przepływają między nimi. Zatem dynamiczne zachowanie systemu transformacyjnego jest w dużej mierze zdeterminowane przez połączenia przepływowe między składnikami lub obiektami. Znacznie bardziej problematyczne są te systemy (zwłaszcza te duże i złożone), które są silnie sterowane kontrolą lub zdarzeniami. Takie systemy nazwano reaktywnymi. Ich rolą w życiu jest reagowanie na wiele różnych rodzajów zdarzeń, sygnałów i warunków w zawiły sposób. Systemy reaktywne niekoniecznie muszą być współbieżne lub rozproszone, chociaż wiele z nich jest. Wiele z nich ma również krytyczne znaczenie czasowe, często muszą reagować na wydarzenia w czasie rzeczywistym. Tutaj mały Runaround lub wiele z nich, jeśli system jest również współbieżny, musi być niezwykle czujny, responsywny i szybki. W rzeczywistości nie będzie przesadą stwierdzenie, że ogromna część skomputeryzowanych systemów jest reaktywna lub ma dominujące reaktywne części. Przykłady obejmują stosunkowo małe systemy, takie jak magnetowidy (VCR), telefony komórkowe i bankomaty, oraz znacznie większe i złożone, takie jak systemy samochodowe i awioniki, zakłady chemiczne, sterowania systemy, sterowniki telefoniczne i komunikacyjne, roboty przemysłowe oraz interaktywne pakiety oprogramowania, takie jak edytory tekstu i edytory programów. Systemy te muszą utrzymywać zawiłe dynamiczne relacje z otoczeniem, odpowiednio i na czas reagować na naciskanie przycisków, wzrost temperatury powyżej poziomu krytycznego, odkładanie odbiorników, przesuwanie kursorów na ekranie i tak dalej. Często zawierają one również wewnętrznie rozległą interakcję reaktywną; to znaczy pomiędzy różnymi komponentami tworzącymi system. A w dobie Internetu coraz więcej systemów intensywnie korzystających z sieci reaguje również, a manipulacja w jednym miejscu powoduje reakcje w innym, z dużą ilością tej reaktywności zachodzącej jednocześnie. Tak więc dominująca część złożoności systemu reaktywnego nie wynika ze złożonych obliczeń lub przepływu danych, ale ze skomplikowanych wzorców przyczynowo-skutkowych, wyzwalaczy/odpowiedzi, zwykle połączonych z wysokim stopniem współbieżności i aspektów czasowych. Głównym problemem związanym z reaktywnością jest określenie zachowania systemu w czasie, jasno i poprawnie oraz w sposób, który można łatwo i niezawodnie wdrożyć, przeanalizować i zweryfikować. Co się stanie i kiedy? Dlaczego te rzeczy się wydarzą i co jeszcze spowodują, że stanie się na ich przebudzeniu? Czy w międzyczasie mogą się zdarzyć inne rzeczy? Czy niektóre rzeczy są obowiązkowe, a inne po prostu dozwolone? Jakie są ograniczenia czasowe, gdy coś się dzieje? Jaki jest skutek tego, że oczekiwane rzeczy nie dzieją się wtedy, gdy powinny? Co może się nie wydarzyć w żadnych okolicznościach? I tak dalej. Co ciekawe, reaktywność nie jest wyłączną cechą systemów komputerowych stworzonych przez człowieka. Występuje również w układach biologicznych, które pomimo tego, że są dużo mniejsze niż my ludzie i nasze domowe artefakty, mogą być również dużo bardziej skomplikowane. I występuje również w

systemach ekonomicznych i społecznych, które są dużo większe niż pojedynczy człowiek. Te również mają skomplikowany charakter reaktywny i są w stanie w pełni zrozumienia i analizowania ich, a także przewidywanie ich przyszłych zachowań, wymaga tego samego rodzaju myślenia, co systemy komputerowe. Prowadzi to do przekonania, że niektóre rozwiązania oferowane przez informatykę i inżynierię systemów mogą być również wykorzystane do radzenia sobie z taką niekomputerową reaktywnością. Podwójnie możemy również dowiedzieć się wiele o tym, jak radzić sobie z skomputeryzowaną reaktywnością, obserwując Matkę Naturę zajmującą się własnymi systemami reaktywnymi.

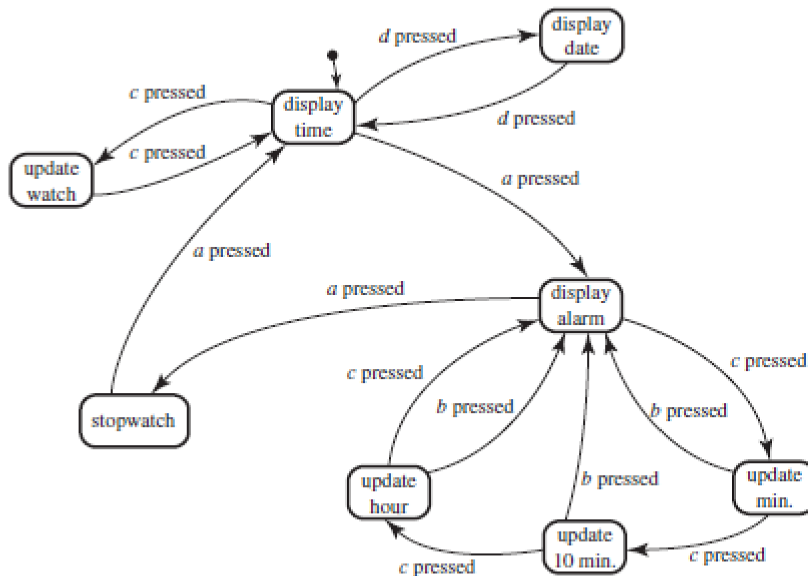
**Formalizmy wizualne dla zachowań reaktywnych**

Głównym artefaktem potrzebnym do opracowania niezawodnego systemu reaktywnego jest ogólny model systemu, który składa się ze starannie połączonej kompleksowej reprezentacji aspektów strukturalnych i behawioralnych systemu. Służy jako narzędzie dla projektantów i projektantów do uchwycenia ich przemyśleń i włączenia elementów z wymagań i projektu, a następnie może prowadzić aż do pomyślnego wdrożenia. Pod pewnymi względami model przypomina zestaw planów narysowanych przez architekta w celu opisania domu lub mostu. Różnica polega jednak na dynamice: systemy reaktywne zmieniają się w czasie; robią rzeczy; zachowują się. Nie dotyczy to domów czy mostów. Tak więc, podczas gdy opis strukturalny systemu reaktywnego można uznać za jego kręgosłup, jego zachowanie jest w zasadniczym sensie jego sercem i duszą. Zachowanie w systemie reaktywnym jest jak silnik w samochodzie; bez niego też nie mogą się „poruszać”. Ponadto zachowanie w czasie jest znacznie mniej namacalne niż ogólna funkcjonalność reaktywnego systemu lub jego fizyczna struktura, a przede wszystkim ten aspekt sprawia, że rozwój reaktywnych systemów jest tak trudny i podatny na błędy. Jednym z podejść do problemu specyfikowania takich systemów są formalizmy wizualne, języki diagramatyczne i intuicyjne, ale matematycznie rygorystyczne. Przy wszystkich innych rzeczach równych, obrazy są zwykle lepiej rozumiane niż tekst czy symbole, a właściwie użyte mogą służyć do myślenia na wyższym poziomie abstrakcji niż tekst. Ale te języki to nie tylko grafika. Podobnie jak języki programowania wysokiego poziomu wymagają nie tylko edytorów i narzędzi do wyświetlania, ale także – i to znacznie ważniejsze! - kompilatory, interpretery i narzędzia do debugowania, więc języki do modelowania zachowania systemów reaktywnych wymagają znacznie więcej niż ładnych diagramów z ładnymi edytorami graficznymi. Potrzebujemy środków do uruchamiania lub wykonywania modeli oraz środków do kompilacji ich w konwencjonalny kod, czynności zwanej generowaniem kodu lub syntezą kodu. Zatem formalizmy wizualne dla zachowań reaktywnych, podobnie jak konwencjonalne języki programowania obliczeń, muszą być uzupełnione składnią, która określa, co jest dozwolone, i semantyką, która określa, co oznaczają dozwolone rzeczy. Wizualizacja często opiera się na użyciu pól i strzałek, ze związkami topologicznymi między nimi, takimi jak hermetyzacja, połączenie i sąsiedztwo. Takie języki są często hierarchiczne i modułowe. Istnieje wiele języków określania zachowań reaktywnych, z których kilka można zaklasyfikować jako w pełni rozwinięte formalizmy wizualne. Niektóre z najbardziej interesujących to sieci Petriego i diagramy SDL = zarówno wizualne formalizmy, jak i Esterel, Signal i Lustre, które z wyglądu przypominają bardziej języki programowania, chociaż mają również graficzne nakładki. Opiszemy teraz jeden przykład formalizmu wizualnego dla zachowań reaktywnych, zwany wykresami stanów.

