۳-۱- مروری بر طراحی سیستم

در این بخش اصول طراحی سیستم را که با مراجعه به کارهای قبلی و اطلاعات خود طراح گردآوری شده است را ارائه خواهیم داد. این اصول کمک زیادی در ارائه یک سند معماری خوب می‌کنند، چرا که نقشه سیستم را با استفاده از نمودارهای بلوکی بیان می‌کنند. بدین ترتیب مراجعه کنندگان به این سند به راحتی خواهند توانست روند کاری سیستم را سریعاً دریابند. بعد از معرفی اصول طراحی سیستم، در فصول بعد اقدام به بررسی روند طراحی سیستم خواهیم کرد. در این فرآیند طراحی نیازمندی‌ها و قواعد حاکم بر سیستم که در بخش‌های قبل ذکر شد را دقیق‌تر بررسی خواهیم کرد؛ آن‌ها را پالایش و نهایتاً با استفاده از عبارت‌های زبان OCL بیان خواهیم کرد. در حقیقت استراتژی طراحی سیستم در این پروژه به این صورت می‌باشد که ساختار استاتیک سیستم را با استفاده از زبان UML، و مدل رفتاری و ساختار پویای آن را با کمک OCL طراحی کنیم، بدین ترتیب روند تبدیل مدل UML به زبان صوری AMN دقیق خواهد بود.

۳-۱-۱- مدلسازی فضای آدرس

هر فرآیند دارای فضای آدرسی می‌باشد که تعیین کننده محدوده‌ایی از حافظه مجازی است که آن فرآیند می‌تواند آدرس‌دهی کند. با مراجعه به [LIED96] و [LIED95] به ایده جالبی برای مدیریت فضاهای آدرس دست می‌یابیم که برای توسعه ریزهسته‌های محض بسیار مناسب می‌باشد.

در این دو مرجع فضای آدرس را با استفاده از توابع بیان کرده‌اند؛ بطوری که فضای آدرس اولیه را با تابعی با نام σ_0، و بقیه فضاهای آدرس را با تابعی با نام σ نشان می‌دهند. تابع σ_0 بصورت $\sigma_0 : V \to R \cup \{\emptyset\}$ تعریف می‌شود که در آن مجموعه V، مجموعه آدرس‌های مجازی می‌باشد که به یک آدرس حقیقی در مجموعه R، یا به یک نماد $\emptyset$ نگاشت می‌شود. لازم به ذکر است که یک نگاشت به $\emptyset$ بیانگر یک خطای صفحه (یا Page Fault) می‌باشد. تابع σ نیز بصورت $\sigma : V \to (\Sigma \times V) \cup \{\emptyset\}$ تعریف گردد. در این تعریف مجموعه Σ مجموعه تمام فضاهای آدرس (از جمله σ_0) می‌باشد. البته اگر دقیق‌تر به این موضوع نگاه کنیم، توصیف مجموعه‌ایی مثل مجموعه Σ در یک زبان فرمال مثل AMN به سادگی و بطور مستقیم میسر نمی‌باشد، چرا که نوع دو تابع σ و σ_0 با هم سازگار نبوده و سیستم نوع زبان AMN اجازه اجتماع این دو نوع را نمی‌دهد.

برای سادگی طراحی سیستم در این مرحله نیازمند این هستیم که توصیف عمومی‌تر و کلی‌تری از فضای آدرس داشته باشیم. بدین منظور ما توصیفی دیگر از فضای آدرس ولی مشابه با توصیف ارائه شده در [LIED96] و [LIED95] را ارائه خواهیم داد. این مدل از فضای آدرس به راحتی هم در زبان UML و هم در زبان AMN قابل توصیف می‌باشد. در حالت کلی می‌توان فضای آدرس هر فرآیندی را با توابعی بصورت $r : M \to \mathbb{N}$ و $i : M \to A \times \mathbb{N}$ و نهایتاً $\sigma : A \times \mathbb{N} \to M$ مدل کرد. در این تعاریف مجموعه‌های M و A زیر مجموعه‌هایی متناهی از بستار یک الفبای متناهی می‌باشند که بیانگر نام اشیاء هستند. مجموعه M، مجموعه نام‌های تمام اشیاء ایجاد شده Maplet Codomain می‌باشد[1]. مجموعه A نیز مجموعه اسامی تمامی فضاهای آدرسی می‌باشد که تا کنون در سیستم ایجاد شده‌اند. بطور دقیق‌تر می‌توان گفت: $M, A \subset \mathrm{T}^+$ که در آن مجموعه T مجموعه‌ایی متناهی از الفبای اسامی اشیاء می‌باشد[2]. در این توصیف، تابع σ فضاهای آدرس را مدل می‌کند که در آن اگر یک دوتایی فضای آدرس–عدد طبیعی به یک شئ Maplet Codomain نگاشت شود، بطوری که این شئ توسط هیچ یک از دو تابع i و r به مقداری نگاشت نگردد، آنگاه آن آدرس مجازی (عدد طبیعی–مؤلفه دوم از دوتایی ورودی تابع σ) در آن فضای آدرس (جمله‌ایی از زبان T^+–مؤلفه اول از دوتایی ورودی تابع σ) به هیچ فریمی نگاشته نمی‌شود، به عبارتی دیگر ارزیابی آن آدرس مجازی در فضای آدرس مذکور منجر به یک خطای صفحه خواهد شد. دو تابع i و r نیز به ترتیب تابع نگاشت غیر مستقیم و تابع نگاشت مستقیم نام دارند؛ یک شئ Maplet Codomain یا توسط تابع r به یک آدرس فیزیکی و یا توسط تابع i به یک آدرس غیر مستقیم نگاشت می‌شود. لازم است بدانید که توابع i و r بصورت Partial و تابع σ بصورت پوشا و Partial تعریف شده‌اند، در ضمن مجموعه M همواره به سه زیر مجموعه M_0، M_i، و M_r افراز شده است که در آن M_0 مجموعه تمامی اشیاء Maplet Codomain می‌باشد که توسط هیچ یک از توابع i و یا r به مقداری نگاشت نشده‌اند، و مجموعه‌های M_i و M_r به ترتیب مجموعه اشیاء نگاشت شده توسط تابع i و r را نشان می‌دهد.

[1] اصطلاح Maplet Codomain برای نشان دادن شئ‌ای که می‌تواند به یک عدد طبیعی یا به یک دوتایی فضای آدرس–عدد طبیعی نگاشت شود بکار می‌رود.

[2] در ادامه بیشتر در مورد اشیاء و معنای مدل‌های شئ‌گرا بحث خواهیم کرد، و مفهوم این اصطلاحات روشن‌تر خواهد شد.

برای طراحی بخش مدیر حافظه یک ریزهسته محض، Liedtke در [LIED95] پیشنهاد می‌کند سه عمل map، grant، و flush کافی است. عمل map برای نگاشت صفحه‌ایی از یک فضای آدرس به یک فضای آدرس دیگر، عمل grant نگاشت صفحه‌ایی از یک فضای آدرس به یک فضای آدرس دیگر و حذف آن از فضای آدرس مبدا، و نهایتاً عمل flush برای باز گرداندن تمامی صفحاتی که توسط فرآیند به سایر فرآیندها map شده‌اند بکار می‌رود. این اعمال را می‌توان بطور دقیق‌تر در مدل جدید فضای آدرس بصورت زیر بیان کرد:

عمل map: فرآیند p با فضای آدرس a صفحه‌ایی معادل با شئ n را به فضای آدرس فرآیند p' با فضای آدرس a' نگاشت می‌کند:

$$i\left(\sigma\left(a',n'\right)\right) \coloneqq \sigma(a,n)$$

عمل grant: فرآیند p با فضای آدرس a صفحه‌ایی معادل با شئ n را به فضای آدرس فرآیند p' با فضای آدرس a' اعـطا می‌کند:

در این مورد خاص کار قدری پیچیده‌تر می‌شود؛ به این صورت که عمل در مقابل حالت مستقیم با حالت غیر مستقیم متفاوت است. در صورتی که $\sigma(a,n)$ بصورت غیرمستقیم (توسط تابع i) نگاشت شود، داریم:

$$i\left(\sigma\left(a',n'\right)\right) \coloneqq i\big(\sigma(a,n)\big)$$

$$i \coloneqq i - \{(\sigma\left(a,n\right), i\left(\sigma\left(a,n\right)\right))\}$$

در غیر این‌صورت، اگر $\sigma(a,n)$ بصورت مستقیم (توسط تابع r) نگاشت شود، داریم:

$$r\left(\sigma\left(a',n'\right)\right) \coloneqq r\big(\sigma(a,n)\big)$$

$$r \coloneqq r - \{(\sigma\left(a,n\right), r\left(\sigma\left(a,n\right)\right))\}$$

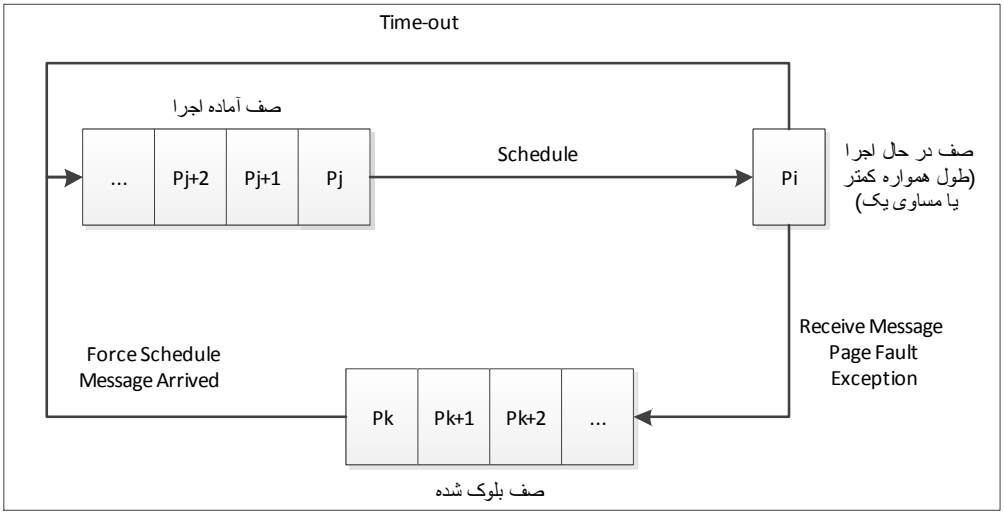
عمل flush (در این پروژه به این عمل reclaim می‌گوئیم): فرآیند p با فضای آدرس a صفحه‌ایی معادل با شئ n را از تمام فضاهای آدرس بازپس می‌گیرد.

$$i \coloneqq i - \left\{\left(m, i\left(m\right)\right) \middle|\, m \in M \wedge i\left(m\right) = \left(a,n\right)\right\}$$

البته این تعاریف قدری با تعاریف موجود در [LIED96] و [LIED95] متفاوت است، اما مفهوم و اثر هر دو تعریف یکسان می‌باشد. در ادامه خواهیم دید که چگونه بر اساس این تعاریف سیستم خود را مدلسازی خواهیم کرد.

