دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکدهٔ مهندسی برق و کامپیوتر

طراحی منطق دامنه بر اساس تبادل ناهمگام پیغام

نگارش

وحید ذوقی شال

استاد راهنما

دکتر رامتین خسروی

پایان‌نامه برای دریافت درجهٔ کارشناسی ارشد در رشتهٔ

مهندسی کامپیوتر - گرایش نرم‌افزار

شهریور ۱۳۹۱

تقدیم به آنان که در خوشی‌هایم همراهی کردند و در ناخوشی‌هایم صبر؛

پدرم، مادرم و همسر مهربانم

قدردانی

خدای سبحان را سپاس می‌گویم که به من توان و قوه‌ی ذهنی عطا فرمود تا از عهده‌ی مشکلات انجام این پژوهش برآیم.

در ابتدا لازم می‌دانم از جناب آقای دکتر رامتین خسروی که در انجام این پژوهش افتخار استفاده از راهنمایی ایشان را داشتم، تشکر و قدردانی کنم. مطمئناً این کار بدون کمک‌های همه‌جانبه و بی‌شائبه‌ی ایشان امکان‌پذیر نبود. از اعضای هیئت داوران محترم نیز برای فرصتی که در اختیار من قرار دادند تشکر می‌کنم.

طراحی منطق دامنه بر اساس تبادل ناهمگام پیغام

چکیده

آزمون مبتنی بر مدل، به معنی تولید خودکار موارد آزمون از مدل کارکردی و صوری سیستم تحت آزمون به صورت جعبه سیاه، که به عنوان راه‌حلی برای مسئله‌ی تولید و اجرای خودکار آزمون‌ها مطرح شده است، استفاده‌ی روزافزونی درسال‌های اخیر داشته است. ایده‌ی آزمون مبتنی بر مدل، اساساً به هدف آزمون سیستم‌های با رفتار پیچیده (و معمولاً همروند) مطرح شده است. با این حال، در این روش‌ها مدل‌سازی رفتار سیستم توسط نمادگذاری‌های سطح پایین (مانند ماشین‌های گذار) انجام می‌شود. برای مثال، در این روش‌ها مدل‌سازی مقادیر داده‌ای به عنوان پارامترهای ورودی و خروجی سیستم تحت آزمون یا اصلاً امکان‌پذیر نیست و یا منجر به تولید مدل‌های بسیار پیچیده‌ای می‌شود.

از سوی دیگر، روش‌هایی برای آزمون خودکار نرم‌افزار معرفی شده‌اند که هدف آن‌ها تولید موارد آزمون شامل داده است. این روش‌ها به جای توصیف رفتار سیستم توسط ماشین‌های گذار، با استفاده از متن برنامه (به صورت جعبه سفید) امکان توصیف روابط میان مقادیر داده‌ای و چگونگی تولید مقادیر داده‌ای به هدف آزمون را مهیا می‌کند.

بر این اساس، در این پژوهش چهارچوبی یک‌پارچه برای مدل‌سازی همزمان رفتارهای مورد انتظار از سیستم تحت آزمون از یک سو و توصیف داده‌های ورودی و خروجی آن از سوی دیگر ارائه شده است. این چهارچوب برای توصیف این موارد از زبان یوام‌ال استفاده می‌کند. به این ترتیب امکان توصیف سیستم‌هایی که هم از نظر رفتاری و هم از نظر داده‌هایی که مبادله می‌کنند، پیچیده هستند فراهم می‌شود. در راستای طراحی این چهارچوب، هم‌چنین آزمون‌گری پیاده‌سازی شده است که موارد آزمون خود را بر اساس مدل‌های یوام‌ال به صورت خودکار تولید می‌کند. **واژه‌های کلیدی:** *طراحی منطق دامنه، تبادل ناهمگام پیغام، مدل بازیگر، همروندی*

فهرست مطالب

فهرست تصاویر

فصل ۱

مقدمه

از دیدگاه مهندسی نیازمندی ن

۱.۱ انگیزه‌ی پژوهش

VIII. CURRENT STATUS AND PERSPECTIVE Actor languages have been used for parallel and dis- tributed computing in the real world for some time (e.g. Charm++ for scientific applications on supercomputers, Er- lang for distributed applications). In recent years, interest in actor-based languages has been growing, among researchers as well as practitioners. This interest is triggered by emerg- ing programming platforms such as multicore computers, cloud computers, Web services, and sensor networks. In some cases, such as cloud computing, web services and sensor networks, the Actor model is a natural programming model because of the distributed nature of these platforms. As multicore architectures are scaled, multicore computers will also look more more like the traditional multicomputer platforms. This is illustrated by the prototype, 48-core Single-Chip Cloud Computer (SCC) developed by Intel [36]. However, the argument for using actor-based programming languages is not simply that they provide a good match for representing computation on

a variety of parallel and dis- tributed computing platforms. The point is that by extending object-based modeling to concurrent agents, actors provide a good starting point for simplifying the task of parallel (distributed, mobile) programming.

(از مقاله‌ی آقا ۲۰۱۰) روش‌های آزمون مبتنی بر مدل کاستی‌هایی نیز دارند. برای مثال در آی‌اوکو، استفاده از ماشین گذار برای توصیف سیستم تحت آزمون می‌تواند منجر به تولید مدل‌های بسیار پیچیده‌ای شود، زیرا ماشین‌های گذار علی‌رغم قدرت بیان بالا، چنان‌چه خواهیم دید، از نظر توصیفی نمادگذاری سطح پایینی محسوب می‌شوند و بنابراین مدل‌سازی جزئیات سیستم ممکن است حجم زیادی از پیچیدگی را در مدل‌ها به وجود آورد.

۲.۱ صورت مسئله

تمرکز اصلی این پژوهش بر آزمون مبتنی بر مدل سیستم‌های وابسته به داده[۱] قرار دارد. سیستم‌های وابسته به داده معمولاً حجم زیادی از اطلاعات را با محیط خود مبادله می‌کنند و رفتار آن‌ها وابسته به محاسباتی است که بر روی مقادیر داده‌ای انجام می‌دهند.

۳.۱ روش پژوهش

ارزیابی عملی با مطالعه موردی

۴.۱ روش ارزیابی

GQM

[۱] data dependent

۵.۱ خلاصه‌ی دستاوردهای پژوهش

برخی از دستاوردهای این پژوهش را می‌توان به این ترتیب برشمرد: داد. به این منظور نمای سطح بالا برای تولید آزمون‌ها و بررسی نتایج در روش اصلی آزمون مبتنی برمدل در شکل

۶.۱ ساختار پایان‌نامه

برای بررسی این موارد، ساختار این متن در ۶ فصل تنظیم گردیده است:

- به طور خلاصه مورد بررسی قرار گرفته‌اند.
- بررسی
- ه‌اند.
-

فصل ۲

پیش‌زمینه تحقیق

در این فصل به طور اجمالی مروری بر پیش‌زمینه‌ی پژوهش انجام شده است. در هر بخش سعی شده است که با حفظ اختصار، تنها جنبه‌های کاربردی مرتبط با پژوهش مطرح گردد.

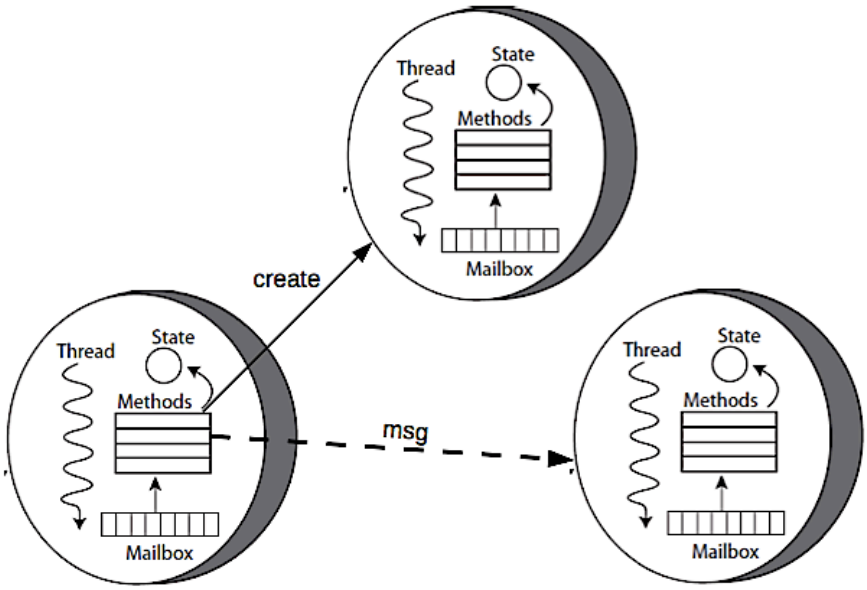
۱.۲ مدل بازیگر

در زمینه‌ی برنامه‌نویسی همروند پژوهش‌های مختلفی صورت گرفته و مدل‌هایی ارائه شده است[۱]. در این میان مدل بازیگر[۱] با توجه به استفاده از ارتباط ناهمگام و قابلیت توزیع بالا توجه زیادی را به خود جذب کرده است. با توجه به ارتباط تنگاتنگ این مدل با پژوهش حاضر، در این بخش به معرفی اجمالی این مدل می‌پردازیم. لفظ بازیگر برای اولین بار در حدود ۳ دهه پیش توسط هیوئیت [۲] به کار گرفته شد. بازیگر در کاربرد هیوئیت به معنی موجودیت‌های فعالی بود که در یک پایگاه دانش به جستجو پرداخته و در نتیجه کنش‌هایی را ایجاد می‌نمودند. در دهه‌های بعدی گروه هیوئیت با تکیه بر بازیگرها به عنوان عامل‌های محاسباتی[۲] مدل بازیگر را به عنوان یک مدل محاسباتی همروند گسترش داد. خلاصه‌ای از تاریخچه‌ی مدل بازیگر در [۳] موجود است. امروزه برداشت عمومی از مدل بازیگر مربوط به آقا[۴] می‌باشد. در ادامه‌ی این بخش مشخصات مدل بازیگر ارائه شده است.