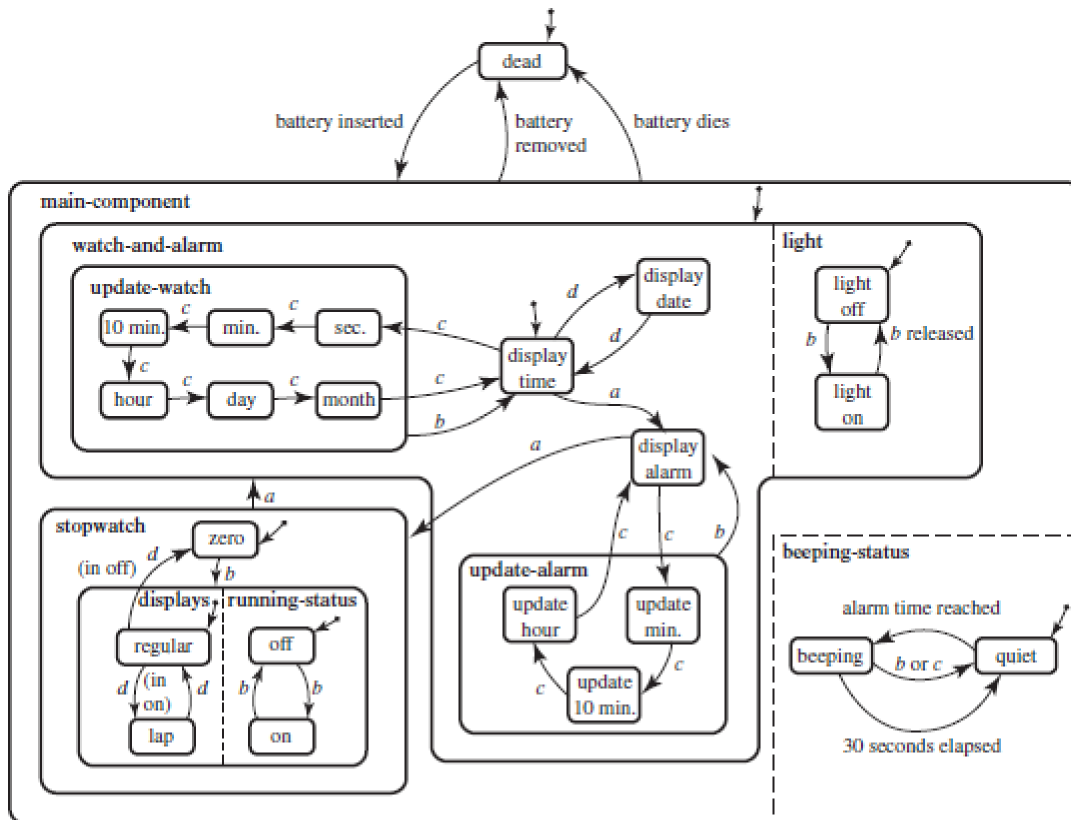
**Schematy zachowań reaktywnych**

Automaty skończone i związane z nimi diagramy przejść stanów wydają się być zadowalającym punktem wyjścia do określenia zachowania reaktywnego. Identyfikujemy stany systemu lub tryby działania i przystępujemy do określania zdarzeń i warunków, które powodują przejścia między stanami oraz działań ( na przykład wysyłanie i odbieranie danych, rozpoczynanie lub zatrzymywanie działań itp.), które są w nich wykonywane. Część 9 zawierał prosty przykład diagramu opisującego część

zachowania zegarka cyfrowego . Istnieje jednak kilka problemów związanych z naiwnym używaniem diagramów stanów w przypadku złożonych przypadków. Po pierwsze, diagramy są „płaskie", podczas gdy reaktywne zachowanie nawet stosunkowo małych systemów w naturalny sposób spada na poziomy szczegółowości. (Odcięcie zasilania elektrycznego jest wydarzeniem bardzo wysokiego poziomu, podczas gdy przesuwanie kursora ekranowego nad jakąś ikoną jest wydarzeniem niskiego poziomu.) Te poziomy są korzystne nie tylko dla jasności i zrozumienia, ale także podczas samego procesu tworzenia. Rozważ zegarek z rysunku.




Chcielibyśmy kontynuować doprecyzowanie go, opisując zachowanie samej funkcji stopera (na przykład, jak i kiedy jest uruchamiana i zatrzymywana) oraz różne możliwości aktualizacji. Wszystko, co mamy do aktualizacji, to stan o nazwie update-watch, który prawdopodobnie oznacza lub zawiera pewną liczbę podstanów, które zajmują się aktualizacją sekund, minut, godzin i miesięcy, podobnie jak istnieją trzy stany do aktualizacji alarmu. Po drugie, diagramy stanów są sekwencyjne i nie uwzględniają w naturalny sposób współbieżności. Załóżmy na przykład, że nasz zegarek ma oświetlenie do podświetlania, które można włączać lub wyłączać niezależnie od innych rzeczy, które dzieją się gdzie indziej. Jeśli światło jest całkowicie oddzielną jednostką sterowaną, powiedzmy, oddzielnym przyciskiem, to wystarczy nam nowy dwustanowy diagram opisujący je jako osobny system. W rzeczywistości jednak funkcja światła w zegarku cyfrowym jest mniej prosta. Może nie zawsze mieć zastosowanie - być może nie działa w stanie stopera. Oczywiście możliwe jest połączenie jasnych informacji z resztą opisu, dając po dwa stany dla każdego ze starych; jeden na wypadek, gdy światło jest włączone, a drugi na wypadek, gdy jest wyłączone. Jeśli jednak podejście to zostanie przyjęte ogólnie do radzenia sobie ze współbieżnością, skutkuje to wykładniczym wzrostem liczby tatów, które należy wyraźnie opisać. (Dlaczego?) Oczywiście te problemy, które pojawiają się nawet w tak małych systemach, jak zegarki cyfrowe, są znacznie bardziej dotkliwe w dużych i złożonych systemach reaktywnych, które zazwyczaj mają ogromną liczbę stanów. Próbując zaradzić tym niedociągnięciom, rozszerzono diagramy stanów, uzyskując język zwany wykresami stanów. Rysunek pokazuje wykres stanu dla bardziej szczegółowej wersji zegarka z rysunku powyżej, który zawiera teraz komponent świetlny, udoskonalone stany stopera i aktualizacji oraz status sygnału dźwiękowego.




(Dla zwięzłości, pominęliśmy słowo „wciśnięty" w zdarzeniach naciśnięcia przycisku). Wykresy stanów pozwalają na wielopoziomowe stany, rozłożone w sposób i/lub sposób, a tym samym obsługują zwartą specyfikację. Aby uchwycić hierarchię stanów, prostokąty regularnych diagramów stanów można ułożyć w klastrowy, hermetyczny sposób. Na przykład stan obserwowany na rysunku składa się z sześciu zamkniętych podstanów, powiązanych między sobą wyłącznym „lub": bycie w trybie obserwowania aktualizacji oznacza w rzeczywistości przebywanie dokładnie w jednym z jego sześciu podstanów. Mogliśmy pogrupować te sześć stanów dla jasności lub w celu stopniowego rozwoju. Być może najpierw zdecydowaliśmy, że będzie stan do aktualizacji (co rzeczywiście mamy), a dopiero później przystąpiliśmy do samodoskonalenia. Co więcej, sześć stanów aktualizacji ma co najmniej jedną wspólną właściwość: wszystkie reagują na naciśnięcie b, zatrzymując proces aktualizacji i powracając do czasu wyświetlania. Gdyby użyto „płaskich" diagramów, wymagałoby to sześciu oddzielnych strzałek, po jednej wychodzącej z każdego stanu. Jeśli jesteśmy w aktualizacji-watch i przycisk b jest wciśnięty, wychodzimy i wprowadzamy czas wyświetlania. Jednakże, ponieważ bycie w trybie obserwacji aktualizacji znajduje się w jednym z podstanów, następuje pożądany efekt. Podobnie, jeśli bateria zostanie wyjęta lub wygaśnie, następuje przejście do stanu martwego, bez względu na to, w jakim byliśmy. Są to czasami nazywane przerwaniami. Aby określić współbieżne składniki stanu, wykresy stanów używają partycjonowania z linią przerywaną. Dodają one wymiar jednoczesności i są nazywane składowymi ortogonalnymi mapy stanów. Relacja między komponentami ortogonalnymi to nie „lub", ale „i". Na przykład, jeśli jesteśmy w stanie stopera z rysunku , ale nie w stanie zero, to musimy być jednocześnie zarówno na wyświetlaczach, jak i w stanie pracy. W każdym z nich istnieją dwie możliwości, powiązane przez „lub" i dające w sumie cztery możliwe konfiguracje stanów. Dla kontrastu z tymi czterema, rozważ składnik światła, w którym dodaliśmy dwa stany zamiast 12, których potrzebowalibyśmy bez składowych ortogonalnych, oraz status dźwiękowy, w którym dodano dwa zamiast 29. (Skąd się wzięły 12 i 29 ?) To zmniejszenie rozmiaru pomaga przezwyciężyć omówiony wcześniej problem wykładniczego rozrostu. Małe strzałki oznaczające stany początkowe w zwykłych

diagramach stanów mogą pojawić się na dowolnym poziomie w schemacie stanów. Tutaj nazywane są wartościami domyślnymi. W ramach głównego komponentu stanu wysokiego, na przykład, stan kombinowany składający się z ortogonalnych komponentów zegarka i alarmu oraz światła jest domyślny, więc jeśli bateria jest włożona w stanie martwym, wprowadzamy tę kombinację, a nie stoper. (Oczywiście wchodzimy również w stan ciszy w stan pikania.) Teraz, w każdym z tych prostopadłych składowych jest również domyślna strzałka - musi być jedna, w przeciwnym razie nie wiedzielibyśmy, w który z ich podstanów wejść - tak że naprawdę kończą się w konfiguracji „czas wyświetlania, wyłączanie, cisza". Wykresy stanów mają szereg dodatkowych funkcji, takich jak możliwość określenia ograniczeń czasowych oraz możliwość oparcia przejść na zachowanie w przeszłości. Powinniśmy jednak pamiętać, że formalizmy, takie jak wykresy stanów, są w stanie opisać tylko część kontrolną systemu reaktywnego, a nie jego przepływ danych czy aspekty strukturalne. Specyfikacja behawioralna musi być połączona ze specyfikacją struktury systemu, tak jak niesilnikowe części samochodu muszą być połączone z silnikiem, a te połączenia nie są takie proste. Jednym z najczęściej stosowanych podejść do tego jest łączenie języka zachowań reaktywnych, takich jak schematy stanów, z ideami orientacji obiektowej. Omówimy to podejście później.