۳-۱-۲- زمانبندی فرآیندها و صف‌های سیستم

همانطور که در بخش‌های قبل مشاهده کردیم در سیستمی که در حال طراحی آن هستیم، فقط سه عدد صف برای فرآیندها در نظر گرفته شده است. در شکل زیر نمودار بلوکی صف‌های سیستم را مشاهده می‌کنید.



شکل ۱- صف‌های سیستم، و رویدادهایی که موجب می‌شوند فرآیندها از صفی به صف دیگر منتقل شوند.

در این طرح ساده دیگر نیازی به ذخیره کردن حالت فرآیند در PCB نیست، چرا که حالت فرآیند از صفی که در آن قرار دارد مشخص می‌شود. برای اینکه طراحی سیستم سر راست‌تر باشد، فرآیند در حال اجرا را نیز در صفی قرار می‌دهیم تا حالت آن را از حالت بقیه فرآیندها متمایز کند، ولی از آنجایی که این سیستم برای کامپیوترهای تک پردازنده‌ایی طراحی شده است طول صف فرآیندهای در حال اجرا همواره کمتر یا مساوی یک خواهد بود.

همانطور که در شکل می‌توان مشاهده کرد، فراخوانی فراخوان سیستم receiveMessage، وقوع یک Page Fault، و یا یک استثناء موجب می‌شود فرآیند مورد نظر از صف در حال اجرا حذف و وارد صف بلوک شده شود. در حالت مسدود یک فرآیند همواره یا منتظر رسیدن یک پیام از سوی یک فرآیند خاص است و یا اینکه منتظر نگاشت و یا اعطای یک صفحه توسط یک فرآیند مسئول این کار

می‌باشد. لازم است بدانیم که این صف در ابتدا تهی بوده و هر فرآیندی که که بدون دلیل وارد این صف شود ممکن است هیچ گاه بیرون نیاید، ولی سیستم هیچ مسئولیتی در قبال چنین فرآیندهایی بر عهده ندارد. در حالت بلوک شده رسیدن یک پیام از سوی یک فرآیند (فرآیندی که فرآیند مسدود شده منتظر رسیدن پیام از سوی آن می‌باشد) و یا فراخوانی فراخوان سیستمی forceSchedule از سوی فرآیندی مسئول این کار، موجب می‌شود که فرآیند وارد صف آماده اجرا شود و از صف بلوک شده حذف گردد. در حالت آماده اجرا زمانبند سیستم، فرآیندی که در سر صف قرار دارد را از صف خارج و آن را به جای فرآیندی که در صف فرآیندهای در حال اجرا قرار دارد، می‌گذارد.

۳-۲- روش تبدیل دقیق مدل‌های UML به مدل‌های B

تفاوت عمده این پروژه با پروژه‌های مشابه در این است که ما از زبان UML برای مدلسازی شئ استفاده کرده‌ایم، و در کنار توسعه یک ریزهسته روشی را برای ترجمه مدل‌های UML به یک زبان مدلسازی فرمال نیز ارائه داده‌ایم. استفاده از UML در کنار زبان‌های مدلسازی فرمال این مزیت را دارد عمل صحت سنجی صوری مدل‌ها سریع‌تر می‌شود؛ در بسیاری از مقالات نیز به استفاده از UML در کنار روش‌های فرمال مانند B توصیه شده است، چرا که نویسندگان این مقالات معتقد هستند که استفاده از متد B به عنوان یک روش فرمال برای بسیاری از مهندسین (ولو با تجربه) دشوار است، و استفاده از UML در کنار B کمک زیادی به مدلسازی سریع جهان واقع خواهد کرد [LEDA01]، [LEDB01]، و [LEDC01]. تحقیقات زیادی در زمینه ترجمه مدل‌های ساخته شده در UML به مدل‌های فرمال صورت گرفته است، اما از آنجایی که زبان مدلسازی UML در بسیاری از موارد خیلی سطح بالا است و زیاد به جزئیات نمی‌پردازد کار ترجمه مدل‌های توسعه داده شده توسط این زبان به مدل‌های دقیق‌تر را دشوار و تقریباً غیرقابل استفاده در سطح وسیع شود (البته لازم است بدانیم که همه روش‌های ارائه شده بدون استثناء، به نحوی عمل صحت سنجی صوری را شتاب بخشیده‌اند). همانطور که می‌دانیم زبان مدلسازی UML در بسیاری از جنبه‌های مدلسازی سیستم از زبان طبیعی برای توصیف استفاده می‌کند که این یکی از مزیت‌های مهم این زبان است؛ این در حالی است که زبان‌های مدلسازی فرمال تا حد امکان از استفاده از زبان‌های طبیعی در توصیفات و مدلسازی پرهیز می‌کنند. یکی دیگر از دلایلی که باعث می‌شود ترجمه مدل‌های UML به مدل‌های فرمال دشوار شود (به عبارتی اصلی‌ترین دلیل) این است که مسئولین ارائه UML هیچگاه معناشناخت دقیقی برای اکثر بخش‌های این زبان ارائه نداده‌اند [BJOR06]، بدین ترتیب کسانی که می‌خواهند چهارچوب کاری و روشی را برای ترجمه مدل‌های UML به مدل‌های فرمال ارائه بدهند مجبور هستند خودشان بر اساس دانشی که از UML دارند، و اطلاعات اندکی که در مستندات مربوط به معناشناخت UML موجود است، یک معناشناخت برای UML تنظیم کنند و با استفاده از آن روش ترجمه را ارائه دهند. ما هم در این پروژه بر اساس دانش خود و بر اساس اطلاعاتی که از مستندات و کتب مربوط به UML [و OCL] دریافت کرده‌ایم روش ترجمه خود را ارائه خواهیم داد.

ایده‌ایی که در این پروژه بکار گرفته شده است این است که مدل‌های UML را تا حد امکان دقیق‌تر و با جزئیات بیشتر کنیم تا بتوانیم راحت‌تر و دقیق‌تر عمل ترجمه را انجام دهیم؛ در [WHIT07]، Lonnie و Whitten روشی را برای مدلسازی سیستم‌ها با استفاده از UML ارائه داده‌اند که در این پروژه بیشتر از این ایده برای دقیق‌تر کردن توصیفات مدل‌ها استفاده کرده‌ایم. بهتر است در ابتدا عمل مدلسازی خود را بدون محدودیت انجام دهیم و سپس بر اساس این مدل درستی و صحت آن را بصورت غیرصوری (البته با توجه به توصیه‌هایی که مهندسی نرم‌افزار در این زمینه ارائه می‌دهد) بررسی کنیم که اصطلاحاً به این فاز، *فاز تحلیل* گویند؛ در این فاز بسیاری از بخش‌هایی از مدل که موجب عدم تطابق آن با جهان واقع می‌شود شناسائی و رفع می‌گردد، بنابراین این فاز برای دقیق‌تر کردن و اطمینان از اینکه مدل حاضر همه جنبه‌های کار اصلی را مدلسازی می‌کند بسیار ضروری است. بـعـد از فاز تحلیل نوبت به *فاز طراحی* می‌رسد، در فاز طراحی مدل خود را با افزودن جزئیات بیشتر به پیاده‌سازی نزدیکتر می‌کنیم. باید دقت کرد که شیوه افزودن جزئیات به یک مدل که قرار است در یک زبان برنامه نویسی پیاده‌سازی شود، با مدلی که قرار است به یک زبان مدلسازی دیگر ترجمه و صحت سنجی شود متفاوت است؛ روش اول در کتاب [WHIT07] و سایر کتاب‌های مهندسی سیستم‌های نرم‌افزاری بررسی شده است و خارج از بحث و اهداف این پروژه می‌باشد؛ بحث اصلی شیوه دقیق‌تر کردن مدل‌ها به بگونه‌ایی می‌باشد که قادر باشیم آن را به یک زبان مدلسازی فرمال مثل AMN تبدیل کنیم و قادر به استفاده از روش B در توسعه سیستم خود باشیم.

در [PRES09]، Pressman طی مثالی از UML به همراه زبان OCL استفاده کرده است؛ با نگاهی دقیق‌تر و واقع بینانه‌تر به این مثال متوجه می‌شویم که برای استفاده از UML به همراه زبان‌های مدلسازی فرمال باید مدل خود را بسیار محدود سرشار از محدودیت‌های مختلف کنیم، که این کار بر خلاف اهداف ابداع UML به عنوان یک زبان مدلسازی با قابلیت افزایش سرعت مدلسازی می‌باشد [FOWL04]. یکی از دلایل دشوار بودن استفاده از روش‌های فرمال در مهندسی نرم‌افزار این است که پیدا کردن و لحاظ کردن تمامی محدودیت‌هایی که در کار موجود است، کار سختی است و نیاز به تجربه زیادی دارد. در این پروژه به جای اینکه مستقیماً سراغ مدلسازی با جزئیات بسیار زیاد برویم اقدام به افزودن میزان جزئیات در هر مرحله از مدلسازی کرده‌ایم که تجربه انجام این پروژه ثابت کرده است که این کار به میزان زیادی دشواری توسعه مدل‌های بسیار دقیق را کاهش می‌دهد.