---

[۱] Actor Model

[۲] agents of computation



شکل ۱.۲: بازیگرها موجودیت‌های همروندی هستند که به صورت ناهمگام تبادل پیغام انجام می‌دهند.

مدل بازیگر که توسط هیوئیت و آقا [۲، ۵، ۶] ایجاد شده‌است، یک نمایش سطح بالا از سیستم‌های توزیع‌شده فراهم می‌کند. بازیگرهااشیای لفافه‌بندی‌شده‌ای هستند که به صورت همروند فعالیت می‌کنند و دارای رفتار[۳] قابل تغییر هستند. بازیگرها حالت مشترک[۴] ندارند و تنها راه ارتباط بین آنها تبادل ناهمگام پیغام است. در مدل اکتور فرضی در مورد مسیر پیغام و میزان تاخیر در رسیدن پیغام وجود ندارد، در نتیجه ترتیب رسیدن پیغام‌ها غیرقطعی است. در یک دیدگاه می‌توان بازیگر را یک شیء در نظر گرفت که به یک ریسمان‌ریسمان[۵] کنترل، یک صندوق پست و یک نام غیر قابل تغییر و به صورت سرارسی یکتا [۶] مجهز شده است. برای ارسال پیغام به یک بازیگر، از نام آن استفاده می‌شود. در این مدل، نام یک بازیگر را می‌توان در قالب پیغام ارسال کرد. پاسخگویی به هر پیام شامل برداشتن آن پیام از صندوق پستی و اجرای عملیات متناسب با آن است. این اجرای عملیات به صورت تجزیه‌ناپذیر[۷] و بی‌وقفه خواهد بود[۴].

همان گونه که گفته‌شد، مدل بازیگر سیستم را در سطح بالایی از انتزاع مدل می‌کند. این ویژگی دامنهٔ سیستم‌های

---

[۳] Behavior

[۴] Shared State

[۵] Thread

[۶] Globally Unique

[۷] Atomic

قابل مدلسازی توسط مدل بازیگر را بسیار وسیع نموده‌است. انواع سیستم‌های سخت‌افزاری و نرم‌افزاری طراحی‌شده برای زیرساخت‌های خاص یا عام، و همچنین الگوریتم‌ها و پروتکل‌های توزیع‌شدهٔ مورد استفاده در شبکه‌های ارتباطی از جملهٔ موارد مناسب برای بهره‌گیری از مدل بازیگر هستند. علاوه بر این، خصوصیت تبادل ناهمگام پیغام، باعث می‌شود مدل بازیگر برای مدل کردن سیستم‌های توزیع شده و متحرک بسیار ایده‌آل باشد[۷]. شکل ۱.۲ شمای کلی از مدل بازیگر و نحوه‌ی تعامل بازیگرها را نشان می‌دهد.

یک بازیگر در نتیجه‌ی دریافت پیغام احتمالا محاسباتی انجام می‌دهد و در نتیجه‌ی آن یک از ۳ عمل زیر را انجام می‌دهد:

- ارسال پیغام به سایر بازیگرها
- ایجاد بازیگر جدید
- تغییر حالت محلی

۱.۱.۲ معناشناسی

[۸] از نظر معناشناسی مشخصه‌های کلیدی مدل محض بازیگر عبارتند از: لفافه‌بندی و تجزیه‌ناپذیری[۹]، انصاف[۱۰]، استقلال از مکان[۱۱]، توزیع[۱۲] و تحرک[۱۳] [۷]. باید توجه داشت که این مشخصه‌ها در مدل محض وجود دارند و این الزاما به این معنی نیست که تمام زبان‌های مبتنی بر مدل بازیگر از این مشخصه‌ها پشتیبانی می‌کنند. ممکن است تعدادی از این مشخصه‌ها در زبان‌های مبتنی بر بازیگر با در نظر گرفتن اهدفی مانند کارایی و سهولت پیاده‌سازی نشده‌اند. در این موارد باید با به کار بردن ابزارهای بررسی ایستا، مترجم‌ها و یا با تکیه بر عملکرد درست برنامه‌نویس از صحت عملکرد برنامه اطمینان حاصل کرد [۸].

---

[۸] Semantics

[۹] Encapsulation and Atomicity

[۱۰] Fairness

[۱۱] Location Transparency

[۱۲] Distribution

[۱۳] Mobility

- **لفافه‌بندی و تجزیه‌ناپذیری:**[۱۴] نتیجه‌ی مستقیم مشخصه‌ی لفافه‌بندی در بازیگرها این است که درهیچ دو بازیگری، به اشتراک گذاری حالت وجود ندارد. این مشخصه، تجزیه‌ی شیءگونه‌ی برنامه را تسهیل می‌کند. در زبان‌های برنامه‌نویسی شیء-بنیان مشخصه منجر به ایجاد تغییر تجزیه‌ناپذیر شده است. به این صورت که وقتی یک شیء، شیء دیگری را فراخوانی می‌کند، شیء مقصد تا پایان محاسبات مربوط به این فراخوانی، به فراخوانی‌های دیگر پاسخ نمی‌دهد. این مشخصه به ما اجازه می‌دهد تا بتوانیم در باره‌ی رفتار یک شیء در قبال دریافت یک پیغام (فراخوانی) با توجه به حالت شیء در زمان دریافت آن استدلال کنیم.

در محاسبات همروند، وقتی یک بازیگر مشغول انجام محاسبات مربوط به یک پیغام است، امکان دریافت پیغام جدید توسط آن وجود دارد اما مشخصه‌ی تجزیه‌ناپذیری تضمین می‌کند که پیغام جدید امکان قطع محاسبات جاری بازیگر و تغییر حالت محلی آن را ندارد. این مشخصه الزام می‌کند که بازیگر گیرنده، در هر لحظه فقط یک پیغام در حال پردازش داشته باشد و محاسبات مربوط به پیغام جاری را در یک قدم بزرگ[۱۵] به صورت تجزیه ناپذیر طی کند. [۳] مشخصه‌های معناشناسی لفافه‌بندی و تجزیه ناپذیری به طور چشم‌گیری از عدم قطعیت مدل بازیگر می‌کاهند و با کوچکتر کردن فضای حالت برنامه‌های نوشته شده در مدل بازیگر، این برنامه‌ها را برای استفاده در ابزارهای آزمون درستی و verification(؟) قابل استفاده می‌کند[۹]. این دو مشخصه مجموعا باعث می‌شوند تا بتوانیم بر اساس پیغام انتخاب شده برای اجرا و وضعیت محلی بازیگر در هنگام شروع به اجرا ، رفتار یک بازیگر قابل پیش‌بینی باشد.

- **انصاف:** انصاف در مدل بازیگر به این مفهوم است که پیغام فرستاده شده نهایتا به بازیگر مقصد خواهد رسید، مگر آنکه بازیگر مقصد به طور دائمی غیر فعال شده باشد. لازم به ذکر است که این تعریف از انصاف در رسیدن پیغام به بازیگر مقصد، متضمن انصاف در زمان‌بندی بازیگرها است. به این مفهوم که در صورتی که یک بازیگر در اثر زمان‌بندی غیر منصفانه، موفق به اخذ نوبت اجرا نشود، پیغام‌های فرستاده شده به مقصد آن بازیگر هرگز به مقصد نخواهند رسید. انصاف علاوه بر تضمین رسیدن پیغام‌ها، امکان استدلال مناسب درباره‌ی نحوه‌ی تداوم اجرای برنامه[۱۶] را فراهم می‌کند. میزان طبیعتا میزان موفقیت در تضمین این مشخصه در محیط‌های مبتنی بر بازیگر وابسته به منابع موجود در سیستم در حال اجرا است [۸].