**Wykonanie modelu**

Jednym z najbardziej interesujących pojęć, jakie wyszły z badań nad systemami i inżynierią oprogramowania, jest specyfikacja wykonywalnych specyfikacji lub, aby lepiej dopasować się do używanej tu terminologii, modele wykonywalne. Wykonywanie modelu może odbywać się albo bezpośrednio, w sposób analogiczny do uruchamiania interpretera w konwencjonalnym programie komputerowym, albo pośrednio, poprzez kompilację do kodu i uruchomienie kodu. Dostępnych jest szereg potężnych narzędzi obsługujących takie możliwości. Sednem wykonania modelu jest możliwość wykonania pojedynczego kroku dynamicznego działania systemu, z uwzględnieniem wszystkich konsekwencji. Podczas kroku środowisko może generować zdarzenia zewnętrzne, modyfikować wartości warunków i zmiennych itp. Takie zmiany wpływają następnie na stan systemu: wyzwalają nowe zdarzenia, aktywują i dezaktywują czynności, modyfikują warunki i zmienne, i tak na. I oczywiście każda z tych zmian może z kolei powodować wiele innych, często wywołując skomplikowane reakcje łańcuchowe. W wielu rodzajach systemów ograniczenia czasowe i czasowe odgrywają ważną rolę w określaniu sposobu wykonania kroku. Semantyka formalizmu użytego do określenia zachowania musi zawierać wszystkie informacje potrzebne do precyzyjnego uchwycenia tych zmian. Obliczenie efektu kroku na podstawie aktualnego stanu i zmian dokonanych przez otoczenie zwykle wiąże się ze skomplikowanymi procedurami algorytmicznymi, które wywodzą się i odzwierciedlają tę semantykę. W przypadku map stanów mechanizm wykonawczy musi być w stanie śledzić dynamikę wszystkich komponentów ortogonalnych we wszystkich aktywnych mapach stanu, biorąc pod uwagę reakcje łańcuchowe zdarzeń i zmian stanu oraz przeprowadzając wszystkie zalecone przez nie zmiany, w tym wszelkie działania które są powiązane ze stanami lub przejściami. Najprostszym sposobem wykonania lub „uruchomienia" modelu za pomocą skomputeryzowanego narzędzia jest interaktywny sposób „krok po kroku". Na każdym kroku użytkownik emuluje system środowiska, generując wydarzenia i zmieniając wartości. Narzędzie z kolei reaguje, przekształcając system w nowy wynikowy status. Jeśli model jest reprezentowany wizualnie, zmiana statusu zostanie również odzwierciedlona wizualnie, powiedzmy, poprzez zmiany koloru lub podkreślenia na diagramach. Tytułem ilustracji powróćmy na chwilę do przykładu zegarka. Obserwując topologiczne właściwości wykresu można łatwo wywnioskować, że zachowanie światła nie ma zastosowania do stanów stopera, ponieważ jest ono ortogonalne do stanu zegarka i alarmu, a wyłączne dla topwatcha. Jednak ten związek między światłem a obserwacją i alarmem ma bardziej subtelne implikacje, których nie można łatwo dostrzec bez faktycznego wykonania modelu. W szczególności załóżmy, że jesteśmy w konfiguracji „godzina, światło wyłączone, sygnał dźwiękowy", co oznacza, że aktualizujemy godzinę, światło jest wyłączone, a

brzęczyk emituje sygnał dźwiękowy, ponieważ nadeszła godzina alarmu. (To tak, jakby powiedzieć, że trzymamy zegarek w dłoniach i właściwie go „obsługujemy", zaczynając od opisanej sytuacji.) Teraz wciskamy przycisk b. Co się dzieje? Cóż, czy nam się to podoba, czy nie, trzy pozornie niepowiązane ze sobą rzeczy wydarzą się jednocześnie. Sygnał dźwiękowy zostanie zatrzymany (z powodu przejścia „b lub c" w tryb cichy), światło włączy się, dopóki b nie zostanie zwolnione, a aktualizacja zakończy się, a zegarek powróci do czasu wyświetlania! Gdy ta sekwencja zostanie uruchomiona za pomocą narzędzia, które wykonuje wykresy stanu, zmiany te pojawią się na wykresie stanu w sposób animowany, zwykle w specjalnym kolorze stanów, w których aktualnie znajduje się system, oraz ostatnio przebytych przejść. Poprzez interaktywne wykonywanie scenariuszy, które odzwierciedlają sposób, w jaki oczekujemy, że nasz system będzie się zachowywał, jesteśmy w stanie zweryfikować, czy rzeczywiście to zrobi, przed ostatecznym wdrożeniem. Jeśli stwierdzimy, że reakcja systemu nie jest zgodna z oczekiwaniami, możemy wrócić do modelu, zmienić go i ponownie uruchomić ten sam scenariusz. Jest to analogiczne do jednoetapowego lub wsadowego debugowania konwencjonalnych programów. Podczas wykonywania, użytkownik odgrywa rolę wszystkich części modelu, które są zewnętrzne w stosunku do wykonywanej części, nawet jeśli te części zostaną ostatecznie określone i staną się w ten sposób wewnętrznymi. Gdy mamy już podstawową umiejętność wykonania kroku, nasz apetyt rośnie. Możemy teraz chcieć, aby model wykonywał się w sposób nieinteraktywny. Aby np. sprawdzić, czy połączenie telefoniczne łączy się wtedy, gdy powinno, możemy przygotować odpowiednią sekwencję zdarzeń i sygnałów w pliku wsadowym, ustawić model tak, aby startował w stanie początkowym i poprosić nasze narzędzie o iteracyjne wykonywanie kroków, wczytanie zmian z pliku. Graficzna informacja zwrotna z takiego wykonania wsadowego staje się (często całkiem atrakcyjną) animacją diagramów. W rzeczywistości nie musimy ograniczać się do tworzenia samodzielnie opracowanych scenariuszy: możemy chcieć zobaczyć, jak model działa w okolicznościach, których sami nie chcemy szczegółowo określać. Chcielibyśmy zobaczyć jego działanie w losowych warunkach, zarówno w typowych, jak i mniej typowych sytuacjach. Możemy chcieć włączyć punkty przerwania do mechanizmu wykonawczego, powodując jego zawieszenie, a narzędzie do podjęcia określonych działań, gdy pojawią się określone sytuacje. Działania te mogą obejmować chwilowe wejście w tryb interaktywny w celu dokładnego monitorowania postępów krok po kroku, aż po wykonanie gotowego kodu opisującego czynność niskiego poziomu. Zdolności te trafiają w sedno potrzeby modeli wykonywalnych - aby zminimalizować nieprzewidywalne w rozwoju złożonych systemów reaktywnych. Wszystko to można posunąć o wiele dalej, ponieważ same wykonania modeli są programowane lub metaprogramowane za pomocą środków zewnętrznych. W ten sposób narzędzie wykonawcze można skonfigurować tak, aby wyszukiwało predefiniowane punkty przerwania i gromadziło informacje dotyczące postępu systemu w miarę jego przebiegu. Jako przykład możemy chcieć wiedzieć, ile razy podczas typowego lotu samolotu, który określamy, radar traci namierzony cel. Ponieważ inżynierowi może być trudno złożyć typowy scenariusz lotu, możemy wykorzystać moc naszego narzędzia, instruując je, aby uruchomiło wiele typowych scenariuszy, używając zgromadzonych wyników do obliczenia informacji o średnim przypadku. Narzędzie będzie następnie podążać za typowymi scenariuszami, generując losowe liczby w celu wybrania nowych zdarzeń zgodnie z predefiniowanymi rozkładami prawdopodobieństwa. Statystyki są następnie gromadzone przy użyciu odpowiednich punktów przerwania i prostych obliczeń. Podstawowe idee stojące za tymi technikami są oczywiście dobrze znane w testowaniu i debugowaniu programów. Chodzi tu jednak o rozszerzenie ich na formalizmy wizualne wysokiego poziomu używane do modelowania złożonych zachowań reaktywnych, na długo przed kosztownymi etapami implementacji i wdrażania finalnego systemu. Wykonanie modelu może ujawnić błąd, tj. zachowanie, które różni się od tego, co zamierzaliśmy. Oczywiście, jeśli tak się stanie, chcielibyśmy dowiedzieć się, co spowodowało anomalne zachowanie. Dlaczego wydarzyło się coś nieoczekiwanego? Dlaczego nie wydarzyło się coś innego, mimo że myśleliśmy, że powinno? Co by się stało, gdyby jakieś zewnętrzne wydarzenie miało miejsce lub miało miejsce wcześniej lub później? I tak

dalej. Wiele takich pytań odnosi się do zachowania modelu przed momentem wykrycia błędu. Aby móc odpowiedzieć na takie pytania, narzędzie musi przechowywać historię swoich przeszłych działań wraz z ich przyczynami. W przypadku braku takich informacji sprowadzamy się do wielokrotnego uruchamiania modelu od początku, zatrzymując się w różnych punktach, aby zaobserwować stany pośrednie i zobaczyć, gdzie jego zachowanie odbiega od tego, czego oczekujemy. Kolejnym krokiem w tym rosnącym apetycie na możliwości analizy zachowań reaktywnych jest oczywiście weryfikacja w rozumieniu Części 5. Typową właściwością, którą bardzo często chcemy zweryfikować, jest osiągalność, która określa, czy po uruchomieniu w jakimś W takiej sytuacji system może kiedykolwiek dojść do sytuacji, w której jakiś określony warunek stanie się spełniony. Warunek ten można postawić w celu odzwierciedlenia niepożądanych lub pożądanych sytuacji. Co więcej, możemy sobie wyobrazić, że test jest skonfigurowany tak, aby raportował pierwszy znaleziony scenariusz, który prowadzi do określonego stanu, lub raportował wszystkie możliwe, tworząc szczegóły samych scenariuszy. Czy takie testy są realistyczne? Czy moglibyśmy na przykład poddać model testowi osiągalności, po którym będziemy wiedzieć na pewno, czy istnieje jakakolwiek możliwość jego wystąpienia w każdych możliwych okolicznościach? Odpowiedź jest podobna do tej, którą udzieliliśmy, omawiając weryfikację w Części 5 oraz ograniczenia obliczeń w Częściach 7 i 8: w zasadzie nie, ale w wielu praktycznych sytuacjach tak. Rzeczywiście, w ostatnich latach poświęcono wiele pracy, aby weryfikacja programu zrobiła duży krok dalej, prowadząc do możliwości weryfikacji modeli wizualnych pod kątem złożonych zachowań reaktywnych. Wierzymy, że automatyczna weryfikacja krytycznych właściwości w systemach reaktywnych stanie się powszechna i że metody i narzędzia modelowania behawioralnego będą stawały się coraz potężniejsze, oferując środki do rutynowej weryfikacji właściwości w miarę rozwoju modelu.