زبان UML از جنبه‌های مختلف یک سیستم نرم‌افزاری را می‌تواند مدلسازی کند، در حالت کلی مدلسازی در UML به دو بخش *مدلسازی رفتاری* و *مدلسازی ساختاری* طبقه‌بندی می‌شود [FOWL04]. بخش ساختاری هر مدل فـضای حالات سیستم را مشخص می‌کند و خوشبختانه روش‌های کارآمد زیادی برای ترجمه آن به زبان‌های مدلسازی فرمال ارائه شده است، در بخش‌های بعدی روشی را برای ترجمه مدل‌های ساختاری سیستم‌های نرم‌افزاری ارائه خواهیم داد، و خواهیم دید که با این روش می‌توانیم بصورت سیستماتیـک عمل ترجمه را انجام دهیم. مدلسازی رفتاری، رفتار سیستم را در هر مورد استفاده نشان می‌دهد. دشوارترین مرحله ترجمه مدل‌های توسعه داده شده توسط UML به مدل‌های فرمال هم ترجمه بخش رفتاری مدل‌ها می‌باشد.

مطالبی در ادامه در دوبخش خواهیم دید مربوط به روش افزایش بهره‌وری تولید ریزهسته‌های نسل سوم است چـرا کـه مـی‌دانـیم از اهداف اصلی این پروژه افزایش بهره‌وری تولید ریز هسته‌های نسل سوم می‌باشد. در بسیاری از مقالات از جمله [LEDA01] ترکیبی از مدلسازی و توصیف فرمال و شئ‌گرا را برای استفاده از روش‌های فرمال در صنعت توصیه کرده‌اند. در این پروژه هم می‌خواهیم از ترکیبی از زبان UML و روش B برای توصیف و مدلسازی سیستم خود استفاده کنیم. از آغازین روزهای معرفی زبان UML همواره سعی بر آن بوده که معنای دقیقی برای زبان UML تعریف کنند، ولی زبان UML یک زبان فرمال برای مدلسازی سیستم‌های نرم‌افزاری نمی‌باشد. با مراجعه به کتب و مراجع زبان UML، مثل [FOWL04] مشاهده می‌کنیم که افراد درعرصه نرم‌افزار افراد بصورت‌های مختلفی از زبان مـدلسازی UML استفاده می‌کنند؛ برخی از این افراد از UML تنها برای بیان نکات مهم طراحی و بصورت خیلی غیر دقیق و زبانی استفاده می‌کنند، حال آنکه برخی دیگر از افراد از UML برای بیان دقیق ساختار و معماری نرم‌افزار استفاده می‌کنند که این افراد در مدلسازی خیلی دقیق‌تر هستند، چرا که می‌بایست بسیاری از جنبه‌های نرم‌افزار را توصیف کنند.

قبل از وارد شدن به بحث اصلی لازم به ذکر است که در این بین برخی از محققین عرصـــه مهندسی نرم‌افزار معتقدند که از UML می‌توان به عنوان یک زبان برنامه نویسی نیز استفاده کرد، این موجب شد که برای زیرمجموعه‌ایی از زبان UML معنای دقیقی تعریف شود [FOWL04]. لازم است بدانیم که تعریف معنای دقیق برای تمام عبارات زبان UML کاری بسیار دشوار بوده و محققین هنوز به توافقی در این زمینه نرسیده‌اند. دشواری تعریف معنای دقیق برای UML در تعریف معنا برای

ساختارهایی است که برای مدلسازی رفتاری بکار می‌روند؛ ما با مطالعه مقالاتی از جمله [FRAN98] به این نتیجه رسیدیم که دشواری تعریف معنای دقیق برای UML، در تعریف معنا برای ساختارهایی از UML است که برای مدلسازی رفتاری بکـــار می‌روند (مثـــــل Activity Diagram و Sequence Diagram) است. افراد زیادی سعی کرده‌اند که معنای دقیقی برای ساختارهای مدلسازی رفتاری UML تعریف کنند ولی تا کنون این روش‌ها در صنعت موفق نبوده‌اند و به همین خاطر استاندارد نشده است.

۳-۲-۱- دیاگرام‌های مدلسازی رفتاری

زبان UML دارای ساختارهای متعددی مانند دایاگرام‌های حالت[۳] و دیاگرام‌های فعالیت[٤] برای مدلسازی رفتاری است. به دلیل پیش زمینه تئوری دیاگرام‌های حالت برای این دیاگرام‌ها روش‌های متعددی توسعه یافته است که می‌توان آن‌ها را بدون خطا به کد تبدیل کرد [WHIT07]، [ROBE98] و یا در مورد درستی آن‌ها استدلال کرد، اما دیاگـرام‌های حالت برای مدلسازی رفتار بسیاری از برنامه‌ها مناسب نیست چرا که گاهی اوقات در رفتار برنامه‌هایی را باید بررسی کرد که به تعداد نامتناهی حـــالت دارند. دیاگرام‌های فعالیت در UML بیشتر برای بیان غیرصوری اعمالی[٥] که باید در هر مورد استفاده برای برطرف کردن نیاز در آن مـورد استـفاده انـجـام شود مورد استفاده قرار می‌گیرند، علی رغم اینکه دیاگرام‌های فعالیت سالهای زیادی است که بصورت تئوری و صوری مورد مطالعه و بررسی قرار می‌گیرند، استدلال در مورد دیاگرام‌های فعالیت بزرگ و پیچیده کاری بس دشوار می‌باشد و هنوز بصورت گسترده و در صنعت از آن روش‌ها استفاده نمی‌شود.

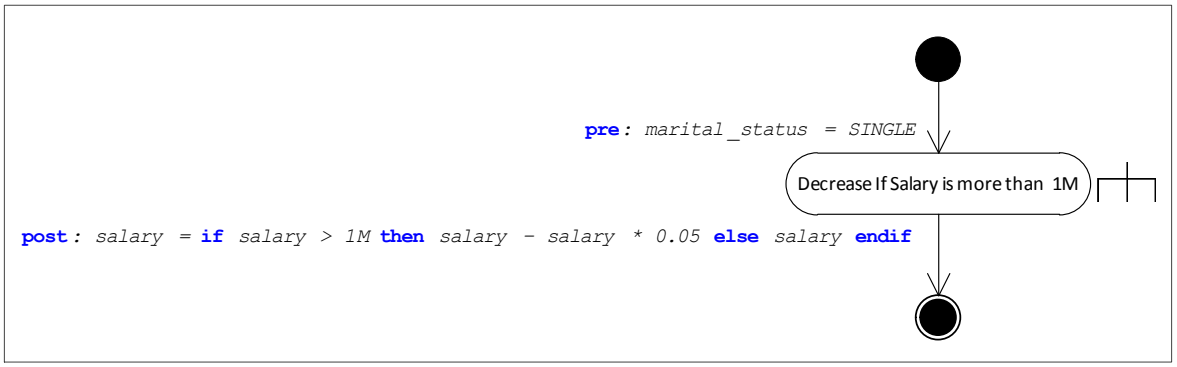
روشی که ما برای ترجمه دیاگرام‌های فعالیت استفاده کرده‌ایم بر اساس شکستن و کوچکتر کردن اعمال (یا همان Action ها) و بیان پیش‌شرط‌ها و اثرات مطلوبی (پس شرط‌ها) که از آنها انتظار داریم به زبان OCL می‌باشد. با وجود اینکه این روش هیچ تأثیری بر روی قابلیت توصیف[٦] دیاگرام‌های فعالیت برای توصیف اعمال موازی ندارد، برای سادگی از توصیف و استدلال در مورد اعمال موازی

[3] State-Chart Diagram
[4] Activity Diagram
[5] Action
[6] Expressiveness

می‌پرهیزیم و همواره فرض می‌کنیم که اعمال بگونه‌ایی هستند که اجرای آن‌ها با هر ترتیبی، و نیز اجرای آن‌ها بصورت موازی معادل با اجرای همان اعمال بصورت ترتیبی است.

ایده طراحی دقیق دیاگرام‌های فعالیت به این صورت است که در هر مرحله توصیف اعمال (یعنی توصیف Action ها) را دقیق و دقیق‌تر کنیم، و همچنین در هر مرحله پیش‌شرط‌ها و اثرات مطلوبی (یا همان پس‌شرط‌هایی) که از هر عمل انتظار داریم را با استفاده از زبان OCL بیان کنیم. دقت کنید که بیان اثرات مطلوب اعمال باید بگونه‌ایی باشد که بتوان آن‌ها را با استفاده از اعمال انتساب در زبان AMN بیان کنیم، بنابراین عمل شکستن اعمال بزرگ به اعمال کوچکتر باید تا حدی انجام شود که اثرات مطلوب اعمال بصورت انتساب‌های ساده باشد. در ادامه مثالی از یک دیاگرام فعالیت را مشاهده می‌کنیم که عمل شکستن و تجزیه اعمال را می‌توان روی آن مشاهده کرد، البته این مثال ساده می‌باشد و به راحتی می‌توان به همین شیوه مثال‌های پیچیده‌تری را نیز بیاوریم.