- **استقلال از مکان، توزیع و تحرک:** در مدل بازیگر، ارسال پیغام به یک بازیگر تنها از طریق دسترسی به نام آن

---

[۱۴] Encapsulation and Atomicity

[۱۵] Macro-Step

[۱۶] Liveness Property

بازیگر ممکن می‌شود. مکان واقعی بازیگر تأثیری روی نام آن ندارد. هر بازیگر دارای فضای آدرس مربوط به خود است که می‌تواند کاملا متفاوت با دیگر بازیگرها باشد. بازیگرهایی که به یکدیگر پیغام می‌فرستند می‌توانند روی یک هسته از یک پردازنده‌ی مشترک اجرا شوند یا اینکه در ماشین دیگری که از طریق شبکه به آنها مرتبط می‌شوند در حال اجرا باشند. مشخصه‌ی استقلال از مکان در مدل بازیگر به برنامه‌نویس این امکان را می‌دهد که فارغ از نگرانی درباره‌ی محل اجرای بازیگر ها به برنامه‌نویسی بپردازد. عدم اطلاع از مکان اجرای بازیگران منجر به قابلیت حرکت در آنها می‌شود. تحرک به صورت قابلیت انتقال پردازش به نودهای دیگر تعریف می‌شود.در سطح سیستم، تحرک از جهت توزین بار [۱۷]، قابلیت تحمل خطا[۱۸] و نیز پیکربندی مجدد[۱۹] حائز اهمیت است. پژوهش‌های پیشین نشان می‌دهد که قابلیت تحرک در رسیدن به کارایی مقیاس‌پذیر به ویژه در کاربردهای بی‌قاعده[۲۰] روی ساختار داده‌های پراکنده مفید است[۱۰]. در کاربردهای دیگر، توزیع بهینه به شرایط زمان اجرا و میزان بار وابسته است. به عنوان مثال، در کاربردهای وب، تحرک با توجه به شرایط شبکه و امکانات کلاینت مورد استفاده قرار می‌گیرد[۱۱]. از سوی دیگر، قابلیت تحرک می‌تواند در کاهش انرژی مصرفی در اثر اجرای کاربردهای موازی مفید باشد. در این کاربردها، محاسبات موازی به صورت پویا بین تعداد هسته‌های بهینه (تعداد هسته‌هایی که منجر به کمترین مصرف می‌شوند) توزین می‌شوند. قسمت‌های مختلف یک کاربرد می‌تواند شامل الگوریتم‌های موازی مختلفی باشد و میزان مصرف انرژی یک الگوریتم به تعداد هسته‌های مشغول اجرای الگوریتم و نیز بسامد اجرای آن هسته‌ها بستگی دارد[۱۲]. در نتیجه، ویژگی تحرک پذیری بازیگرها، ویژگی مهمی برای برنامه نویسی در معماری‌های چند-هسته‌ای به شمار می‌آید.

۲.۱.۲ پیاده‌سازی‌ها

برای مدل بازیگر زبان‌ها و چارچوب‌های زیادی توسعه داده شده است. ABCL، POOL، ConcurrentSmalltalk، ACT++ و CEiffel تعدادی از پیاده‌سازی‌های اولیه از این مدل می‌باشند. مرجع [۱] به بررسی این زبان‌ها پرداخته است. شاید بتوان زبان ارلانگ[۲۱][۱۳] را معروفترین پیاده‌سازی مدل بازیگر دانست. این زبان در حدود ۲۲ سال قبل برای

[۱۷] Load-Balancing

[۱۸] Fault Tolerance

[۱۹] Reconfiguration

[۲۰] Irregular

[۲۱] Erlang

برنامه‌نویسی سوئیچ‌های مخابراتی شرکت اریکسون[۲۲] توسعه داده شد. علاوه بر ارلانگ زبان‌ها و چارچوب‌های مبتنی بر مدل بازیگر دیگری نیز در سال‌های اخیر مورد استفاده گرفته‌اند که کتابخانه‌ی بازیگر اسکالا [۲۳] [۱۴]، Ptolemy [۱۵]، SALSA [۱۶]، CHARM++ [۱۷]، ActorFoundry [۱۸]، Library Agents Asynchronous [۱۹] از جمله‌ی آنها هستند. از کاربردهای متن-باز که بر مبنای مدل بازیگر توسعه داده شده‌اند می‌توان به سیستم تبادل پیغام توئیتر[۲۴] و چارچوب تحت وب لیفت[۲۵] و از میان کاربردهای تجاری می‌توان به سیستم گپ[۲۶] فیسبوک و موتور بازی وندتا[۲۷] اشاره کرد. در این پژوهش برای پیاده‌سازی نسخه‌ی مبتنی بر تبادل ناهمگام پیغام از کتابخانه‌ی بازیگر اسکالا استفاده شده است (چرا؟) که در بخش ؟ معرفی شده است.

۲.۲ معرفی زبان اسکالاو کتابخانه‌بازیگر اسکالا

همان طور که در بخش ۲.۱.۲ اشاره شد، پیاده‌سازی‌های مختلفی از مدل بازیگر در زبان‌ها و چارچوب‌های برنامه‌نویسی ارائه شده است. مقاله‌ی [۸] به بررسی و مقایسه‌ی این پیاده‌سازی‌ها پرداخته است. در این پژوهش زبان اسکالا و کتابخانه‌ی بازیگر آن برای پیاده‌سازی مطالعه‌ی موردی انتخاب شده است. گستردگی ابزار و همچنین فعال بودن جامعه[۲۸]ی برنامه‌نویسی این زبان اصلی‌ترین انگیزه‌های انتخاب این زبان برای پیاده‌سازی بوده‌اند. ضمنا با توجه به انتخاب زبان جاوا برای پیاده‌سازی نسخه‌ی متداول مورد مطالعه و ارتباط تنگاتنگ زبانهای اسکالا و جاوا، انتخاب زبان اسکالا منجر به سهولت ارزیابی مقایسه‌ای مطالعه‌ی موردی شده است. در این بخش به معرفی اجمالی زبان اسکالا و کتابخانه‌ی بازیگر آن پرداخته شده است. هدف از این معرفی، سهولت درک روش طراحی پیشنهادی در فصل ۳ می باشد و به همین دلیل از توضیح جزئیات و امکانات اضافی این زبان خودداری شده است. کتاب [۲۰] به عنوان منبع این بخش استفاده شده است.

---

[۲۲] Ericsson

[۲۳] Scala Actor Library

[۲۴] Twitter

[۲۵] Lift

[۲۶] Chat

[۲۷] Vendetta game engine

[۲۸] Community

۱.۲.۲ زبان اسکالا

اسکالا مخفف عبارت *"زبان مقیاس‌پذیر"*[۲۹] است و اشاره به این نکته دارد که اسکالا برای رشد بر اساس نیاز کاربر طراحی شده است. اسکالا را می‌توان برای گستره‌ی وسیعی از کاربردها از نوشتن اسکریپت‌های کوچک گرفته تا پیاده‌سازی سیستم‌های بزرگ به کار برد. برنامه‌های اسکالا بر روی محیط اجرایی جاوا[۳۰] قابل اجرا هستند و در برنامه‌های اسکالا می‌توان از کتابخانه‌های استاندارد جاوا استفاده کرد. زبان اسکالا ترکیبی از ویژگی‌های زبان‌های تابعی و شیءگرا را در خود دارد. در زبان‌های تابعی، توابع مانند انواع داده‌ها قابل ارجاع هستند. اسکالا مانند جاوا دارای بررسی گونه‌های، ایستا است.

```
1  class Course(var id: String, var name: String, var units: Int,
2    var preRequisites: List[Course]) extends BaseDomain {
3
4    override def equals(other: Any): Boolean =
5      other match {
6        case that: Course =>
7            id == that.id
8        case _ => false
9      }
10
11   def printPrerequisites() = {
12     for (pre <- preRequisites)
13       println(pre)
14   }
15
16   override def toString = "[id= " + id + ",name=" + name + ",units=" + units + "]"
17 }
```

شکل ۲.۲: قطعه کد نمونه برای زبان اسکالا

در ادامه مشخصات نحوی زبان اسکالا در قالب یک مثال توضیح داده می‌شود. در شکل ۲.۲ قطعه کد اسکالا مربوط به کلاس Course نمایش داده شده است. برای آشنایی با نحو زبان اسکالا به بررسی این کد می‌پردازیم:

---

[۲۹] Scalable Language

[۳۰] JRE

در خطوط ۱ و ۲ کلاس Course و متغیرهای id، name، units و prerequisites به عنوان فیلدهای آن تعریف شده‌اند. در خط ۴ تابع equals از این کلاس override شده است. در اسکالا همانند جاوا هر کلاس به طور پیش‌فرض دارای یک تابع equals است که در صورت لزوم می‌توان آن را override کرد. همان‌طور که در کد مشخص است، تعریف تابع در اسکالا با کلمه‌ی کلیدی **def** انجام می‌گیرد. در خطوط ۴ تا ۸ شرط لازم برای یکسان بودن یک شیء از نوع Course با شیء حاضر پیاده‌سازی شده است. نوع و مقدار یک متغیر را می‌توان با استفاده از دستور match .. case .. با انواع و مقادیر دلخواه مقایسه کرد. نتیجه‌ی دستورات خطوط ۶ و ۷ این است که اگر متغیر other از نوع Course باشد و مقدار فیلد id آن با مقدار فیلد id از شیء حاضر یکسان باشد تابع مقدار true را برمی‌گرداند. خط ۸ به این معنا است که اگر هر حالت دیگری به جز حالت قبل بود مقدار false برگردانده می‌شود. در خط ۱۲ نمونه‌ای از حلقه‌ی for نمایش داده شده است. در اسکالا حلقه‌ها به صورت‌های متنوعی می‌توانند بیان شوند که در این مثال یک حالت از آنها نمایش داده شده است. در خط ۱۲ متغیر pre برای گرفتن مقدار موقت حلقه تعریف شده است. نکته‌ی جالب توجه این است که در این خط، نوع متغیر تعریف نشده است. در بخش قبل ذکر شد که اسکالا دارای خاصیت بررسی گونه‌های ایستا [۳۱] است. ظاهرا این دو امر در تناقض با یکدیگر هستند اما باید توجه داشت که در زبان اسکالا نوعی از استنتاجِ گونه[۳۲] در زمان ترجمه اتفاق می‌افتد. در این مورد با توجه به اینکه متغیر pre از لیست prerequisites مقداردهی می‌شود، گونه‌ی آن در زمان ترجمه قابل استنتاج است. خط ۱۶ تابع دیگری را نشان می‌دهد که در آن تابع toString، override شده است. نکته‌ی قابل توجه در مورد این قسمت از کد عدم استفاده از علامت { } برای تعیین حوزه‌ی تابع است. در زبان اسکالا به دلیل وجود ویژگی‌های زبان‌های تابعی، می‌توانیم با توابع مانند متغیرها و داده‌ها رفتار کنیم که این بخش از کد مثالی از این ویژگی است. همانطور که در این مثال مشخص است، در زبان اسکالا استفاده از نقطه‌ویرگول (;) در اکثر موارد اختیاری است.

۲.۲.۲ کتابخانه‌ی بازیگر اسکالا

همانطور که در بخش ۲.۱.۲ اشاره شد، یکی از پیاده‌سازی‌های مدل بازیگر، کتابخانه‌ی بازیگر اسکالا است. در این بخش به معرفی اجمالی کتابخانه‌ی بازیگر اسکالا و طرز استفاده از آن برای برنامه‌نویسی همروند می‌پردازیم.

---

[۳۱] static type checking

[۳۲] type inference

ایجاد بازیگر

بازیگرها در اسکالا از کلاس scala.actors.Actor مشتق می‌شوند. شکل ۳.۲ کد مربوط به یک بازیگر ساده را نشان می‌دهد. این بازیگر کاری به صندوق پیغام‌ها ندارد و صرفا پنج بار پیغام !I'm acting را چاپ می‌کند و سپس اجرای آن خاتمه می‌یابد.

```
1 import scala.actors._
2 object SillyActor extends Actor {
3   def act() {
4     for (i <- 1 to 5) {
5       println("I'm acting!")
6       Thread.sleep(1000)
7     }
8   }
9 }
```

شکل ۳.۲: کد یک بازیگر ساده در زبان اسکالا

بازیگرها در اسکالا با دستور start() شروع به فعالیت می‌کنند. با شروع به فعالیت یک بازیگر، تابع act() آن فراخوانی می‌شود و تا زمانی که اجرای این تابع به اتمام نرسد، بازیگر به طور همروند در حال اجرا باقی می‌ماند. در صورتی که بخواهیم بازیگر به طور دائمی در حال اجرا بماند دو راه وجود دارد. راه اول این است که تابع act() را در پایان کار خود مجدداً فراخوانی کنیم. و راه دیگر استفاده از عبارت **loop** در اسکالا است. دستورات درون حلقه‌ی loop به صورت بی‌پایان اجرا می‌شوند. شکل ۴.۲ کدهای مربوط به این ۲ روش را نمایش می‌دهد.

تبادل پیغام

عملگر ! برای فرستادن پیغام ناهمگام استفاده می‌شود. دستور dest ! message پیغام message را برای بازیگر dest ارسال می‌کند بدون آنکه برای دریافت جواب منتظر بماند. با اینکه در مدل اکتور دستوری برای تبادل همگام پیغام وجود ندارد، در اکثر پیاده‌سازی‌ها این امکان به مدل اضافه شده است[۸]. در کتابخانه‌ی بازیگر اسکالا، عملگر !? به این منظور به کار گرفته می‌شود. در صورت استفاده از این دستور، فرستنده‌ی پیغام تا گرفتن جواب متوقف می‌ماند. برای برداشتن پیغام از صندوق پیغام‌ها، از دو دستور receive و react استفاده می‌شود (تفاوت این دو دستور در بخش

```
1 object SillyActor extends Actor {
2   def act() {
3     loop {
4       for (i <- 1 to 5) {
5           println("I'm acting!")
6           Thread.sleep(1000)
7       }
8     }
9   }
10 }
```

(ب)

```
1 object SillyActor extends Actor {
2   def act() {
3     for (i <- 1 to 5) {
4       println("I'm acting!")
5       Thread.sleep(1000)
6     }
7     act()
8   }
9 }
```

(الف)

شکل ۴.۲: تداوم اجرای بازیگر با استفاده از الف)فراخوانی بازگشتی و ب)حلقه‌ی loop

۲.۲.۲ توضیح داده شده است). شکل ۵.۲ مثالی از نحوه‌ی تبادل پیغام بین بازیگران را نمایش می‌دهد. در این برنامه دو بازیگر PingActor و PongActor به تبادل پیغام می‌پردازند. در ابتدا بازیگر PingActor که متغیر آن با مقدار ۱۰۰ مقداردهی شده است یک پیغام Ping برای بازیگر PongActor می‌فرستد و در ادامه در یک حلقه‌ی loop منتظر پاسخ Pong می‌ماند. بازیگر PongActor با گرفتن هر پیغام Ping پاسخ Pong را برای فرستنده ارسال می‌کند. کلمه‌ی کلیدی **sender** در کلاس Actor اشاره‌گری به فرستنده‌ی پیغام در حال پردازش می‌باشد (خط ۶ از کد قسمت (ب) شکل ۵.۲). بازیگر PingActor با دریافت پاسخ Pong مقدار متغیر pingsLeft را چک می‌کند و در صورت مثبت بودن آن پیغام Ping بعدی را ارسال می‌کند و در غیر این صورت پیغام Stop را ارسال می‌کند. نهایتا با صفر شدن متغیر pingsLeft، بازیگر PingActor پیغام Stop را برای PongActor می‌فرستد. دستور exit که در پایان کار هر دو بازیگر استفاده شده است باعث می‌شود ریسمان اجرایی بازیگر رها شود و پس از اجرای این دستور بازیگر قادر به دریافت پیغام نخواهد بود.

زیرساخت اجرای همروند در کتابخانه‌ی بازیگر اسکالا

پردازش‌های همروند مانند بازیگرها با دو نوع استراتژی پیاده‌سازی می‌شوند:

- **پیاده‌سازی ریسمان-بنیان:** در این نوع پیاده‌سازی رفتار پردازش همروند به وسیله‌ی یک ریسمان کنترل می‌شود.

```
class PingActor(count: int, pong: Actor) extends Actor {
  def act() {
    var pingsLeft = count - 1
    pong ! Ping
    loop {
      receive {
        case Pong =>
          if (pingsLeft > 0) {
            pong ! Ping
            pingsLeft -= 1
          } else {
            pong ! Stop
            exit()
          }
      }
    }
  }
}
```

(الف) بازیگر Ping که فرستنده‌ی اولیه‌ی پیغام است

```
class PongActor extends Actor {
  def act() {
    loop {
      receive {
        case Ping =>
          sender ! Pong
        case Stop =>
          Console.println("Pong: stop")
          exit()
      }
    }
  }
}
```

(ب) بازیگر Pong که به پیغام ping پاسخ می‌دهد.

```
object pingpong extends Application {
  val pong = new PongActor
  val ping = new PingActor(100, pong)
  ping.start
  pong.start
}
```

(ج) کد اجرای برنامه‌ی PingPong

شکل ۲.۵: مثالی از نحوه‌ی تبادل پیغام بین بازیگرها

حالت اجرا[۳۳] به وسیله‌ی پشته‌ی ریسمان[۲۱]

- **پیاده‌سازی رویداد-بنیان:** در این مدل رفتار به کمک یک سری مجری رویداد[۳۴] پیاده‌سازی می‌شوند. این مجری‌ها از یک حلقه‌ی رویداد فراخوانی می‌شوند. حالت اجرای پردازش‌های همروند در این روش به کمک رکوردها یا اشیاء مشخصی که به همین منظور طراحی شده‌اند نگهداری می‌شود [۲۲].

