نشریه علمی ترویجی محاسبات نرم
سال ششم/ شمارهٔ دوم/ پاییز و زمستان ۱۳۹۶/ صفحهٔ ۲ـ۱۷

دریافت مقاله: ۱۳۹۵/۷/۲۸
پذیرش مقاله: ۱۳۹۷/۷/۸

تولید خودکار یک سیستم چندعاملی برای مدیریت بحران به روش مدل‌رانده

سمانه حسین‌دوست[۱]، طاهره آدم‌زاده[۲]، بهمن زمانی[۳*]، افسانه فاطمی[۴]

دانشجوی کارشناسی ارشد دانشکده مهندسی کامپیوتر، دانشگاه اصفهان، اصفهان، ایران
s.hoseiondoost@eng.ui.ac.ir
دانش‌آموختهٔ کارشناسی ارشد، دانشکده مهندسی کامپیوتر، دانشگاه اصفهان، اصفهان، ایران
t.adamzadeh@eng.ui.ac.ir
استادیار، دانشکده مهندسی کامپیوتر، دانشگاه اصفهان، اصفهان، ایران
zamani@eng.ui.ac.ir
استادیار، دانشکده مهندسی کامپیوتر، دانشگاه اصفهان، اصفهان، ایران
a_fatemi@eng.ui.ac.ir

چکیده: با توجه به وقوع روزافزون حوادث غیرمترقبه و نیاز به برنامه‌ریزی پیش از بحران به‌منظور کاهش خطرات و خسارات وارده، نیاز به مدل‌سازی محیط‌های واکنش اضطراری بیش از پیش احساس می‌شود. با استفاده از مدل‌سازی می‌توان برای عملیات پاسخ‌گویی به بحران، مانند تشکیل تیم، تخصیص وظایف به تیم‌ها و انجام وظایف توسط تیم‌ها، برنامه‌ریزی دقیق‌تری انجام داد. یکی از چالش‌های رایج در این مسیر، این است که مدل باید برای مدیر بحران به شکلی قابل فهم درآید تا وی بتواند از نتایج حاصل از مدل‌سازی بهره‌برداری کند. برای اجرای مدل و مشاهدهٔ نتایج، لازم است مدل به برنامه تبدیل شود. مدیر بحران با اجرای آن برنامه می‌تواند چگونگی اجرای عملیاتی از مدل مانند تشکیل تیم، تخصیص وظایف و انجام وظایف را مشاهده کند.

در این مقاله، کد قابل اجرای یک سیستم چندعاملی به‌طور خودکار از روی مدل‌های طراحی‌شده بر اساس روش مدل‌رانده تولید شده است. این کار با استفاده از زبان مدل‌سازی خاص منظورهٔ ERE-ML و ابزار مربوط به آن انجام شده و امکانات جدیدی به این زبان افزوده شده است. به‌منظور ارزیابی خروجی کدهای تبدیل، مطالعهٔ موردی زلزله بم انجام شده و سناریوهای تعریف‌شده در سیستم تولیدشده به نمایش درآمده است.

واژه‌های کلیدی: مهندسی نرم‌افزار مدل‌رانده، تولید خودکار کد، سیستم‌های چندعاملی، محیط‌های واکنش اضطراری، ERE-ML.

۱. مقدمه

امروزه یکی از نیازهای ضروری بشر برای زندگی بهتر، پیش‌بینی حوادث طبیعی مانند زلزله، سیل، سونامی و حوادث مداخله‌جویانهٔ بشر همچون آتش‌سوزی و حملات تروریستی است. وقتی امکان پیش‌بینی وجود ندارد یا پیش‌بینی‌ها به شکست می‌انجامد، باید بتوان در مقابل این حوادث (که از این پس آن‌ها را بحران می‌نامیم) واکنش بهنگام نشان داد. در این‌گونه موارد مدیران بحران می‌توانند با سازمان‌دهی مناسب تیم‌ها و تخصیص وظایف مربوط به هریک از آن‌ها بر اساس منابع و تخصص‌های موجود، نقش مهمی را در کاهش خسارات و تلفات ایفا کنند. مدل‌سازی یکی از روش‌هایی است که مدیران بحران برای سازمان‌دهی تیم‌های امدادرسانی، می‌توانند از آن استفاده کنند. این کار می‌تواند کارایی تیم‌های امدادرسان را بهبود بخشد و باعث کاهش تلفات و خطرات، و نجات جان افراد شود [۱].

محیط‌های واکنش اضطراری، محیط‌های تعریف‌شده برای پاسخ به بحران‌ها هستند. از آنجا که محیط‌های واکنش اضطراری محیط‌هایی پویا و پیچیده‌اند [۲]، پیاده‌سازی سیستم‌های نرم‌افزاری برای این گونه محیط‌ها کاری دشوار و پیچیده خواهد بود. برای غلبه بر این پیچیدگی، استفاده از رویکردهای جدید توسعهٔ نرم‌افزار، مثل مهندسی نرم‌افزار عامل‌گرا و توسعهٔ مدل‌رانده، می‌تواند بسیار کمک‌کننده باشد. به کمک این رویکردها می‌توان علاوه بر غلبه بر پیچیدگی این سیستم‌ها، از روی مدل به کدهایی رسید که به‌صورت کاملاً خودکار تولید شده‌اند. در توسعهٔ مدل‌رانده، کد به‌صورت خودکار از مدل‌ها تولید می‌گردد، که این کار با تولید اسکلت کد شروع می‌شود و تا تولید کامل محصول ادامه می‌یابد [۳].