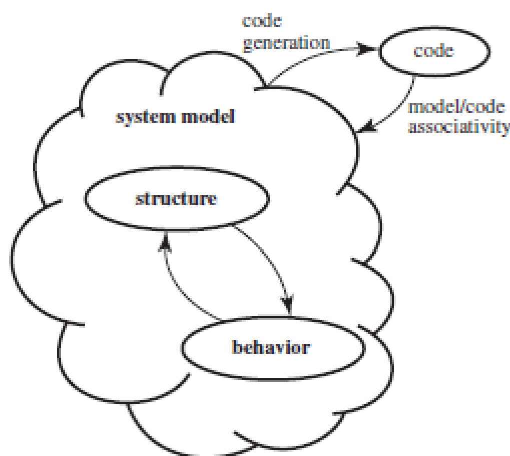
**Synteza kodu**

Jak wyjaśniono wcześniej, bezpośrednie wykonanie modelu jest analogiczne do uruchamiania programów za pomocą interpretera. Jednak wiele narzędzi generuje automatycznie z kodu modelu w jakimś konwencjonalnym języku, takim jak C++ lub JAVA, a następnie wykonuje ten kod. To jest analogia kompilacji. Kiedy patrzysz na model w wykonaniu, tak naprawdę nie możesz odróżnić.6 Jednym z głównych zastosowań danych wyjściowych generatora kodu jest obserwowanie działania systemu w warunkach zbliżonych do rzeczywistych. Na przykład kod może zostać przeniesiony i wykonany w rzeczywistym środowisku docelowym lub, jak to często bywa na wcześniejszych etapach, w symulowanej wersji środowiska docelowego. W ten sposób kod można połączyć z graficznym interfejsem użytkownika (GUI) systemu – makieta ekranowa tablic kontrolnych systemu, wraz z obrazami wyświetlaczy, przełączników, dźwigni, pokręteł i wskaźników – która reprezentuje rzeczywisty interfejs użytkownika końcowego systemu. GUI można wtedy manipulować w realistyczny sposób za pomocą myszy i klawiatury. Ważną kwestią jest to, że zachowanie symulowanego systemu nie jest napędzane przez pospiesznie napisany kod przygotowany specjalnie na potrzeby prototypu, ale przez kod, który został wygenerowany automatycznie z modelu, który zazwyczaj zostanie dokładnie przetestowany i przeanalizowany przed generowaniem kodu. Co więcej, gdy dostępne są części rzeczywistego środowiska docelowego, można je również połączyć z kodem, a przebiegi stają się jeszcze bardziej realistyczne. Generowanie kodu z modeli zachowań reaktywnych zbudowanych za pomocą formalizmów wizualnych może być zatem wykorzystane do celów wykraczających poza zespół programistów. Interfejsy GUI systemu oparte na kodzie mogą być używane jako część standardowej komunikacji między klientem a wykonawcą lub wykonawcą a podwykonawcą. Nie jest nierozsądne, aby taka działająca wersja modelu systemu była wymaganym produktem na niektórych etapach rozwoju. Dobre narzędzie do generowania kodu będzie miało również mechanizm debugowania, za pomocą którego użytkownik może prześledzić wykonywane części kodu z powrotem do modelu graficznego. Punkty przerwania można wstawić, aby zatrzymać przebieg, gdy wystąpią określone

zdarzenia, w którym to momencie można sprawdzić stan modelu, a elementy można modyfikować w locie przed wznowieniem przebiegu. Jeśli pojawią się poważne problemy, zmiany można wprowadzić w oryginalnym modelu, który jest następnie ponownie kompilowany do kodu i ponownie uruchamiany. Można zażądać plików śledzenia, rejestrując kluczowe informacje do przyszłej inspekcji i tak dalej.

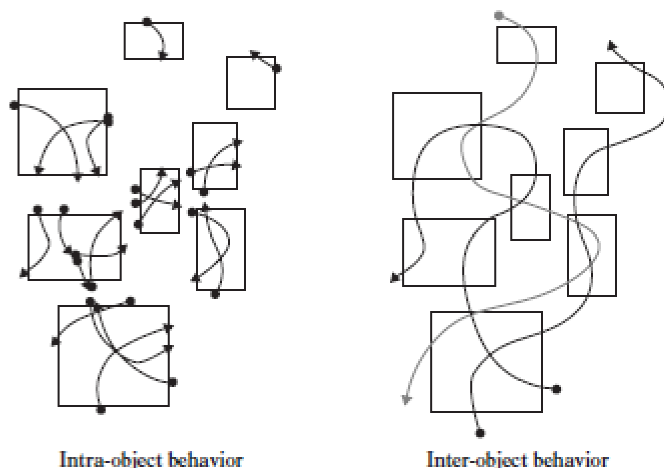
**Dwa style zachowań**

Jak określana jest struktura reaktywnego systemu i jak ta specyfikacja jest połączona z jego zachowaniem? Zaproponowano kilka metod, a jedną z bardziej rozpowszechnionych jest oparcie całego procesu modelowania i rozwoju na paradygmacie obiektowym. Prowadzi to do tak zwanej specyfikacji i analizy obiektowej. Główną ideą jest podniesienie pojęć z poziomu programowania obiektowego, jak opisano w rozdziale 3, na poziom modelowania i wykorzystanie formalizmów wizualnych. W przypadku struktury systemu do określania klas i ich wzajemnych relacji używany jest język diagramów zwany diagramami modeli obiektowych. Jeśli chodzi o określanie zachowania, większość podejść do modelowania obiektowego opiera zachowanie obiektu na maszynie stanów, a wiele z nich zaleca skonstruowanie wykresu stanu dla każdej klasy, przechwytującego pożądane zachowanie dowolnej jego instancji. Po utworzeniu instancji klasy kopia schematu stanu klasy rozpoczyna działanie, kontrolując zachowanie tej instancji. Szczegóły tego związku między strukturą a zachowaniem są o wiele bardziej skomplikowane, niż można to tutaj przedstawić. Klasy reprezentują dynamicznie zmieniające się kolekcje konkretnych obiektów, a modelowanie behawioralne musi dotyczyć kwestii związanych z tworzeniem i niszczeniem obiektów, delegowaniem komunikatów, modyfikacją relacji i konserwacja, kolejkowanie zdarzeń, agregacja klas, dziedziczenie, przetwarzanie wielowątkowe i tak dalej. Powiązania między zachowaniem a strukturą muszą być zdefiniowane wystarczająco szczegółowo i wystarczająco rygorystycznie, aby wspierać konstrukcję narzędzi, które umożliwiają omówione powyżej rodzaje wykonywania modeli i generowania kodu. Rysunek poniższy ilustruje te podstawowe części modelowania systemu.




Innymi słowy, zachowanie jest określone w tej konfiguracji w sposób oparty na stanie obiekt po obiekcie, zapewniając reaktywność każdego obiektu za pomocą, powiedzmy, mapy stanu. Gdyby to była wymyślona przez nas gra piłkarska, to podejście wymagałoby od nas szczegółowego zaprogramowania każdego z graczy – a także sędziów, piłki, drewnianych ram bramek itd. – określając sposób, w jaki reagują i reagują na każde zdarzenie lub wydarzenie, które nadchodzi ich droga. Po wykonaniu tej czynności możemy „uruchomić” system lub zasymulować grę, ponieważ mamy pełną informację o każdej możliwej parze wyzwalacz/odpowiedź dostępna dla każdego obiektu. Jak na razie dobrze. Jednak kiedy ludzie myślą o systemach reaktywnych, najczęściej myślą naturalnie w

kategoriach scenariuszy zachowań. Nie ma zbyt wielu ludzi mówiących takie rzeczy jak „Cóż, mechanizm nagrywania mojego magnetowidu może być w trybie bezczynności, w trybie nagrywania lub w trybie selekcji środkowej; w pierwszym przypadku są to możliwe dane wejściowe i reakcje magnetowidu, . . . ; w drugim przypadku oto, co się dzieje, itd.” Raczej zauważasz, że mówią rzeczy takie jak „Jeśli nacisnę ten przycisk, a następnie obrócę pokrętło, aby wskazać tutaj, na wyświetlaczu pojawi się następujący komunikat” lub mniej szczegółowe rzeczy, takie jak „Wybór daty i godziny, a następnie naciśnięcie Nagrywaj, powoduje, że magnetowid wykonuje następujące czynności . . . Mówią również o zabronionych scenariuszach, takich jak „Dopóki jest podłączony do gniazdka elektrycznego, magnetowid nigdy się nie wyłączy, gdy szpula kasety się obraca”. Wiele osób uważa, że o wiele bardziej naturalne jest opisywanie i omawianie zachowania reaktywnego systemu na podstawie jego scenariuszy, a nie na podstawie reaktywności każdego z jego składników opartej na stanie. Dotyczy to w szczególności niektórych wczesnych i późniejszych etapów procesu tworzenia systemu — np. podczas przechwytywania i analizy wymagań oraz podczas testowania i konserwacji. Mamy więc tutaj do czynienia z ciekawą i subtelną dychotomią. Jedna strona ma opisy behawioralne oparte na stanie, które pozostają w obiekcie i opierają się na zapewnieniu pełnego opisu reaktywności każdego z nich. Rodzaj podejścia wewnątrzobiektowego: „wszystkie fragmenty historii dla każdego obiektu”. Druga strona zawiera opisy zachowań oparte na scenariuszach, które przecinają granice obiektów systemu w celu dostarczenia zrozumiałych opisów scenariuszy zachowań (i zachowań zakazanych). Rodzaj podejścia międzyobiektowego: „każda historia podana poprzez wszystkie jej istotne obiekty”. To drugie jest bardziej intuicyjne i naturalne do uchwycenia przez ludzi i dlatego pasuje do wymagań i etapów testowania, ale to pierwsze podejście wydaje się być tym, które jest potrzebne do wdrożenia. Rzeczywiście, wydaje się, że implementacja systemu wymagałaby dostarczenia do każdego obiektu pełnego opisu obsługiwanych przez niego reakcji behawioralnych, tak aby można go było wykonać bezpośrednio lub poddać procesowi, który wygeneruje kod implementujący ten opis. Rysunek 3 jest próbą graficznego zilustrowania tych dwóch podejść.