مدل ریسمان-بنیان معمولا پیاده‌سازی راحتتری دارد ولی به دلیل مصرف حافظه‌ی بالا و پرهزینه بودن تعویض متن[۳۵] می‌تواند منجر به کارایی کمتری شود[۲۳]. از طرف دیگر مدل رویداد-بنیان معمولا کاراتر است ولی در طراحی‌های بزرگ پیاده‌سازی آن مشکل‌تر است[۲۴]. استفاده از مدل رویداد-بنیان منجر به ایجاد نوعی از وارونگی کنترل[۳۶] می‌شود: یک برنامه به جای فراخوانی عملیات مسدود کننده [۳۷]، صرفا تمایل خود به ادامه‌ی کار در صورت رخ دادن رویدادهای مشخص (مانند فشردن یک دکمه) را به محیط اجرا اعلام می‌کند. این اعلام تمایل با ثبت یک مجری رویداد در محیط انجام می‌شود. برنامه هیچ وقت این مجری‌های رویداد را فراخوانی نمی‌کند بلکه محیط اجرایی با وقوع هر رخداد، مجری‌های ثبت شده برای آن رویداد را فراخوانی می‌کند. به این ترتیب کنترل اجرای منطق برنامه نسبت به حالت بدون رویداد **وارونه** می‌شود. به دلیل پدیده‌ی وارونگی کنترل، تبدیل یک مدل ریسمان-بنیان به مدل رویداد-بنیان معادل معمولا نیاز به دوباره‌نویسی برنامه دارد[۲۵].

در پیاده‌سازی زیرساخت همروندی در کتابخانه‌ی بازیگر اسکالا هر دو رویکرد معرفی شده پیاده‌سازی شده‌اند و قابل دسترسی هستند. اصلی‌ترین عمیات مسدود کننده در مدل اکتور انتظار برای دریافت پیغام است. کنترل اجرا در صورتی مسدود می‌شود که پیغامی که بازیگر منتظر دریافت آن است در صندوق پیغام موجود نباشد. در بازیگرهای اسکالا، عمل برداشتن پیغام با دو دستور انجام می‌شود:

- دستور :receive با استفاده از این دستور، در صورتی که در صندوق پیغام بازیگر، پیغامی که با یکی از الگوهای معرفی شده در بدنه‌ی receive موجود باشد کد مربوط به الگوی مربوطه اجرا می‌شود. در غیر این صورت ریسمان اجرای این بازیگر مسدود می‌شود. در این حالت پشته‌ی فراخوانی تابع act() در بازیگر به صورت خودکار توسط محیط اجرایی ذخیره می‌شود و در صورت ورود پیغام متناسب اجرا به صورت ترتیبی از سر گرفته می‌شود. بنابراین

[۳۳] execution state

[۳۴] event handler

[۳۵] context switch

[۳۶] Inversion of Control

[۳۷] blocking operation

در پیاده‌سازی این دستور از رویکرد ریسمان-بنیان استفاده شده است.

- دستور :react با استفاده از این دستور، در صورتی که هیچ پیغام متناسبی در صندوق پیغام وجود نداشته باشد، به جای مسدود کردن ریسمان اجرای بازیگر، از رویکرد رویداد-بنیان استفاده می‌شود. این کار از طریق نوع خاصی از تابع در زبان اسکالا انجام می‌شود که هیچ‌گاه به طور معمولی اجرای آن خاتمه نمی‌یابد. بلکه پس از ثبت مجری رویداد مناسب در محیط اجرا، با استفاده از ایجاد یک استثناء[۳۸] اجرای تابع react و توابع شامل آن در بازیگر خاتمه می‌یابد. در این نوع توقف اجرا با توجه به اینکه ریسمان اجرا مسدود نمی‌شود، پشته‌ی فراخوانی تابع نیز ذخیره نمی‌شود و با برگشت به اجرای این تابع، محیط هیچ تاریخچه‌ای از اجرای قبلی آن ندارد. در نتیجه در هر بار بازگشت مانند اولین اجرا رفتار می‌کند. نتیجه‌ی مهم این خصوصیت این است که در صورت استفاده از react در یک بازیگر، هیچ کدی که بعد از این تابع نوشته شده باشد اجرا نخواهد شد. به همین دلیل برنامه‌نویس باید دقت کند که تابع react از نظر ترتیب اجرا همیشه آخرین کد بدنه‌ی یک بازیگر باشد. نتیجه‌ی استفاده از رویکرد رویداد-بنیان در بازیگرهای اسکالا افزایش چشمگیر کارایی در صورت استفاده از تعداد بسیار زیاد بازیگر در سیستم است.

به برنامه نویسان توصیه شده است که به جز در موارد خاص که نیاز به مسدود کردن ریسمان اجرای بازیگر وجود دارد، در بقیه‌ی موارد از رویکرد رویداد-بنیان استفاده کنند. توضیحات تکمیلی در مورد نحوه‌ی پیاده‌سازی هر دو رویکرد در کتابخانه‌ی بازیگر اسکالا و آنالیز کارایی و مقایسه با سایر پیاده‌سازی‌های مدل بازیگر در [۲۶] قابل دسترس می‌باشد.

[۳۸] exception

فصل ۳

کارهای پیشین

در این فصل به ارائه‌ی برخی کارهای پیشین و مرتبط به موضوع این پژوهش خواهیم پرداخت. در مورد هر یک از این موارد به ارتباط آن با بحث جاری، کاربرد و یا نقاط تأثیرگذار آن در موضوع این پژوهش و هم‌چنین ضعف ها و نقایص آن‌ها پرداخته شده است.

۱.۳ الگوهای برنامه‌نویسی بازیگر

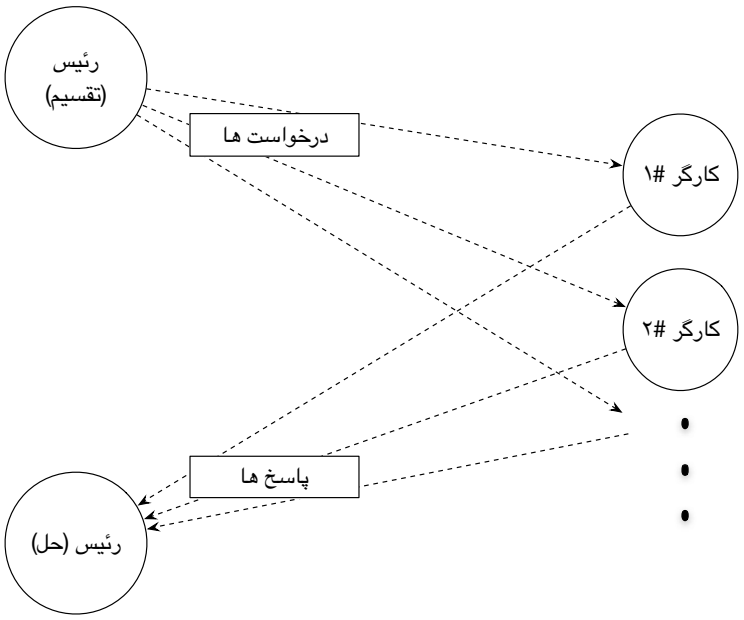
در برنامه‌نویسی همروند با بازیگرها دو نوع الگوی کلی معرفی شده است [۶]: یکی *تقسیم-و-حل*[۱] و دیگری *خط لوله*[۲]. در روش تقسیم-و-حل مسئله‌ی مورد بحث به زیربخش‌های کوچکتر و مستقل تقسیم می‌شود که هرکدام به صورت مستقل حل می‌شوند و نتایج هر زیربخش برای نتیجه‌گیری کلی ادغام می‌شوند. در برنامه‌نویسی به مدل بازیگر، برای پیاده‌سازی این الگو یک بازیگر رئیس[۳] در نظر گرفته می‌شود که تعدادی بازیگر کارگر[۴] را برای حل زیربخش‌های مسئله ایجاد می‌کند. عمل تقسیم به وسیله‌ی فرستادن پیغام حاوی حالت لازم برای حل زیر بخش به کارگرها انجام می‌شود.

[۱] devide and conquer

[۲] pipeline

[۳] master

[۴] worker

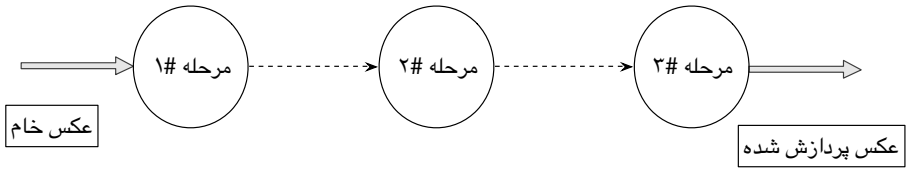


شکل ۱.۳: شمای کلی از الگوی تقسیم-و-حل در مدل بازیگر

کارگرها به نوبه‌ی خود منطق لازم برای حل زیر بخش را ایجاد نموده و نتیجه را به صورت پیغام دیگری برای بازیگر رئیس ارسال می‌کنند. نهایتا رئیس با ادغام نتایج جواب نهایی مسئله را تولید می‌کند. شایان ذکر است که فازهای تقسیم و حل لزوما توسط بازیگر یکسان اجرا نمی‌شوند. ممکن است اجرای فاز حل به بازیگر دیگری سپرده شود.[۲۷] مثال دیگری از پیاده‌سازی الگوی تقسیم-و-حل در مدل بازیگر در [۲۷] آمده است که در آن الگوریتم جستجوی سریع[۵] توسط این الگو پیاده شده است. شکل ۱.۳ شمایی از نحوه‌ی پیاده‌سازی الگوی تقسیم-و-حل در مدل بازیگر را نمایش می‌دهد.
الگوی خط لوله برای حالت‌هایی مناسب است که فعالیت قابل تقسیم به بخش‌های افزایشی باشد. در این صورت هر بازیگر تغییرات مربوطه را در مدل ایجاد می‌کند و آن را به عنوان پیغام به بازیگر بعدی در خط لوله منتقل می‌کند.

