مکانیزم تبدیل دوجهته نمودار کلاس UML و توصیف Object - Z

عباس رسول‌زادگان و احمد عبداله‌زاده بارفروش

چکیده: در این مقاله مکانیزمی برای تبدیل ساختاری دوطرفه نمودار کلاس UML و توصیفات Object - Z پیشنهاد می‌گردد. در مکانیزم پیشنهادی برای تبدیل المان‌های مدل‌سازی نمودار کلاس و توصیفات Object - Z به یکدیگر، قواعد ساخت‌یافته‌ای تعریف شده است. تبدیل نمودار کلاس به‌عنوان یکی از پرکاربردترین نمودارهای زبان بصری UML و توصیف زبان صوری Object - Z به یکدیگر، بستر مناسبی را برای بهره‌مندی توأمان از مزایای منحصر به فرد روش‌های مدل‌سازی صوری و بصری فراهم می‌نماید. به منظور امکان‌سنجی مکانیزم پیشنهادی، یک مطالعه موردی بر روی سیستم آسانسور چندکابینه ارائه می‌گردد. نتایج مطالعه مذکور حاکی از امکان‌پذیر بودن مکانیزم پیشنهادی است.

کلید واژه: **تبدیل مدل، نمودار UML، Object - Z، سیستم آسانسور چندکابینه.**

۱- مقدمه

مدل‌ها در فازهای مختلف توسعه نرم‌افزار از نیازمندی‌ها تا طراحی برای توصیف، تحلیل، اعتبارسنجی و صحت‌سنجی نیازمندی‌های مشتری و همچنین مدل‌سازی راه حل مسأله مورد استفاده قرار می‌گیرند. این همان ایده‌ای است که مهندسی نرم‌افزار مدل- رانده[1] (MDSE) بر مبنای آن مطرح شده است [۱]. مهندسی نرم‌افزار مدل- رانده بر استفاده از مدل‌ها به‌جای کد به‌عنوان فراورده‌های اولیه توسعه نرم‌افزار تأکید دارد. مهندسی نرم‌افزار مدل- رانده توسط دو گروه بزرگ از روش‌های مدل‌سازی پشتیبانی می‌شود. این دو گروه عبارتند از روش‌های مدل‌سازی صوری[2] (FMMs) و روش‌های مدل‌سازی بصری[3].

روش‌های مدل‌سازی صوری به صورت گسترده در قالب ابزارها و نمادگذاری‌هایی با معنای دقیق و قطعی (نظیر زبان توصیف Object - Z) تعریف می‌شوند [۲]. این روش‌ها به‌صورت ریاضی، سازگاری[4] و کمال[5] توصیف نیازمندی‌های ذی‌نفعان را اثبات می‌نمایند. چنین اثبات‌هایی به شناسایی به موقع و زودهنگام خطاهای طراحی منجر می‌گردد. بنابراین دیگر نیازی به انتظار تا زمان آزمایش کد نهایی نیست. هرچه خطاها دیرتر شناسایی شوند، هزینه بیشتری برای اصلاح آنها باید صرف شود. به‌علاوه، صحتی که از طریق اثبات تضمین می‌شود به مراتب جامع‌تر و قابل اعتمادتر از صحتی است که توسط آزمایش ارزیابی می‌شود. بنابراین روش‌های صوری در تحلیل، مدل‌سازی، توصیف و صحت‌سنجی نیازمندی‌های نرم‌افزارهایی که نیاز حیاتی به تضمین صحت دارند، از مزیت منحصر به فردی برخوردار هستند. به‌کارگیری روش‌های مذکور همچنین به توصیف‌گران، پیاده‌سازان و آزمایش‌کنندگان در برقراری تعاملات، صحت‌سنجی و اعتبارسنجی غیر مبهم و در برخی موارد تولید کد خودکار کمک می‌نماید [۳]. علی‌رغم مزایای فراوان و منحصر به فرد روش‌های مدل‌سازی صوری، کمبود توسعه‌دهندگان تعلیم‌دیده، کاربرد روش‌های مذکور را به توسعه نرم‌افزارهای حیاتی[6] و فوق صحیح[7] که تضمین کامل صحت در آنها ضروری است، محدود می‌نماید [۴] و [۵] تا [۷].

روش‌های مدل‌سازی بصری با استفاده از زبان‌های بصری (نظیر UML)، رویکردی عمل‌گرا به توسعه نرم‌افزار دارند. این روش‌ها بر مبنای نیاز به دوری از جزئیات کد و به‌منظور بازنمایی بصری[8] کلیات ساختار و رفتار سیستم پدید آمده‌اند. عمده‌ترین نقاط قوت روش‌های بصری عبارتند از: نمادگذاری‌های شناخته‌شده و شهودی که تعامل ذی‌نفعان پروژه را با یکدیگر تسهیل می‌نمایند، پشتیبانی متدولوژیکی با تأکید بر تجزیه مسأله[9] و فراهم‌سازی بستر مناسب برای به‌کارگیری تکنیک‌های شهودی و ابتکاری مهندسی نرم‌افزار که موجب افزایش کیفیت نرم‌افزار می‌شود. نبود یک مبنای ریاضی کامل و جامع، عمده‌ترین نقطه ضعف روش‌های مذکور است که منجر به تفسیرهای مختلف [۲]، توصیف ناسازگار، غیر دقیق و مبهم نیازمندی‌های نرم‌افزار و نهایتاً کاهش قابلیت اعتماد نرم‌افزار می‌شود. روش‌های بصری غالباً از نبود ابزارهایی برای تحلیل و صحت‌سنجی خودکار رنج می‌برند.

بررسی مزایا و کمبودهای روش‌های مدل‌سازی صوری و بصری از طریق مطالعه منابع و مراجع مرتبط [۸] و انجام مطالعه موردی سیستم آسانسور چندکابینه [۹] نشان می‌دهد که استفاده ترکیبی دو روش مذکور برای توصیف، اعتبارسنجی و صحت‌سنجی دقیق و کامل نیازمندی‌ها و همچنین طراحی انعطاف‌پذیر و قابل اعتماد راه حل مسأله، ضروری است. چنین ترکیبی دستیابی به نرم‌افزار باکیفیت را تضمین می‌نماید. اگرچه تاکنون تلاش‌های ارزشمندی در زمینه تجمیع روش‌های مذکور برای بهره‌مندی توأمان از مزایای منحصر به فردشان انجام پذیرفته است، اما همچنان راهی طولانی تا دست‌یابی به اهداف وعده داده شده باقی مانده است [۳]، [۴] و [۱۰].

در این مقاله مکانیزمی برای تبدیل ساختاری دوطرفه نمودار کلاس UML و توصیفات Object - Z پیشنهاد می‌گردد. در واقع، ارائه مکانیزم

این مقاله در تاریخ ۴ مرداد ماه ۱۳۹۰ دریافت و در تاریخ ۱۲ مهر ماه ۱۳۹۱ بازنگری شد.

عباس رسول‌زادگان، دانشکده مهندسی کامپیوتر، دانشگاه صنعتی امیرکبیر، تهران، (email: rasoolzadegan@aut.ac.ir).

احمد عبداله‌زاده بارفروش، دانشکده مهندسی کامپیوتر، دانشگاه صنعتی امیرکبیر، تهران، (email: ahmad@aut.ac.ir).

1. Model - Driven Software Engineering
2. Formal Modeling Methods
3. Visual Modeling Methods
4. Consistency
5. Completeness
6. Critical
7. High Integrity
8. Visualizing
9. Problem Decomposition

مذکور گامی است در جهت استفاده ترکیبی از روش‌های مدل‌سازی صوری و بصری به منظور بهره‌مندی از مزایای هر دو روش مدل‌سازی. شایان ذکر است که هدف از بازنمایی بصری توصیفات Object - Z در قالب نمودار کلاس UML بازبینی آنها از منظر الگوهای طراحی است. بنابراین فقط قسمت‌هایی از توصیفات صوری، بازنمایی بصری می‌شوند که برای بازبینی از منظر الگوهای طراحی مورد نیاز هستند. قابلیت استفاده مکانیزم پیشنهادی توسط یک مطالعه موردی بر روی سیستم آسانسور چندکابینه ارزیابی می‌شود. نتایج ارزیابی مذکور امکان‌پذیر بودن مکانیزم پیشنهادی را نشان می‌دهند.

ادامه این مقاله به‌صورت زیر سازماندهی شده است: بخش دوم مکانیزم تبدیل پیشنهادی را تشریح می‌کند. بخش سوم نحوه به‌کارگیری قواعد تبدیل پیشنهادی را در فرایند توسعه سیستم آسانسور چندکابینه به‌صورت عملی ارائه می‌نماید. در بخش چهارم کارهای مشابه آمده است و بخش پنجم به نتیجه‌گیری و گفتگو پیرامون کارهای آینده می‌پردازد.

۲- مکانیزم تبدیل پیشنهادی

این مقاله یک مکانیزم جدید برای ترکیب توصیفات صوری Object-Z و نمودار بصری کلاس UML ارائه می‌نماید. ارائه مکانیزم مذکور تلاشی است در جهت فراهم‌سازی بستر لازم برای تولید نرم‌افزار باکیفیت. مدل‌های صوری (توصیفات Object-Z) به همراه پالایش صوری، صحت و قابلیت اعتماد را تضمین می‌نمایند. مدل‌های بصری (UML) تعامل ذی‌نفعان مختلف (نظیر تحلیل‌گران و طراحان) را که لزوماً آشنایی کافی با مفاهیم پیچیده ریاضی روش‌های صوری ندارند، تسهیل می‌نماید. در این حالت، طراحان می‌توانند با دخالت مستقیم و مؤثر خود در فرایند توسعه نرم‌افزار با اعمال تکنیک‌های غیر صوری و مبتنی بر خلاقیت مهندسی نرم‌افزار نظیر الگوهای طراحی و چندریختی موجب افزایش انعطاف‌پذیری نرم‌افزار در حال توسعه گردند. بنابراین می‌توان در یک فرایند مبتنی بر تکرار و تکامل و با استفاده ترکیبی از مدل‌های صوری و بصری، شرایط لازم را برای تولید یک نرم‌افزار باکیفیت فراهم نمود.

تبدیل مدل در مهندسی نرم‌افزار مدل- رانده، یک مدل را که منطبق بر یک فرا- مدل[1] است به‌عنوان ورودی دریافت می‌کند و یک مدل دیگر را که آن هم منطبق بر یک فرا-مدل است به‌عنوان خروجی تولید می‌نماید. اگر فرا- مدل‌های مبدأ و مقصد یکسان باشند، تبدیل مذکور درونی[2] نامیده می‌شود. اگر فرا- مدل‌های مذکور متفاوت باشند تبدیل مدل، بیرونی[3] نامیده می‌شود. اگر در فرایند تبدیل سطح تجرید تغییر نکند، نکند، تبدیل مذکور یک تبدیل افقی[4] است، در غیر این صورت عمودی[5] است. مکانیزم پیشنهادی شامل دو تبدیل است: ۱) بازنمایی بصری که شامل تبدیل توصیفات Object - Z به نمودار کلاس متناظر است و ۲) بازنمایی صوری که شامل تبدیل نمودار کلاس UML به توصیفات متناظر در Object - Z است. با توجه به تعاریف پیش‌گفته، هر دو تبدیل مذکور از نوع افقی بیرونی هستند.

در ادامه به معرفی مکانیزم تبدیل پیشنهادی می‌پردازیم. این مکانیزم، قواعد دوجهته زیادی را برای تبدیل کامل توصیفات Object - Z به نمودار کلاس UML و بالعکس ارائه می‌نماید و کلیه ویژگی‌های مهم مشترک نمودار کلاس UML و توصیفات Object - Z را پوشش می‌دهد. برای تبدیل دوجهته کلیه ویژگی‌های مشترک نمودار کلاس Object - Z و UML قواعدی ارائه شده است. این ویژگی‌ها عبارتند از: ۱) کلاس و اجزای سازنده آن و مفاهیم مرتبط نظیر: صفات اولیه[6]، صفات ثانویه[7]، ثابت‌ها[8]، عملیات[9]، قابلیت رؤیت[10]، انواع تعریف‌شده توسط کاربر[11]، تعدد[12] تعدد[12] و مقداردهی اولیه[13]، ۲) انواع روابط بین کلاس‌ها شامل وراثت، وراثت عمومی[14]، رابطه انجمنی یک‌جهته[15]، رابطه انجمنی دوجهته[16]، رابطه تجمع[17]، رابطه ترکیب[18]، کلاس ارتباطی[19] و رابطه وابستگی و ۳) چندریختی[20]. با توجه به این که هر دو زبان مذکور دارای مفاهیم مشترک شیءگرایی هستند، ساخت یک تبدیل سیستماتیک بین آنها امکان‌پذیر می‌نماید. هیچ یک از کارهای مرتبط موجود که در بخش چهارم معرفی می‌شوند ویژگی‌های فوق‌الذکر را به طور کامل پوشش نداده‌اند.

۲-۱ قواعد دوجهته تبدیل کلاس

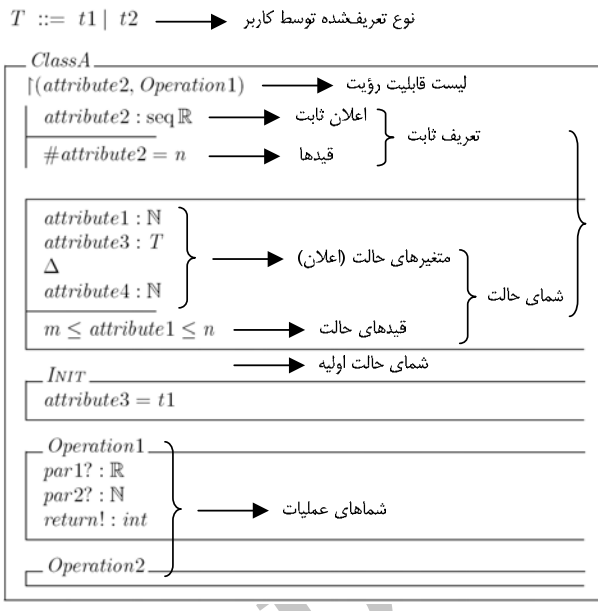
در Object - Z، یک کلاس شامل شِمای حالت و شِماهای عملیات به ترتیب برای تعریف متغیرهای حالت (اولیه و ثانویه) و عملیات است. همچنین یک کلاس شامل یک شِمای حالت اولیه به منظور مقداردهی اولیه متغیرهای حالت است. یک ثابت که در یک کلاس UML با علامت {*frozen*} مشخص می‌شود، در کلاس Object - Z به شکل یک قاعده تعریف می‌شود که در بخش قیدهای آن، محدودیت‌های ثابت تعریف‌شده در بخش اعلان بیان می‌شوند. صفات ثانویه که در UML با علامت / مشخص می‌شوند، درون شِمای حالت Object - Z به کمک علامت Δ از صفات اولیه متمایز می‌شوند. در UML و Object - Z، انواع تعریف‌شده توسط کاربر مانند T قابل تعریف هستند. در Object - Z، متغیرهای ورودی و خروجی یک عملیات به‌ترتیب با علامت‌های ? و ! مشخص می‌شوند. شکل‌های ۱ و ۲ به ترتیب، نحوه تعریف یک کلاس عمومی در Object - Z و UML را نشان می‌دهند.