یکی از کارهایی که برای مدل‌سازی یک واقعه در یک محیط واکنش اضطراری و نیز تولید کد اولیهٔ عامل‌ها انجام شده، زبان ERE-ML [۴] است. ERE-ML یک زبان مدل‌سازی برای طراحی سیستم‌های چندعاملی در محیط‌های واکنش اضطراری است که در آن مفاهیمی چون «حادثه»، «وظیفه»، «عامل»، «سازمان» و «محیط» تعریف شده است. کاربر به کمک این مفاهیم و روابط موجود در زبان مدل‌سازی، می‌تواند یک سناریوی واکنش اضطراری را مدل‌سازی کند. سپس مدل به‌عنوان ورودی به کدهای تبدیل داده می‌شود و یک برنامهٔ جاوا تولید می‌گردد.

در این پژوهش، به‌منظور تبدیل مدل تولیدشده به یک برنامهٔ قابل اجرا، ویژگی‌هایی به برخی از عناصر موجود در زبان مدل‌سازی ERE-ML افزوده شده است که در بخش ۴ به آن‌ها پرداخته می‌شود. با توجه به نیاز به بصری‌سازی عملیات طراحی‌شده در یک محیط واکنش اضطراری و نمایش عملیاتی از قبیل تشکیل تیم، تخصیص وظایف و انجام وظیفه به‌صورت کاملاً گرافیکی و قابل فهم برای مدیر بحران، نیاز به مکانیزمی است که بتوان از مدل تولیدشده (به زبان ERE-ML) برنامه‌ای (به زبان جاوا) تولید کرد. به‌طور خلاصه، در این مقاله قصد بر آن است تا با استفاده از گسترش چارچوب JAMDER [۵]، و نیز تکمیل کدهای تبدیل مدل ERE-ML به‌صورت خودکار به یک برنامهٔ قابل اجرا روی سکوی جاوا تبدیل شود تا مدیر بحران بتواند نتایج حاصل از تشکیل تیم، تخصیص وظایف و انجام وظیفه در یک محیط واکنش اضطراری را به‌صورت گرافیکی مشاهده کند.

ساختار این مقاله بدین شکل است: در بخش ۲، کارهای مرتبط با این تحقیق ارائه شده است. بخش ۳، یک مثال انگیزشی در زمینهٔ تحقیق را بیان می‌کند. در بخش ۴ راه‌حل پیشنهادی، شامل انجام تغییرات در زبان مدل‌سازی ERE-ML، تکمیل چارچوب JAMDER و کدهای تبدیل ارائه شده‌اند. در نهایت به‌منظور ارزیابی برنامهٔ تولیدشده، در بخش ۵ مطالعهٔ موردی زلزله بم به کمک ابزار ERE-ML مدل‌سازی می‌شود، و سپس مدل طراحی‌شده به‌صورت خودکار به یک برنامهٔ قابل اجرا روی سکوی جاوا تبدیل می‌گردد.

۲. کارهای مرتبط

بنا بر آنچه در مقدمه بیان شد، قلمرو پژوهش حاضر در سه حوزهٔ محیط‌های واکنش اضطراری، سیستم‌های چندعاملی و توسعهٔ مدل‌رانده مطرح می‌شود. در ادامه، به ارائهٔ کارهای

مرتبط با توسعهٔ مدل‌راندهٔ سیستم‌های چندعاملی، توسعهٔ مدل‌راندهٔ محیط‌های واکنش اضطراری و توسعهٔ مدل‌راندهٔ سیستم‌های چندعاملی در محیط‌های واکنش اضطراری و نقاط ضعف هریک از آن‌ها پرداخته می‌شود.

۱.۲. توسعهٔ مدل‌راندهٔ سیستم‌های چندعاملی

در این زمینه، چندین زبان مدل‌سازی و ابزار ارائه شده است که در ادامه به آن‌ها پرداخته می‌شود.

بائور[1] و همکاران [۶] زبانی به نام AUML[2] را معرفی کرده‌اند که از گسترش UML [۷] به کمک مفاهیم مرتبط با سیستم‌های چندعاملی ایجاد شده است. با استفاده از این زبان می‌توان پروتکل‌های تعاملی میان عامل‌ها و نیز رفتار درونی آن‌ها را مدل‌سازی کرد. AUML از نمادهای گرافیکی UML برای مدل‌سازی پروتکل‌ها استفاده می‌کند. همچنین ابزاری برای پشتیبانی از مدل‌سازی به کمک AUML طراحی شده است. اما این ابزار قابلیت تولید خودکار کد از روی مدل را ندارد [۸].