به عنوان مثالی از الگوی خط لوله یک برنامه‌ی پردازش تصویر را در نظر بگیرید. هر مرحله از خط لوله، تغییراتی را در تصویر دریافتی ایجاد می‌کند و تصویر نتیجه را به مرحله‌ی بعد منتقل می‌کند. در پیاده‌سازی با روش بازیگر، هر مرحله به صورت یک بازیگر مدل می‌شود و تصویر به صورت پیغام بین مراحل رد و بدل می‌شود. در شکل ۲.۳ شمایی از این الگو نشان داده شده است.
در پژوهش‌های انجام شده مشخص شد که الگوهای ارائه شده صرفا الگوهای کلی همروندی هستند و جزئیات این الگوها در طراحی منطق دامنه، نحوه‌ی طراحی پیغام‌ها بررسی نشده اند .

[۵] quick sort



شکل ۲.۳: مثالی از الگوی خط لوله (پردازش تصویر)

۲.۳ همگام‌سازی و هماهنگی بازیگرها

همان‌طور که در بخش‌های قبل ذکر شد، مدل بازیگر دارای خاصیت ناهمگامی است و ترتیب پیغام‌هایی که یک بازیگر دریافت می‌کند وابسته به ترتیب فرستاده شدن پیغام‌ها نیست. نتیجه‌ی این خاصیت این است که تعداد ترتیب[۶]های دریافت پیغام‌ها در مدل بازیگر نمایی است[۷]. به دلیل اینکه فرستنده‌ی پیغام از حالت محلی بازیگر گیرنده اطلاعی ندارد، ممکن است بعضی از ترتیب‌های ذکر شده برای پیغام‌ها مطلوب نباشد. به عنوان مثال الگوریتمی را در نظر بگیرید که زیر بخش‌های مختلف آن به بازیگرهایی فرستاده شده و نتایج آن دریافت می‌شود ولی در آن ترتیب دریافت نتایج اهمیت داشته باشد. نیاز به این نوع اولویت‌بندی‌ها در مدل بازیگر منجر به ایجاد پیچیدگی در محاسبات همروند می‌شود و در صورت پیاده‌سازی نامناسب باعث ایجاد ناکارامدی در برنامه‌ها می‌شود. راه حل این مسئله در مدل اکتور همگام‌سازی است. در مدل بازیگر، بازیگرها برای همگام‌سازی باهم ارتباط برقرار می‌کنند. در این قسمت دو نوع الگوی هماهنگی بازیگرها را معرفی می‌کنیم: تبادل پیغام شبه آرپی‌سی (فراخوانی رویه راه دور)[۷] و قیود همگام‌سازی محلی[۸]. [۶، ۲۸، ۲۹، ۷]

[۶] ordering

[۷] Remote Procedure Call

[۸] Local Synchronization Constraints

۱.۲.۳ تبادل پیغام شبه-آرپی‌سی

در ارتباط شبه-آرپی‌سی، فرستنده پس از ارسال پیغام منتظر گرفتن پیغام پاسخ از طرف گیرنده می‌ماند. رفتار بازیگر در این مدل به ترتیب زیر است:

۱. بازیگر فرستنده درخواست را در قالب یک پیغام به بازیگر گیرنده ارسال می‌کند.

۲. سپس فرستنده صندوق پیغام‌ها را بررسی می‌کند

۳. اگر پیغام بعدی پاسخ درخواست ارسال شده باشد اقدام مناسب صورت می‌گیرد و فعالیت بازیگر ادامه پیدا می‌کند.

۴. اگر پیغام بعدی پاسخ درخواست ارسال شده نباشد پیغام جاری در صورت امکان (بسته به منطق برنامه) پردازش می‌شود و در غیر این صورت برای پردازش در آینده به صندوق پیغام‌ها برگردانده می‌شود.

شکل ۳.۳ مثالی از پیاده‌سازی ارتباط شبه-آرپی‌سی در مدل بازیگر را نشان می‌دهد. ارتباط شبه-آرسی‌پی در دو نوع سناریوی خاص مفید و ضروری است: یک سناریو این است که بازیگر نیاز به ارسال پیغام به صورت ترتیبی به یک یا چند بازیگر خاص دارد و تا حاصل شدن اطمینان از رسیدن پیغام قبلی پیغام بعد را ارسال نمی‌کند. سناریوی دوم این است که حالت[۹] بازیگر فرستنده بستگی به محتوای پاسخ دارد. در این حالت بازیگر قبل از دریافت پاسخ مورد نظر، نمی‌تواند پیغام‌های بعدی را به درستی پردازش کند. نکته‌ی قابل توجه این است که با توجه به شباهت ارسال پیغام شبه-آرپی‌سی به فراخوانی رویه[۱۰]ها در زبان‌های ترتیبی[۱۱]، معمولا برنامه‌نویسان گرایش به استفاده‌ی بیش از حد از این نوع تبادل پیغام دارند که این ممکن است با ایجاد وابستگی‌های بی‌مورد در اشیاء برنامه، علاوه بر کاهش کارایی، منجر به ایجاد بن‌باز[۱۲] در برنامه شود (حالتی که یک بازیگر به علت انتظار برای پاسخی که هرگز دریافت نخواهد کرد، از پیغام‌های جدید مرتباً چشم‌پوشی می‌کند یا پردازش آنها را به تأخیر می‌اندازد).
امکان تبادل پیغام شبه-آرپی‌سی تقریبا در تمامی پیاده‌سازی‌های مدل بازیگر به صورت امکانات سطح زبان وجود دارد[۸].

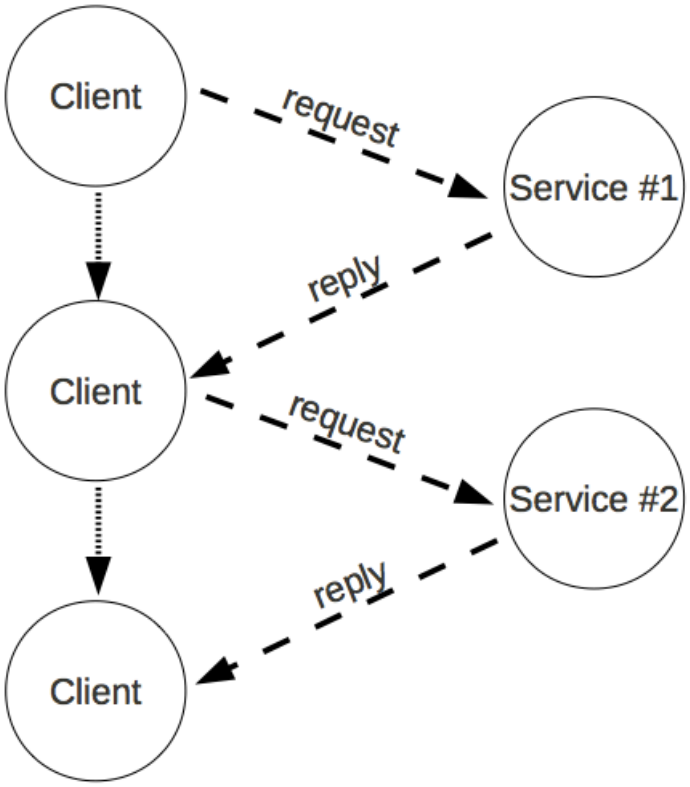
---

[۹] state

[۱۰] procedure

[۱۱] sequential

[۱۲] live lock



شکل ۳.۳: مثالی از ارتباط شبه-آرپی‌سی در بازیگرها)