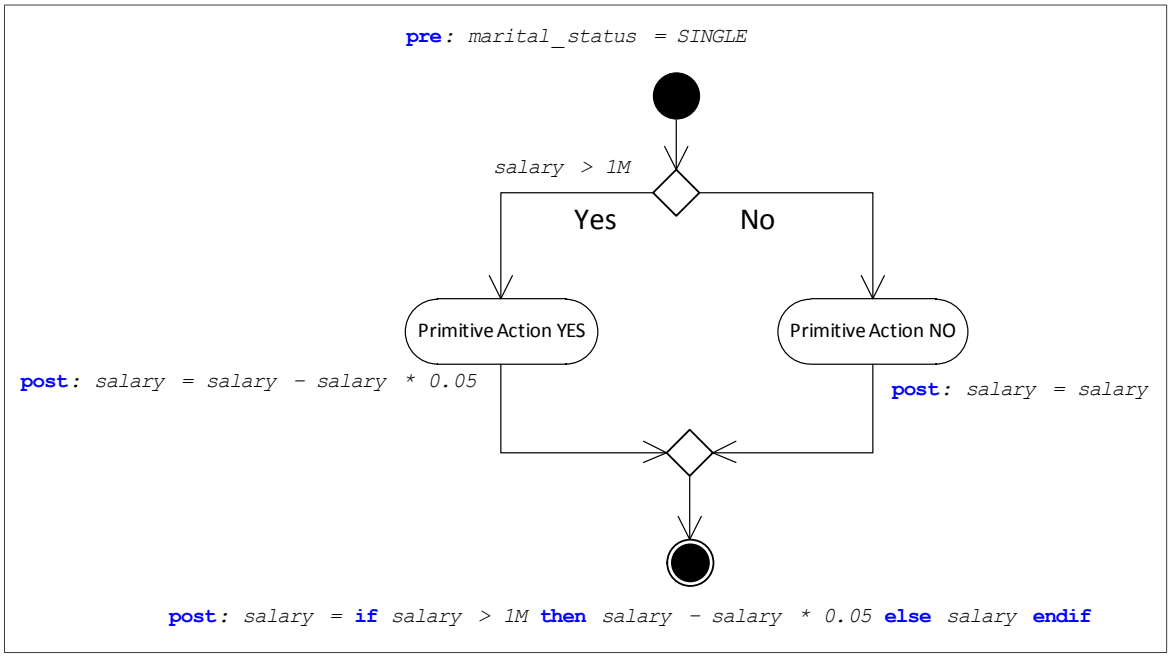
دیاگرام فعالیت زیر برای مدلسازی رفتار یک مورد استفاده که در آن با فرض اینکه کارمندی مجرد است، در صورتی که دستمزد وی بیش از یک میلیون تومان باشد ۵٪ از دستمزد او بکاهیم می‌باشد؛ لازم به ذکر است که این دیاگرام فعالیت و عبارات OCL استفاده شده در آن در متن کلاسی که یک خصیصه با نام marital_status دارد که از نوع شمارشی MARITAL_STATUS = {SINGLE, MARRIED} تعریف شده است.



شکل ۲- دیاگرام فعالیت در سطح یک.

همانطور که مشاهده می‌شود در این سطح بصورت خیلی کلی اثر مطلوب را بیان کرده‌ایم. البته این اثر را می‌توان باز هم تجزیه کرد (هرچند که اثر این عمل به اندازه کافی ساده است و در اصل نیازی به

تجزیه بیشتر نمی‌باشد، ولی برای نشان دادن ایده این کار را انجام می‌دهیم). در سطح دوم شرط if را می‌توان با استفاده از ساختار تصمیم‌گیری دیاگرام فعالیت بیان کرد:



شکل ۳- دیاگرام فعالیت سطح دوم.

حال به راحتی می‌توان این دیاگرام فعالیت را به زبانی مانند AMN تبدیل کرد، چرا که اعمال به اندازه کافی ساده بیـان شده‌اند و ابهامی در ترجمه آن به وجود نخواهد آمد (در ادامه روشی برای ترجمه بدون ابهـــام و دقیق عبارات OCL به زبان‌های دیگر بیان خواهیم کرد):

```
PRE
        marital_status = SINGLE
THEN
        IF salary > 1000000 THEN
                salary := salary – (salary × 5) / 100
        ELSE
                salary := salary
        END
END
```

شکل ٤- توصیف B معادل با دیاگرام فعالیت شکل ۳.

حتی می‌توانستیم این توصیف را خیلی تمیزتر و کوتاه‌تر هم بنویسیم ولی در این صورت راهکار ترجمه ما فرآیند ساده خود را از دست می‌دهد، بنابراین از دستکاری بیشتر می‌پرهیزیم.

در این مرحله و قبل از ورود به بحث ترجمه عبارات OCL به AMN بد نیست در مورد یکی دیگر از دیاگرام‌های UML به نام دیاگرام مورد استفاده بحث کنیم؛ لازم به یادآوری است که دیاگرام موارد

استفاده برای بیان تصویری نیازمندی‌های عملکردی سیستم می‌باشد و برای مدلسازی رفتاری مورد استفاده قرار نمی‌گیرد. ولی باید توجه کرد که دیاگرام موارد استفاده واسط کاربر با سیستم را نیز نشان می‌دهد. در [SCHN01]، Schneider از دیاگرام‌های موارد استفاده برای نشان دادن عملگرهای یک ماشین استفاده کرده است؛ ما هم از این ایده استفاده خواهیم کرد، و در توسعه نهایی خود سعی خواهیم کرد به تعداد موارد استفاده و حتی‌الامکان هم‌نام با آن‌ها در ماشین خود عملگر تعبیه کنیم.

۳-۲-۲- ترجمه عبارات OCL به عبارات AMN

در حالت کلی ترجمه یک جمله از یک زبان در زبانی دیگر را بصورت زیر تعریف می‌کنیم.

تعریف اول: اگر e جمله‌ایی از زبان A یا B باشد، $[\![e]\!]$ معناشناخت دلالتی آن را نشان می‌دهد؛ به عبارتی دیگر این عملگر نگاشتی است که یک عبارت را به یک ساختار ریاضی نگاشت می‌کند.

تعریف دوم: جمله $\hat{s} \in B$ را ترجمه جمله $s \in A$ گوئیم هرگاه $[\![\hat{s}]\!] = [\![s]\!]$.

البته متاسفانه بیش از این نمی‌توانیم در این مورد دقیق شویم و تنها ادعا می‌کنیم که در صورت در دست داشتن معناشناخت دلالتی زبان‌های OCL و AMN می‌توانیم از تعریف دوم استفاده کنیم. معناشناخت زبان‌های OCL و AMN را می‌توان به ترتیب در [OCL203] و [SCHN01] یافت؛ البته در بخش‌های بعدی نگاهی به معناشناخت زبان OCL خواهیم انداخت، فعلاً از آوردن مطالب زیاد و پرداختن به جزئیات ترجمه و نیز بررسی معناشناخت زبان OCL پرهیز می‌کنیم و به مثالی ساده بسنده می‌کنیم: فرض کنید **`salary - (salary * 5) / 100`** یک عبارت از زبان OCL باشد. با مراجعه به معناشناخت دلالتی زبان OCL داریم:

$$[\![salary - (salary * 5)/100]\!] =$$
$$\begin{cases} [\![salary]\!] - [\![(salary * 5)/100]\!] & ; [\![salary]\!] \in \mathbb{Z} \wedge [\![(salary * 5)/100]\!] \in \mathbb{Z} \\ \bot & ; [\![salary]\!] = \bot \vee [\![(salary * 5)/100]\!] = \bot \end{cases}$$

با قدری ساده سازی داریم:

$$[\![salary - (salary * 5)/100]\!] =$$
$$\begin{cases} \sigma_{ATT}(salary) - [\![(salary * 5)/100]\!] & ; \sigma_{ATT}(salary) \in \mathbb{Z} \wedge [\![(salary * 5)/100]\!] \in \mathbb{Z} \\ \bot & ; \sigma_{ATT}(salary) = \bot \vee [\![(salary * 5)/100]\!] = \bot \end{cases}$$

حال با ساده سازی بیشتر داریم:

$$\begin{cases} [\![salary - (salary * 5)/100]\!] = & \sigma_{ATT}(salary) - f \quad ; \sigma_{ATT}(salary) \in \mathbb{Z} \wedge f \in \mathbb{Z} \\ & \bot \qquad\qquad ; \sigma_{ATT}(salary) = \bot \vee f = \bot \end{cases}$$

$$f = \quad [\![(salary * 5)/100]\!] = \begin{cases} [\![salary * 5]\!]/[\![100]\!] & ; [\![salary * 5]\!] \in \mathbb{Z} \wedge [\![100]\!] \in \mathbb{Z} \\ \bot & ; [\![salary * 5]\!] = \bot \vee [\![100]\!] = \bot \end{cases}$$

با ساده سازی بیشتر زیرعبارت f داریم:

$$f = \begin{cases} g/100 & ; g \in \mathbb{Z} \wedge 100 \in \mathbb{Z} \\ \bot & ; g = \bot \vee 100 = \bot \end{cases}$$

$$g = [\![salary * 5]\!] = \begin{cases} [\![salary]\!] * [\![5]\!] & [\![salary]\!] \in \mathbb{Z} \wedge [\![5]\!] \in \mathbb{Z} \\ \bot & [\![salary]\!] = \bot \vee [\![5]\!] = \bot \end{cases}$$

$$= \begin{cases} \sigma_{ATT}(salary) * 5 & ; \sigma_{ATT}(salary) \in \mathbb{Z} \ \wedge 5 \in \mathbb{Z} \\ \bot & ; \sigma_{ATT}(salary) = \bot \vee 5 = \bot \end{cases}$$

نهایتاً با ساده سازی داریم:

$$[\![salary - (salary * 5)/100]\!] =$$
$$\begin{cases} \sigma_{ATT}(salary) - (\sigma_{ATT}(salary) * 5)/100 & ; \sigma_{ATT}(salary) \in \mathbb{Z} \\ \bot & ; \sigma_{ATT}(salary) = \bot \end{cases}$$

در صورتی که شئ مورد نظر که salary یکی از خصائص آن هست دارای نام this باشد، عبارت AMN زیر را می‌توانیم به عنوان یک ترجمه در نظر بگیریم در صورتی که تضمین کنیم مقدار خصیصه salary همواره عددی صحیح خواهد بود:

salary (this) – (salary (this) * 5) / 100

با مراجعه به معناشناخت زبان AMN و قاعده ترجمه که در ادامه مفصل‌تر بیان خواهیم کـــرد، در می‌یابیم عبارت AMN ارائه شده ترجمه عبارت OCL مورد نظر است. بدین ترتیب می‌توانیم عبارات OCL پیچیده‌تری را نیز بصورت سیستماتیک و بدون نگرانی از بروز خطا در روند ترجمه به عبارات معادل AMN ترجمه نمائیم؛ البته لازم به ذکر است که برخی از ساختارهای زبان OCL مانند **pre** و **post** را باید با توجه به مفهوم آن‌ها در دو زبان ترجمه کرد. برای مثال عبارت **pre** را می‌توان با ساختار **PRE** P **THEN** S **END** در AMN معادل دانست به شرطی که عبارت P به شیوه گفته شده ترجمه شود و نیز عبارت **post** را می‌توان با عملگر انتساب معادل دانست.