قواعد تبدیل از Object - Z به UML عبارتند از:

۱) اگر یک کلاس Object - Z وجود داشته باشد، آنگاه یک کلاس UML با نام یکسان وجود خواهد داشت.

۲) اگر یک ویژگی (صفت یا عملیات) درون لیست قابلیت رؤیت یک کلاس Object - Z وجود داشته باشد، آنگاه ویژگی مذکور با علامت + (عمومی) در کلاس UML متناظر تعریف می‌شود.

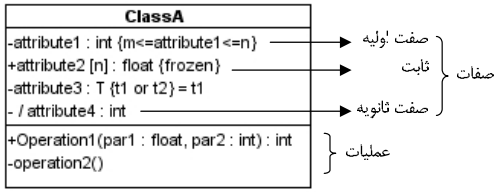
1. Meta - Model
2. Endogenous
3. Exogenous
4. Horizontal
5. Vertical
6. Primary Attributes
7. Derived Attributes
8. Constants
9. Operations
10. Visibility
11. User - Defined Types
12. Multiplicity
13. Initialization
14. Generic Inheritance
15. Unidirectional Association
16. Bidirectional Association
17. Aggregation
18. Composition
19. Association Class
20. Polymorphism



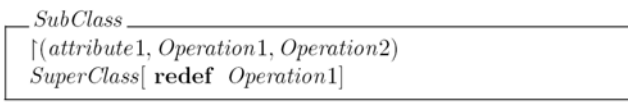
شکل ۱: یک کلاس عمومی در Object-Z.

۳) اگر یک ویژگی درون لیست قابلیت رؤیت یک کلاس Object - Z وجود نداشته باشد، آنگاه ویژگی مذکور با علامت – (خصوصی) در کلاس UML متناظر تعریف می‌شود.
۴) اگر یک متغیر درون شِمای حالت یک کلاس Object - Z بالای جداکننده Δ (در صورت وجود) وجود داشته باشد، آنگاه یک صفت اولیه با نام یکسان در کلاس UML متناظر تعریف می‌شود.
۵) اگر یک متغیر درون شِمای حالت یک کلاس Object - Z پایین جداکننده Δ وجود داشته باشد، آنگاه یک صفت ثانویه با نام یکسان در کلاس UML متناظر تعریف می‌شود.
۶) اگر یک ثابت درون یک شِمای تعریف ثابت در یک کلاس Object - Z اعلان شده باشد، آنگاه یک ثابت با نام یکسان درون کلاس UML متناظر تعریف می‌شود.
۷) اگر یک متغیر به‌صورت یک رشته متناهی از یک نوع پایه یا تعریف‌شده توسط کاربر در توصیف Object - Z موجود باشد، آنگاه یک صفت با نام یکسان به‌صورت آرایه‌ای از نوع مشابه با اندازه یکسان درون کلاس متناظر در UML تعریف می‌شود.
۸) اگر عبارت $attribute۳ = t_۱$ در شِمای حالت اولیه یک کلاس Object-Z موجود باشد، آنگاه مقدار اولیه $t_۱$ به متغیر $attribute۳$ درون کلاس متناظر در UML منتسب می‌شود.
۹) اگر یک شِمای عملیات در یک کلاس Object - Z وجود داشته باشد، آنگاه یک عملیات با نام و پارامترهای ورودی و خروجی یکسان درون کلاس متناظر در UML تعریف می‌شود.
۱۰) اگر یک نوع آزاد مانند T شامل مقادیر گسسته $(\{t_۱, t_۲\})$ درون توصیف Object - Z موجود باشد، آنگاه یک نوع تعریف‌شده توسط کاربر به‌صورت $T\{t_۱, t_۲\}$ در UML تعریف می‌شود.

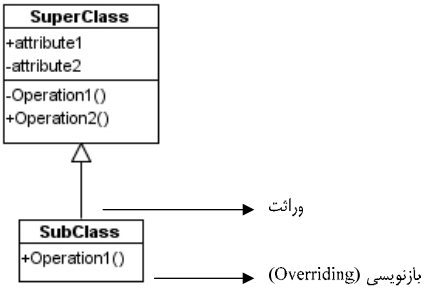
قواعد تبدیل از UML به Object - Z عبارتند از:
۱) اگر یک کلاس UML وجود داشته باشد، آنگاه یک کلاس Object - Z با نام یکسان وجود خواهد داشت.
۲) اگر یک ویژگی (صفت یا عملیات) با علامت + (عمومی) در یک کلاس UML وجود داشته باشد، آنگاه ویژگی مذکور به لیست قابلیت رؤیت کلاس متناظر در Object - Z اضافه می‌شود.
۳) اگر یک ویژگی با علامت – (خصوصی) یا بدون علامت در



شکل ۲: یک کلاس عمومی در UML.



شکل ۳: نحوه تعریف رابطه وراثت در Object-Z.

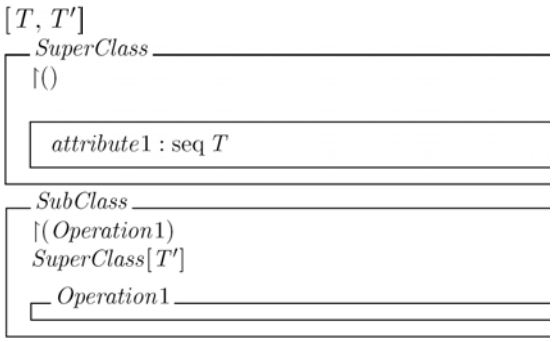


شکل ۴: نحوه تعریف رابطه وراثت در UML.

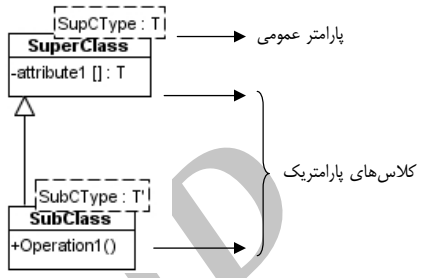
یک کلاس UML وجود داشته باشد، آنگاه ویژگی مذکور به لیست قابلیت رؤیت کلاس متناظر در Object - Z اضافه نمی‌شود.
۴) اگر یک صفت اولیه در یک کلاس UML وجود داشته باشد، آنگاه یک متغیر با نام یکسان درون شِمای حالت کلاس متناظر در Object - Z بالای جداکننده Δ (در صورت وجود) اعلان می‌شود.
۵) اگر یک صفت ثانویه در یک کلاس UML وجود داشته باشد، آنگاه یک متغیر با نام یکسان درون شِمای حالت کلاس متناظر در Object - Z پایین جداکننده Δ اعلان می‌شود.
۶) اگر یک ثابت در یک کلاس UML وجود داشته باشد، آنگاه یک ثابت با نام یکسان درون یک شِمای تعریف ثابت مجزا در کلاس متناظر Object - Z اعلان می‌شود.
۷) اگر یک صفت مانند $attribute۲$ با تعدد بیش از ۱ در یک کلاس UML وجود داشته باشد، آنگاه یک متغیر به صورت یک رشته متناهی از نوع مشابه به همراه یک قید تعداد درون کلاس متناظر در Object - Z اعلان می‌شود.
۸) اگر یک مقدار اولیه مانند $t_۱$ به یک متغیر مانند $attribute۳$ در یک کلاس UML منتسب شود، آنگاه عبارت $attribute۳ = t_۱$ به شِمای حالت اولیه کلاس متناظر در Object - Z افزوده می‌شود.
۹) اگر یک عملیات در یک کلاس UML وجود داشته باشد، آنگاه یک شِمای عملیات با نام و پارامترهای ورودی و خروجی یکسان درون کلاس متناظر در Object - Z اعلان می‌شود.
۱۰) اگر یک نوع تعریف‌شده توسط کاربر مانند T به‌صورت یک مجموعه متناهی از مقادیر $(\{t_۱, t_۲\})$ در UML موجود باشد، آنگاه عبارت $T :: t_۱ \mid t_۲$ به‌صورت یک نوع آزاد درون توصیف متناظر در Object - Z تعریف می‌شود.

۲-۲ قواعد دوجهته تبدیل رابطه تعمیم (وراثت)

وراثت رابطه‌ای است بین چند کلاس که در آن یک کلاس (وارث یا فرزند) در ساختار، رفتار یا هر دو با یک کلاس (پدر یا والد) یا چند کلاس دیگر شراکت دارد. شکل‌های ۳ و ۴ به ترتیب، نحوه تعریف رابطه وراثت



شکل ۵: نحوه تعریف وراثت عمومی در Object - Z.



شکل ۶: نحوه تعریف وراثت عمومی در UML.

در Object - Z و UML را نشان می‌دهند. کلاس وارث *SubClass* شامل ویژگی‌های کلاس *SuperClass* نیز می‌باشد. با این وجود، لیست قابلیت رؤیت از ارث‌بری مستثنی است. این نکته امکان تعریف یک رابط جدید برای کلاس وارث را فراهم می‌سازد. لذا لیست قابلیت رؤیت باید مجدداً به طور صریح در کلاس وارث تعریف شود. عملیات بازنویسی شده (نظیر *Operation1*) در UML با استفاده از عبارت رزروشده *redef* در Object - Z بازنمایی می‌شود.

قواعد دوجهته تبدیل رابطه وراثت بین Object-Z و UML عبارتند از:

۱) اگر یک رابطه وراثت بین یک کلاس فرزند مانند *SubClass* و یک کلاس پدر مانند *SuperClass* در یک نمودار کلاس UML موجود باشد، آنگاه نام کلاس پدر (*SuperClass*) بلافاصله بعد از لیست قابلیت رؤیت کلاس فرزند (*SubClass*) درون توصیف متناظر در Object - Z بیان خواهد شد و بالعکس.

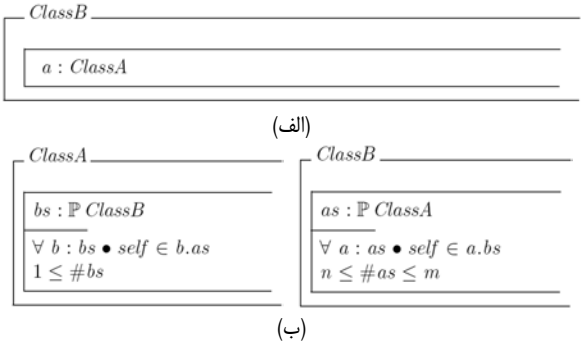
۲) اگر یک عملیات بازنویسی شده مانند *Operation1* در یک کلاس فرزند مانند *SubClass* در یک نمودار کلاس UML موجود باشد، آنگاه عبارت [*redef Operation1*] *SuperClass* بلافاصله بعد از لیست قابلیت رؤیت کلاس فرزند (*SubClass*) درون توصیف متناظر در Object - Z بیان خواهد شد و بالعکس.

۲-۳ قاعده دوجهته تبدیل وراثت عمومی

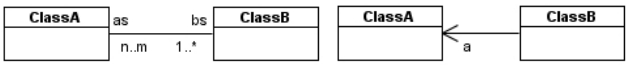
وراثت عمومی مکانیزمی برای تعریف ساختارهای عام منظوره است. شکل‌های ۵ و ۶ به ترتیب، نحوه تعریف وراثت عمومی در Object - Z و UML را نشان می‌دهند.

قاعده دوجهته تبدیل وراثت عمومی بین Object - Z و UML عبارت است از:

اگر یک رابطه وراثت عمومی بین یک کلاس پدر مانند *SuperClass* با یک نوع عمومی (T) و یک کلاس فرزند مانند *SubClass* با یک نوع متفاوت (T') در یک نمودار کلاس UML موجود باشد، آنگاه عبارت $SuperClass[T']$ بلافاصله بعد از لیست قابلیت رؤیت کلاس فرزند (*SubClass*) درون توصیف متناظر در Object - Z بیان خواهد شد و بالعکس.



شکل ۷: (الف) نحوه تعریف رابطه انجمنی یک‌جهته در Object - Z و (ب) نحوه تعریف رابطه انجمنی دوجهته در Object - Z.



شکل ۸: (الف) نحوه تعریف رابطه انجمنی یک‌جهته در UML و (ب) نحوه تعریف رابطه انجمنی دوجهته در UML.

۲-۴ قواعد دوجهته تبدیل رابطه انجمنی

رابطه انجمنی، قابلیت یک شیء را در ارسال یک پیغام به یک شیء دیگر نشان می‌دهد. روابط انجمنی در Object - Z از طریق تعریف صفات حالت اضافی، بسته به جهت تعیین‌شده بر روی خط رابطه انجمنی در UML، بازنمایی می‌شوند. در Object - Z، رابطه انجمنی از طریق حضور یک یا چند نوع کلاس در بخش تعریف متغیرهای کلاس دیگر مشخص می‌شود. بخش‌های (الف) و (ب) در شکل ۷ به ترتیب، رابطه انجمنی یک‌جهته و دوجهته در Object - Z را نشان می‌دهند. بخش‌های (الف) و (ب) در شکل ۸ نیز به ترتیب، رابطه انجمنی یک‌جهته و دوجهته در UML را نشان می‌دهند.

قواعد دوجهته تبدیل رابطه انجمنی بین Object - Z و UML عبارت هستند از:

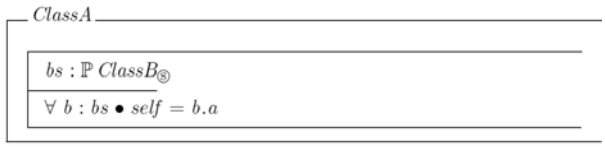
۱) اگر یک رابطه انجمنی یک‌جهته بین دو کلاس مانند *ClassA* (در سمت نوک پیکان) و *ClassB* (در سمت انتهای پیکان) در یک نمودار کلاس UML موجود باشد، آنگاه یک متغیر در شِمای حالت *ClassB* به منظور تعریف یک شیء از نوع *ClassA* درون توصیف متناظر در Object - Z اعلان خواهد شد و بالعکس.