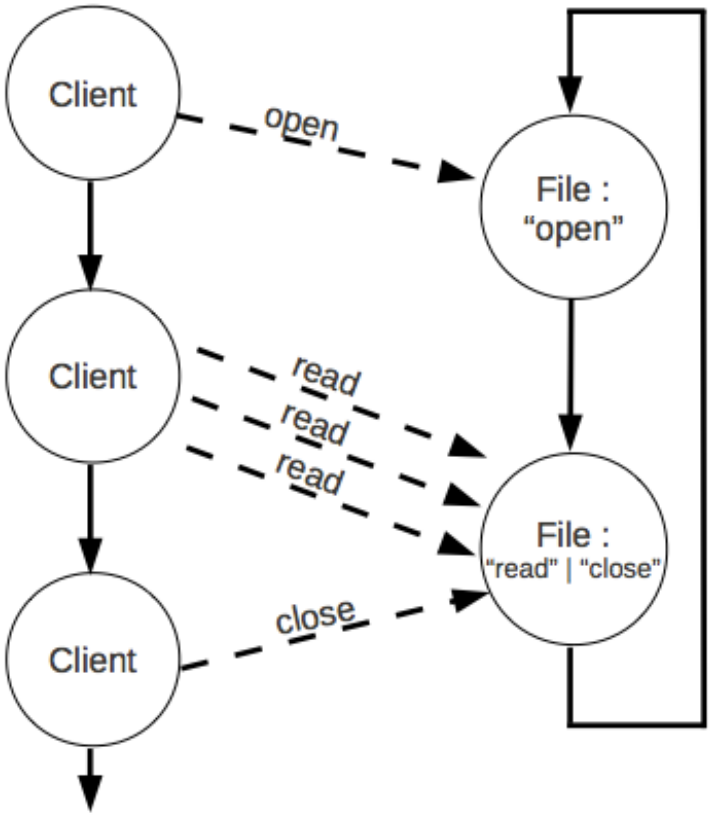
۲.۲.۳ قیود همگام‌سازی محلی

استفاده از قیود همگام‌سازی محلی روشی برای اولیت‌بندی پردازش پیغام‌ها در مدل بازیگر است[۳۰]. برای توضیح مفهوم همگام‌سازی محلی مثالی در شکل ۴.۳ ارائه شده است. در این مثال بازیگر فایل پس از دریافت پیغام باز کردن فایل[۱۳]، با استفاده از قیود همگام‌سازی خود را محدود به پردازش پیغام‌های *بستن*، *خواندن* می‌کند. در صورت عدم وجود امکانات مناسب برای قیود همگام‌سازی، برنامه‌نویس ناگزیر خواهد بود تا در میان منطق اجرای پیغام‌ها، میانگیر صندوق پیغام‌ها را بررسی و ترکیب یا ترتیب آنها را تغییر داده و یا با جستجو در آنها پیغام مناسب را انتخاب کند. این امر موجب مخلوط شدن منطق چگونگی پردازش پیغام (چگونه) با منطق زمانی انتخاب پیغام (چه زمانی) می‌شود که در اصول نرم‌افزار پدیده‌ی نامطلوبی به حساب می‌آید[۷]. به همین دلیل بسیاری از زبان‌ها و چارچوب‌های مبتنی بر بازیگر امکانات مناسبی برای پشتیبانی از قیود همگام‌سازی محلی ارائه داده‌اند. به عنوان مثال در کتابخانه‌ی بازیگر اسکالا که در بخش ۲.۲.۲ معرفی شد، از مکانیزم تطابق الگو[۱۴] برای اولیت بندی پردازش پیغام‌ها بدون اینکه با منطق اجرایی برنامه مخلوط گردد استفاده می‌شود.

---

[۱۳] open

[۱۴] pattern matching



شکل ۴.۳: مثالی از قیود همگام‌سازی محلی. بازیگر فایل به وسیله‌ی قیود همگام‌سازی محدود شده است. فلش عمودی به معنی ترتیب زمانی و برچسب‌های داخل دایره به معنی پیغام‌های قابل پردازش در هر حالت هستند.)

فصل ۴

روش طراحی پیشنهادی

در فصول گذشته روش آزمون ...

۱.۴ معرفی مطالعه‌ی موردی

- خصوصیت ۱
- خصوصیت ۲

۱.۱.۴ زیر بخش

این فصل فرض بر آن است که موارد کاربرد برای سیستم مورد نظر تهیه شده و موجود است.

تولید مدل رفتاری سیستم

در مدلی که در این مرحله تولید می‌شود، نیازی به مشخص کردن کنش‌هایی که باعث حرکت بین حالت‌های مختلف ماشین می‌شوند وجود ندارد. در واقع نمودار به دست آمده در این مرحله، تنها به هدف مدل‌سازی نمای سطح بالای عملکرد سیستم و مجموعه‌ی حالت‌های آن طراحی می‌ش

تولید مدل رفتاری محیط

آن‌چه که در ادامه‌ی این متن *محیط عملکرد* و یا اختصاراً محیط نامیده می‌شود، به طور دقیق عبارت است از اکتورهایی که در یک مورد کاربرد با سیستم در ارتباطند. همان‌طور که پیش از این نیز گفته شد، برای حفظ هم‌خوانی ماشین‌های طراحی شد نمودار رفتار سیستم) نمی‌شود، بنابراین این مسیر هرگز اجرا نخواهد ش

مشخص کردن و تعریف گونه‌های داده‌ای

اگرچه این بخش به توصیف مدل‌سازی رفتاری سیستم و محیط اختصاص دارد، اما باید توجه کرد که تکمیل مدل‌های رفتاری نمی‌تواند کاملاً مستقل از داده‌هایی که بین اجزای مدل مبادله می‌شوند، انجام شود. همان‌طور که پیش از این اصلی است. در چهارچوب پیشنهادی اجازه‌ی تعریف گونه‌های مستقل (یعنی گونه‌ای که از گونه‌ی دیگری گسترش نیافته است) وجو

۲.۴ طراحی سیستم به روش ناهمگام

۳.۴ الگوها و سبک‌های طراحی

پیش از این، نحو نمادگذاری مربوط به چهارچوب پیشنهادی این پژوهش تشریح شد. در این بخش، در مورد معنای هر یک از اجزای این نمادگذاری بحث خواهد شد. علاوه بر این، مطابقت معنایی این نمادگذاری را با مفاهیم آی‌اوکو مورد بررسی قرار داده و سپس الگوریتم جامعی برای تولید و اجرای یک‌پارچه‌ی آزمون‌هایی که با این نمادگذاری توصیف شده‌اند، ارائه خواهد شد.

۱.۳.۴ روشهای coordination

روش یک

همان‌طور که پیش از این اشاره شد، در چهارچوب پیشنهادی، توصیف‌های رفتاری چه برای سیستم و چه برای محیط در چندین نمودار حالت بیان می‌شود که هدف از آن کاهش پیچیدگی در طراحی مدل‌هاست. بنا به قرارداد، روش ترکیب این ماشین‌های حالت، روش میان‌گذاری است. به این معنی که

روش ۲

همان‌طور که پیش از این اشاره شد، توصیف‌های رفتاری محیط نقش محدودکننده را در تولید موارد آزمون ایفا می‌کند. به عبارت دقیق

۲.۳.۴ سبک های طراحی

سناریوهای آزمون در چهارچوب پیشنهادی، تعیین کننده‌ی رو با توجه به این تعاریف

۴.۴ پیاده‌سازی

نحو و معنایی که برای توصیف چهارچوب پیشنهادی در این پژوهش مورد استفاده قرار گرفته است، در قالب یک مجموعه‌ی ابزار نیز پیاده‌سازی نیز شده

فصل ۵

ارزیابی

در فصل قبل اجزای چهارچوب پیشنهادی این پژوهش به تفصیل تشریح شد و در م

۱.۵ روش ارزیابی

۲.۵ ارزیابی کارایی

سبیس

۳.۵ ارزیابی تغییرپذیری

سبیس

۱.۳.۵ بررسی معیارهای ایستا

با توجه به بزرگی سیستم مورد مطالعه، برای این مطالعه‌ی موردی دو مورد کاربرد از مجموعه‌ی مهم‌تری

۲.۳.۵ اعمال تغییرات

تغییر اول

۴.۵ نتایج ارزیابی

قبل از بررسی نتایج، لازم است برخی نکات در مورد اجرای آزمون‌ها مورد بررسی قرار گیرد. مطابق آن‌چه در فص

۱.۴.۵ تحلیل نتایج

با داشتن نتای

فصل ۶

جمع‌بندی و نکات پایانی

به عنوان جمع بندی متن حاضر، در این فصل به فهرستی از مهم‌ترین دستاوردهای این پژوهش خواهیم پرداخت. در مورد هر یک از این دستاوردها برخی نکات مهم نیز ذکر شده است. بعد از این، برخی از مهم‌ترین کاستی‌های چهارچوب ارائه شده آورده شده است. این کاستی‌ها در هر دو جنبه‌ی نظری و عملی مورد بررسی قرار گرفته‌اند. در نهایت، بر مبنای این موارد برخی جهت‌گیری‌های ممکن برای ادامه‌ی این پژوهش در آینده آورده شده است.

۱.۶ دستاوردهای این پژوهش

این پژوهش، چهارچوبی بدیع برای آزمون سیستم‌های نرم‌افزاری بر پسیستم واقعی استفاده می‌شود.