هان[3] [۹] یک زبان مدل‌سازی خاص دامنه برای سیستم‌های چندعاملی به نام DSML4MAS ارائه کرده است. این زبان از مفاهیم اصلی موجود در یک سیستم چندعاملی همچون عامل‌ها، ساختارهای سازمانی و پروتکل‌های تعاملی پشتیبانی می‌کند. همچنین تبدیل‌هایی برای تولید خودکار کد از روی مدل‌های طراحی‌شده با DSML4MAS نوشته شده است. اما در عمل، کدهای تولیدشده برای اجرا به مداخلهٔ برنامه‌نویس نیاز دارند.

INGENIAS یک متدولوژی مهندسی نرم‌افزار عامل‌گراست که زبانی را برای مدل‌سازی سیستم‌های چندعاملی نیز فراهم کرده است [۱۰]. این زبان توسط شامل پنج فرامدل است که جنبه‌های مختلف سیستم را از لحاظ ساختاری و رفتاری توصیف می‌کند. به‌منظور پشتیبانی از فرایند توسعه در INGENIAS ابزاری به نام IDK[4] نیز طراحی شده است که شامل ویرایشگری برای ایجاد مدل‌هایی از یک سیستم چندعاملی بر اساس زبان مدل‌سازی INGENIAS می‌باشد [۱۱]. در این ابزار، ماژولی تحت عنوان IAF[5] [۱۲] وجود دارد که از روی مدل‌های رسم‌شده، کد جاوا تولید می‌کند. اما این کدها تنها اسکلت کد یک سیستم چندعاملی هستند و برای اجرا به مداخلهٔ برنامه‌نویس نیاز دارند.

داسیلوا[6] و همکاران [۱۳] یک زبان مدل‌سازی برای سیستم‌های چندعاملی به نام MAS-ML ارائه کرده‌اند که از گسترش زبان UML به کمک مفاهیم موجود در چارچوب TAO[7] [۱۴] ایجاد شده است. TAO مفاهیم لازم برای مدل‌سازی یک سیستم چندعاملی را فراهم می‌آورد. برای پشتیبانی از زبان مدل‌سازی MAS-ML، ابزاری با نام MAS-ML Tool [۱۵] نیز توسعه داده شده که امکان مدل‌سازی گرافیکی سیستم‌های چندعاملی را فراهم می‌کند. در سال‌های اخیر یک رویکرد مدل‌رانده برای مدل‌سازی و تولید خودکار کد با استفاده از زبان و ابزار مدل‌سازی MAS-ML ارائه شده است [۱۶]. اما ضعف کار در این است که کد تولیدشده به‌صورت خودکار، قابلیت اجرایی نداشته و فقط با اجرای کدهای تبدیل، کلاس‌های موجود در مدل به کلاس جاوای معادل تبدیل می‌شوند.

۲.۲. توسعهٔ مدل‌راندهٔ سیستم‌های واکنش اضطراری

در این بخش به معرفی زبان‌ها و ابزارهای مدل‌سازی ارائه‌شده برای توسعهٔ مدل‌رانده سیستم‌های واکنش اضطراری پرداخته می‌شود.

عثمان[8] و همکاران [۱۷] فرامدلی برای مدیریت حادثه به نام DMM[9] معرفی کرده‌اند که می‌توان برای مدل‌سازی مراحل چهارگانهٔ مدیریت بحران از آن استفاده کرد. این مراحل به‌ترتیب عبارت‌اند از: پیشگیری، آمادگی، واکنش و بهبود. همچنین ابزاری به‌نام DMKR[10] [۱۸] برای ایجاد، ذخیره و

1. Bauer
2. Agent UML
3. Hahn
4. INGENIAS Development Kit

5. INGENIAS Agent Framework
6. Da Silva
7. Taming Agents and Objects
8. Othman
9. Disaster Management Meta-model
10. Disaster Management Knowledge Repository

بازیابی دانش مدیریت بحران بر اساس مفاهیم موجود در زبان DMM ایجاد شده است. این ابزار از نمادهای گرافیکی موجود در UML برای مدل‌سازی استفاده می‌کند و فقط سیستمی برای ذخیره و بازیابی دانش مدیریت بحران است. این سیستم از رویکرد مدل‌رانده برای تولید خودکار کد و تولید برنامه‌ای برای نمایش عملیات‌هایی که باید در طول یک حادثه انجام شوند، استفاده نمی‌کند.

بتکه[1] [۱۹]، یک فرامدل عمومی به نام DRP[2] برای نمایش فرایندهای واکنش به بحران ارائه داده است. به کمک این فرامدل، ساختار داده‌ای صحیحی را برای توسعهٔ سیستم‌های آگاه از فرایند در مدیریت واکنش به حادثه فراهم می‌آید. تاکنون ابزاری برای پشتیبانی از این زبان و نیز تبدیل خودکار مدل به کد ارائه نشده است.

۳.۲. توسعهٔ مدل‌راندهٔ سیستم‌های چندعاملی در محیط‌های واکنش اضطراری

مصطفی و همکاران [۱] یک چارچوب متدولوژیکی سازمانی ارائه کرده‌اند که مدل‌سازی و شبیه‌سازی حوادث طبیعی را نشان می‌دهد. در این کار از رویکرد معماری مدل‌رانده[3] برای توسعهٔ سیستم استفاده شده است. بدین منظور ابتدا یک مدل‌سازی مستقل از سکو برپایهٔ مدل سازمان–نقش مفهومی[4] انجام شده و سپس این مدل به مدل سازمان–عامل مفهومی[5] بهبود پیدا می‌کند. با این عمل مدل‌ها، به مدل‌های وابسته به سکو تبدیل شده و عامل‌ها معنی پیدا می‌کنند. اما در اصل، پیاده‌سازی این چارچوب برای حوادث طبیعی انجام نشده و این کار به‌عنوان کارهای آتی پژوهش پیشنهاد شده است.