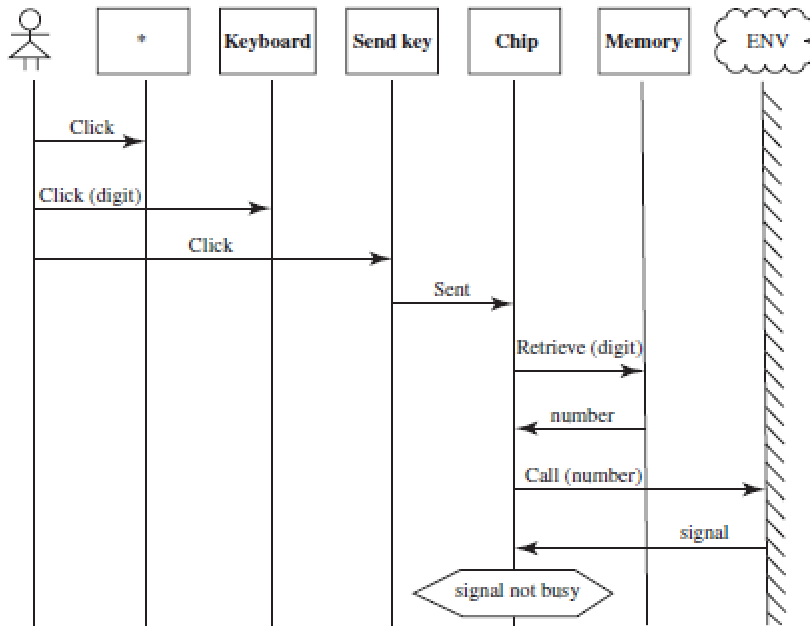



Po lewej stronie mamy każdy obiekt wypełniony wszystkimi jego małymi elementami zachowania - globalne zachowanie całego systemu wyprowadzone z wielkiej kombinacji tych wszystkich - a po prawej każda sekwencja zachowań przechodzi przez wszystkie istotne obiekty. Gdybyśmy chcieli na przykład opisać „zachowanie” typowego biura, dużo bardziej naturalne byłoby opisanie scenariuszy międzyobiektowych, takich jak wysyłanie przez pracownika 50 kopii dokumentu (może to dotyczyć pracownika sekretariatu, kserokopiarki, pokoju pocztowego itp.), czynności biurowych, których nie wolno wykonywać, o ile nie są one inicjowane przez szefa najwyższego szczebla, lub w jaki sposób informacje o urlopach i zwolnieniu lekarskim są organizowane i przekazywane do listy płac gabinet.

Porównajmy to ze stylem intra-obiektowym, w którym musielibyśmy podać pełną informację o sposobach działania i reaktywności szefa, sekretarza, pracowników, kserokopiarki, pokoju pocztowego itp. Tak więc, w przeciwieństwie do scenariuszy, stosowanych zazwyczaj do określania wymagań we wczesnych etapach rozwoju systemu, modelowanie za pomocą wykresów stanu lub bezpośrednio z kodem jest zwykle przeprowadzane na późniejszym etapie (zwykle projektowania) i skutkuje behawioralnym specyfikację dla każdej instancji obiektu (lub zadania lub procesu), podając szczegóły jego zachowania we wszystkich możliwych warunkach i we wszystkich „historiach". Ta specyfikacja obiekt po obiekcie będzie później testowana pod kątem scenariuszy i ewoluuje w implementację systemu, ponieważ pod koniec dnia ostateczny system będzie składał się z kodu (lub sprzętu) sterującego dynamicznym zachowaniem każdego obiektu. U podstaw tego procesu leży założenie - które wkrótce zakwestionujemy = że zachowanie oparte na scenariuszach międzyobiektowych nie jest wykonalne ani możliwe do wdrożenia. Rzeczywiście, jak działałby system opisany przez scenariusze? Co by zrobił w ogólnych warunkach dynamicznych? Skąd mamy wiedzieć w każdym punkcie, które scenariusze powinny się uruchomić i zacząć działać? W jaki sposób powinniśmy aktywnie upewniać się, że rzeczy, które muszą się wydarzyć, rzeczywiście się wydarzą, rzeczy, które mogą się wydarzyć, czasami się wydarzą, a rzeczy, które mogą się nie wydarzyć, rzeczywiście się nie wydarzą? Jak to wszystko egzekwujemy i co robimy, gdy napotykamy na niedookreślenie (brakujące informacje), przespecyfikowanie (niedeterminizm) i sprzeczności (zderzenia między „konieczne" i „nie wolno")?

**LSCs dla zachowania międzyobiektowego**

Formalizmem wizualnym używanym od wielu lat do określania scenariuszy, wywodzącym się z branży telekomunikacyjnej, jest język wykresów sekwencji komunikatów (MSC). Scenariusze są określone w MSC jako sekwencje interakcji komunikatów między instancjami obiektów. MSC są popularne w świecie zorientowanym obiektowo w fazie wymagań, w której inżynierowie identyfikują przypadki użycia – ogólne wzorce zachowań wysokiego poziomu – a następnie określają scenariusze, które je tworzą. Przechwytuje to pożądane zależności między instancjami obiektów oraz między nimi a środowiskiem zewnętrznym (np. użytkownikiem). Instancje obiektów są reprezentowane w MSC za pomocą pionowych linii, a wiadomości między tymi instancjami są reprezentowane przez poziome (lub czasami pochylone) strzałki. Warunkowe osłony, przedstawione jako wydłużone sześciokąty, określają stwierdzenia, które po osiągnięciu mają być prawdziwe. Ogólnym efektem takiego wykresu jest określenie scenariusza zachowania, składającego się z komunikatów przepływających między obiektami i rzeczy, które po drodze muszą być prawdziwe. Rysunek przedstawia prosty przykład MSC dla funkcji szybkiego wybierania telefonu komórkowego. Sekwencja komunikatów, które przedstawia, składa się z następujących elementów: użytkownik klika klawisz ∗, a następnie klika cyfrę na klawiaturze, a następnie klawisz Send, który wysyła wskazanie Sent do wewnętrznego Chipa, który z kolei wysyła cyfrę do Pamięci, aby pobrać numer telefonu skojarzony z klikniętą cyfrą. Chip następnie wysyła ten numer do otoczenia (np. anteny firmy komórkowej) w celu wykonania połączenia, po czym odbierany jest sygnał z otoczenia. Wreszcie, wykres zawiera warunek ochronny, który stwierdza, że sygnał rzeczywiście nie jest zajęty. Semantyka MSC jest egzystencjalna: wykres potwierdza, że scenariusz, który opisuje, reprezentuje możliwą sekwencję zdarzeń w życiu systemu. Zależne od czasu znaczenie samego scenariusza określają dwie proste zasady. Najpierw wzdłuż pionowej linii obiektu czas biegnie od góry do dołu. Po drugie, zdarzenie wysłania wiadomości poprzedza zdarzenie jej odebrania. Dlatego MSC nie mówią zbyt wiele o tym, co system faktycznie zrobi po uruchomieniu. Można ich użyć do powiedzenia, co może się wydarzyć, ale nie do określenia tego, co musi się wydarzyć. Na wykresie z rysunku 14.4, możemy na przykład zapytać, czy pamięć może „zdecydować", że nie będzie odsyłać numeru w odpowiedzi na żądanie od Chipa?

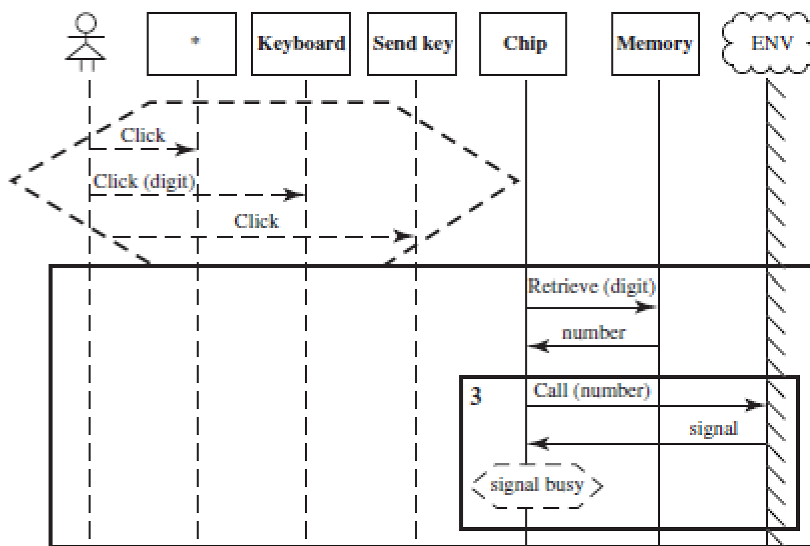



Czy warunek stwierdzający, że sygnał nie jest zajęty, musi być prawdziwy? Co się stanie, jeśli tak nie jest?