در پایان لازم به ذکر است که زبان‌های OCL و AMN تفاوت‌های زیادی با هم دارند چرا که OCL بر مبنای توصیف محض می‌باشد حال آنکه زبان AMN تک تک حالات و نحوه تغییر حالات را توصیف می‌کند؛ به عبارت بهتر زبان AMN خیلی سطح پایین‌تر از زبان OCL بوده و برای اینکه ترجمه (علی الخصوص عباراتی مانند **post**) صحیح و قابل اطمینان باشد نیاز است که عبارات OCL را به اندازه کافی سطح پایین بنویسیم. در مثال دیاگرام فعالیت مشاهده کردیم که برای ترجمه صحیح و بدون ابهام نیازمند این هستیم که عمل‌ها را تا حد امکان شکسته و بصورت اعمال ابتدائی مانند ساختارهای شرطی و توابعی مثل جمع، ضرب، و تقسیم بیان کنیم؛ در این صورت است که خواهیم توانست اثرات اعمال و عبارات پیچیده‌تر را بصورت انتساب‌هایی در زبان AMN بیان کنیم.

۳-۲-۳- دیاگرام‌های مدلسازی ساختاری

تا کنون مطالبی را در مورد شیوه ترجمه دیاگرام‌های مدلسازی رفتار به زبان AMN ارائه دادیم با وجود این از روش‌های ذکر شده برای ترجمه مدل‌های رفتاری پروژه سیستم‌عامل خود استفاده نکرده‌ایم چرا که در این پروژه هدف تولید سیستم عاملی که درستی ترجمه مدل‌های آن کاملاً اثبات شده باشد، نیست و توسعه یک سیستم نرم‌افزاری که تا این حد بصورت صوری در موردش اثبات شود زمانی بیش از یک سال و تیمی بیش از یک نفر را می‌طلبد. لازم به ذکر است که مطالب این بخش که در سه زیربخش بعدی ارائه می‌گـردد، بیشتر بـــر اسـاس سند رسمی مشخصات زبان OCL در [OCL203] می‌باشد.

۳-۲-۳-۱- *ترجمه مدل‌های ساختاری شئ‌گرا به زبان AMN*

اولین گام در تبدیل مدل‌های UML به یک زبان مدلسازی فرمال خارج کردن آن‌ها از شکل تصویری بصورت متنی است؛ در بسیاری از تحقیقات مدل‌های ساختاری را با استفاده از یک زبان که شبیه XML است و توسط OMG توسعه داده شده است نشان می‌دهند [LIJU05]. این کار برای شناسایی اجزا و بیان دقیق مدل مناسب می‌باشد، ولی ما قصد نداریم از زبان XMI برای نشان دادن مدل خود استفاده کنیم. در این پروژه ما از زبانی دیگر برای نشان دادن مدل‌های ساختاری خود استفاده خواهیم کرد که برای ترجمه به زبان AMN مناسب‌تر می‌باشد.

اولین گام در روش ترجمه پیشنهادی ما خارج کردن مدل‌های ساختاری UML از شکل تصویری و نشان دادن آن با استفاده از زبان ارائه شده در این سند است. این زبان از ساختارهای نحوی رایج در ریاضیات برای نشان دادن ساختارهای ریاضی استفاده می‌کند (ارائه دقیق قواعد نحوی، معنایی، و کاربردی این زبان خارج از اهداف این پروژه است)، بنابراین مفهوم عبارات این زبان بدون ابهام و تعریف شده می‌باشند؛ و بدون اینکه به دنبال تعریف نحو و معناشناخت این زبان باشیم به استفاده از آن می‌پردازیم. عمل ترجمه مدل‌های ساختاری UML از شکل تصویری به زبان مدلسازی فرمال این پروژه، توسط نگاشتی فرضی انجام می‌شود که تعریف بدنه این نگاشت منجر می‌شود به پیاده‌سازی یک کامپایلر که فراتر از اهداف این پروژه می‌باشد، و در این پروژه فرض می‌کنیم کاربر با نگاه کردن به مدل و تشخیص اجزای آن اقدام به ترجمه می‌کند. از آنجایی که این نگاشت مدل UML را از

شکل تصویری خارج می‌کند و آن را بـه دنباله‌ایی از نمادهای ریاضی تبدیل می‌کند، نام آن را flat نهاده‌ایم و به این صورت تعریف می‌کنیم:

$$flat: Valid\ UML\ Structural\ Models \rightarrow Our\ Formal\ Modeling\ Notation\ Models$$

همانطور که مشاهده می‌شود این نگاشت هر مدل ساختاری UML را که از نظر نحوی صحیح باشد را دریافت و آن را به یک توصیف در زبان مدلسازی فرمال پیشنهادی ما تبدیل می‌کند. حال وقت آن فرا رسیده است که در مورد زبان مدلسازی فرمال خود بیشتر صحبت کنیم. در این زبان هر مدل بصورت یک سه‌تایی مرتب بیان می‌شود:

$$Model = (D(M), M, \sigma(M))$$

مؤلفه اول این سه‌تایی *تعاریف مدل*، مؤلفه دوم آن *توصیف مدل*، و نهایتاً مؤلفه سوم آن *حالت مدل* نام دارد. منظور از تعاریف مدل، مجموعه تمام نوع‌ها و اعمال روی این نوع‌ها می‌باشد که در مدل تعریف شده‌اند، و منظور از حالت مدل در اصل حالت سیستمی است که مدل M آن را تـوصیف می‌کند، و در هر لحظه اشیاء ایجاد شده و مقادیر خصائص اشیاء (حالت سیستم) را مشخص می‌کند. در مورد توصیف مدل در بخش‌های بعدی بحث خواهیم کرد در اینجا ذکر این نکته کافی است که این مؤلفه در حقیقت اجزای مختلف مدل را نشان می‌دهد.

۳-۲-۳-۲- *مدل‌های شئ‌گرا در زبان مدلسازی فرمال پیشنهادی*

در این بخش به بررسی بخش کوچکی از زبان مدلسازی شئ‌گرا و شیوه بیان مدل‌هـای ساختاری شئ‌گرا در زبان مدل‌سازی فرمال پیشنهادی خود که در این پروژه از آن استفاده کرده‌ایم، می‌پردازیم. در همین ابتدا لازم است روشی را برای نام گذاری اجزاء مدلهای خود ارائه دهیم؛ با فرض اینکه مجموعه T، مجموعه نمادهای الفبا باشد، مجموعه غیر تهی و متناهی N که $N \subseteq T^+$ مجموعه نام‌ها می‌باشد. بر این اساس می‌توان مهفوم نوع را تعریف کرد؛ قصد نداریم زیاد در مورد نوع و مفاهیم آن دقیق شویم، فعلاً فرض می‌کنیم ساختاری بصورت $D(M) = (\tau, \Omega)$ وجود دارد که در آن τ مجموعه تمام اسامی نوع‌ها می‌باشد، و نیز Ω مجموعه تمام عملگرهایی است که بر روی نوع‌های موجود در τ قابل اعمال هستند. مجموعه τ شامل نوع‌های پایه Integer، Real، Boolean، و String می‌باشد؛ این نوع‌ها در اصل نوع‌های از قبل تعریف شده زبان OCL می‌باشند. نکته مهم در اینجا، این است که

دامنه مقادیر تمامی انواع شامل یک مقدار تعریف نشده، تهی، و یا ناشناخته می‌باشد که با نام null شناخته می‌شوند. عملگرهای موجود در Ω نیز برای مثال شامل عملگرهای معمول برای حساب اعداد صحیح نظیر +، –، *، و یا / می‌باشد. لازم است بدانیم که نوع‌های مجموعه‌ایی مثل Set (String) و Sequence (Integer) نیز عضو مجموعه τ در نظر گرفته می‌شوند.

اصلی‌ترین مفهوم در زبان UML برای مدلسازی موجودیت‌های یک مسأله، مفهوم کلاس است. یک کلاس در اصل توصیفی در مورد گروهی از اشیاء می‌باشد که دارای خصوصیات مشترکی می‌باشند.

تعریف ۱ (کلاس‌ها): کلاس‌های یک مدل در اصل زیرمجموعه‌ایی متناهی از نام‌ها می‌باشد. به عبارتی دیگر $CLASS \subset N$.

متناظر با هر کلاس $c \in CLASS$، یک *نوع شئ* (Object Type) در مجموعه τ در نظر گرفته می‌شود که هم‌نام با آن کلاس می‌باشد. یک مقدار از یک نوع شئ در حقیقت یک شئ از نوع کلاس متناظر با آن نوع شئ می‌باشد. تفاوت اصلی بین نوع‌های شئ و کلاس‌ها در این است که برای نوع‌ها همواره یک مقدار مخصوص null در دامنه آن در نظر می‌گیریم. در این پـروژه از روابـط ساده‌ایی بین کلاس‌ها استفاده کرده‌ایم، برای همین نیازی نداریم به بررسی روش ترجمه تمامی بخش‌های زبان UML بپردازیم و درگیر سختی‌های معنای روابط Aggregation و Composition شویم.

خصیصه‌ها (Attributes) بخشی از اعلان کلاس‌ها را در مدل‌های UML تشکیل می‌دهند. هر شئ با مقادیر خصیصه‌ها همراه می‌باشد که خصوصیات و ویژگی‌های (Properties) آن شـــئ را نشان می‌دهند. هر خصیصه دارای یک نام و یک نوع می‌باشد که دامنه مقادیر آن خصیصه را نشان می‌دهد.