در واقع چهارچوب پیشنهاد شده تلاش می‌کند تا مجموعه‌ی به هم‌پیوسته‌ای از فعالیت‌ها برای آزمون را، از اولین مراحل طراحی تا نتیجه‌گیری از مجموعه‌ی آزمون‌ها، پیشنهاد کند. در زیر برخی از مهم‌ترین دستاوردهای هر یک از مراحل این کار آمده است:

۲.۶ کاستی‌های چهارچوب

چهارچوب پیشنهاد شده در این پژوهش دارای کاستی‌هایی نیز هست که کار بیشتری را می‌طلبد. در این بخش به طور فهرست‌وار به برخی از آن‌ها اشاره می‌کنیم:

۳.۶ جهت‌گیری‌های پژوهشی آینده

بهره می‌برند را نیز می‌توان به شکل زیر برشمرد:

پیوست آ

تطبیق نمادگذاری‌ها

متن برنامه‌ی طراحی شده به روش ارسال ناهمگام پیغام

سلام

متن برنامه‌ی طراحی شده به روش شیء‌گرا

تعریف ذکر شین گذار نمادین به طور ساده به این شکل است:

کتاب‌نامه

[1] J. pierre Briot, R. GUERRAOUI, K.-P. Löhr, and K. peter L, "Concurrency and distribution in object-oriented programming," tech. rep., 1998. 5, 9

[2] C. Hewitt, *Description and Theoretical Analysis (Using PLANNER: A Language for Proving Theorems and Manipulating Models in a Robot)*. Ph.D. thesis, Department of Computer Science, MIT, 1972. 5, 6

[3] G. Agha, I. A. Mason, S. F. Smith, and C. L. Talcott, "A foundation for actor computation," *J. Funct. Program.*, vol.7, no.1, pp.1–72, 1997. 5, 8

[4] G. Agha. *Actors: A Model of Concurrent Computation in Distributed Systems*. MIT Press, Cambridge, Mass, 1986. 5, 6

[5] G. Agha and C. Hewitt, "Concurrent programming using actors," pp.37–53, 1987. 6

[6] G. Agha, "Concurrent object-oriented programming," *Commun. ACM*, vol.33, no.9, pp.125–141, 1990. 6, 19, 21

[7] R. K. Karmani and G. Agha, "Actors," in *Encyclopedia of Parallel Computing*, pp.1–11, 2011. 7, 21, 24

[8] R. K. Karmani, A. Shali, and G. Agha, "Actor frameworks for the jvm platform: a comparative analysis," in *Proceedings of the 7th International Conference on Principles and Practice of Programming in Java*, PPPJ '09, (New York, NY, USA), pp.11–20, ACM, 2009. 7, 8, 10, 13, 22

[9] S. Lauterburg, R. K. Karmani, D. Marinov, and G. Agha, "Evaluating ordering heuristics for dynamic partial-order reduction techniques," in *FASE*, pp.308–322, 2010. 8

[10] W. Kim and G. Agha, "Efficient support of location transparency in concurrent object-oriented programming languages," in *SC*, 1995. 9

[11] P.-H. Chang and G. Agha, "Towards context-aware web applications," in *DAIS*, pp.239–252, 2007. 9

[12] V. A. Korthikanti and G. Agha, “Towards optimizing energy costs of algorithms for shared memory architectures,” in *SPAA*, pp.157–165, 2010. 9

[13] J. Armstrong, R. Virding, C. Wikström, and M. Williams. *Concurrent Programming in Erlang, Second Edition*. Prentice-Hall, second ed. , 1996. 9

[14] P. Haller and M. Odersky, “Actors that unify threads and events,” in *Coordination Models and Languages*, vol.4467 of *Lecture Notes in Computer Science*, pp.171–190, Springer Berlin / Heidelberg, 2007. 10

[15] E. A. Lee, “Overview of the ptolemy project,” Tech. Rep. UCB/ERL M03/25, University of California, Berkeley, 2003. 10

[16] C. A. Varela and G. Agha, “Programming dynamically reconfigurable open systems with salsa,” *SIGPLAN Notices*, vol.36, no.12, pp.20–34, 2001. 10

[17] L. V. Kale and S. Krishnan, “Charm++: a portable concurrent object oriented system based on c++,” *SIGPLAN Not.*, vol.28, pp.91–108, Oct. 1993. 10

[18] M. Astley, “The actor foundry: A java-based actor programming environment,” Open Systems Laboratory, Uni- versity of Illinois at Urbana-Champaign, 1998-99. 10

[19] Microsoft Corporation, “Asynchronous agents library,” `http://msdn.microsoft.com/en-us/library/dd492627.aspx`. 10

[20] B. V. Martin Odersky, Lex Spoon. *Programming In Scala*. WALNUT CREEK, CALIFORNIA: artima, 2 ed. , 2010. 10

[21] D. Lea. *Concurrent Programming in Java*. Addison-Wesley, 1996. 16

[22] M. Welsh, D. Culler, and E. Brewer, “Seda: an architecture for well-conditioned, scalable internet services,” in *Proceedings of the eighteenth ACM symposium on Operating systems principles*, SOSP ’01, (New York, NY, USA), pp.230–243, ACM, 2001. 16

[23] John Ousterhout, “Why threads are a bad idea (for most purposes),” Invited talk at USENIX, January 1996. 16

[24] R. von Behren, J. Condit, and E. Brewer, “Why events are a bad idea (for high-concurrency servers),” in *IN HOTOS*, 2003. 16

[25] B. Chin and T. Millstein, “T.d.: Responders: Language support for interactive applications,” in *In: Proc. ECOOP*, pp.255–278, 2006. 16

[26] P. Haller and M. Odersky, “Scala actors: Unifying thread-based and event-based programming,” *Theoretical Computer Science*, vol.410, no.2â€“3, pp.202 – 220, 2009. <ce:title>Distributed Computing Techniques</ce:title>. 17

[27] T. H. Feng and E. A. Lee, “Scalable models using model transformation,” 2008. 20

[28] G. Agha, S. Frølund, W. Kim, R. Panwar, A. Patterson, and D. Sturman, “Abstraction and modularity mechanisms for concurrent computing,” *IEEE Parallel and Distributed Technology: Systems and Applications*, vol.1, pp.3–21, 1993. 21

[29] T. Papaioannou, “On the structuring of distributed systems : the argument for mobility.,” 2000. 21

[30] S. Frølund. *Coordinating distributed objects: an actor-based approach to synchronization*. Cambridge, MA, USA: MIT Press, 1996. 24

واژه‌نامه‌ی فارسی به انگلیسی

ارلانگ ..... erlang
استثناء ..... exception
استدلال ..... reason
انصاف ..... fairness
ایستا ..... static
بازیگر ..... actor
بررسی گونه‌ها ..... type checking
بن‌باز ..... livelock
بی‌قاعده ..... Irregular
پراکنده ..... sparse
پشته ..... stack

تابعی ..... functional
تجزیه ..... decomposition
تجزیه‌ناپذیر ..... atomic
ترتیبی ..... sequential
تعویض متن ..... context switch
تقسیم-و-حل ..... devide and conquer
حالت مشترک ..... shared state
خط لوله ..... pipeline
رفتار ..... behavior
رویداد-بنیان ..... event-based
ریسمان ..... thread
ریسمان-بنیان ..... thread-based
زمان‌بندی ..... scheduling
شیء ..... object
شیء-بنیان ..... object-based
شیءگونه ..... object-style
غیرقطعی ..... non-deterministic, indeterminate
لفافه‌بندی‌شده ..... encapsulated
مجری رویداد ..... event handler
مسدود کننده ..... blocking
معناشناسی ..... semantics
مقیاس‌پذیر ..... scalable
وارونگی کنترل ..... inversion of control
همروند ..... concurrent

Integrating Functional and Structural Methods In Model-Based Testing

Abstract

Model-based testing (i.e. automatic test-case generation based on functional models of the system under test) is now widely in use as a solution to automatic software testing problem. The goal this testing method is to test complex systems (e.g. systems with concurrent behaviors). By the way, it exploits low-level notations (e.g. transition systems) to describe system specifications. Therefore, modeling some aspects of the system, such as input/output data values, may result in high-complexity of the resulted model or it may not possible at all. On the other hand, there are methods that focus on data-dependent systems. Hereby, they analyze the source code (in a white-box manner), instead if high-level behavioral models, to infer data dependencies and to define test data valuation method. In spite of their power in modeling data items, test behaviors in these methods should be designed manually and therefore, defining numerous and complex behaviors for the test process may lead to difficulties.

In this work, we introduce an integrated framework for modeling both the expected system behaviors and the input/output data structures, consistently. To this end, we have used UML language for modeling purposes. This enables us to describe systems that are both complex in behavior and the data. We have also developed a tool which automatically generates test-cases based on the defined UML models.

Keywords: *model-based testing, automatic test generation, test framework, testing data dependent systems, category partitioning methods.*

University of Tehran

School of Electrical and Compuer Engineering

Integrating Functional and Structural Methods In Model-Based Testing

by

Hamid Reza Asaadi

Under supervision of

Dr. Ramtin Khosravi

A thesis submitted to the Graduate Studies Office
in partial fulfillment of the requirements
for the degree of M.Sc

in

Computer Engineering

June 2010