۲) اگر یک رابطه انجمنی دوجهته بین دو کلاس مانند *ClassA* با تعدد M_A و نقش *as* و *ClassB* با تعدد M_B و نقش *bs* در یک نمودار کلاس UML موجود باشد، آنگاه شِماهای حالت دو کلاس متناظر در Object - Z شامل موارد زیر خواهد شد: ۱) یک متغیر که به صورت یک مجموعه توانی (P) از نوع کلاس دیگر اعلان شده است، ۲) یک قید که وجود رابطه واقعی بین اشیای ۲ کلاس را تضمین می‌نماید و ۳) یک قید که نمایانگر تعدد است و بالعکس.

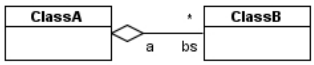
۲-۵ قاعده دوجهته تبدیل رابطه تجمع

رابطه تجمع نوع خاصی از رابطه انجمنی است که رابطه "کل به جزء" را مدل می‌نماید. در این رابطه حیات کلاس جزء از حیات کلاس کل مستقل است. Object - Z، یک نماد ویژه برای مدل‌سازی رابطه تجمع دارد (Ⓢ). شکل‌های ۹ و ۱۰ به ترتیب، نحوه تعریف رابطه تجمع در Object - Z و UML را نشان می‌دهند.

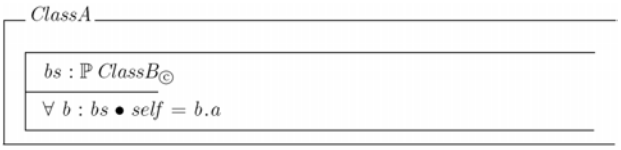
قاعده دوجهته تبدیل رابطه انجمنی بین Object - Z و UML عبارت است از:



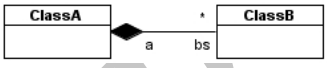
شکل ۹: نحوه تعریف رابطه تجمع در Object - Z.



شکل ۱۰: نحوه تعریف رابطه تجمع در UML.



شکل ۱۱: نحوه تعریف رابطه ترکیب در Object - Z.



شکل ۱۲: نحوه تعریف رابطه ترکیب در UML.

اگر یک رابطه تجمع بین دو کلاس مانند *ClassA* (به عنوان کل) با نقش *a* و *ClassB* (به عنوان جزء) با نقش *bs* در یک نمودار کلاس UML موجود باشد، آنگاه عبارات $bs : \mathrm{P}ClassB_{©}$ و $\forall b : bs \bullet self = b.a$ به شِمای حالت کلاس کل (*ClassA*) به ترتیب در قالب یک اعلان و یک قید، درون توصیف متناظر در Object - Z اضافه خواهند شد و بالعکس.

۲-۶ قاعده دوجهته تبدیل رابطه ترکیب

رابطه ترکیب دقیقاً مشابه رابطه تجمع است با این تفاوت که حیات کلاس جزء به حیات کلاس کل وابسته است. در رابطه ترکیب زمانی که حیات کلاس کل تمام می‌شود، حیات کلاس‌های جزء نیز پایان می‌پذیرد. Object - Z یک نماد ویژه برای مدل‌سازی رابطه ترکیب دارد (©). شکل‌های ۱۱ و ۱۲ به ترتیب، نحوه تعریف رابطه ترکیب در Object - Z و UML را نشان می‌دهند.

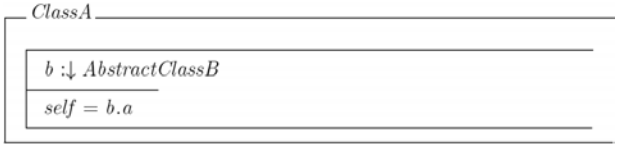
قاعده دوجهته تبدیل رابطه انجمنی بین Object - Z و UML عبارت است از:

اگر یک رابطه ترکیب بین دو کلاس UML مانند *ClassA* (به عنوان کل) با نقش *a* و *ClassB* (به عنوان جزء) با نقش *bs* موجود باشد، آنگاه عبارات $bs : \mathrm{P}ClassB_{©}$ و $\forall b : bs \bullet self = b.a$ به شِمای حالت کلاس کل (*ClassA*) به ترتیب در قالب یک اعلان و یک قید، درون توصیف متناظر در Object - Z اضافه خواهند شد و بالعکس.

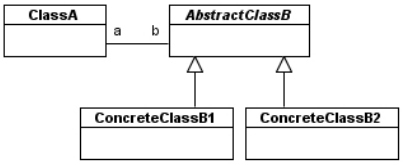
۲-۷ قاعده دوجهته تبدیل چندریختی

چندریختی مکانیزمی است که به کمک آن می‌توان متغیری از نوع یکی از کلاس‌های یک مجموعه کلاس تعریف کرد. مجموعه کلاس‌های مذکور از طریق رابطه ارث‌بری از یک کلاس مجرد مشتق می‌شوند. شکل‌های ۱۳ و ۱۴ به ترتیب، نحوه تعریف چندریختی در Object - Z و UML را نشان می‌دهند. در Object-Z عبارت $b :\downarrow AbstractClassB$ شیء *b* را از نوع کلاس *AbstractClassB* یا هر یک از کلاس‌هایی که از *AbstractClassB* ارث برده‌اند، تعریف می‌نماید.

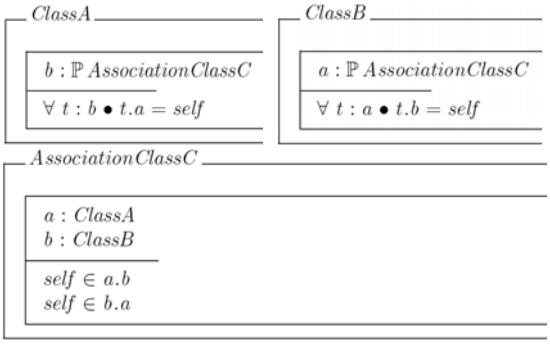
قاعده دوجهته تبدیل چندریختی بین Object - Z و UML عبارت است از:



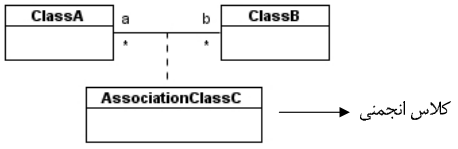
شکل ۱۳: نحوه تعریف چندریختی در Object - Z.



شکل ۱۴: نحوه تعریف چندریختی در UML.



شکل ۱۵: نحوه تعریف کلاس انجمنی در Object - Z.



شکل ۱۶: نحوه تعریف کلاس انجمنی در UML.

اگر رابطه یک کلاس UML مانند *ClassA* (با نقش *a*) با سلسله مراتبی از کلاس‌ها مانند *AbstractClassB* (با نقش *b*) و فرزندانش بر مبنای چندریختی باشد، آنگاه عبارات $b :\downarrow AbstractClassB$ و $self = b.a$ به شِمای حالت کلاس *ClassA* به ترتیب در قالب یک اعلان و یک قید، درون توصیف متناظر در Object - Z اضافه خواهند شد و بالعکس.

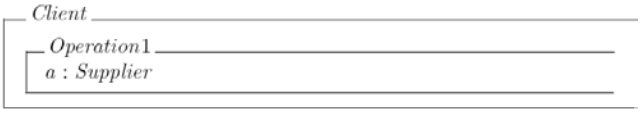
۲-۸ قاعده دوجهته تبدیل کلاس انجمنی

یک کلاس انجمنی شامل اطلاعاتی پیرامون یک رابطه انجمنی است و همانند یک کلاس معمولی بازنمایی می‌شود. تعریف کلاس‌های انجمنی اغلب در روابط انجمنی یک- به- چند و چند- به- چند که خود رابطه دارای ویژگی‌هایی است، ضرورت پیدا می‌کند. در UML یک کلاس انجمنی توسط یک خط نقطه‌چین به رابطه انجمنی متناظر متصل می‌گردد. شکل‌های ۱۵ و ۱۶ به ترتیب، نحوه تعریف کلاس انجمنی در Object - Z و UML را نشان می‌دهند.

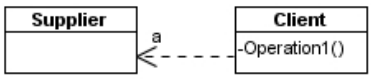
قاعده دوجهته تبدیل کلاس انجمنی بین Object - Z و UML عبارت است از:

اگر یک کلاس ارتباطی مانند *AssociationClassC* به یک رابطه انجمنی که دو کلاس UML مانند *ClassA* (با نقش *a*) و *ClassB* (با نقش *b*) را به یکدیگر متصل می‌نماید، الصاق شده باشد، آنگاه درون توصیف متناظر در Object - Z:

- اعلان $b : \mathrm{P}AssociationClassC$ و قید $\forall t : b \bullet t.a = self$ به شِمای حالت *ClassA* اضافه خواهند شد.



شکل ۱۷: نحوه تعریف رابطه وابستگی در Object-Z.



شکل ۱۸: نحوه تعریف رابطه وابستگی در UML.

- اعلان $a : \mathrm{P}AssociationClassC$ و قید $\forall t : a \bullet t.b = self$ به شِمای حالت *ClassB* اضافه خواهند شد.
- اعلان‌های $a : Class$ و $b : ClassB$ و قیود $self \in a.b$ و $self \in b.a$ به شِمای حالت *AssociationClassC* اضافه خواهند شد و بالعکس.

۲-۹ قواعد دوجهته تبدیل رابطه وابستگی

رابطه وابستگی برای بازنمایی وابستگی یک کلاس به کلاس دیگر مورد استفاده قرار می‌گیرد. در UML کلاسی که در سمت انتهای پیکان قرار دارد به کلاسی که در سمت نوک پیکان قرار دارد وابسته است. شکل‌های ۱۷ و ۱۸ به ترتیب، نحوه تعریف رابطه وابستگی در Object-Z و UML را نشان می‌دهند.

قواعد دوجهته تبدیل رابطه وابستگی بین Object - Z و UML عبارتند از:

۱) اگر در Object-Z یک متغیر از نوع *Supplier* درون عملیات *Operation1* کلاس *Client* اعلان شده باشد، آنگاه در نمودار کلاس متناظر در UML یک رابطه وابستگی بین کلاس‌های *Supplier* و *Client* تعریف می‌شود به قسمی که *Client* در انتهای پیکان وابستگی قرار خواهد گرفت.

۲) اگر یک کلاس UML (*Client*) شامل عملیاتی مانند *Operation1* باشد که برای انجام وظایف خود وابسته به کلاس دیگری (*Supplier*) است، آنگاه درون توصیف متناظر در Object - Z یک متغیر برای تعریف شیءای از نوع *Supplier* درون عملیات *Operation1* کلاس *Client* اعلان می‌شود.

۳- مطالعه موردی: سیستم آسانسور چندکابینه

مطالعه موردی از متداول‌ترین روش‌های ارزیابی در مهندسی نرم‌افزار است [۴] و [۱۱]. بر مبنای این روش، تأثیر و کارایی دست‌آوردهای علمی و تحقیقاتی به‌صورت عملی ارزیابی می‌شود. هدف از تعریف مطالعه موردی سیستم آسانسور چندکابینه، فراهم‌سازی یک بستر آزمایش مناسب برای ارزیابی و امکان‌سنجی عملی مکانیزم تبدیل پیشنهادی است. نتایج ارزیابی به طور مفصل در [۹] ذکر شده است. با انجام مطالعه موردی مذکور، قابلیت استفاده مکانیزم پیشنهادی به صورت عملی امکان‌سنجی می‌شود.

در ادامه این بخش، نحوه مدل‌سازی یک سیستم آسانسور چندکابینه به طور خلاصه ارائه می‌گردد. در ابتدا سیستم آسانسور چندکابینه توسط Object - Z توصیف می‌شود. سپس توصیف مذکور توسط مکانیزم تبدیل پیشنهادی به‌صورت بصری در قالب نمودار کلاس UML بازنمایی شده و به کمک تعدادی الگوی طراحی و چندریختی، نمودار کلاس مذکور بازبینی می‌شود. سپس نمودار کلاس بازبینی شده توسط مکانیزم تبدیل پیشنهادی به صورت صوری بازنمایی می‌گردد. در این حالت، مدل صوری مذکور آماده پالایش‌های صوری بیشتر است. به دلیل کمبود فضا، مدل‌های تولیدشده به‌صورت خلاصه در این بخش ارائه می‌شوند. بدین معنی که از ذکر بخش‌های تکراری (از نظر نوع پیچیدگی و المان‌های مدل‌سازی مورد استفاده) صرف نظر شده است. نسخه کامل توصیفات صوری، بصری و غیر صوری سیستم آسانسور چندکابینه در [۹] ارائه شده است.

سیستم آسانسور چندکابینه توسط یک نرم‌افزار توزیع‌شده و موازی کنترل می‌شود. نرم‌افزار مذکور باید کلیه اطلاعات مرتبط درباره آسانسورها را پردازش نماید و موتور را به اندازه لازم و در جهت درست حرکت دهد تا آنها را به مقصد مورد نظر برساند. سیستم‌های آسانسور چندکابینه با فراهم‌کردن امکان حرکت مستقل آسانسورها در یک چاه آسانسور واحد، تحول وسیعی در کاهش فضای مورد استفاده در مجتمع‌های شهری ایجاد نموده است. در حال حاضر فضای مفید زیادی از ساختمان، صرف چاه‌های آسانسور می‌شود که هر کدام مسئول حرکت‌دادن فقط یک آسانسور در فضای بزرگ خالی اشغال‌شده‌ای در تعداد زیادی طبقه می‌باشند. سیستم‌های آسانسور چندکابینه به معماران امکان صرفه‌جویی زیادی در فضا از طریق حرکت تعداد زیادی آسانسور در یک چاه مشترک می‌دهد. سیستم مذکور یک بستر آزمایش متداول برای نمایش قدرت زبان‌های مدل‌سازی در توصیف سیستم‌های تعاملی هم‌زمان پیچیده است. پیچیدگی سیستم آسانسور چندکابینه ناشی از تعاملات هم‌زمان ذاتی مؤلفه‌های مختلف آن است.