آدم‌زاده [۴] زبان MAS-ML را با استفاده از مفاهیم موجود در فرامدل DMM گسترش داده و زبان مدل‌سازی جدیدی به نام ERE-ML ایجاد کرده است. این زبان، یک زبان مدل‌سازی خاص مرحلهٔ واکنش از محیط‌های واکنش اضطراری است.

همراه با این زبان، یک ابزار پشتیبان جهت مدل‌سازی با ERE-ML نیز ارائه شده که گسترش‌یافتهٔ ابزار MAS-ML [۱۵] است. ضعف پژوهش فوق در این است که قابلیت تولید کد قابل اجرا به‌صورت خودکار را ندارد. این مقاله قصد دارد این ضعف را برطرف کرده و کد قابل اجرا به‌صورت خودکار از روی مدل تولید کند.

۳. مثال انگیزشی

یکی از حوادثی که در سال‌های اخیر در ایران، خسارات جانی و مالی فراوانی به بار آورد، زمین‌لرزه‌ای به بزرگی ۶/۵ ریشتر بود که در بامداد ۵ دی‌ماه ۱۳۸۲، در شهر باستانی بم در جنوب شرق استان کرمان به وقوع پیوست [۱۴]. در طول زلزله، تقریباً همهٔ زیرساخت‌های شهر آسیب دید. خطوط برق و ارتباطات تلفنی قطع شدند و تنها ۳۰٪ از آن، دو هفته پس از حادثه وصل شد. با وجود آنکه شهر در یک منطقه مسطح واقع شده است، حمل‌ونقل شهری نیز مختل شد و به‌دلیل تخریب گستردهٔ جاده‌ها و پیاده‌روها، نجات مصدومان بسیار دشوار بود [۱۴].

در این مقاله، قصد بر آن است تا با ارائهٔ یک نمودار کلاس از سناریوی شرح‌داده‌شده، یک برنامهٔ کاربردی برای شبیه‌سازی آن، به زبان جاوا و به‌صورت کاملاً خودکار تولید شود. ازاین‌رو در ادامه، به بیان راه‌حل پیشنهادی در این زمینه پرداخته می‌شود.

۴. راه‌حل پیشنهادی

همان طور که در بخش ۱ بیان شد، هدف این پژوهش، تولید خودکار یک برنامهٔ قابل اجرا از روی مدل ERE-ML است. برای این منظور نیاز به افزودن ویژگی‌هایی به زبان مدل‌سازی ERE-ML، تکمیل چارچوب JAMDER و کدهای تبدیل نوشته‌شده به زبان MTL خواهد بود، که در ادامه به آن‌ها پرداخته می‌شود.