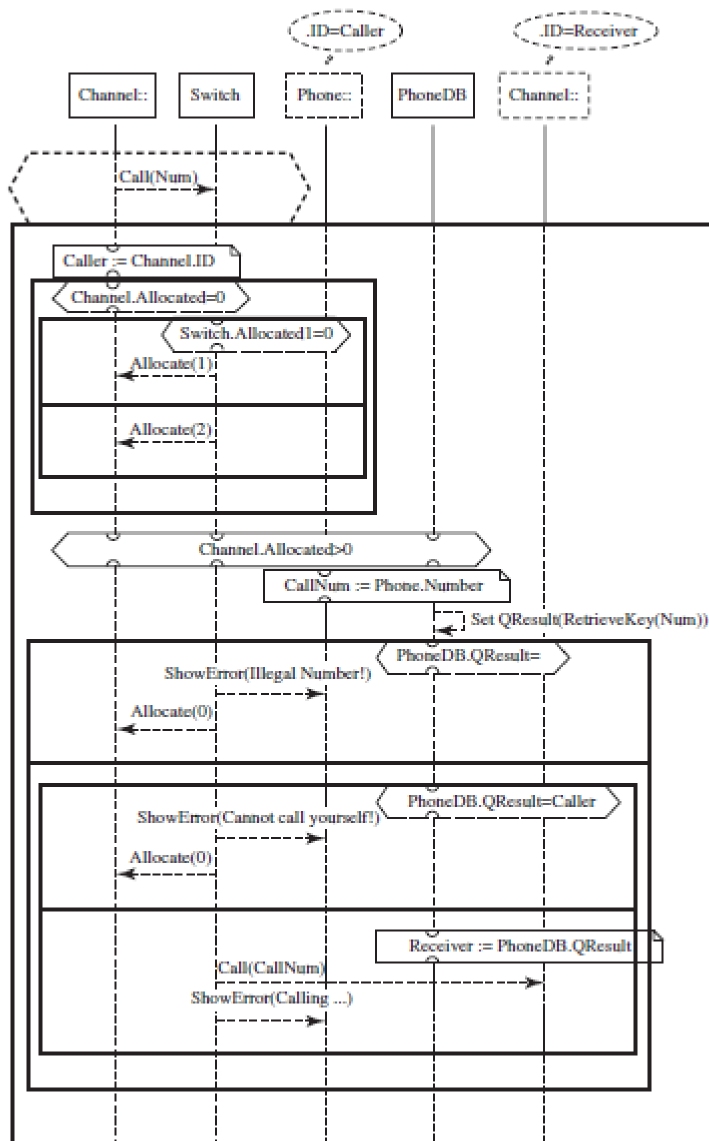
Takie wykresy rzeczywiście mogą być użyte do uchwycenia przykładowych scenariuszy oczekiwanego zachowania, które później zostaną porównane z ostatecznym systemem wykonywalnym. Jednak nie wystarczą, jeśli chcemy ich użyć do faktycznego stwierdzenia i potwierdzenia tego, co robi system. Chcielibyśmy móc powiedzieć, co może się wydarzyć, a co musi się wydarzyć, a także – jak wspomniano powyżej – co może się nie wydarzyć. Takie zabronione zachowania są czasami nazywane antyscenariuszami. Jeśli wystąpią podczas wykonywania systemu, jest coś bardzo nie tak: albo coś w specyfikacji behawioralnej nie zostało prawidłowo potwierdzone, albo implementacja nie spełnia poprawnie specyfikacji. Chcielibyśmy również móc określić wiele scenariuszy, które łączą się ze sobą, a nawet ze sobą, w subtelny sposób. Chcemy mieć możliwość określania ogólnych scenariuszy, tj. takich, które reprezentują wiele konkretnych scenariuszy, ponieważ mogą być tworzone przez różne obiekty tej samej klasy. Potrzebujemy zmiennych i środków do określania ograniczeń czasowych i tak dalej. MSC zostały rozszerzone na kilka sposobów, aby pomóc rozwiązać niektóre z tych problemów. Jedno z ostatnich rozszerzeń, zwane wykresami sekwencji na żywo lub LSC, wzięło swoją nazwę od możliwości określania żywotności, tj. rzeczy, które muszą wystąpić. LSC umożliwiają rozróżnienie między możliwym a koniecznym zachowaniem, zarówno globalnie, na poziomie całego wykresu, jak i lokalnie, podczas określania zdarzeń, warunków ochrony i postępu w czasie na wykresie. Tak więc w języku LSC występują dwa rodzaje wykresów: uniwersalne (oznaczone ciągłą linią graniczną) i egzystencjalne (opisane przerywaną linią graniczną). Bardziej interesujące są wykresy uniwersalne, które służą do określania zachowania opartego na scenariuszu, które ma zastosowanie do wszystkich możliwych uruchomień systemu. Uniwersalny wykres składa się z dwóch części, co zamienia go w rodzaj konstrukcji jeśli-to: prechart, który określa scenariusz, który, jeśli jest spełniony, zmusza system do spełnienia również drugiej części, głównego wykresu. udostępnia zestaw par scenariuszy akcji/reakcji, które muszą być spełnione przez cały czas podczas każdego uruchomienia systemu. W LSC, żywe elementy, określane jako gorące, oznaczają rzeczy, które muszą się wydarzyć, a inne, określane jako zimne, oznaczają rzeczy, które mogą się wydarzyć. Elementy gorące umożliwiają wymuszanie zachowania (i anty-zachowania), a elementy zimne mogą być używane do określania

struktur kontrolnych, takich jak rozgałęzienie i iteracja. Rysunek 14.5 pokazuje uniwersalny LSC, który jest w rzeczywistości wzbogaconą wersją MSC z rysunku powyżej

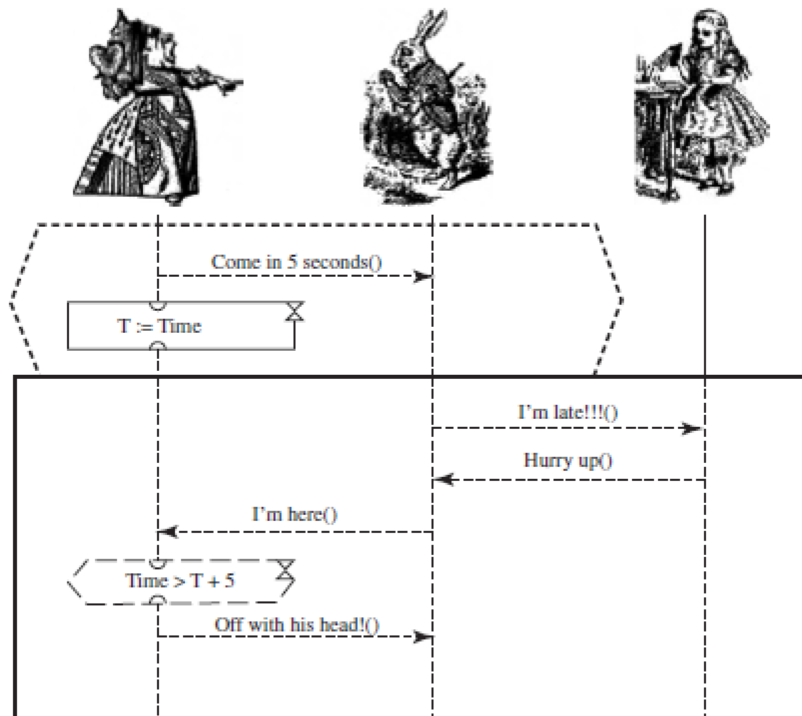
.




Pierwsze trzy zdarzenia znajdują się na wykresie wstępnym, a pozostałe na wykresie głównym. W związku z tym LSC stwierdza, że ilekroć użytkownik kliknie ∗, po którym następuje cyfra, a następnie klawisz Wyślij, musi zostać spełniona reszta scenariusza. (W szczególności, jeśli trzy zdarzenia z wykresu wstępnego nie zostaną zakończone, np. użytkownik nie naciśnie klawisza Wyślij, nic się nie stanie i niczego nie oczekuje się od systemu.) Komunikaty na głównym wykresie są gorące (przedstawione za pomocą ciągłych strzałek , w przeciwieństwie do linii przerywanych na precharte), podobnie jak linie pionowe. W związku z tym musi nastąpić postęp wzdłuż wszystkich linii na głównym wykresie, a wiadomości muszą zostać wysłane i odebrane, aby wykres był satysfakcjonujący. Dodatkowo dodano pętlę, w ramach której chip może wykonać do trzech prób odbioru niezajętego sygnału z otoczenia. Pętla jest kontrolowana przez warunek zimny (linia przerywana): dopóki sygnał jest zajęty, pętla trzyrundowa jest kontynuowana, ale jeśli tak nie jest, pętla zostaje zakończona (co oznacza, że cały wykres jest spełniony). W tym przykładzie wykorzystaliśmy semantykę zimnego warunku: jeśli jest prawdziwy, gdy zostanie osiągnięty podczas działania systemu, to dobrze, ale nawet jeśli jest fałszywy, nic złego się nie dzieje, a wykonanie po prostu przesuwa się w górę o jeden poziom, poza najbardziej wewnętrzny wykres lub podwykres. W przeciwieństwie do tego, stan wysokiej temperatury musi być prawdziwy po osiągnięciu. Jeśli to nieprawda, to bardzo źle rzeczywiście; w rzeczywistości jest to niewybaczalny błąd lub naruszenie, a system musi przerwać. Na przykład dobrym sposobem na określenie scenariusza przeciwnego z wykorzystaniem gorących warunków (np. otwarcie drzwi windy, gdy nie powinno, lub wystrzelenie pocisku, gdy radar nie jest namierzony na celu) jest uwzględnienie całego niepożądanego scenariusza w prechart, po którym następuje główny wykres, który zawiera jeden gorący stan, który zawsze jest fałszywy. (Dlaczego to działa?) LSC obsługują wiele dodatkowych funkcji, które nie zostaną tutaj szczegółowo opisane. Język jest wystarczająco potężny, aby określić większość aspektów zachowań reaktywnych. Rysunek jest częścią pełnej specyfikacji centrali wielotelefonicznej, której nie będziemy dalej wyjaśniać.




Zawiera między innymi symboliczne instancje, które odnoszą się do dowolnego telefonu lub kanału, warunki wiążące, konstrukcje jeśli-to-inaczej, gorące stany i nie tylko. Rysunek pokazuje proste użycie czasu w LSC, w połączeniu z instrukcjami przypisania i zimnym warunkiem.