سیستم آسانسوری که در این پروژه به عنوان مطالعه موردی تعریف و طراحی شده است شامل کارکردهای اصلی یک سیستم آسانسور واقعی نظیر حرکت به سمت بالا و پایین، باز و بسته‌شدن درب‌ها و البته سوارکردن مسافر می‌باشد. فرض بر این است که سیستم بالابر مذکور در یک مجتمع بزرگ مسکونی- تجاری چندطبقه واقع شده است و شامل چندین آسانسور است. طبقات ساختمان از ۱ تا *MaxFloor* شماره‌گذاری شده‌اند. درون هر آسانسور یک تابلو شامل چندین دکمه وجود دارد و هر دکمه متناظر با یک طبقه است. در هر طبقه دو دکمه جهت برای درخواست حرکت به سمت پایین یا بالا وجود دارد. پایین‌ترین و بالاترین طبقات فقط به یک دکمه جهت نیاز دارند. هر یک از دکمه‌های مذکور در هر لحظه می‌تواند فشار داده شود. هر یک از دکمه‌های بیرونی در هر طبقه از لحظه‌ای که فشار داده می‌شود تا لحظه‌ای که یک آسانسور با همان جهت حرکت درخواستی در طبقه مذکور توقف کرده و درب آن باز می‌شود، روشن می‌ماند. همچنین از لحظه‌ای که کنترل‌کننده مرکزی بالابر، یک آسانسور را برای پاسخ‌گویی به یک درخواست خارجی تخصیص می‌دهد، دکمه داخلی متناظر با طبقه‌ای که درخواست مذکور از آنجا اعلام شده است در آسانسور تخصیص داده شده به‌صورت چشمک‌زن روشن خواهد شد (در صورتی که پیش از این در اثر یک درخواست داخلی روشن نشده باشد) و تا لحظه رسیدن به مقصد روشن خواهد ماند. هر یک از دکمه‌های درون یک آسانسور نیز از لحظه‌ای که فشار داده می‌شود تا لحظه‌ای که آسانسور طبقه متناظر را ملاقات می‌کند، روشن می‌ماند.

همچنین هر آسانسور دارای یک نمایشگر جهت حرکت و یک نمایشگر موقعیت می‌باشد که در داخل آن تعبیه شده‌اند. نمایشگر موقعیت یک آسانسور همیشه روشن است و موقعیت آسانسور را در هر لحظه نمایش می‌دهد. نمایشگر جهت حرکت، فقط در لحظاتی که آسانسور در حال حرکت می‌باشد روشن است و جهت حرکت آسانسور را نمایش می‌دهد. با هدف مشابه در هر طبقه به تعداد آسانسورهای موجود نمایشگر جهت حرکت و موقعیت آسانسورها تعبیه شده است. ضمناً آسانسورهای سیستم

بالابر مذکور در داخل خود دارای دکمه‌هایی برای بازکردن و بستن دستی درب‌ها، تهویه و اعلام خطر هستند. علاوه بر این، ویژگی‌های مطروحه زیر نیز باید در مدل‌سازی سیستم بالابر مذکور لحاظ شوند:

- ویژگی ایمنی ۱ ($S1$): درب یک آسانسور قبل از هر حرکتی به سمت بالا یا پایین باید بسته باشد.
- ویژگی ایمنی ۲ ($S2$): در صورت قطع برق، به کمک برق اضطراری، هر آسانسور در حال حرکت باید در نزدیک‌ترین طبقه متوقف شود و درب آن باز گردد.
- ویژگی سیستمی ۱ ($P1$): وقتی یک آسانسور هیچ درخواستی ندارد، بسته به استراتژی حرکت یا باید در آخرین مقصدش متوقف بماند و درب آن بسته باشد و یا به طبقه آماده‌باش اعزام شود.
- ویژگی سیستمی ۲ ($P2$): یک آسانسور باید فقط در صورتی یک درخواست خارجی را برآورده نماید که ۱) به سمت درخواست مذکور در حال حرکت باشد و جهت درخواست مذکور با جهت حرکت آسانسور یکی باشد یا ۲) آسانسور بیکار باشد و ۳) از بین آسانسورهایی که یکی از شرایط دوگانه فوق را دارا هستند به محل درخواست مذکور نزدیک‌ترین باشد. شرایط فوق‌الذکر توسط کنترل‌کننده مرکزی بررسی می‌شود.
- ویژگی سیستمی ۳ ($P3$): وقتی که یک درخواست از یک طبقه می‌رسد، کنترل‌کننده مرکزی درخواست مذکور را در انتهای صف درخواست‌های خارجی قرار خواهد داد.
- ویژگی سیستمی ۴ ($P4$): اگر یکی از درخواست‌های خارجی توسط یکی از آسانسورها برآورده شود، آن درخواست باید از صف درخواست‌ها حذف گردد.
- ویژگی سیستمی ۵ ($P5$): همواره اولویت پاسخ‌گویی به درخواست‌های خارجی از درخواست‌های داخلی بیشتر است.
- ویژگی سیستمی ۶ ($P6$): همواره یک آسانسور تا زمانی که به تمام درخواست‌های داخلی یا خارجی در جهت حرکت فعلی خود پاسخ ندهد، به منظور پاسخ‌گویی به درخواست‌های در خلاف جهت خود تغییر جهت نمی‌دهد.
- ویژگی سیستمی ۷ ($P7$): تا زمانی که چراغ یکی از دکمه‌های داخلی یا خارجی روشن است، فشردن مجدد آن، به‌عنوان درخواست جدید محسوب نمی‌شود.
- ویژگی زمانی ۱ ($T1$): هرگاه صف درخواست‌های خارجی خالی نباشد، درخواستی که دارای طولانی‌ترین زمان انتظار است، از صف خارج و به نزدیک‌ترین آسانسور واجد شرایط (بر مبنای مکانیزم بیان‌شده در $P2$) تخصیص داده می‌شود.
- ویژگی زمانی ۲ ($T2$): به همه درخواست‌های خارجی (در نتیجه فشردن کلیدهای طبقات) در نهایت باید پاسخ داده شود.
- ویژگی زمانی ۳ ($T3$): به همه درخواست‌های داخلی (در نتیجه فشردن کلیدهای داخل آسانسورها) در نهایت باید پاسخ داده شود.
- ویژگی زمانی ۴ ($T4$): اگر درب یکی از آسانسورها باز باشد، در نهایت باید بسته شود.

کنترل‌کننده مرکزی مسؤل کنترل آسانسورها از طریق کنترل‌کننده‌های محلی آنها است. در ابتدا همه آسانسورها در طبقه آماده‌باش قرار می‌گیرند. اگر یک مسافر وارد یک آسانسور شده و دکمه طبقه k را فشار دهد، اطلاعات درخواست مذکور به کنترل‌کننده مرکزی ارسال می‌شود. سپس آسانسور مذکور توسط کنترل‌کننده محلی خود به سمت طبقه k بر اساس استراتژی حرکت تعیین‌شده توسط کنترل‌کننده مرکزی اعزام می‌شود. استراتژی حرکت برای هر آسانسور بر اساس چندین معیار نظیر سیاست‌های مدیریت و وضعیت ترافیک مشخص می‌شود. زمانی که آسانسور به مقصد می‌رسد، درب آن باز شده و برای مدت معینی به همان حالت باقی می‌ماند. سپس مجدداً بسته می‌شود و بسته به استراتژی تعیین‌شده یا در همان طبقه بیکار می‌ماند و یا به طبقه آماده‌باش اعزام می‌شود. به‌علاوه، زمانی که یک مسافر در طبقه m درخواستی را از طریق فشردن یکی از دکمه‌های بالا یا پایین طبقه مذکور اعلام می‌نماید، مناسب‌ترین آسانسور توسط کنترل‌کننده مرکزی به طبقه مذکور ارسال و درب آن در مقصد باز می‌گردد.

کنترل‌کننده مرکزی مناسب‌ترین آسانسور را برای پاسخ‌گویی به هر یک از درخواست‌های خارجی بر مبنای تعدادی معیار ارزیابی نظیر موقعیت و جهت حرکت آسانسورها تعیین می‌نماید. کارایی کنترل‌کننده مرکزی توسط چندین معیار نظیر میانگین زمان پاسخ‌گویی به مسافران، درصد مسافرانی که بیش از ۶۰ ثانیه انتظار کشیده‌اند و مصرف انرژی ارزیابی می‌شود. کنترل‌کننده مرکزی، آسانسورها را به‌گونه‌ای مدیریت می‌نماید که معیارهای ارزیابی مذکور کمینه شوند. البته برآوردن همه این معیارها به‌صورت همزمان دشوار است. بنابراین کنترل‌کننده مرکزی به‌گونه‌ای طراحی می‌شود که هر یک از معیارهای مذکور را در سطح مشخصی برآورده نماید. امروزه مدیران سیستم‌ها خواهان تعریف استراتژی کنترل برای کنترل‌کننده مرکزی هستند. بدین معنی که بعضی از مدیران خواستار کاهش میانگین زمان پاسخ‌گویی و برخی دیگر خواهان کاهش مصرف انرژی هستند. لذا طراحی یک کنترل‌کننده انعطاف‌پذیر که امکان تغییر پویای استراتژی کنترل را داشته باشد ضروری است.

توصیف صوری اولیه سیستم آسانسور چندکابینه به‌صورت خلاصه به شرح زیر است:

Floor

$\uparrow()$

$num : \mathbb{N}$
$centc : CentralController$

$1 \leq num \leq \#floors$
$self \in centc.floors$

همان طور که ملاحظه می‌شود، مفهوم طبقه در قالب یک کلاس با نام *Floor* مدل شده است. شماره طبقه به‌صورت یک متغیر از نوع اعداد طبیعی با نام *num* تعریف شده است. به‌علاوه، کلاس طبقه شامل یک متغیر از نوع کلاس *CentralController* نیز می‌باشد. این متغیر که با نام *centc* مشخص شده است، نمایانگر وجود یک رابطه انجمنی بین کلاس *Floor* (طبقه) و کنترل‌کننده مرکزی (*CentralController*) می‌باشد. تعدی کلاس *CentralController* در رابطه انجمنی مذکور ۱ است. قید اول، حداقل و حداکثر تعداد طبقات را مشخص می‌نماید. قید دوم نیز بر این نکته تأکید می‌کند که هر شیء از نوع کلاس طبقه باید متعلق به مجموعه طبقات (*floors*) کنترل‌کننده مرکزی باشد.

Passenger

$\uparrow(PressLiftButton, PressAlarmButton, PressAirButton)$

PressLiftButton
$lbut : LiftButton$

PressAlarmButton
$albutton : AlarmButton$

PressAirButton
$aibutton : AirButton$

مسافر نیز در قالب یک کلاس با نام *Passenger* تعریف می‌شود. این کلاس دارای ۳ متد با نام‌های *PressLiftButton*، *PressAlarmButton* و *PressAirButton* می‌باشد که به ترتیب جهت فشردن دکمه‌های طبقه، هشدار و تهویه مورد استفاده قرار می‌گیرند. در ضمن، قابلیت رؤیت هر سه متد مذکور از نوع عمومی است. زیرا نام آنها در لیست رؤیت کلاس مسافر قرار گرفته است. به همین ترتیب سایر اجزای سیستم آسانسور در قالب کلاس‌هایی شامل متغیرها و متدهای مورد نیاز تعریف شده‌اند. تنوع کلاس‌های مذکور به‌گونه‌ای است که کلیه المان‌های مدل‌سازی مشترک نمودار کلاس و Object - Z را پوشش می‌دهد و لذا بستر لازم را برای ارزیابی عملی قابلیت استفاده مکانیزم تبدیل پیشنهادی فراهم می‌نماید. به دلیل محدودیت فضا از توضیح سایر کلاس‌های تعریف‌شده خودداری می‌نماییم.

$[T, Requests]$

LiftButton
↾(*Press*)

liftb : *Lift*
self ∈ *liftb.fbuts*

Press

AlarmButton
↾(*Press*)

lif : *Lift*
self = *lif.albut*

Press

RequestQueue
↾(*items*, *Remove*)
NRQueue[*Requests*]

LocalController

centcont : *CentralController*
lift : *Lift*
self ∈ *centcont.locconts*
self = *lift.lc*

ExternalRequestAllocator
↾(*AllocateExternalRequest*)

centralcontroller : *CentralController*
centralcontroller.extreqalc = *self*

AllocateExternalRequest

ec : *EvaluationCriteria*	
return! : ℕ	

$LiftStatus ::= inUse \mid Idle$

TrafficFeature
↾(*MeasureFeature*)

MeasureFeature

tmr : *TrafficManager*	
return! : ℝ	

ControlStrategyGenerator
↾(*CalculateCurrentTrafficMode*)

ceco : *CentralController*	
chmanc : *ChangeManager*	
ceco.csg = *self*	
chmanc.csgen = *self*	

CalculateCurrentTrafficMode

tmd : *TrafficMode*	

CentralController
↾(*TurnOffFloorBut*)

tmcc : *TrafficManager*©	
locconts : seq *LocalController*	
reqQ : *RequestQueue*	
extreqalc : *ExternalRequestAllocator*©	
csg : *ControlStrategyGenerator*©	
floors : seq *Floor*	
#*locconts* = #*floors*	
tmcc.cctrl = *self*	
∀ *l* : ran *locconts* • *l.centcont* = *self*	
extreqalc.centralcontroller = *self*	
csg.ceco = *self*	
∀ *f* : ran *floors* • *f.centc* = *self*	