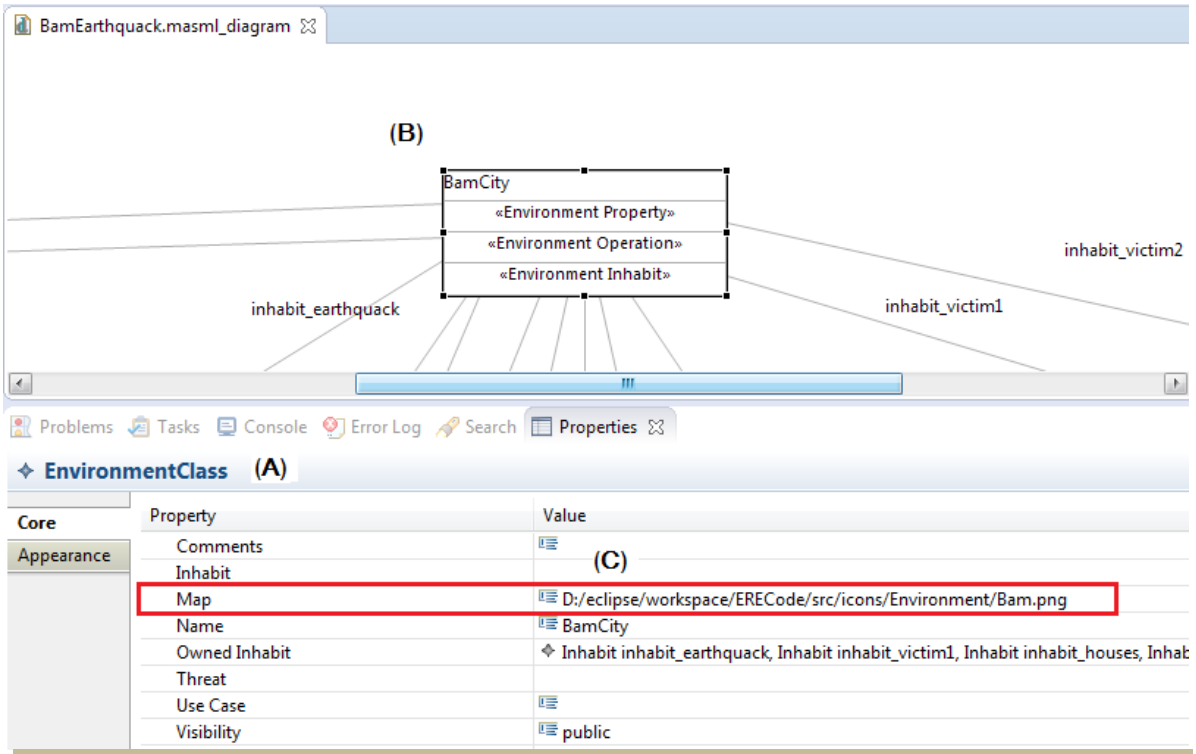
۱.۴. تکمیل زبان مدل‌سازی ERE-ML

برای تولید یک نمایش بصری از سیستم چندعاملی طراحی‌شده با زبان ERE-ML، لازم است ویژگی‌هایی به

1. Betke
2. Disaster Response Process
3. Model Driven Architecture (MDA)
4. Conceptual Role Organizational Model (CROM)
5. Conceptual Agent Organizational Model (CAOM)

عناصر موجود در زبان افزوده شود. بنابراین در مفاهیم موجود در زبان ERE-ML، تغییراتی داده شده که در این بخش به آن‌ها پرداخته می‌شود.

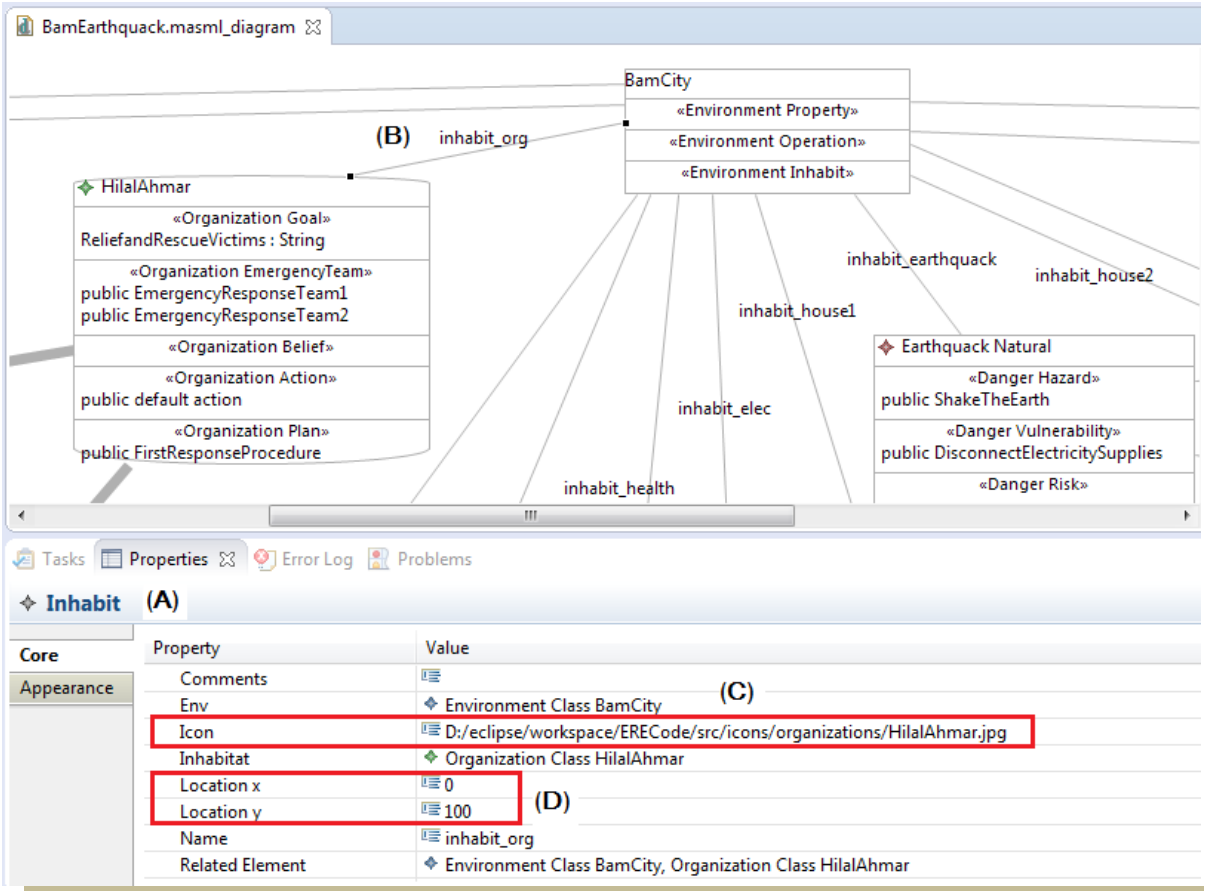
- EnvironmentClass: یک محیط عنصری است که عامل‌ها، اشیا و سازمان‌ها در آن ساکن می‌شوند [۱۴]. گفتنی است که به‌منظور نمایش گرافیکی محیط، صفتی به نام Map (که مسیر تصویر نقشهٔ جغرافیایی محیط در سیستم خواهد بود) به این مفهوم اضافه شده است و در حین مدل‌سازی باید توسط مدل‌ساز در مدل قرار گیرد. در شکل (۱)، نمونه‌ای از این متاکلاس برای شهر بم نشان داده شده است.