Prechart pokazuje Królową Kier, która instruuje Białego Królika, aby przybył na swoje miejsce za pięć sekund, a następnie patrzy na zegarek, odnotowując czas. W rezultacie Biały Królik spotyka Alicję i mówi jej, że się spóźnia (wiadomość jest zimna, więc nie musi tego robić, ale może). Alicja pośpiesza Białego Królika, a po przybyciu na miejsce Królowej posłusznie informuje o swoim przybyciu. Królowa następnie sprawdza, czy nie upłynęło więcej niż pięć sekund. Jeśli rzeczywiście tak jest, wydaje rozkaz usunięcia głowy Białemu Królikowi; w przeciwnym razie (tj. jeśli zimny stan jest fałszywy) scenariusz kończy się spokojnie, a Biały Królik pozostaje z nienaruszoną anatomią . . .

**Podejście play-in/play-out**

Wokół LSC opracowano ostatnio dwuaspektową metodologię, zwaną playin/play-out. Pozwala użytkownikowi wygodnie określić zachowanie międzyobiektowe w oparciu o scenariusz, a następnie wykonać je bezpośrednio. Tak więc jest to tak naprawdę tylko sposób na zaprogramowanie systemu za pomocą LSC, a następnie uruchomienie programu. Pierwsza technika polega na przyjaznym dla użytkownika sposobie „zagrywania" zachowania bezpośrednio z GUI systemu (lub jego abstrakcyjnej wersji, takiej jak diagram modelu obiektowego), podczas którego LSC są generowane automatycznie. Druga technika umożliwia „odtworzenie" zachowania, to znaczy wykonanie systemu jako ograniczonego przez sumę informacji opartych na scenariuszu, symulując w ten sposób zachowanie systemu dokładnie tak, jak gdyby zostało określone w konwencjonalnym stanie. w oparciu o modę wewnątrzobiektową. Techniki te są obsługiwane przez narzędzie zwane Play-Engine. Główną ideą procesu play-in jest podniesienie poziomu abstrakcji w specyfikacji behawioralnej oraz praca z podobną wersją opracowywanego systemu nie tylko przy uruchamianiu modelu, ale także przy jego przygotowaniu. Dzięki temu osoby, które nie są zaznajomione z LSC lub nie chcą bezpośrednio pracować z takimi językami formalnymi, mogą określić wymagania behawioralne systemów za pomocą wysokopoziomowego i intuicyjnego mechanizmu. Mogą to być eksperci dziedzinowi, inżynierowie aplikacji, inżynierowie wymagań, a nawet potencjalni użytkownicy. „play-in" oznacza, że programista systemu najpierw buduje GUI systemu, bez wbudowanego zachowania, ale z podziałem na obiekty i

ich podstawowe izolowane możliwości. Na przykład przełącznik ma możliwość włączenia lub wyłączenia, a wyświetlacz kalkulatora jest określony jako zdolny do pokazania dowolnej sekwencji do, powiedzmy, 10 znaków. W przypadku systemów bez GUI lub zestawów obiektów wewnętrznych możemy po prostu użyć reprezentacji strukturalnej, takiej jak diagram modelu obiektowego jako GUI. W każdym razie użytkownik „odtwarza" GUI, klikając przyciski, obracając pokrętła i wysyłając wiadomości do obiektów w intuicyjny sposób przeciągnij i upuść. W podobny sposób grając w GUI, często używając myszki do wyboru spośród możliwości, użytkownik opisuje pożądane reakcje systemu i warunki, które mogą lub muszą wystąpić. W tym czasie odpowiednie LSC są konstruowane automatycznie. Pożądane modalności konstruowanego wykresu i jego elementów (uniwersalne/egzystencjalne, gorące/zimne) mogą być wybrane w procesie. Podczas odtwarzania użytkownik po prostu odtwarza aplikację GUI, tak jak zrobiłby to podczas wykonywania konwencjonalnego modelu opartego na stanie wewnątrz obiektu, ale ogranicza się do działań użytkownika końcowego i środowiska zewnętrznego. Proces rozgrywania wymaga, aby silnik gry monitorował odpowiednie premapy wszystkich uniwersalnych wykresów, a jeśli zakończy się pomyślnie, wykona ich główne wykresy, wypatrując naruszeń. Podstawowy mechanizm można przyrównać do nadmiernie posłusznego obywatela, który przez cały czas kręci się i trzyma Wielką Księgę Zasad. Tacy ludzie nie robią nic, chyba że zostaną o to poproszeni, i nigdy nie robią niczego, jeśli narusza to inną zasadę. Aby to osiągnąć, przez cały czas skanują i monitorują wszystkie reguły, a po wykonaniu dowolnej czynności (np. uniesienie palca) wielokrotnie dokonują wymaganych konsekwencji. Oczywiście, w ten sposób można dokonać wyborów i odkryć niespójności w przepisach. Silniejszym sposobem na wykonanie LSC jest użycie techniki zwanej smart play-out, w której wykorzystuje się zaawansowane techniki sprawdzania modelu z weryfikacji programu do obliczenia najlepszego sposobu, w jaki system ma reagować na działanie użytkownika, unikając w ten sposób wielu pułapki naiwnej egzekucji. Szczegóły jednak nie zostaną tutaj opisane. Istnieją dwa sposoby wykorzystania możliwości wykonywania LSC. Pierwszym z nich jest postrzeganie LSC jako wzbogacających konwencjonalny cykl rozwoju systemu, a drugim jest używanie ich jako samej możliwej do wdrożenia specyfikacji behawioralnej, która może prowadzić do nowego rodzaju cyklu rozwojowego. W pierwszym podejściu wykonywalne zachowanie oparte na scenariuszach oferuje ulepszenia niektórych standardowych etapów rozwoju systemu: wygodniejsze przechwytywanie wymagań, możliwość określenia bardziej zaawansowanych wymagań behawioralnych, sposób wykonywania bogatych przypadków użycia, narzędzia do dynamicznego testowania wymagania przed zbudowaniem rzeczywistego modelu systemu lub implementacją oraz środki do testowania systemów poprzez dynamiczne porównanie dwóch plików wykonywalnych z podwójnym widokiem w czasie wykonywania. Drugie podejście jest jednak bardziej radykalne. Wymaga rozważenia możliwości alternatywnego sposobu projektowania rzeczywistego zachowania systemu, który ma charakter scenariuszowy i międzyobiektowy. Ten pogląd proponuje ideę, że LSC (lub inny porównywalny język międzyobiektowy, taki jak diagramy czasowe lub logika temporalna) mogą faktycznie stanowić implementację systemu. Mechanizm play-out stanowiłby wówczas rodzaj „uniwersalnego systemu reaktywnego", który wykonuje zbiór LSC w sposób interpretacyjny, jakby była to konwencjonalna implementacja. Z tego punktu widzenia specyfikacja behawioralna systemu reaktywnego nie musiałaby wiązać się z dużym modelowaniem wewnątrzobiektowym (np. za pomocą maszyn stanów, wykresów stanów lub kodu). Ten pomysł jest wciąż dość wstępny, ale wydaje się obiecujący, ponieważ zachowanie oparte na scenariuszach jest sposobem, w jaki większość ludzi myśli o reaktywności. Kiedy stanie się możliwe uchwycenie tego myślenia w naturalny sposób i bezpośrednie wykonanie go, będziemy mieli środki do określenia możliwego do wdrożenia zachowania naszych systemów, które jest dobrze dopasowane do sposobu, w jaki o nim myślimy. A to może mieć wpływ na jakość, niezawodność i szybkość rozwoju złożonego systemu.

**Opracowywanie systemów czasu rzeczywistego**

Niektóre systemy reaktywne, zwane systemami czasu rzeczywistego, charakteryzują się tym, że czas odgrywa kluczową rolę w ich zachowaniu. Nie wystarczy, aby taki system prawidłowo reagował na bodźce, musi to robić w ściśle określonych terminach. Na przykład system zapobiegania kolizjom statku powietrznego musi wcześnie wydać ostrzeżenie , aby dać pilotowi czas na podjęcie działań mających na celu uniknięcie kolizji, a system przeciwhamulcowy w samochodzie musi działać wystarczająco szybko, aby zapobiec wypadkowi. Podobnie, system antyrakietowy musi zidentyfikować trajektorię nadlatującego pocisku na czas, aby wystrzelić własną broń przechwytującą. Tworzenie systemów czasu rzeczywistego jest szczególnie trudne, ponieważ rzeczywisty czas każdego elementu systemu musi być uwzględniony w obliczeniach jego czasów reakcji. W szczególności konwencjonalne systemy operacyjne są zwykle nieodpowiednie do wdrażania systemów czasu rzeczywistego, ponieważ nie zapewniają niezbędnych gwarancji czasu. W tym celu istnieją specjalne systemy operacyjne czasu rzeczywistego; zazwyczaj oferują one bardziej ograniczone usługi niż ich odpowiedniki ogólnego przeznaczenia, ale gwarantują ścisły czas reakcji. Określanie zachowania systemów czasu rzeczywistego jest również szczególnie trudne, ponieważ oprócz kwestii reaktywności i konwencjonalnych obliczeń, muszą one przestrzegać krytycznych ograniczeń czasowych. Chociaż wiele podejść do reaktywnej specyfikacji systemu może również dotyczyć aspektów czasu rzeczywistego, ostatnia propozycja jest dostosowana specjalnie do reaktywności zależnej od czasu. Nazywa się MASS, akronimem języka specyfikacji schematu aktywacji marionetek. Metafora marionetki sugeruje oddzielenie mechanizmu aktywacji od działań marionetek. W MASS niereaktywne aspekty systemu można określić przy użyciu dowolnego z wielu proponowanych formalizmów dla programów sekwencyjnych lub systemów transformacyjnych. Aspekty reaktywne, szczególnie te zależne od czasu, są ujmowane przez zestaw reakcji, z których każda określa reakcję systemu na jakieś zdarzenie, z możliwym ograniczeniem czasowym. Na przykład reakcja