TurnOffFloorBut

Queue[*T*]
↾(*items*)

items : seq *T*	

INIT

items = ⟨⟩	

TrafficManager
↾()

trafficInfo : seq ℝ	
cctrl : *CentralController*	
cctrl.tmcc = *self*	

TInfoChanged

AirButton
↾(*Press*)

lf : *Lift*	
self = *lf.aibut*	

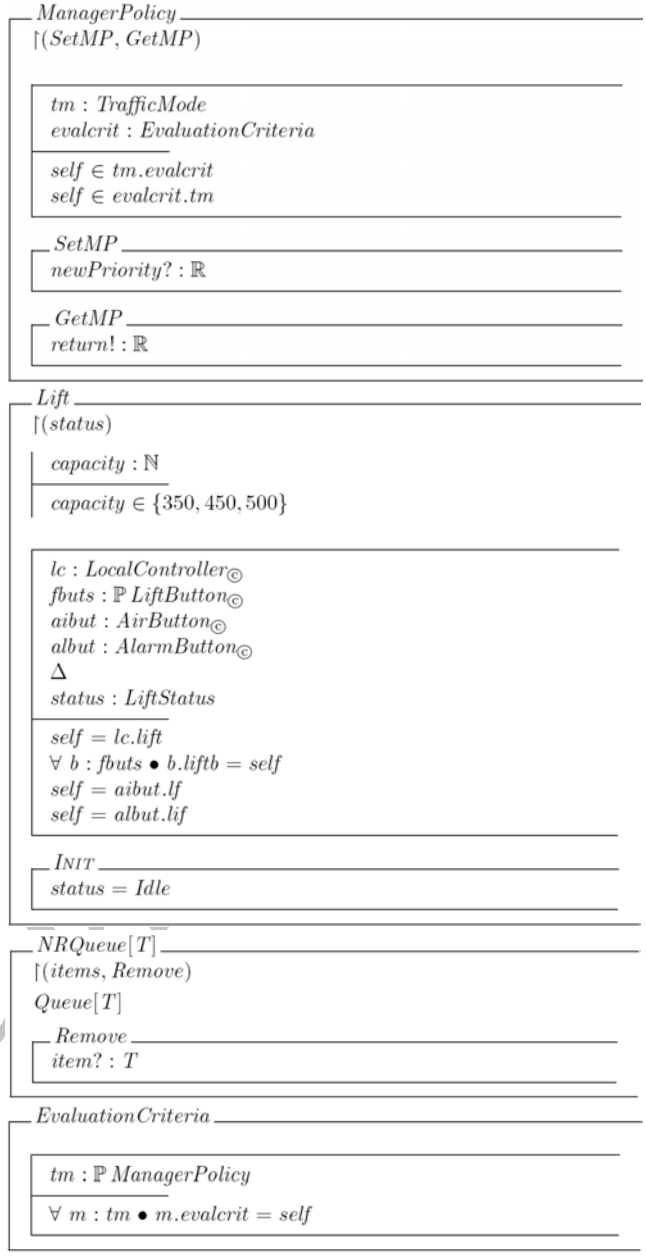
Press

TrafficMode
↾(*CalculateSuitabilityPercentage*)

evalcrit : ℙ *ManagerPolicy*	
∀ *m* : *evalcrit* • *m.tm* = *self*	

CalculateSuitabilityPercentage

tf : *TrafficFeature*	
return! : ℝ	



شکل ۱۹ نسخه اولیه نمودار کلاس سیستم آسانسور چندکابینه را که متناظر با توصیف صوری فوق‌الذکر است، نمایش می‌دهد. این نمودار کلاس از طریق اعمال قواعد تبدیل پیشنهادی بر توصیف صوری فوق‌الذکر حاصل شده است. شکل‌های ۲۰ تا ۲۳ قسمت‌های مستعد نمودار کلاس اولیه برای بازبینی را از منظر چندریختی و الگوهای طراحی رفتاری میانجی، مشاهده‌گر و استراتژی [۱۲] نشان می‌دهند. هر یک از شکل‌های مذکور دارای دو قسمت هستند: الف) قبل از بازبینی و ب) پس از بازبینی. اعمال الگوهای رفتاری مذکور و چندریختی بر مدل‌های بصری، انعطاف‌پذیری آنها را بهبود می‌بخشد.

به عنوان نمونه، کلاس *Floor* در نمودار کلاس ارائه‌شده در شکل ۱۹ بر مبنای کلاس *Floor* در توصیفات Object - Z به ترتیب به کمک قواعد شماره ۱، ۳ و ۴ از مجموعه قواعد تعریف‌شده برای تبدیل کلاس Object - Z به کلاس UML (بخش ۲-۱) تولید می‌شود. کلاس *Passenger* نیز به طور مشابه توسط قواعد ۱، ۲ و ۹ از همان مجموعه قواعد از حالت صوری به بصری تبدیل می‌شود. رابطه ترکیب موجود بین کلاس‌های کنترل‌کننده مرکزی (*CentralController*) و تخصیص‌دهنده درخواست خارجی (*ExternalRequestAllocator*) نیز به کمک قاعده تعریف‌شده در بخش ۲-۶ تولید می‌شود. سایر اجزا نیز به همین ترتیب به کمک قواعد تعریف‌شده تبدیل می‌شوند.

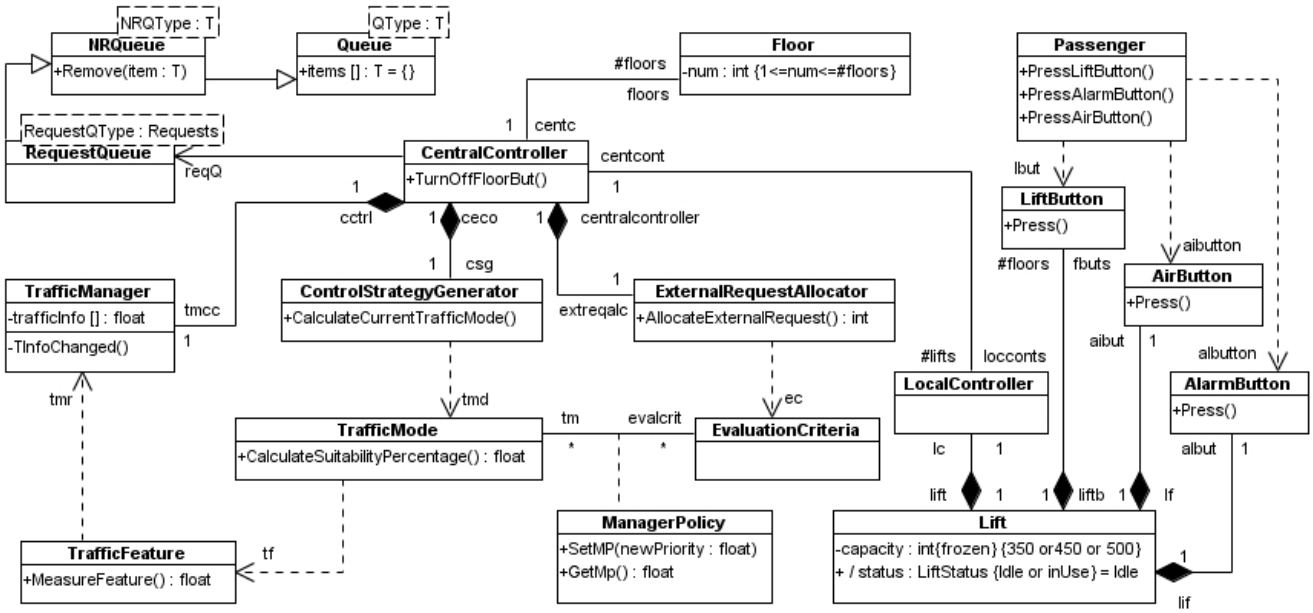
الگوهای طراحی بر روی کلاس‌های سیستم اعمال می‌شوند. بسته به سطح انتزاع، کلاس‌های سیستم یا در قالب نمودار کلاس مدل‌سازی می‌شوند یا در قالب مجموعه‌ای کد پیاده‌سازی می‌گردند. معروف‌ترین و پرکاربردترین الگوهای طراحی در [۱۲] معرفی شده‌اند. در [۲۳] الگوی طراحی معرفی شده است که هر یک به ارائه یک راه حل کلی برای حل یک مسأله کلی می‌پردازد. شایان ذکر است که برای افزایش انعطاف‌پذیری هر سیستم نرم‌افزاری، تعداد خاصی از الگوهای طراحی قابل استفاده است. به عنوان نمونه در سیستم آسانسور چندکابینه، سه مورد از پرکاربردترین و معروف‌ترین الگوهای طراحی رفتاری یعنی Mediator، Observer و Strategy قابل استفاده هستند.

همان طور که در شکل ۲۰- الف مشاهده می‌شود، کلاس *ControlStrategyGenerator* وضعیت جاری ترافیک را از طریق متُد *CalculateCurrentTrafficMode* به‌روزرسانی می‌نماید. به‌روزرسانی مذکور زمانی رخ می‌دهد که درصد مناسب‌بودن[1] یک وضعیت ترافیک تغییر کند که به نوبه خود وابسته به تغییر مقادیر مشخصه‌های ترافیک است. به‌علاوه مشخصه‌های ترافیک نیز خود وابسته به اطلاعات ترافیک هستند که توسط مدیر ترافیک مدیریت می‌شوند. با توجه به وابستگی‌های زیاد موجود بین اشیا، این قسمت کاندیدای مناسبی برای بازبینی از منظر الگوی طراحی مشاهده‌گر است. نسخه بازبینی شده در شکل ۲۰- ب مشاهده می‌شود. در الگوی مشاهده‌گر، موضوعات[2]، مشاهده‌گران خود را می‌شناسند. تعداد زیادی شیء مشاهده‌گر می‌توانند یک موضوع را مشاهده نمایند. موضوعات، واسطی را برای اضافه[3] و حذف‌شدن[4] مشاهده‌گران فراهم می‌نمایند. هر موضوعی به محض وقوع یک تغییر، کلیه مشاهده‌گران خود را که ممکن است دچار ناسازگاری شوند از طریق فراخوانی متُد Update آنها مطلع می‌سازد. سپس یک مشاهده‌گر ممکن است اطلاعاتی را از موضوع درخواست نماید. مشاهده‌گر از این اطلاعات برای هماهنگ‌سازی حالت خود با حالت شیء استفاده می‌نماید.

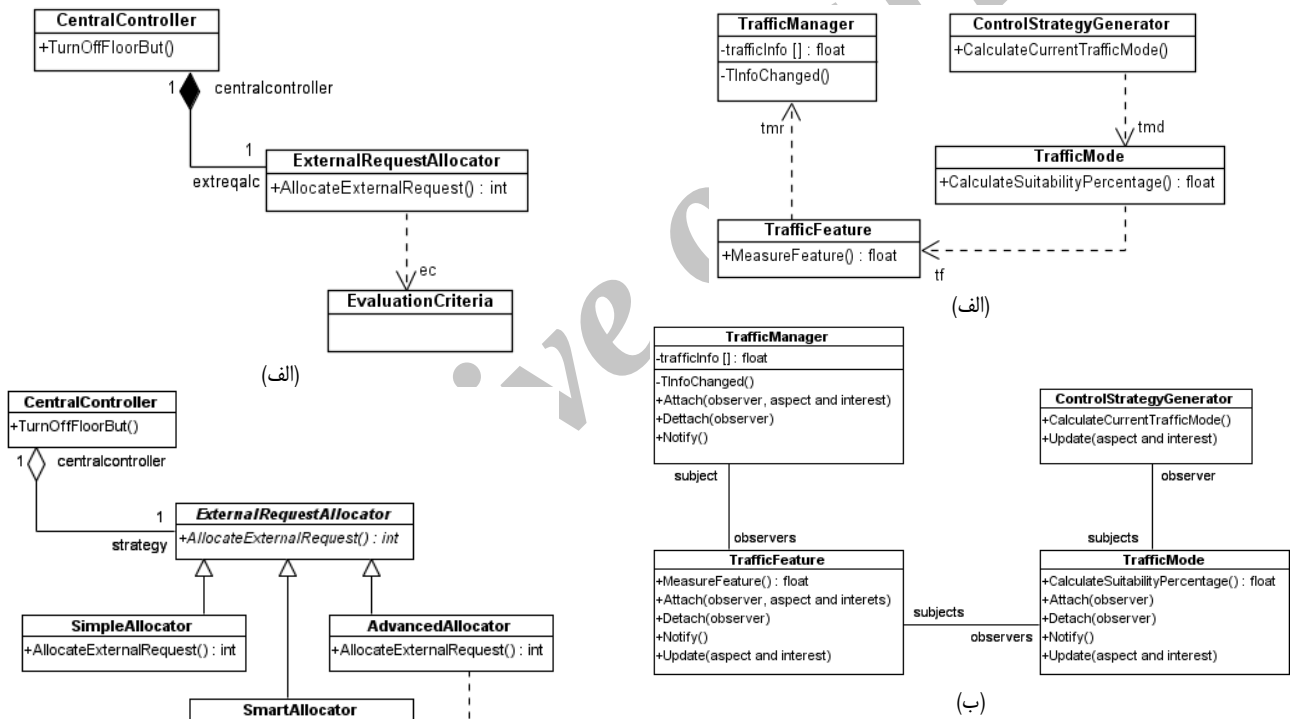
در شکل ۲۱- الف کنترل‌کننده مرکزی شامل یک تخصیص‌دهنده درخواست خارجی (کلاس *ExternalRequestAllocator*) است. وظیفه چنین تخصیص‌دهنده‌ای انتخاب مناسب‌ترین آسانسور برای پاسخ‌گویی به درخواست خارجی جاری بر اساس پارامترهایی نظیر مقدار معیارهای ارزیابی است. همان طور که پیش از این نیز بیان شد، استراتژی‌های مختلفی برای پاسخ‌گویی به درخواست‌های خارجی بر اساس متغیرهای مختلفی نظیر سیاست‌های مدیران و وضعیت جاری ترافیک وجود دارد. این استراتژی‌ها باید در زمان اجرا بر اساس مقدار متغیرهای مذکور تغییر نمایند. لذا این بخش از نمودار کلاس باید از منظر الگوی طراحی استراژی بازبینی شود. نسخه بازبینی شده در شکل ۲۱- ب مشاهده می‌شود.

نمودار ارائه‌شده شکل ۲۲- الف پیش از این توسط الگوی مشاهده‌گر بازبینی شده است. انعطاف‌پذیری نمودار مذکور را می‌توان با استفاده از الگوی میانجی باز هم افزایش داد. زمانی که رابطه وابستگی بین موضوعات و مشاهده‌گران پیچیده است، نیاز به یک شیء به نام *ChangeManager* برای مدیریت روابط مذکور است. مسؤلیت‌های

1. Suitability Percentage
2. Subjects
3. Attach
4. Detach



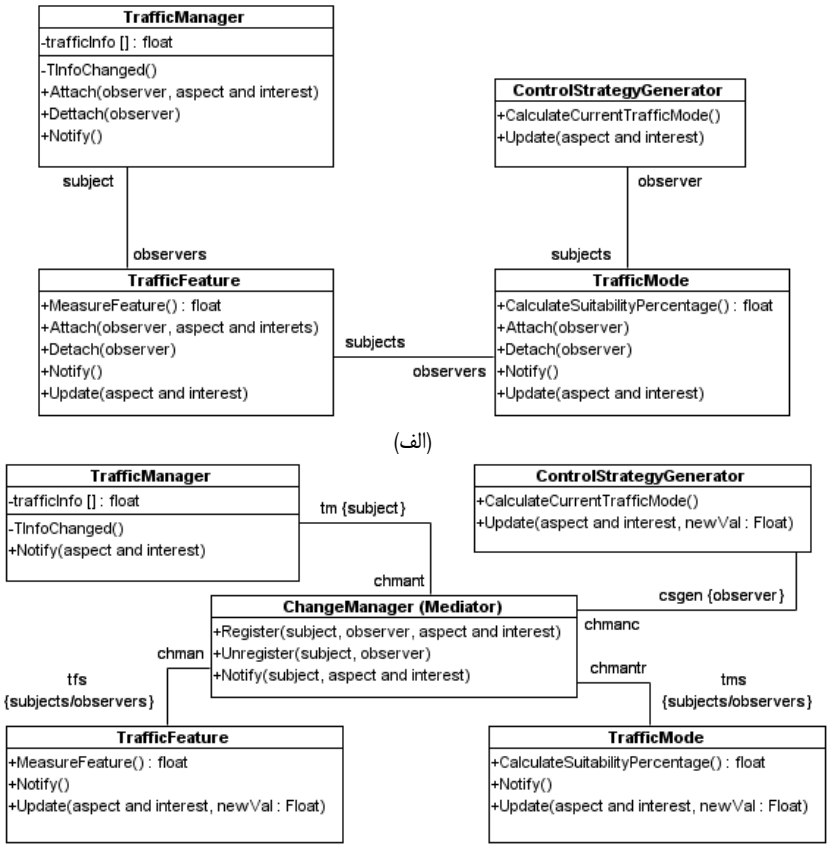
شکل ۱۹: نمودار کلاس اولیه سیستم آسانسور چندکابینه.