تعریف ۲ (خصیصه‌ها): فرض کنید $t \in \tau$ یک نوع باشد؛ خصیصه‌های کلاس $c \in CLASS$ با ATT_c به عنوان مجموعه‌ایی از نگاشت‌های $a : t_c \to t$ تعریف می‌شود که در آن a نام خصیصه و عضوی از مجموعه N، و $t_c \in \tau$ نوع اشیاء کلاس c می‌باشد.

در کنار این تعریف باید به این نکته هم اشاره کنیم که نام خصیصه‌ها باید منحصر بفرد باشند، به عبارتی دیگر نام یک خصیصه با یک نوع نمی‌تواند برای تعریف یک خصیصه دیگر با نوعی متفاوت بکار رود. البته خصیصه‌هایی هم‌نام (و یا هم‌نوع) می‌توانند در کلاس‌های متعدد که با رابطه Generalization به هم مرتبط شده‌اند، یافت شود.

در اعلان کلاس‌ها علاوه بر تعریف خصیصه‌ها عملگرهایی هم برای کلاس تعریف می‌شود. برخی از این عملگرها بدون آثار جانبی و برخی دیگر با آثار جانبی می‌باشد؛ اما در حالت کلی نام عملگرهایی که برای تمامی نوع‌های مدل (چه از پیش تعریف شده، و چه تعریف شده توسط کاربر) موجود است در اصل اعضایی از مجموعه Ω می‌باشند که اشیائی را که روی آن‌ها عمل می‌کنند بصورت پارامتر به آن‌ها پاس می‌شوند. در زیر بخش بعدی هنگامی که معنای جملات زبان OCL را بصورت دقیق بیان کردیم، خواهیم توانست رفتار این عملگرها را بصورت خیلی دقیق‌تر بررسی و ترجمه کنیم.

در مدل‌های UML جزئی با نام Association و نیز Role Name وجود دارد؛ بطوری که اشیائی از کلاس‌های مختلف در یک یا چند رابطه Association با نقش‌های مختلف که توسط Role Name مشخص می‌شود، شرکت می‌کنند. با مراجعه به [FOWL04] در می‌یابیم که روابط Association و حتی گونه‌های خاص‌تر آن مثل Composition و Aggregation در اصل روشـــی برای بیان خصیصه‌های یک کلاس از اشیاء می‌باشد؛ بدین ترتیب می‌توانیم با افزودن محدودیت‌های بیشتر در گزاره‌های همواره ثابت کلاس‌ها این روابط را نیز به وسیله تعریف خصیصه در اعلان کلاس بیان کنیم.

در صورتی که بخواهیم مطالب گفته شده تا کنون را جمع‌بندی کنیم، می‌توانیم توصیف یک مدل شئ‌گرا را در حالت کلی بصورت زیر تعریف کنیم.

تعریف ۳ (توصیف مدل شئ‌گرا): یک مدل شئ گرا با نام M با استفاده از یک ساختار بصورت زیر توصیف می‌شود:

$$M = (CLASS, ATT, OP, \prec)$$

که در آن:

۱- $CLASS$ مجموعه‌ایی از اسامی کلاس‌های تعریف شده در مدل می‌باشد.

۲- ATT عبارت است از مجموعه‌ایی دربرگیرنده تمامی مجموعه‌های صورت توابعی که این توابع هر شئ موجود در سیستم را به مقدار یک خصیصه نگاشت می‌کنند؛ به عبارتی بهتر:

$$.ATT = \bigcup_{c \in CLASS} ATT_c$$

۳- OP عبارت است از مجموعه‌ایی دربرگیرنده تمامی مجموعه‌های صورت توابع عملگرهای شئ؛ به عبارتی بهتر:

$$OP = \bigcup_{c \in CLASS} OP_c$$

(در زیر بخش بعدی رفتار این عملگرها را بصورت دقیق‌تر بیان خواهیم کرد).

۴- $\prec$ یک ترتیب جزئی اکید بر روی مجموعه $CLASS$ می‌باشد که سلسله مراتب Generalization را نشان می‌دهد.

در همین مقطع لازم است مطالبی را هم در مورد ارث‌بری ذکر کنیم. با توجه به مفهومـــی که از ارث‌بری در منابع مربوط به مدلسازی شئ‌گرا و UML ذکر می‌شود ([FOWL04] و [WHIT07])، این است که یک کلاس فرزند تمامی عملگرها و خصائص کلاس پایه را به ارث می‌برد. در صورتی که بخواهیم غیر صوری و غیر دقیق صحبت کنیم، ارث بردن در اصل به این معنا است که عملگرها و خصائص کلاس پایه یک‌بار دیگر در کلاس‌های فرزند اعلان می‌گردند. در صورتی که بخواهیم این مفهوم را بطور دقیق و در زبان مدلسازی فرمال خود بیان کنیم در اصل باید یک مفهوم دیگر را در کنار تعاریف ATT_c و OP_c داشته باشیم. این مفهوم در UML *توصیف کامل*[۷] کلاس نام دارد؛ در حقیقت دو مجموعه مجزا از خصائص و عملگرها برای هر کلاس تعریف می‌شود، یکی آن‌هایی که در خود کلاس بطور صریح اعلان شده‌اند و دیگری آن‌هایی که کلاس مورد نظر از کلاس‌های والد خود به ارث برده است. ما این مفهوم را در زبان مدلسازی فرمال خود به این صورت تعریف می‌کنیم:

$$ATT_c{}^* = ATT_c \cup \bigcup_{ć \in CLASS \wedge c \preccurlyeq ć} ATT_ć$$

$$OP_c{}^* = OP_c \cup \bigcup_{ć \in CLASS \wedge c \preccurlyeq ć} OP_ć$$

البته همانطور که می‌دانیم و ذکر شد، مدل‌های ساختاری UML اجزائی بیشتر از اسامی کلاس‌ها، خصائص، عملگرها، و روابط سلسله مراتبی دارند، ولی خیلی از این اجزاء در مدل بکار رفته در پروژه ما وجود ندارد، بنابراین از توضیح دقیق‌تر آن‌ها پرهیز می‌کنیم. در انتهای این زیر بخش حالت مدل‌های UML را تعریف خواهیم کرد.

از آنجایی که قصد داریم مدل‌های UML را در نهایت به مدل‌های B تبدیل کنیم، لازم است تمام اشیائی که تا یک لحظه معین ایجاد شده‌اند، تمام مقادیر خصائص، و ... را دنبال کنیم؛ به عبارتی دیگر

[7] Full Description.

حالت سیستم را بیان کنیم. در ادامه خواهیم دید که چگونه می‌توان حالت سیستمی را که مدل شئ‌گرا آن را مدل می‌کند را چگونه می‌توان تعریف کرد. در زبان مدلسازی فرمال پیشنهادی، ما این مفهوم را با $\sigma(M)$ نشان خواهیم داد؛ البته قبل از اینکه بتوانیم حالت سیستم را تعریف کنیم نیاز داریم که به تعریف چند مفهوم پایه‌ایی‌تر بپردازیم که در ادامه آن‌ها را مشاهده خواهیم کرد.

دامنه یک کلاس مجموعه تمامی اشیائی می‌باشد که می‌توان از آن کلاس و فرزندان آن کلاس نمونه سازی کرد. در این پروژه از آنجایی که از رابطه Generalization در بین کلاس‌ها استفاده نشده است، تعریف دامنه کلاس قدری ساده خواهد بود. البته باید قبل از تعریف دامنه کلاس شناسه اشیاء را برای یک کلاس تعریف کنیم.

تعریف ٤ (شناسه اشیاء): برای یک کلاس، $c \in CLASS$، مجموعه شناسه اشیاء برای آن کلاس با مجموعه‌ایی نامتناهی از اسامی بصورت $oid\ (c) = \{\hat{c_1}, \hat{c_2}, \dots\}$ تعریف می‌شود.

تعریف ٥ (دامنه): دامنه یک کلاس برابر است با مجموعه شناسه اشیاء برای آن کلاس، و آن را برای یک کلاس c بصورت $D_{CLASS}\ (c)$ نشان می‌دهیم که بطور خاص اگر رابطه پدر–فرزندی در مدل موجود نباشد[٨] $D_{CLASS}\ (c) = oid\ (c) \cup \{\perp\}$.

تعریف ٦ (حالت کلاس): حالت هر کلاس $c \in CLASS$ را با یک دوتایی بصورت زیر نشان می‌دهیم:

$$\sigma(c) = (\sigma_{CLASS}(c), \sigma_{ATT}(c))$$

که در آن مؤلفه اول در یک مقطع از زمان عبارت است از تمامی اشیائی که تا آن لحظه از کلاس مورد نظر (در اینجا کلاس c) نمونه‌سازی شده است؛ به عبارت بهتر، همواره داریم: $\sigma_{CLASS}(c) \subset \mathrm{oid(c)}$. مؤلفه دوم این ساختار نیز مجموعه‌ایی از صورت‌های توابع به فرم $a: t_c \to t$ می‌باشد که در آن t و t_c اعضایی از τ (مجموعه اسامی نوع‌های موجود در مدل) هستند که t_c بطور خاص نوع اشیاء کلاس c می‌باشد. تابع a در اصل مقدار خصیصه (با نام) a تمام اشیاء ایجاد شده از کلاس c را تعیین می‌کند.

[٨] البته در حالت کلی D_{CLASS} را برای یک کلاس c بصورت $D_{CLASS}(c) = (\bigcup_{\acute{c} \in CLASS \wedge \acute{c} \preccurlyeq c} oid(\acute{c})) \cup \{\perp\}$ تعریف می‌کنیم؛ به عبارتی دیگر هر شئ در اصل یک نمونه از کلاس بالاتر خود می‌باشد.

در انتهای این زیر بخش، بعد از اینکه تعاریف لازم را برای تعریف حالت سیستم آوردیم، می‌توانیم حالت یک سیستم را که مدل توصیف شده آن را مدل می‌کند را بصورت زیر تعریف کنیم. این تعریف برای ارزیابی عبارت‌های OCL و نیز برای تفسیر مدل‌های شئ‌گرا لازم است.