[Włącz > Włącz → Aktywuj piec] < 2 s

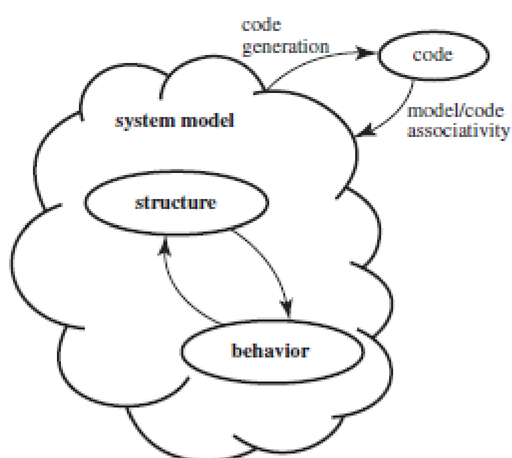
oznacza, że w ciągu dwóch sekund od momentu włączenia przełącznika zadanie Aktywuj piec musi zostać zakończone. Szczegóły tego, co dokładnie jest zaangażowane w to zadanie, są określone osobno, ponieważ nie współdziała z żadną inną czynnością w systemie. Reakcje mogą również mieć przerwane zdarzenia; na przykład reakcja [Train-out→Gate(Open)] : Train-in < 15sec określa, że brama musi zakończyć otwieranie nie później niż 15 sekund od momentu, gdy pociąg wyjechał z przejazdu kolejowego; jeśli jednak inny pociąg wjedzie na skrzyżowanie przed zakończeniem tego zadania, jest on przerywany. Inna reakcja, taka jak [Train-in→Gate(Close)] < 10sek określa, że brama powinna zakończyć zamykanie w ciągu 10 sekund od zdarzenia Train-in.

Tutaj pierwsza reakcja musi zostać przerwana, w przeciwnym razie wystąpią sprzeczne wymagania. Również w tym przykładzie szczegóły dotyczące otwierania i zamykania bramy mogą być określone innymi środkami i mogą być zaimplementowane w dowolnym języku programowania, dla którego można wyprowadzić ograniczenia czasowe; w praktyce ogranicza to wybór do języków asemblerowych lub języków niskiego poziomu, takich jak C. W ten sposób MASS pozwala projektantowi systemu czasu rzeczywistego skoncentrować się na aspektach aktywacji systemu w czasie rzeczywistym bez martwienia się o specyfikę aktywowane zadania w tym samym czasie.

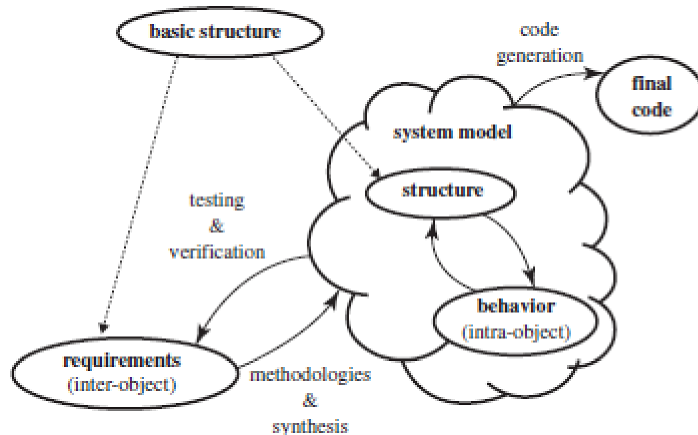
**Badania nad systemami reaktywnymi**

Większość tematów poruszanych w części poświęconej badaniom w poprzedniej części dotyczy wszelkiego rodzaju dużych systemów komputerowych, przy czym szczególny przypadek systemów reaktywnych nie stanowi wyjątku. To samo dotyczy tematów badawczych dotyczących poprawności i weryfikacji z Części 5. Jeśli już, wiele problemów staje się bardziej dotkliwych, gdy systemy są

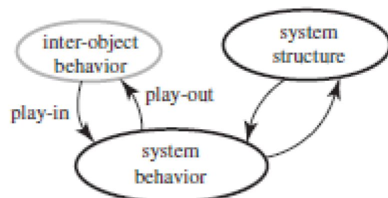
reaktywne, a zwłaszcza gdy mają rygorystyczne ograniczenia czasu rzeczywistego. Ponadto prowadzi się wiele badań nad formalizmami wizualnymi, ich wygodą i siłą wyrazu oraz ich implementacją i analizą. Jednym z godnych uwagi wysiłków jest zunifikowany język modelowania, UML, który rzekomo zebrać pod jednym dachem wiele powiązanych ze sobą schematycznych notacji dla rozwoju systemu (niekoniecznie systemów reaktywnych). Sercem behawioralnym UML jest zorientowana obiektowo wersja schematów stanów. MSC są również częścią UML i są używane do określania wymagań i sekwencji testowych pod nazwą diagramów sekwencji. UML jest oficjalnym standardem rozwoju systemu, koordynowanym i wydawanym przez Object Management Group. Jest to ciągły wysiłek, z okresowymi zaproszeniami do składania wniosków i rozszerzeniami, a od czasu do czasu wydawane są nowe wersje normy. Zespół UML podjął szeroko zakrojone próby zdefiniowania i zapisania znaczeń różnych języków, które go tworzą, oraz ich wzajemnych powiązań, ale wysiłki te są nieformalne i często niekompletne. Innym problemem, jaki niektórzy ludzie mają z UML, jest jego rozległy zakres, obejmujący wiele różnych języków używanych do wielu różnych celów. W szczególności UML udostępnia kilka różnych języków do określania reaktywnego zachowania, które można łatwo wykorzystać do nieumyślnego określenia rzeczy więcej niż raz na różne sposoby. Rodzi to subtelne kwestie spójności, które nie zostały jeszcze odpowiednio rozwiązane. Ponieważ UML jest akceptowanym standardem, i prawdopodobnie będzie tylko rósł w użyciu, semantycy są zajęci pracą, próbując zdefiniować niektóre z bardziej centralnych części UML w rygorystyczny sposób, czyniąc go podatnym na analizę komputerową i wymyślając sposoby zapewnienia spójności modeli UML. Weryfikacja systemów reaktywnych jest tematem szeroko zakrojonych prac, a niektóre techniki weryfikacji zostały przystosowane do pracy z wizualnymi modelami reaktywnymi i są włączane do narzędzi programistycznych na skalę przemysłową. Według wszelkiego prawdopodobieństwa trend ten będzie się utrzymywał, a przewidywalna przyszłość powinna nieść ze sobą możliwość automatycznej weryfikacji pewnych krytycznych właściwości złożonych systemów. Relacje między zachowaniem międzyobiektowym i wewnątrzobiektowym oraz pojawienie się języków takich jak LSC, rodzą wiele problemów związanych z różnicą między wymaganiami dotyczącymi zachowania a konwencjonalnym zachowaniem możliwym do wdrożenia. Podczas gdy weryfikacja dotyczy sprawdzenia, czy to pierwsze jest prawdziwe w odniesieniu do drugiego, inny temat badań dotyczy syntezy tego drugiego z pierwszym. Zobacz Rysunek, który pokazuje model systemu z rysunku




po prawej, ponieważ odnosi się do wymagań po lewej stronie.




Relacje polegają na (1) upewnieniu się, że to drugie obowiązuje dla pierwszego (poprzez testowanie i weryfikację) oraz (2) konstruowaniu pierwszego z drugiego (poprzez metodologie i syntezę). Na przykład byłoby miło, gdybyśmy mogli zapewnić wydajny algorytm do syntezy zwartych wykresów stanu z LSC. Niestety, złożoność najgorszego przypadku większości wersji problemu syntezy dla systemów skończonych jest bardzo zła - przynajmniej w czasie wykładniczym. Dobrą wiadomością jest jednak to, że chociaż to samo dotyczy problemu z weryfikacją takich systemów, nie przeszkodziło to w opracowaniu niezwykle przydatnych narzędzi weryfikacyjnych. W każdym razie synteza jest również tematem wielu badań. Rysunek  jest modyfikacją rysunku, mającą na celu zilustrowanie możliwości wykorzystania odtwarzanego zachowania międzyobiektowego jako rzeczywistej implementacji systemu reaktywnego.




Pomysł ten wymaga również dalszych badań, takich jak: opracowanie metodologii i heurystyk pozwalających ustalić, jakie rodzaje systemów byłyby najbardziej podatne na podejście; opracowanie kryteriów i wytycznych dla ustalenia kompletności takiej specyfikacji; oraz opracowanie algorytmów do określania wewnętrznej spójności specyfikacji opartej na scenariuszu oraz równoważności takiej specyfikacji z konwencjonalną. Ponadto, inteligentna rozgrywka wydaje się być interesującą linią przyszłej pracy, w której techniki weryfikacji są wykorzystywane nie do udowodnienia czegoś na temat modelu lub programu, ale do faktycznej pomocy w jego uruchomieniu w sposób, który pozwala uniknąć naruszeń i pułapek, przyczyniając się w ten sposób do „prawidłowego” ich wykonania. Wydaje się, że te i inne pokrewne tematy będą zajmować badaczy w tej dziedzinie już od dłuższego czasu. Wreszcie, kuszący nowy obszar badań obejmuje wykorzystanie technik, języków i narzędzi do reaktywnego rozwoju systemów w celu modelowania natury. Wydaje się, że wiele rodzajów układów biologicznych wykazuje reaktywność w dużym stopniu i na wielu poziomach szczegółowości, w tym na poziomie molekularnym i komórkowym, a także na poziomie całego organizmu. W ciągu ostatnich kilku lat nastąpił gwałtowny wzrost prac w tej dziedzinie i możliwe, że zaczniemy oglądać skomplikowane modele złożonych systemów biologicznych zbudowanych przy użyciu technik wywodzących się z informatyki. Takie modele będą wykorzystywane nie tylko do wspomagania biologów w wizualizacji i

zrozumieniu systemów biologicznych w animowanym zachowaniu, ale także do odkrywania luk lub błędów w wiedzy biologicznej, a nawet do przyczynienia się do rzeczywistych odkryć poprzez przewidywanie, które będą napędzać eksperymenty laboratoryjne.