شکل ۲۰: اولین بازبینی نمودار کلاس بر اساس الگوی طراحی مشاهده‌گر، (الف) قبل از بازبینی و (ب) بعد از بازبینی.

ChangeManager عبارتند از: ۱) نگاشت یک موضوع به مشاهده‌گران آن و فراهم‌سازی یک واسط برای مدیریت نگاشت مذکور که موضوع و مشاهده‌گران را از نگهداری آدرس یکدیگر بی‌نیاز می‌سازد، ۲) تعریف یک استراتژی مناسب به‌هنگام‌سازی و ۳) به‌هنگام‌سازی به موقع کلیه مشاهده‌گران وابسته به تغییرات یک موضوع. *ChangeManager* نمونه‌ای از الگوی میانجی است. شکل ۲۲- ب نسخه بازبینی‌شده جدید این بخش را پس از به‌کارگیری الگوی میانجی نشان می‌دهد.

شکل ۲۳ کاربرد چندریختی را در کاهش پیوستگی بین اشیا که منجر به افزایش قابل توجه انعطاف‌پذیری می‌شود نشان می‌دهد. همان طور که پیش از این نیز بیان شد، هدف از چندریختی تحقق یک سبک برنامه‌نویسی است که در آن اشیای انواع مختلف، واسط مشترکی از عملیات را برای کاربران تعریف می‌نمایند. بدین ترتیب که یک درخواست واحد به اشیا مختلف ارسال می‌گردد، اما چند ریخت جواب دریافت می‌گردد. در این بازبینی، از بازنویسی (متُد *Press*) برای تحقق مفهوم چندریختی استفاده شده است. در بازنویسی، یک زیرکلاس (نظیر زیرکلاس‌های *LiftButton* و *AirButton*) یک یا چند متُد کلاس پدر (نظیر متُد *Press* کلاس *Button*) را مجدداً پیاده‌سازی می‌نماید. شکل ۲۳- الف قسمتی از نمودار کلاس سیستم آسانسور را که مستعد بازبینی از منظر چندریختی است، نشان می‌دهد. شکل ۲۳- ب نیز نسخه بازبینی شده را نشان می‌دهد.



شکل ۲۲: سومین بازبینی نمودار کلاس بر اساس الگوی طراحی میانجی، (الف) قبل از بازبینی و (ب) بعد از بازبینی.

شکل ۲۴ نمودار کلاس نهایی سیستم آسانسور چندکابینه را پس از اعمال الگوهای طراحی رفتاری مذکور و چندریختی بر نسخه اولیه نشان می‌دهد. شایان ذکر است که کلیه مراحل میانی بازبینی نمودار کلاس سیستم آسانسور چندکابینه به‌صورت کامل و مفصل در [۹] ارائه شده است.

نسخه نهایی توصیف صوری سیستم آسانسور چندکابینه بر مبنای نمودار کلاس فوق‌الذکر با استفاده از مکانیزم تبدیل پیشنهادی به شرح زیر است:

LocalController

$centcont : CentralController$
$lift : Lift$

$self \in centcont.locconts$
$self = lift.lc$

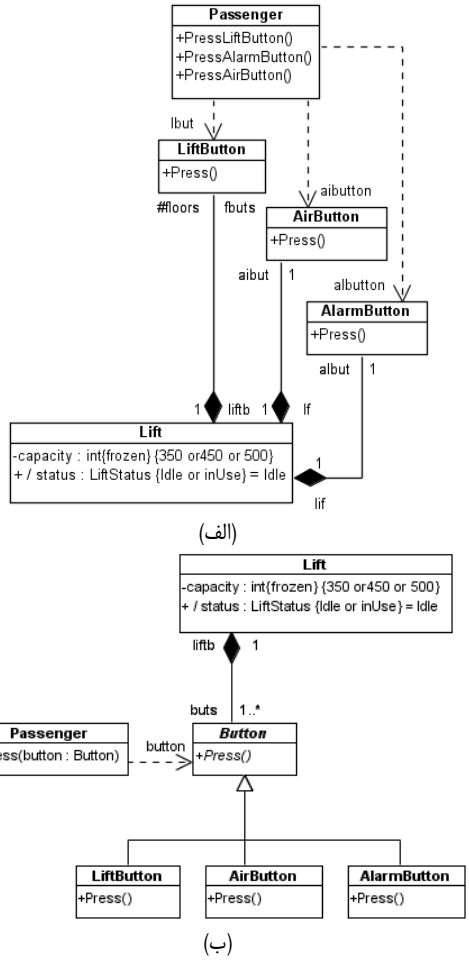
CentralController

$\upharpoonright (TurnOffFloorBut)$

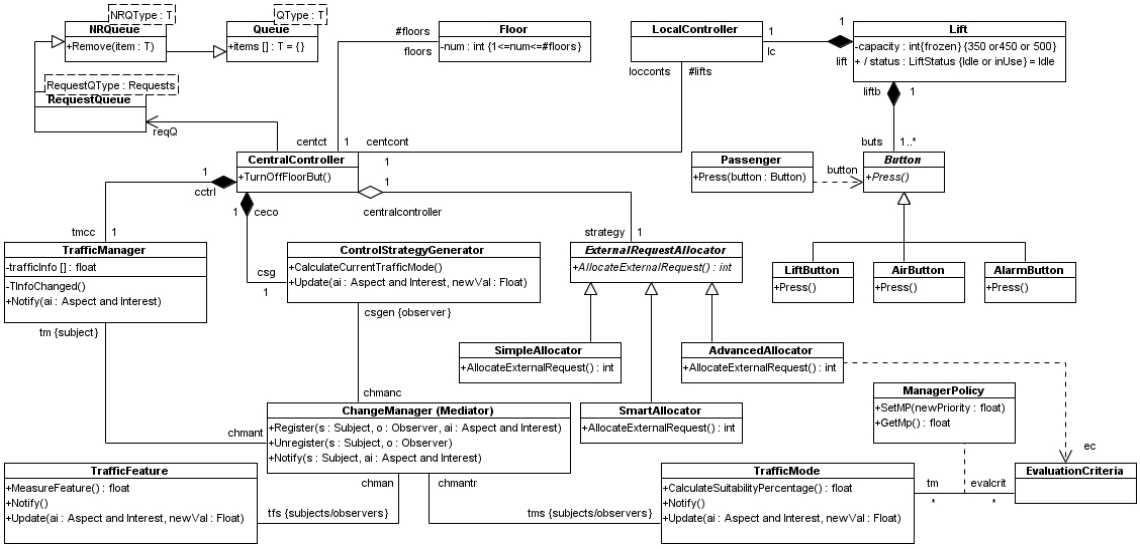
$tmcc : TrafficManager_{©}$
$locconts : \mathrm{seq}\ LocalController$
$reqQ : RequestQueue$
$strategy :\downarrow ExternalRequestAllocator_{Ⓢ}$
$csg : ControlStrategyGenerator_{©}$
$floors : \mathrm{seq}\ Floor$

$\#locconts = \#floors$
$tmcc.cctrl = self$
$\forall\ l : \mathrm{ran}\ locconts \bullet l.centcont = self$
$strategy.centralcontroller = self$
$\forall\ f : \mathrm{ran}\ floors \bullet f.centc = self$
$csg.ceco = self$

TurnOffFloorBut



شکل ۲۳: چهارمین بازبینی نمودار کلاس بر اساس چندریختی، (الف) قبل از بازبینی و (ب) بعد از بازبینی.



شکل ۲۴: نمودار نهایی کلاس سیستم آسانسور چندکابینه.

قسمت‌هایی از توصیف صوری فوق‌الذکر با علائم ⊗، * و – نشانه‌گذاری شده است. اولین علامت یعنی ⊗ زمانی استفاده می‌شود که مورد یا مواردی نسبت به نسخه اولیه توصیف صوری از نسخه جاری حذف شده باشد. دومین علامت یعنی * در محل اضافه‌شدن قسمت‌های جدید به نسخه اولیه استفاده شده است. علامت آخر یعنی – نشان‌دهنده مواردی است که نسبت به نسخه اولیه دچار تغییراتی شده‌اند. بدین ترتیب، قابلیت استفاده از مکانیزم تبدیل پیشنهادی توسط مطالعه موردی سیستم آسانسور چندکابینه امکان‌سنجی می‌شود.

٤- کارهای مرتبط

تمرکز اصلی این مقاله بر نقش مدل‌سازی و تبدیل مدل در چرخه توسعه نرم‌افزار به منظور تولید نرم‌افزار باکیفیت است. تاکنون تلاش‌های زیادی در زمینه تبدیل مدل با هدف‌های مختلف نظیر افزایش قابلیت استفاده مدل‌های صوری و افزایش دقت روش‌های بصری صورت پذیرفته است [۵] تا [۷]. توماس تیلی در [۱۳] تلاش می‌کند که قابلیت استفاده از زبان توصیف Z را از طریق نمودارهای خطی[1] که نمایانگر تحلیل مفهوم صوری[2] هستند، افزایش دهد. این رساله همچنین برای پشتیبانی ابزاری

1. Line Diagrams
2. Formal Concept Analysis

جدول ۱: مهم‌ترین کارهای مرتبط در زمینه تبدیل مدل.

کارهای مرتبط / ویژگی‌ها		مدل منبع	مدل مقصد	جهت تبدیل	هدف (اهداف)
تبدیل از مدل‌های صوری به مدل‌های بصری	[۷۰]	توصیف B	نمودار حالت	یک‌جهته	افزایش دقت نمودار حالت
	[۱۳]	توصیف Z	تحلیل مفهوم صوری	یک‌جهته	افزایش قابلیت استفاده از Z
	[۴]، [۱۸] و [۱۹]	توصیف Z	نمودارهای UML	یک‌جهته	افزایش قابلیت استفاده از Z
	[۵۱]	توصیف Z	نمودار حالت	دوجهته	طراحی سیستم‌های کنترلی حیاتی- ایمنی
	[۱۴]	توصیف Object - Z	نمودارهای کلاس و حالت	یک‌جهته	توصیف و طراحی کارکردهای کلیدی محیط وب با استفاده از Object - Z و بازنمایی بصری در UML به منظور افزایش قابلیت استفاده از Object - Z
تبدیل از مدل‌های بصری به مدل‌های صوری	[۶۳] و [۶۵]	نمودارهای OMT	توصیف B	یک‌جهته	افزایش دقت نمودارهای OMT
	[۷۸]	نمودارهای OMT	توصیف Object - Z	دوجهته	توصیف دقیق و قابل فهم
	[۶۰]، [۶۲]، [۶۸]، [۶۹]، [۷۱] و [۸۱]	نمودارهای UML	توصیف B	یک‌جهته	افزایش دقت نمودارهای UML و پشتیبانی از قابلیت ردیابی در توسعه سیستم‌های اطلاعاتی حساس
	[۲۷]، [۲۹] تا [۳۱]، [۴۸] و [۵۴]	نمودارهای UML	توصیف Z	یک‌جهته	فراهم‌سازی بستر صوری برای جنبه‌های مختلف UML در Z
	[۷۹]، [۲۰] تا [۲۲]، [۲۸]، [۸۰] و [۸۲]	نمودارهای UML	توصیف Object - Z	یک‌جهته	افزایش دقت نمودارهای UML

[۴]، [۱۴]، [۱۵] تا [۱۷]، به‌عنوان راهکاری دیگر برای افزایش قابلیت استفاده و پذیرش روش‌های صوری Z مانند، نمونه اولیه ابزاری را برای بازنمایی بصری توصیفات Z بر مبنای تحلیل مفهوم صوری ارائه می‌نماید. تاکنون تلاش‌های زیادی به منظور بازنمایی بصری توصیفات Z از طریق UML صورت پذیرفته است [۴]. سان، دونگ و وانگ یک بازنمایی XML [۱۸] برای زبان‌های خانواده Z با نام ZML ارائه می‌نمایند [۱۴]. ZML قابل تبدیل به UML است. نمونه مشابه دیگر، کارهای کارینگتن و کیم است [۱۹] تا [۲۲]. بسیاری از رویکردهای پیشنهادی بر جنبه ساختاری توصیف تمرکز دارند [۲۳]. کیم و کارینگتن [۱۹] معتقدند که فراتر از ساختار یک توصیف، رفتار آن نیز باید جهت درک کامل یک توصیف به‌صورت بصری بازنمایی شود. برای این منظور آنها دو بازنمایی گرافیکی دیگر علاوه بر UML، یکی برای قیدهای پیچیده و دیگری برای شِماهای عملیات ارائه نمودند.