تعریف ۷ (حالت مدل): حالت مدل برای یک مدل شئ‌گرا M را با $\sigma(M)$ نشان می‌دهند که بصورت ساختاری به شکل زیر تعریف می‌شود:

$$\sigma(M) = (\sigma_{CLASS}, \sigma_{ATT})$$

در این ساختار:

۱- σ_{CLASS} در یک مقطع از زمان متشکل از تمامی اشیائی می‌باشد که تا آن لحظه در سیستم ایجاد شده‌اند؛ به عبارتی بهتر:

$$\sigma_{CLASS} = \bigcup_{c \in CLASS} \sigma_{CLASS}(c) .$$

۲- σ_{ATT} مجموعه‌ایی متناهی از تمامی صورت‌های توابعی می‌باشد که هر شئ موجود در سیستم (تا کنون ایجاد شده‌اند؛از هر نوعی که باشند) را به مقادیر خصیصه‌های آن‌ها نگاشت می‌کنند؛ به عبارتی بهتر:

$$\sigma_{ATT} = \bigcup_{c \in CLASS} \sigma_{ATT}(c) .$$

در انتها لازم به ذکر است که این تعاریف برای ترجمه ساختار استاتیک در AMN کافی است. در ادامه معناشناخت زبان OCL را بررسی خواهیم کرد. البته بخشی از تعاریف در این بخش مانند OP_c ناقص رها شدند که در زیربخش‌های بعدی آن‌ها را تکمیل خواهیم کرد.

۳-۲-۳-۳- *معناشناخت عبارات زبان OCL*

در این بخش قصد داریم معناشناخت آن دسته از عبارات زبان OCL را که در پروژه استفاده شده است را بررسی کنیم. زبان OCL یک زبان دارای نوع و Strongly Typed و تابعی می‌باشد، یعنی به هر عبارت نوشته شده در این زبان یک نوع اختصاص داده می‌شود. در این بخش نمی‌خواهیم که سیستم نوع این زبان را توضیح دهیم و صرفاً به معناشناخت انواع پایه و برخی از عبارات OCL بسنده خواهیم کرد، و علاقه مندان در صورت نیاز می‌توانند به منبع اصلی OCL در [OCL203] رجوع کنند.

همانطور که در زیربخش قبل مشاهده کردیم که می‌توان نوع‌های تغریف شده (و موجود؛ یعنی از قبل تعریف شده) در مدل‌های UML و نیز اعمال مجاز بر روی داده‌هایی از آن نوع‌ها را می‌توان با

ساختار نحوی $D(M) = (\tau, \Omega)$ نشان داد که در آن τ مجموعه تمامی اسامی نوع‌ها از جمله نوع‌های پایه می‌باشد. در زیر معناشناخت و نحو نوع‌های OCL را مشاهده می‌کنیم.

تعریف ۸ (نحو نوع‌های پایه OCL): نوع‌های پایه OCL به عنوان زیر مجموعه‌ایی از نوع‌های تعریف شده در مدل UML در نظر گرفته می‌شوند، و با مجموعه $\tau_B = \{\text{Integer, Real, Boolean, String}\}$ نشان می‌دهند.

تعریف ۹ (معناشناخت نوع‌های پایه OCL): با فرض اینکه T^* مجموعه تمامی رشته‌های با طول متناهی بر روی الفبای T باشد، در این صورت معناشناخت نوع‌های پایه (یا همان دامنه مقادیر آن‌ها) بصورت زیر تعریف می‌گردد:

$D_{CLASS}(\text{Integer}) = \mathbb{Z} \cup \{\bot\}$
$D_{CLASS}(\text{Real}) = \mathbb{R} \cup \{\bot\}$
$D_{CLASS}(\text{Boolean}) = \{true, false\} \cup \{\bot\}$
$D_{CLASS}(\text{String}) = T^* \cup \{\bot\}$

همانطور که مشاهده می‌کنید معناشناخت نوع‌های پایه در حقیقت دامنه مقادیر عبارات از این نوع‌ها را نشان می‌دهد. دقت کنید که هر یک از این دامنه‌ها شامل یک مقدار تعریف نشده است که با نماد $\bot$ نشان می‌دهیم. این مقدار برای مدلسازی این موضوع که اطلاعاتی از این نوع در دسترس نیست و یا حاصل عبارت از این نوع تعریف نشده است، بکار می‌رود.

بعد از اینکه نحو و معناشناخت نوع‌های پایه و نوع‌های تعریف شده توسط کاربر را در این زیربخش و زیربخش قبل مشاهده کردیم، لازم است نحو و معناشناخت عملگرهای نوع‌های پایه و نیز عملگرهای نوع‌های تعریف شده بر روی نوع‌های تعریف شده توسط کاربر را بررسی کنیم.

تعریف ۱۰ (نحو عملگر): یک عملگر، صورت تابعی بصورت $\omega : t_1 \times t_2 \times \ldots \times t_n \to t$ که در آن t_1، t_2، ...، t_n، و t اسامی نوع‌های پایه و یا تعریف شده توسط کاربر (عضوی از مجموعه τ)، و ω نیز نام عملگر می‌باشد (عضوی از مجموعه Ω).

در زیربخش قبل مشاهده کردیم که عملگرهای تعریف شده برای یک کلاس $c \in CLASS$ را توسط مجموعه‌ایی با نام OP_c نشان می‌دهند، در این زیربخش قصد داریم این مجموعه را تعریف و سپس معناشناخت آن را بررسی کنیم.

تعریف ۱۱ (مجموعه عملگرهای یک کلاس): مجموعه تمامی عملگرهایی که توسط کاربر برای یک کلاس $c \in CLASS$ تعریف شده است را مجموعه‌ایی با نام OP_c نشان می‌دهند، که این مجموعه در اصل مجموعه تمامی صورت‌های تابعی است که یک حداقل عملوند از نوع خود کلاس دریافت و یک مقدار را بر می‌گردانند، بصورت صوری‌تر:

$$OP_c = \{\omega : t_c \times t_1 \times t_2 \times \dots \times t_n \to t \mid t_c, t_1, \dots, t_n \in \tau \wedge \omega\,(t_1, t_2, \dots, t_n) : t \text{ در کلاس } c \text{ تعریف باشد}\}$$

که در آن t_c اسم نوع متناظر با کلاس c می‌باشد.

تعریف ۱۲ (معناشناخت عملگرها): در حالت کلی معنای هر عملگر بصورت

$$\omega : t_1 \times t_2 \times \dots \times t_n \to t$$

را با $[\![\omega]\!]$ نشان می‌دهیم، و بصورت

$$[\![\omega : t_1 \times t_2 \times \dots \times t_n \to t]\!] = D_{CLASS}(t_1) \times D_{CLASS}(t_2) \times \dots \times D_{CLASS}(t_n) \to D_{CLASS}(t)$$

تعریف می‌شود[۹].

با این تعاریف می‌توان به راحتی معنای تمامی عملگرها را بیان کنیم. برای مثال عملگر + که روی نـــوع‌های پایه به جز نوع Boolean عمل می‌کند را می‌توان بصورت زیر تعریف کرد:

$$[\![+]\!](i_1, i_2) = \begin{cases} i_1 + i_2 & ; \quad i_1 \neq \perp \wedge\ i_2 \neq \perp \\ \perp & ; \quad \text{در غیر اینصورت} \end{cases}$$

لیست کاملی از معانی عملگرهایی که بر روی نوع‌های پایه عمل می‌کنند، را می‌توان در [OCL203] یافت. در این منبع همچنین معنای عملگرهایی که برروی تمامی نوع‌ها عمل می‌کنند را می‌توان مشاهده کرد، برای مثال معنای عملگر = که در این پروژه بسیار استفاده شده، بصورت زیر تعریف می‌شود:

[۹] در اینجا ذکر نکته‌ایی لازم است، و آن هم این است که می‌توان مفهوم یک اسم نوع را معادل با دامنه آن در نظر گرفت؛ بطور دقیق‌تر:
$[\![t]\!] = D_{CLASS}(t)$

$$[\![=]\!](i_1, i_2) = \begin{cases} true & ; \quad i_1 = i_2 \wedge i_1 \neq \perp \wedge i_2 \neq \perp \\ \perp & ; \quad i_1 = \perp \vee i_2 = \perp \\ false & ; \quad \text{در غیر اینصورت} \end{cases}$$

بعد از تعریف معنی عملگرهای از پیش تعریف شده زبان OCL که برروی نوع‌های پایه و نوع‌های تعریف شده توسط کاربر عمل می‌کردند نوبت آن رسیده است که معنی آن دسته از عملگرهایی را بیان کنیم که بر روی نوع‌های تعریف شده توسط کاربر عمل می‌کنند و به حالت سیستم نیز وابسته‌اند؛ در حالت کلی این عملگرها به سه دسته زیر تقسیم می‌شوند:

۱- عملگرهای از پیش تعریف شده؛

۲- عملگرهای خصیصه‌ایی؛

۳- عملگرهای شئ.