شکل (۱): ویژگی‌های مربوط به متاکلاس EnvironmentClass

همان طور که در شکل مشخص است، در محیط مدل‌سازی ERE-ML، نمونه‌ای از کلاس EvironmentClass (A)، با نام BamCity ایجاد شده است (B) و ویژگی Map در آن، برابر تصویری که کاربر مسیر آن را از سیستم خود انتخاب کرده، قرار گرفته است (C).

- Inhabit: مفهومی در زبان مدل‌سازی MAS-ML است. هر عنصری که باید در محیط استقرار یابد، از طریق رابطهٔ Inhabit به محیط متصل می‌شود [۱۵]. اما به‌منظور نمایش گرافیکی این عناصر در برنامهٔ کاربردی، لازم است که هریک از آن‌ها دارای یک صفت Icon (شکل ظاهری عنصر) و Location (موقعیت قرارگیری عنصر در محیط) باشد که هنگام مدل‌سازی، توسط مدل‌ساز تکمیل شود. لذا در این پژوهش، این دو صفت به رابطهٔ Inhabit اضافه شده‌اند. در شکل (۲)، نمونه‌ای از این رابطه برای نمایش استقرار «سازمان هلال احمر» در «شهر بم» نشان داده شده است.

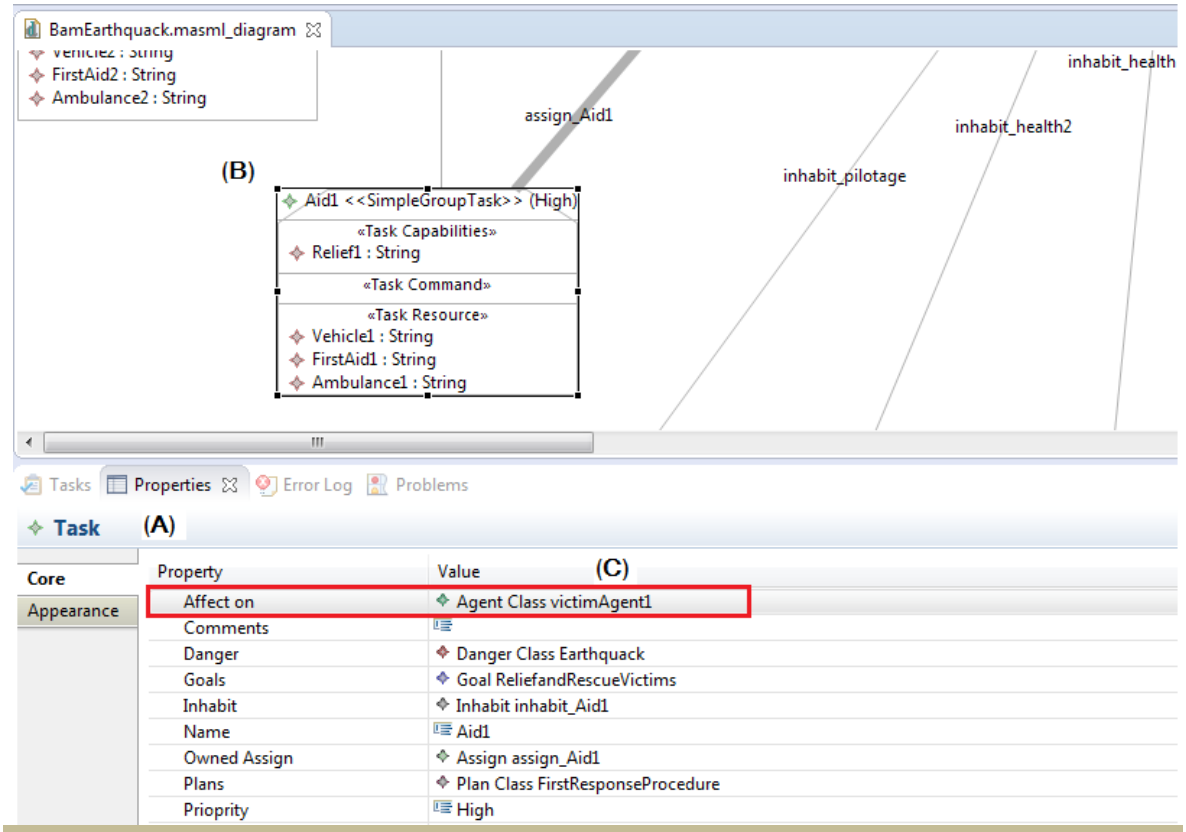


شکل (۲): ویژگی‌های مربوط به متاکلاس Inhabit

همان طور که در شکل مشخص است، در محیط مدل‌سازی ERE-ML، یک رابطهٔ Inhabit (A)، با نام inhabit_org ایجاد شده است (B)، و ویژگی‌های Icon و Location آن، برابر با مقادیر دلخواه کاربر قرار گرفته است (C و D).