استفاده از روش‌های صوری در تجمیع[1] به تدریج تکامل یافته است [۲۴] تا [۲۶]. در ابتدا تجمیع با هدف دست‌یابی به معنای دقیق انجام می‌شد اما تمرکز رویکردهای تجمیع با پیشرفت ابزارهای پشتیبان روش‌های صوری به سمت استفاده از روش‌های صوری در تحلیل و کنترل سازگاری مدل‌های بصری سوق پیدا کرد. ایوانز و کلارک [۲۷] و مایو، لیو و لی [۲۸] Z و UML را با یکدیگر ترکیب کرده‌اند. در کارهای ترکیبی مذکور به جای بازنمایی بصری توصیفات Z از طریق UML، تمرکز بر فراهم‌سازی یک بستر صوری برای جنبه‌های مختلف UML در Z است. Alloy، یک روش صوری سبک‌وزن[2] مرتبط با Z شامل هر دو نوع مؤلفه گرافیکی و متنی است که یک نگاشت مستقیم از UML به یک نماد صوری ارائه می‌نماید [۲۹] تا [۳۱]. روش مذکور، سبک‌وزن است زیرا رویکردی کاملاً صوری به توصیف، اعتبارسنجی و تست ارائه نمی‌نماید [۳۱] و [۳۲]. هیچ کدام از روش‌های پیشنهادی، UML را به طور کامل صوری نمی‌کنند بلکه بر روی زیرمجموعه محدودی از UML تمرکز می‌نمایند. اغلب روش‌های مذکور حتی همه ویژگی‌های یک نمودار را نیز صوری نمی‌سازند، بلکه بر روی آن دسته از ویژگی‌ها که برای کاربرد خاصی مورد نیاز هستند، تمرکز می‌نمایند. روش‌های ترکیبی با پیشرفت روش‌های مدل‌سازی صوری و بصری تکامل یافته‌اند. تمرکز روش‌های ترکیبی در ابتدا بر روی رویکرد ساخت‌یافته، سپس بر روی رویکرد شیءگرا و اخیراً به طور ویژه بر روی UML است. تاکنون رویکردهای زیادی برای تجمیع روش‌های ساخت‌یافته نظیر Yourdon و SSADM با زبان‌های مدل‌سازی صوری نظیر VDM [۳۳] تا [۳۷]، Z [۳۸] تا [۴۱]، B [۴۲]، CCS [۴۳] و زبان‌های جبری [۴۴] ارائه شده است. گزارش کاملی از فعالیت‌های مرتبط در [۴۵] آمده است. رویکردهای تجمیع ارائه‌شده برای روش‌های ساخت‌یافته هم باید به صوری‌سازی مفاهیم گرافیکی می‌پرداختند و هم به هماهنگ‌سازی آنها با روش‌های ساخت‌یافته که این حجم زیاد کار بر کیفیت صوری‌سازی اثر منفی گذاشته و غالباً آنها را به کارهایی سطحی بدل کرده است.

رویکردهای ترکیبی روش‌های شیءگرا به دنبال صوری‌سازی مفاهیم بنیادی شیءگرایی و تجمیع با روش‌های شیءگرا هستند. تاکنون تلاش‌های زیادی برای تجمیع روش‌های شیءگرا با Z [۴۶] تا [۵۷]، Object - Z، B [۵۸] تا [۷۱] و CSP [۷۲] تا [۷۵] صورت پذیرفته است. در اینجا با توجه به اهداف این مقاله در ترکیب Object - Z و UML، کارهای مرتبطی که تاکنون در این زمینه انجام شده است را با جزئیات بیشتر مورد بررسی قرار می‌دهیم:

- Metamorphosis روشی است که Object - Z را با ویژگی‌های مشترک روش‌های شیءگرایی مجتمع می‌سازد [۷۶]. این روش شامل قواعدی برای تبدیل مدل‌های پویا و ایستا است.
- روش آچاتز و شُلت Fusion و Object - Z را با هم ترکیب می‌نماید [۷۷].
- دوپوی مدل‌های کلاس شیءگرا و نمودارهای حالت را در Object - Z صوری می‌نماید [۵۷] و [۷۸].
- کیم و کارینگتن نمودارهای کلاس [۷۹] و [۲۰] و حالت [۲۲] و [۸۰] UML را در Object - Z صوری می‌نمایند. با این وجود، این کار کلیه ویژگی‌های نمودارهای کلاس و حالت را صوری نمی‌سازد.

جدول ۱ کارهای مرتبطی را که به رویکرد پیشنهادی ما نزدیک‌تر هستند و ارجاع بیشتری به آنها صورت پذیرفته است را ارائه می‌نماید. کارهای مرتبط انتخابی به دو گروه عمده تقسیم می‌شوند: ۱) کارهایی که

1. Integration
2. Lightweight

جدول ۲: مقایسه رویکرد پیشنهادی با مشابه‌ترین کارهای مرتبط موجود.

کارهای مرتبط / معیارهای ارزیابی	[۸۲]	[۱۴]	[۷۹]	مکانیزم پیشنهادی جدید
مدل منبع	توصیف Object - Z	توصیف Object - Z	توصیف Object - Z	توصیف Object - Z
مدل مقصد	نمودار کلاس UML	نمودار کلاس UML	نمودار کلاس UML	نمودار کلاس UML
جهت تبدیل	یک‌جهته	یک‌جهته	یک‌جهته	دوجهته
مکانیزم تبدیل	قواعد نادقیق و غیر صوری	قواعد نادقیق و غیر صوری	نگاشت صوری	قواعد ساخت‌یافته
روش ارزیابی	مطالعه موردی جزئی	مطالعه موردی جزئی	در دسترس نمی‌باشد	مطالعه موردی غیر جزئی
ویژگی‌های تحت پوشش	صفات اولیه و ثانویه، ثابت‌ها، عملیات و مقداردهی اولیه درون کلاس، وراثت، رابطه انجمنی یک‌جهته و دوجهته و تجمع	صفات و عملیات درون کلاس و وراثت	ویژگی‌های مدل‌سازی مشترک Object - Z و نمودار کلاس به جز ثابت‌ها، انواع تعریف‌شده توسط کاربر و مقداردهی اولیه درون کلاس، وراثت عمومی، تجمع، وابستگی و چندریختی	همه ویژگی‌های مدل‌سازی مشترک Object - Z و نمودار کلاس UML که در بخش ۲ ذکر شده‌اند
هدف (اهداف)	افزایش قابلیت استفاده از Object - Z	توصیف و طراحی کارکردهای کلیدی محیط وب توسط Object - Z و افزایش قابلیت توسط Object - Z	افزایش دقت نمودارهای UML	فراهم‌سازی بستر لازم برای بهره‌مندی توأمان از مزایای UML و Object - Z

مدل‌های صوری در B، Z و Object - Z را به مدل‌های بصری در FCA و UML تبدیل می‌نمایند و ۲) کارهایی که مدل‌های بصری در OMT و UML را به مدل‌های صوری در B، Z و Object - Z تبدیل می‌نمایند. در جدول ۱ کارهای مذکور از منظر مدل منبع، مدل مقصد، جهت تبدیل (یک‌جهته یا دوجهته) و هدف مورد بررسی قرار گرفته‌اند. تبدیل یک‌جهته فقط می‌تواند در یک جهت عمل نماید. بنابراین مدل منبع را به مدل مقصد تبدیل می‌نماید. در تبدیل یک‌جهته مدل مقصد به مدل منبع قابل تبدیل نیست. تبدیل دوجهته در هر دو جهت می‌تواند عمل تبدیل را انجام دهد.

۵- نتیجه‌گیری و کارهای آینده

کاربرد گسترده روش‌های مدل‌سازی بصری در فرایند توسعه نرم‌افزار به دلیل فراهم‌سازی بستر مناسب برای به‌کارگیری تکنیک‌های ابتکاری مهندسی نرم‌افزار، منجر به تولید نرم‌افزارهای باکیفیت (مثلاً از منظر انعطاف‌پذیری و قابلیت استفاده مجدد) می‌گردد. با این وجود، روش‌های مدل‌سازی بصری رویکرد دقیقی به توسعه نرم‌افزار به‌ویژه در مواردی که وجود قابلیت اعتماد به طور حیاتی ضروری است، ندارند. از جمله سیستم‌هایی که نیاز حیاتی به قابلیت اعتماد دارند، عبارتند از: سیستم‌های فوق صحیح و ایمنی- حیاتی. معنای روش‌های مدل‌سازی بصری دقیق نیست. روش‌های مدل‌سازی صوری قابلیت توصیف و صحت‌سنجی نرم‌افزار را با استفاده از منطق ریاضی دارند. معنای دقیق آنها، امکان طراحی مدل‌های دقیق از سیستم را فراهم می‌نماید. با این وجود، کاربرد آنها به دلیل کمبود تخصص و هزینه‌های زیاد محدود است. تحقیقاتی که تاکنون بر روی روش‌های ترکیبی صورت پذیرفته است، نشان می‌دهند که: ۱) روش‌های صوری و بصری مدل‌سازی را می‌توان در فرایند توسعه نرم‌افزار به صورت ترکیبی مورد استفاده قرار داد به قسمی که مکمل یکدیگر باشند، ۲) این همزیستی مفید است و فواید بسیاری را به همراه دارد و ۳) تبدیل مدل‌های صوری (نظیر توصیف Object - Z) و بصری (نظیر مدل‌های UML) به یکدیگر کاری جزئی و ساده نیست.

این مقاله یک مکانیزم دوجهته مبتنی بر قاعده را برای تبدیل نمودار کلاس UML و توصیف Object - Z به یکدیگر ارائه می‌نماید. در واقع، مکانیزم پیشنهادی گامی است در جهت بهره‌مندی توأمان از مزیت‌های هر دو دسته روش‌های مدل‌سازی صوری و بصری که در نهایت مقدمات تولید نرم‌افزار باکیفیت مطلوب را فراهم می‌نماید. مطالعه موردی سیستم آسانسور چندکابینه نیز قابلیت استفاده از مکانیزم مذکور را به صورت عملی ارزیابی می‌نماید. جدول ۲ رویکرد پیشنهادی جدید را با سه مورد از مشابه‌ترین کارهای موجود از منظر هفت معیار ارزیابی مقایسه می‌نماید.

در این مقاله، رویکرد پیشنهادی به صورت غیر خودکار در چرخه توسعه نرم‌افزار مورد استفاده قرار می‌گیرد. به علاوه، درستی و کارایی رویکرد مذکور با استفاده از مطالعه موردی سنجیده می‌شود. در آینده ما درصدد خودکارسازی مکانیزم مذکور هستیم. ما همچنین قصد داریم که صحت و کارایی رویکرد پیشنهادی را در برآوردن اهداف مورد انتظار به صورت صوری و ریاضی اثبات کنیم. برای این منظور، قواعد تبدیل باید به صورت صوری بین فرا-مدل دو زبان مبدأ و مقصد (UML و Object-Z) تعریف شوند [۲۲] و [۸۰]. همچنین باید توجه داشت که UML ۲/۰ دارای سیزده نمودار است که در کنار یکدیگر به مدل‌سازی جنبه‌های مختلف ساختار و رفتار سیستم می‌پردازند. در پژوهش جاری فقط به تبدیل نمودار کلاس UML پرداخته‌ایم. در آینده لازم است برای تبدیل سایر نمودارها نیز قواعد مشخصی تعریف گردد.

مراجع

[1] D. C. Schmidt, "Model-driven engineering," *IEEE Computer*, vol. 39, no. 2, pp. 25-31, Feb. 2006.

[2] Q. Charatan and A. Kans, *Formal Software Development: from VDM to Java*, Palgrave Macmillan, 2004.

[3] D. Bjorner, Software Engineering III: Domains, Requirements, and Software Design, Springer, 2006.

[4] J. R. Williams, *Automatic Formalization of UML to Z*, MSc. Thesis, Dept. Computer Science, Univ. York, 2009.

[5] A. Hall, "Seven myths of formal methods," *IEEE Software*, vol. 7, no. 5, pp. 11-19, Sep. 1990.

[6] J. Bowen and M. Hinchey, "Seven more myths of formal methods," *IEEE Software*, vol. 12, no. 4, pp. 34-41, Jul. 1995.

[7] D. Bojic and D. Velasevic, "Reverse engineering of use case realizations in UML," *in Proc. Symp. on Applied Computing - SAC2000, ACM*, vol. 2, pp. 741-747, 2000.

[8] A. Rasoolzadegan and A. Abdollahzadeh, "A new approach to software development process with formal modeling of behavior