عملگرهای از پیش تعریف شده، عملگرهایی است که جزو عملگرهای زبان OCL هستند و صرفاً بر روی نوع‌های تعریف شده توسط کاربر عمل می‌کنند. در [OCL203] می‌توان بطور کامل‌تر نحو و معنای این عملگرها را مشاهده کنیم؛ در اینجا صرفاً به توضیح عملگری که بیش از همه عملگرهای از پیش تعریف شده در این پروژه استفاده شده است بسنده می‌کنیم. عملگر allInstances وقتی که بر روی نام کلاسی اعمال می‌شود، مجموعه تمامی اشیائی که تا به این لحظه از این کلاس نمونه‌سازی شده‌اند را بر می‌گرداند؛ همانطور که مشاهده می‌شود، این عملگر وابسته به حالت سیستم می‌باشد. بصورت صوری‌تر و بر اساس آن چیزی که در زیربخش قبل در مورد حالت سیستم تعریف کرده‌ایم، معنای عملگر allInstances را می‌توان بصورت زیر تعریف کرد:

$$[\![\text{allInstances}]\!](c) = \sigma_{CLASS}(c)$$

عملگرهای خصیصه‌ایی، عملگرهایی از کلاس می‌باشند که برای دسترسی به مقادیر خصائص یک شئ در یک حالت از سیستم بکار می‌روند. اعمال این عملگرها در زبان OCL معادل این است که به یک خصیصه از یک شئ با استفاده از عملگر نقطه (.) دسترسی پیدا کنیم. در حالت کلی نحو این عملگر را بصورت $a : t_c \to t$، به عنوان عضوی از مجموعه Ω تعریف می‌کنند. همانطور که مشاهده می‌کنید این تعریف از عملگر خصیصه‌ایی، آن را از اعضای مجموعه ATT_C برای یک کلاس تعریف کرده‌ایم؛ در حقیقت هر خصیصه در عین حال که عضوی از مجموعه ATT_C است، به عنوان عملگری

که بر روی اشیاء کلاس c عمـل می‌کند نیز تعریف می‌شود؛ در نتیجه عضوی از مجموعه Ω نیز در نظر گرفته می‌شود. با این مقدمات معنای یک عملگر خصیصه‌ایی بصورت زیر تعریف می‌شود:

تعریف ۱۳ (نحو و معنای عملگر خصیصه‌ایی): در صورتی که a یک خصیصه از نوع t باشد که در کلاس c تعریف شود؛ نحو عملگر خصیصه‌ایی a بصورت $a : t_c \to t$ و نام این تابع عضوی از مجموعه Ω تعریف می‌شود. معنای این عملگر به صورت

$$[\![a: t_c \to t]\!](\hat{c}) = (a: D_{CLASS}(t_c) \to D_{CLASS}(t))(\hat{c})$$

تعریف می‌کنند؛ لازم است بدانیم که مقدار این عملگر برای اشیاء ناشناخته تعریف نشده و برای Propertyهای ناشناخته و یا تهی برابر است با $\perp$.

عملگرهای شئ هم عملگرهایی می‌باشند که توسط کاربر تعریف شده‌اند، و معنای آن‌ها هم به وسیله معنای عبارت‌های OCL که برای توصیف بدنه این عملگرها بکار می‌رود، تعریف می‌شود. این عملگرها را باید قبل از اجرا در مورد بر قرار بودن پیش‌شرط ذکر شده در بند pre عملگر مذکور اطمینان حاصل کنیم؛ برای این کار کافی است که معنای عبارت مقابل pre را در حالتی از سیستم که می‌خواهیم عملگر را فراخوانی کنیم ارزیابی نمائیم و مطمئن شویم که این عبارت صحیح است. در مورد بند post عملگرها نیز کافی است متغییرهایی که در انتهای کار عملگر می‌خواهیم مقدار خاصی داشته باشند را با انتساب‌هایی ترجمه کنیم.

توضیح کامل معناشناخت زبان OCL مطالب بیشتری را می‌طلبد که ذکر تمامی آن‌ها در اینجا ممکن نیست و برای بدست آوردن اطلاعات بیشتر می‌توان به سند رسمی زبان OCL در [OCL203] مراجعه نمود. البته نتایج سودمندی را نیز برای ترجمه OCL به AMN در [LEDB01] و [LEDC01] می‌توان یافت، که در این پروژه از این نتـایج استفاده‌های زیادی کرده‌ایم.

از مهم‌ترین مواردی که در این پروژه از مطالب دو مقاله [LEDB01] و [LEDC01] استفاده شده است، مربوط می‌شود به الگوهای تبدیل عملگرهای forAll، exists، including، و چند عملگر دیگر؛ این الگوهای تبدیل با توجه به معنایی که در [OCL203] بصورت زبانی تعریف شده است مطابقت داشته، و استفاده از این الگوها کمک زیادی در سرعت بخشیدن به عمل ترجمه مدل‌های UML پروژه داشته است. جدول که زیر بر گرفته از جداول ٦، ٨، و ٩ مقاله [LEDB01] است، حاوی الگوهای مفیدی برای ترجمه عبارات از OCL به AMN می‌باشد:

جدول ۱- عملگرهای **OCL** و عبارات معادل آن‌ها در **AMN** (برگرفته از **[LEDB01]**).

عملگر در OCL	مفهوم در B
ss : Set (T)	$ss \subseteq T$
ee : Sequence (T)	$ee \in seq(T)$
ss1->union (ss2)	$ss1 \cup ss2$
ee1->union (ee2)	$ee1 ^\frown ee2$
ss1->intersection (ss2)	$ss1 \cap ss2$
ss->size()	$card(ss)$
ee->size()	$size(ee)$
ss->including (tt)	$ss \cup \{tt\}$
ee->asSet()	$ran(ee)$
ee->at(i)	$ee(i)$
ss->forAll (tt|boolexp$_{tt}$)	$\forall(tt).(tt \in ss \Rightarrow boolexp_{tt})$
ee->forAll (tt|boolexp$_{tt}$)	$\forall(tt).(tt \in ran(ee) \Rightarrow boolexp_{tt})$
ss->exists (tt|boolexp$_{tt}$)	$\exists(tt).(tt \in ss \wedge boolexp_{tt})$
ee->exists (tt|boolexp$_{tt}$)	$\exists(tt).(tt \in ran(ee) \wedge boolexp_{tt})$

در فصول بعدی بر اساس روش‌های ارائه شده در این فصل، اقدام به مدلسازی سیستم در UML و ترجمه آن به AMN و نهایتاً اثبات درستی سیستم و پالایش آن به منظور تولید کد خواهیم کرد.

۳-۳- بررسی صحت ترجمه عبارات OCL

همانطور که مشاهده نمودید رفتاری که از عملگرهای کلاس‌ها، و یا اعمال دیاگرام‌های فعالیت انتظار داریم را می‌توان بر حسب عبارات زبان OCL بیان کنیم. در این فصل روشی را برای ترجمه عبارات OCL به زبان AMN ارائه دادیم، اما برای کامل‌تر شدن بحث لازم است مطالبی را نیز در مورد صحت سنجی ترجمه ارائه دهیم. فرض کنید به کمک یک دیاگرام فعالیت عبارتی به زبان AMN به صورت **BEGIN** S **END** نوشته‌ایم و ادعا می‌کنیم که این عبارت اثری معادل با اثر مطلوبی که عبارت e به زبان OCL توصیف می‌کند، دارد. کافی است گزاره زیر را برای درستی ترجمه اثبات کنیم:

$$[\![e_{pre}]\!]\,(P) \Rightarrow [\textbf{BEGIN}\ S\ \textbf{END}\ (\mathrm{P})]\,([\![e_{post}]\!])$$

در این عبارت P در اصل حالت فعلی سیستم را نشان می‌دهد، و به معنی ارزیابی $[\![e_{pre}]\!]$ در حالت فعلی سیستم می‌باشد. همچنین $([\![e_{post}]\!])$ **[BEGIN** S **END** (P)] به این معنا است که ارزیابی عبارت **BEGIN** S **END** در حالت فعلی سیستم منجر به حفظ درستی عبارت $[\![e_{post}]\!]$ خواهد شد. در حالت کلی این عبارت به این معنا است: *در صورت بر قرار بودن پیش‌شرط، پس از اجرای عبارت ترجمه شده باید پس‌شرط ذکر شده به* true *ارزیابی شود.*

۳-۴- نتیجه‌گیری

مطالب این فصل هسته اصلی کار ما را در این پروژه تشکیل می‌دهد. در این فصل مطالبی را در مورد شیوه ترجمه مدل‌های رفتاری و مدل‌های ساختاری را دیدیم. برای ترجمه مدل‌های رفتاری، دیاگرام‌های فعالیت را انتخاب کردیم، و بر اساس آنچه که از منابع مربوط به UML از مفهوم دیاگرام‌های فعالیت دریافت کردیم این روش را پایه‌گذاری نمودیم. بعد از ارائه روش برای ترجمه مدل‌های رفتاری، مطالبی را نیز در مورد ترجمه مدل‌های ساختاری بیان کردیم. همانطور که مشاهده نمودیم در این بخش روشی نظام‌مند را برای ترجمه ارائه دادیم، که با استفاده از آن‌ها به راحتی می‌توان اقدام به ترجمه مدل‌های ساختاری نمائیم. در انتهای این فصل نیز روشی را برای صحت‌سنجی و اثبات درستی ترجمه‌ها ارائه دادیم.

در پایان لازم است قدری در مورد نتیجه نهایی مطالب این بخش بحث کنیم. در این فصل، بر اساس دانشی که از معناشناخت UML داشتیم اقدام به ارائه روشی برای ترجمه مدل‌های UML به مدل‌های B (و یا هر زبان مدلسازی فرمال) کردیم. اما باید دقت کرد که این ترجمه بر اساس معناشناختی است که ما خودمان برای UML تعریف کرده‌ایم؛ در اصل همه کارهایی که در جهت ترجمه مدل‌های UML به مدل‌های فرمال صورت گرفته هم به همین ترتیب هستند. بسیاری از محققین عرصه مهندسی نرم‌افزار معتقدند که هر گونه تلاش برای ترجمه مدل‌های UML به مدل‌های فرمال کار بیهوده‌ایی است؛ این دسته از محققین برای این ادعای خود سه دلیل دارند: ۱- زبان UML اساس روشن و دقیقی ندارد؛ این زبان در اصل یک نمایش گرافیکی از افکار مهندسین می‌باشد [FOWL04]؛ ۲- زبان UML فاقد ساختارهایی برای اعمال تجرید و نیز فاقد قدرت استدلال می‌باشد؛ برای مثال هیچ راهی برای اینکه در مورد معادل بودن دو دیاگرام کلاس استدلال کنیم وجود ندارد؛ ۳- زبان UML همواره دچار تغییر می‌شود و ثابت نمی‌باشد؛ [BJOR06]. البته ما نیز با این طرز تفکر موافق هستیم.