- Task: مسئولیت‌ها و وظایفی که باید توسط اعضای تیم اضطراری در یک بازهٔ زمانی محدود انجام شود [۲۰]. در ERE-ML، یک وظیفه به‌صورت مجموعه‌ای از توانمندی‌ها، منابع و دستورات تعریف می‌شود [۲۱]. به‌منظور نمایش گرافیکی نحوهٔ انجام وظیفه در محیط، نیاز به یک ویژگی برای این مفهوم خواهد بود که تعیین کند این وظیفه روی چه عنصری از محیط اثر می‌گذارد. این ویژگی با نام Affect_on برای متاکلاس Task تعریف می‌شود. در شکل (۳)، نمونه‌ای از این متاکلاس برای وظیفهٔ Aid نشان داده شده است.

همان طور که در شکل مشخص است، در محیط مدل‌سازی ERE-ML، نمونه‌ای از کلاس Task (A) با نام Aid1 ایجاد شده است (B)، و ویژگی Affect_on در آن برابر با victiomAgent1 قرار گرفته است (C).



شکل (۳): ویژگی‌های مربوط به متاکلاس Task

شایان ذکر است که سایر مفاهیم از جمله Class، AgentClass، OrganizationClass، Disaster و نیز روابط موجود در زبان مدل‌سازی ERE-ML [۴] تغییری نکرده‌اند و از همان مفاهیم و روابط در این پژوهش استفاده شده است.

۲.۴. گسترش چارچوب JAMDER

پس از آنکه مدل یک محیط واکنش اضطراری، در ابزار ERE-ML طراحی شد، می‌تواند به‌صورت خودکار به یک برنامهٔ قابل اجرا روی سکوی جاوا تبدیل شود. بدین منظور و همچنین به‌علت جلوگیری از طولانی شدن کد تولیدشده، از گسترش‌یافتهٔ چارچوب JAMDER [۵] که بر اساس مفاهیم معرفی‌شده در زبان مدل‌سازی ERE-ML نوشته شده است، استفاده می‌شود. این چارچوب، به زبان جاوا نوشته شده و چارچوب JAMDER را که مختص سیستم‌های چندعاملی بوده، برای محیط‌های واکنش اضطراری گسترش داده است.

هدف این پژوهش این است که محیط و عناصر مستقر در آن به‌صورت قابل نمایش برای کاربر درآیند و نیز امکانی در اختیار کاربر قرار گیرد تا بتواند در اثر تعامل با برنامه، نحوهٔ تشکیل تیم، تخصیص وظایف و انجام وظایف طراحی‌شده در مدل را به‌صورت بصری مشاهده کند. لذا چارچوب JAMDER برای پیاده‌سازی یک محیط گرافیکی و قابل نمایش، دوباره گسترش داده می‌شود. بدین منظور، برخی از کلاس‌های موجود در چارچوب JAMDER گسترش یافته‌اند که در ادامه به آن‌ها پرداخته می‌شود.

- کلاس Environment: این کلاس با کتابخانه‌های swing و awt جاوا گسترش داده می‌شود؛ به‌نحوی که نقشهٔ جغرافیایی، عناصر مستقر در محیط، و پنلی برای برقراری ارتباط با کاربر و ارائهٔ گزارش‌های عملیات به او نشان داده می‌شود.
- کلاس Object: کلاس جدیدی به نام Object.java نیز به چارچوب JAMDER اضافه می‌شود که بیانگر اشیاء

ایجادشده از مفهوم Class [۱۵] در هنگام مدل‌سازی است.

- کلاس Team: به‌منظور نمایش گرافیکی تشکیل تیم‌ها و تخصیص وظایف به هر تیم توسط سازمان مربوط، لازم است متدهای مربوط به این کلاس اضافه شوند. برای مثال، برای اضافه شدن عضو به یک تیم لازم است شمارهٔ تیم، شمارهٔ عضو، نام تیم، عاملی که باید به تیم اضافه شود، سازمانی که مسئولیت تشکیل تیم را دارد، و محیط، به‌عنوان پارامترهای ورودی به متدی با نام addMembers داده شوند. وظیفهٔ این متد به این صورت است که عامل‌هایی که در محیط پراکنده‌اند برای تشکیل تیم، به سمت سازمان مربوط حرکت می‌کنند و با استقرار همهٔ آن‌ها در یک مکان، تیم تشکیل می‌شود.

پس از تشکیل تیم، باید به هریک از آن‌ها وظایف تعیین‌شده در مدل، تخصیص یابد. برای این منظور، با اجرای برنامه در ابتدا وظایف در محل‌هایی که مدل‌ساز در مدل تعریف کرده است، قرار می‌گیرند. سپس برای تخصیص وظایف به هریک از تیم‌ها، تصویر مربوط به هر وظیفه، به سمت تیم مربوط حرکت می‌کند. اینکه کدام وظیفه به کدام تیم تعلق دارد، هنگام مدل‌سازی بر اساس رابطهٔ assign مشخص شده است.