[36] J. Dick and J. Loubersac, "Integrating structured and formal methods: a visual approach to VDM," in *Proc. ESEC'91: European Softw. Eng. Conf.*, vol. 550 of LNCS, pp. 37-59, Milan, Italy, 1991.
[37] V. Hamilton, "Experience of combining Yourdon and VDM," in *Proc. the Methods Integration Workshop*, pp. 529-555, 1991.
[38] L. T. Semmens and P. Allen, "Using Yourdon and Z: an approach to formal specification," in *Proc. Fifth Annual Z User Meeting, Oxford, Springer*, pp. 228-253, 1991.
[39] F. Polack, M. Whiston, and K. C. Mander, "The SAZ project: integrating SSADM and Z," in *Proc. FME'93: Industrial-Strength Formal Methods*, vol. 670 of LNCS, pp. 541-557, 1993.
[40] F. Polack, "SAZ: SSADM version 4 and Z," in *Proc. Software Specification Methods: An Overview Using a Case Study*, pp. 21-38, 21-38, London, UK, 2001.
[41] K. C. Mander and F. Polack, "Rigorous specification using structured systems analysis and Z," *Information and Software Technology*, vol. 37, no. 5, pp. 285-291, 1995.
[42] N. Nagui - Raiss, "A formal software specification tool using the entity - relationship model," in *Proc. ER'94: Entity - Relationship Approach*, vol. 881 of LNCS, pp. 316-332, 1994.
[43] A. Galloway, Integrated Formal Methods with Richer Methodological Profiles for the Development of Multi - Perspective Systems, Ph. D Thesis, University of Teesside, School of Computing and Mathematics, 1996.
[44] R. B. France, "Semantically extended data flow diagrams: a formal specification tool," *IEEE Trans. on Softw. Eng.*, vol. 18, no. 4, pp. 329-346, Apr. 1992.
[45] L. T. Semmens, R. B. France, and T. W. G. Docker, "Integrated structured analysis and formal specification techniques," *The Computer Journal*, vol. 35, no. 6, pp. 600-610, 1992.
[46] D. Harel, "Statecharts: a visual formalism for complex systems," *Science of Computer Programming*, vol. 8, no. 3, pp. 231-274, 1987.
[47] A. Hall, "Using Z as a specification calculus for object - oriented systems," in *Proc. VDM'90*, vol. 428 of LNCS, pp. 290-318, Dublin, Ireland, 1990.
[48] A. Hall, "Specifying and interpreting class hierarchies in Z," in *Proc. Z User Workshop, Cambridge, Workshops in Computing*, pp. 120-138, 1994.
[49] J. Hammond, "Producing Z specifications from object - oriented analysis," in *Proc. Z User Workshop, Cambridge, Workshops in Computing, Springer*, pp. 316-336, 1994.
[50] M. Rawson and P. Allen, "Synthesis: an integrated, object - oriented method and tool for requirements specification," in *Proc. Method Integration Workshop, Leeds, UK, Electronic Workshops in Computing (eWIC*), 320-334, Leeds, UK, 1996.
[51] M. Weber, "Combining statecharts and Z for the design of safety - critical control systems," in *Proc. FME'96: 3rd Int. Symposium of Formal Methods Europe*, vol. 1051 of LNCS, pp. 307-326, Oxford, UK, 1996.
[52] R. B. France, J. M. Bruel, and G. Raghavan, "Towards rigorous analysis of fusion models: the MIRG experience," in *Proc. 2nd Northern Formal methods Workshop, Electronic Workshops in Computing, British Computer Society*, 280-289, Ilkley, UK, 1996.
[53] R. B. France and M. M. Larrondo - Petrie, "An integrated object - oriented and formal model environment," *J. of Object-Oriented Programming*, vol. 10, no. 7, pp. 25-34, 1997.
[54] R. B. France, E. Grant, and J. M. Bruel, *UMLtranZ: An UML - Based Rigorous Requirements Modeling Technique*, Tech. Rep., Colorado State University, 2000.
[55] R. B. France, J. M. Bruel, M. M. Larrondo - Petrie, and E. Grant, "Rigorous object - oriented modeling: integrating formal and informal notations," in *Proc. Algebraic Methodology and Software Technology, AMAST'97*, vol. 1349 of LNCS, pp. 216-230, Munich, Germany, 1997.
[56] J. M. Bruel and R. B. France, "Transforming UML models to formal specifications," in *Proc. UML'98: Beyond the Notation*, vol. 1618 of LNCS, Springer, pp. 165-18, Mulhouse, France, 1998.
[57] S. Dupuy, Couplage de Notations Semi - Formelles et Formelles Pour la Specification des Systemes D'information, Ph.D. Thesis, Universite Joseph Fourier, Grenoble I, 2000.
[58] K. Lano and H. Haughton, "Improving the process of system specification and development in B," in *Proc. 6th Refinement Workshop, Workshops in Computing*, pp. 42-56, Oxford, UK, 1994.
[59] K. Lano, H. Haughton, and P. Wheeler, "Integrating formal and structured methods in object - oriented system development," *Formal Methods and Object Technology*, pp.113-157, 1996.
[60] P. Facon, R. Laleau, and H. P. Nguyen, "Mapping object diagrams into B specifications," in *Proc. Method Integration Workshop, UK, Electronic Workshops in Computing (eWIC)*, pp. 1-13, 1996.
based on visualization," in *Proc. 6th Int. Conf. on Softw. Eng. Advances, ICSEA*, pp. 104-111, Barcelona, Spain, 2011.
[9] A. Rasoolzadegan and A. Abdollahzadeh, *Specifying a Parallel, Distributed, Real-Time, and Embedded System: Multi-Lift System Case Study*, Tech. Rep., Information Technology and Computer Eng. Faculty, Amirkabir Univ. Technology, Tehran, Iran, 2011.
[10] J. Kong, K. Zhang, J. Dong, and D. Xu, "Specifying behavioral semantics of UML diagrams through graph transformations," *The J. of Systems and Software*, vol. 82, no. 2, pp. 292-306, Apr. 2009.
[11] R. Duke and G. Rose, Formal Object - Oriented Specification Using Object - Z, MacMillan Press, 2000.
[12] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Pattern: Elements of Reusable Object - Oriented Software*, Addison - Wesley Publishing Company, 5th Ed., 1995.
[13] T. Tilley, Formal Concept Analysis Applications to Requirements Engineering and Design, Ph. D. Dissertation, the Univ. Queensland, Australia, 2004.
[14] J. Sun, J. S. Dong, J. Liu, and H. Wang, "Object - Z web environment and projections to UML," in *Proc. 10th Int. World Wide Web Conf., New York, ACM*, pp. 725-734, 2001.
[15] J. Bowen (2003) frequently asked questions, [Online], Available: http://www.faqs.org/faqs/z-faq.
[16] J. Bowen (2003) The World Wide Web virtual library: The Z notation, Available: http://www.zuser.org/z.
[17] J. Sun, J. S. Dong, J. Liu, and H. Wang, "A formal object approach to the design of ZML," *Annals of Software Engineering: an International J.*, vol. 13, no. 14, pp. 329-356, 2002.
[18] Extensible Markup Language-XML (2003) World Wide Web Consortium, Available: http://www.w3.org/XML.
[19] S. K. Kim and D. Carrington, "Visualization of formal specifications," in *Proc. 6th Asia - Pacific Softw. Eng. Conf.*, pp. 38-45, Dec. 1999.
[20] S. K. Kim and D. Carrington, "Formalizing the UML class diagram using Object - Z," in *Proc. the Second IEEE Conf. on UML, UML'99*, pp. 83-98, 1999.
[21] S. K. Kim and D. Carrington, "An integrated framework with UML and Object - Z for developing a precise and understandable specification: the light control case study," in *Proc. Seventh Asia - Pacific Software Engineering Conf.*, pp. 240-248, 2000.
[22] S. Kim and D. Carrington, "A formal meta - modeling approach to a transformation between the UML state machine and Object-Z," in *Proc. ICFEM 2002: Int. Conf. Formal Eng. Methods*, pp. 548-560, Shanghai, China, 2002.
[23] E. Wafula and P. Swatman, "FOOM: a diagrammatic illustration of inter-object communication in Object-Z specifications," in *Proc. the Asia-Pacific Software Engineering Conf., APSEC'95*, 23 pp., 1995.
[24] S. German, "Research goals for formal methods," *ACM Computing Surveys*, vol. 28, no. 4es, p. 118. 1996.
[25] E. Borger, "High level system design using abstract state machines," in *Proc. Current Trends in Applied Formal Methods (FM - Trends 98), Springer LNCS 1641*, pp. 1-43, 1998.
[26] E. Clarke and J. Wing, "Formal methods: state of the art and future directions," *ACM Computing Surveys*, vol. 28, no. 4, pp. 626-643, Dec. 1996.
[27] A. Evans, R. France, K. Lano, and B. Rumpe, "The UML as a formal modeling notation," in *Proc. UML'98: Beyond the Notation*, vol. 1618 of LNCS, Springer, pp. 336-348, Mulhouse, France, 1998.
[28] H. Miao, L. Liu, and L. Li, "Formalizing UML models with Object-Z," in *Proc. ICFEM2002: Conf. on Formal Eng. Methods*, pp. 523-534, 2002.
[29] D. Jackson, "A comparison of object modeling notations: alloy, UML and Z," unpublished manuscript, [Online], Available: http://sdg.lcs.mit.edu/~dnj/pubs/alloy-comparison.pdf, 1999.
[30] D. Jackson, I. Schechter, and I. Shlyakhter, "Alcoa: the alloy constraint analyzer," in *Proc. of the Int. Conf. on Software Engineering*, pp. 730-733, Limerick, Ireland, 2000.
[31] D. Jackson, Software Abstractions: Logic, Language, and Analysis, MIT Press, 2006.
[32] S. Agerholm and P. Larsen, "A lightweight approach to formal methods," in *Proc. the Int. Workshop on Current Trends in Applied Formal Methods*, pp. 168-183, 1998.
[33] P. G. Larsen, J. V. Katwijk, N. Plat, K. Pronk, and H. Toetenel, "SVDM: an integrated combination of SA and VDM," in *Proc. Methods Integration Workshop*, 1991.
[34] N. Plat, J. V. Karwijk, and K. Pronk, "A case for structured analysis/formal design," in *Proc. Formal Software Development Methods, VDM'91*, vol. 552 of LNCS, pp. 81-105, 1991.
[35] M. D. Fraser, K. Kumar, and V. K. Vaishnavi, "Informal and formal requirements specification languages: bridging the gap," *IEEE Trans. on Softw. Eng.*, vol. 17, no. 5, pp. 454-465, May 1991.

[61] A. Malioukov, "An object - based approach to the B formal method," in *Proc. B'98: Int. B Conf.*, vol. 1393 of LNCS, pp. 162-181, Montpellier, France, 1998.
[62] H. Nguyen, *Derivation De Specifications Formelles B a partir de Specifications Semi-Formelles*, Ph.D. Thesis, Laboratoire CEDRIC, Conservatoire National des Arts et Metiers, Evry, France, 1998.
[63] E. Meyer and J. Souquiμeres, "Systematic approach to transform OMT diagrams to a B specification," in *Proc. FM'99*, vol. 1708 and 1709 of LNCS, pp. 875-895, Toulouse, France, 1999.
[64] C. Snook and M. Butler, Tool - supported use of UML for constructing B specifications, [Online], Available: http://www.ecs.soton.ac.uk/~mjb/, 2000.
[65] P. Facon, R. Laleau, and H. P. Nguyen, "From OMT diagrams to B specifications," in *Proc. Softw. Spec. Methods: an Overview Using a Case Study*, pp. 57-77, Potsdam, German, 2001.
[66] R. Laleau and F. Polack, "A rigorous metamodel for UML static conceptual modeling of information systems," in *Proc. CAiSE 2001: Advanced Information Systems Eng.*, vol. 2068 of LNCS, pp. 402-416, Interlaken, Switzerland, 2001.
[67] R. Laleau and F. Polack, "Specification of integrity - preserving operations in information systems by using a formal UML - based language," *Information and Software Technology*, vol. 43, no. 12, pp. 693-704, 2001.
[68] R. Laleau and F. Polack, "Coming and going from UML to B: a proposal to support traceability in rigorous IS development," in *Proc. ZB 2002: Formal Specification and Development in Z and B*, vol. 2272 of LNCS, pp. 517-534, Grenoble, France, 2002.
[69] H. Treharne, "Supplementing a UML development process with B," in *Proc. FME 2002: Formal Methods-Getting it Right*, vol. 2391 of LNCS, pp. 568-586, Copenhagen, Denmark, 2002.
[70] A. Hammad, B. Tatibouet, J. Voisinet, and W. Weiping, "From B specification to UML statechart diagrams," in *Proc. ICFEM 2002: Int. Conf. of Formal Engineering Methods*, vol. 2495 of LNCS, pp. 511-522, Shanghai, China, 2001.
[71] C. Snook and M. Butler, "UML - B: formal modeling and design aided by UML," *ACM Trans. Softw. Eng. Methodol*, vol. 15, no. 1, pp. 92-122, 2006.
[72] B. Selic and J. Rumbaugh, "Using UML for modeling complex real - time systems," in *Proc. the ACM SIGPLAN Workshop on Languages, Compilers, and Tools for Embedded Systems, LCTES'98)*, pp. 250-260, Montreal, Canada, 1998.
[73] C. Fischer, E. Olderog, and H. Wehrheim, "A CSP view on UML - RT structure diagrams," in *Proc. Fundamental Approaches to Softw. Eng.*, vol. 2029 of LNCS, Springer, pp. 91-108, Genova, Italy, 2001.
[74] G. Engels, J. M. Kuster, and R. Heckel, "A methodology for specifying and analyzing consistency of object - oriented behavioral models," in *Proc. 9th ACM SIGSOFT Symp. on Foundations of Softw. Eng.*, pp. 186-195, Vienna, Austria, 2001.
[75] J. Davies and C. Crichton, "Concurrency and refinement in the UML," in *Proc. Refine 2002: the BCS FACS Refinement Workshop, Electronic Notes in Theoretical Computer Science*, vol. 70, no. 3, pp. 217-243, Nov. 2002.
[76] J. Araujo, Metamorphosis: an Integrated Object-Oriented Requirements Analysis and Specification Method, Ph.D. Thesis, Dept. of Computing, University of Lancaster, 1996.
[77] K. Achatz and W. Schulte, "A formal object-oriented method inspired by fusion and Object - Z," in *Proc. ZUM'97: the Z Formal Specification Notation, Int. Conf.*, vol. 1212 of LNCS, pp. 91-111, Reading, UK, 1997.
[78] S. Dupuy, Y. Ledru, and M. Chabre-Peccoud, "Integrating OMT and Object-Z," in *Proc. BCS FACS/EROS ROOM Workshop*, pp. 347-366, London, UK, 1997.
[79] S. Kim and D. Carrington, "A formal mapping between UML models and Object - Z specifications," in *Proc. ZB2000: Formal Specification and Development in Z and B, York, UK, Lecture Notes in Computer Science*, vol. 1878, pp. pp. 2-21, York, UK, 2000.
[80] S. Kim and D. Carrington, "A formal model of the UML meta - model: the UML state machine and its integrity constraints," in *Proc. ZB 2002*, vol. 2272 of LNC, pp. 497-516, Grenoble, France, 2002.
[81] F. Bouquet, F. Dadeau, and J. Groslambert, "Checking JML specifications with B machines," in *Proc. ZB 2005*, vol. 3455 of LNCS, pp. 434-453, Guildford, UK, 2005.
[82] Y. Chen and H. Miao, "From an abstract Object - Z specification to UML diagram," *J. of Information & Computational Science*, vol. 1, no. 2, pp.319-324, 2004.

عباس رسول‌زادگان در سال ۱۳۸۳ مدرک کارشناسی مهندسی نرم‌افزار خود را از دانشگاه علوم و فنون هوایی شهید ستاری و در سال ۱۳۸۵ مدرک کارشناسی ارشد مهندسی نرم‌افزار خود را از دانشگاه صنعتی امیرکبیر دریافت نمود. ایشان از سال ۱۳۸۶ در حال تحصیل در مقطع دکتری مهندسی نرم‌افزار در دانشکده مهندسی کامپیوتر دانشگاه صنعتی امیرکبیر می‌باشد. زمینه‌های تحقیقاتی مورد علاقه نام‌برده عبارتند از: مهندسی نرم‌افزار، مهندسی نیازمندی‌ها، روش‌های زمان‌بندی، تخمین، پایگاه داده پویا و روش‌های یادگیری.

احمد عبداله‌زاده بارفروش همکاری خود را با دانشکده مهندسی کامپیوتر دانشگاه صنعتی امیرکبیر در سال ۱۳۷۰ و پس از اخذ مدرک دکتری مهندسی کامپیوتر از دانشگاه بریستول انگلستان، آغاز نمود و هم اکنون استاد دانشکده مهندسی کامپیوتر دانشگاه صنعتی امیرکبیر می‌باشد. ایشان از سال ۱۳۷۹ الی ۱۳۸۱ به عنوان استاد مدعو در دانشگاه‌های مریلند آمریکا و اُراسی (پاریس II) فرانسه و در سال ۱۳۸۸ تا ۱۳۹۰ به عنوان استاد مهمان در دانشگاههای ترنتو ایتالیا و لاف‌برو انگلستان مشغول به کار بوده است. دکتر عبداله‌زاده کتاب‌های "مقدمه‌ای بر هوش مصنوعی توزیع شده" و "کلیات متدولوژی تامین کیفیت" را نیز تالیف نموده است. زمینه‌های تحقیقاتی مورد علاقه نام‌برده عبارتند از: تکنیک‌های هوش مصنوعی، هوش مصنوعی توزیع شده، مذاکره خودکار، سیستم‌های خبره، پردازش زبان طبیعی، سیستم‌های تصمیم‌یار، هوش تجاری، پایگاه داده تحلیلی، داده‌کاوی، و مهندسی نرم‌افزار.