۳.۴. تبدیل مدل به کد به‌صورت خودکار

حال زمان آن است که مدل ترسیم‌شده در ابزار ERE-ML بر اساس چارچوب ارائه‌شده، به‌صورت خودکار به کد جاوا تبدیل شود. در این مقاله، کدهای تبدیلی ارائه می‌شود تا مدل طراحی‌شده به زبان ERE-ML را به یک برنامهٔ قابل اجرا روی سکوی جاوا تبدیل کند. با اجرای برنامهٔ تولیدشده، چگونگی تشکیل تیم‌ها، تخصیص وظایف و انجام وظایف توسط تیم‌ها، به‌صورت گرافیکی نشان داده خواهد شد.

برای نوشتن کدهای تبدیل، از ابزار Acceleo استفاده شده است. این ابزار افزونه‌ای برای اکلیپس است که تولید کد از روی مدل را انجام می‌دهد. تبدیل کدی که این ابزار انجام می‌دهد، با استفاده از فایل‌های تبدیلی است که با زبان MTL نوشته شده است.

به‌منظور تبدیل مدل ترسیم‌شده برای سیستم‌های چندعاملی در محیط‌های واکنش اضطراری، از شش فایل تبدیل Disaster.mtl، Task.mtl، Agent.mtl، AgentRole.mtl، Environment.mtl و Organization.mtl استفاده شده است. در این پژوهش برای تولید یک برنامهٔ بصری قابل اجرا روی سکوی جاوا، باید فایل‌های مذکور گسترش یابند. در ادامه، کدهای تبدیل نوشته‌شده برای هریک از موارد تشکیل تیم، تخصیص وظایف و انجام وظیفه نشان داده شده است.

- **تشکیل تیم**: هر سیستم چندعاملی حداقل یک سازمان دارد که عامل‌ها را گروه‌بندی می‌کند [۱۴]. یک سازمان شامل تعدادی از تیم‌های واکنش اضطراری است و هر تیم شامل یک یا چندین عضو است. اعضای تیم‌ها را عامل‌های موجود در MAS-ML تشکیل می‌دهند. هر سازمان وظیفهٔ تشکیل تیم و تخصیص وظایف مربوط به هریک از تیم‌ها را بر عهده دارد. بنابراین فایل Organization.mtl به‌صورت نشان داده‌شده در شکل (۴) تکمیل می‌شود.

Organization.mtl

```
[for(t:EmergencyManagmentTeam | org.ownedTeam)]
  final Team [t.name.replaceAll('\\s','').concat ('T')/] = new
Team("[t.name/]");
[/for]

    env.OrganizeTeamButton.setEnabled(true);
    env.OrganizeTeamButton.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent e) {
        [for(t:EmergencyManagmentTeam | org.ownedTeam)]
            addTeam("[t.name.concat('T')/]", [t.name.
            replaceAll('\\s', '').concat('T')/]);

            env.reportText.append("[t.name/] was created.\n");
            [for (m:AgentClass | t.Members)]
                [t.name.replaceAll('\\s', '').concat
                ('T')/].addMembers([org.ownedTeam-> indexOf(t)-
                1/], [t.Members->indexOf(m)-1 /], "[m.name/]",
                env.getAgent("[m.name/]"), [org.name /].this ,
                env);
            [/for]
        [/for]
    }
    });
```

شکل (۴): قطعه کد تبدیل مربوط به تشکیل تیم

- **تخصیص وظایف**: در زمان وقوع یک حادثه، لازم است وظایف تعریف‌شده در محیط به تیم‌ها تخصیص داده شود. کدهای تبدیل در این بخش بر اساس رابطه‌های Assign موجود در مدل نوشته می‌شوند. Assign رابطه‌ای

موجود در زبان ERE-ML است که فقط می‌تواند بین مفاهیم «سازمان» و «وظیفه» ترسیم شود [۴]. بدین ترتیب، یک وظیفه به یک تیم خاص از سازمان، تخصیص داده می‌شود. در شکل (۵)، کد تبدیل مربوط به تخصیص وظایف به تیم‌ها نشان داده شده است.

Organization.mtl

```
env.AssignTaskButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
     // Assigned Tasks to teams
     try {
        [for (a:Assign | org.ownedAssign)]
            [for (t:EmergencyManagmentTeam | a.OwnedTeam )]
                [t.name.replaceAll('\\s', '').
                    concat('T')/].addTask([org.ownedTeam-
                    >indexOf(t)-1/], "[a.AssignSource.name/]",
                    env.getTask("[a.AssignSource.name/]",
                    [org.name /].this, env);
            [/for]
        [/for]
            } catch (IOException e1) {
                // TODO Auto-generated catch block
                e1.printStackTrace();
            }
        }
});
```

شکل (۵): قطعه کد تبدیل مربوط به تخصیص وظایف

- **انجام وظیفه:** همان طور که در بخش ۱.۴ بیان شد، به‌منظور نمایش نحوهٔ انجام وظیفه در محیط، یک ویژگی به نام Affect_on روی متاکلاس Task تعریف شده است. کدهای تبدیل برای این بخش، بدین صورت نوشته می‌شوند که ابتدا Location عنصری که به‌عنوان Affect_on برای Task تعریف شده است، یافت می‌شود و سپس عامل‌های تیم مربوط، برای انجام آن وظیفه، به سمت مکان مربوط حرکت می‌کنند. سپس اگر عنصر Affect_on، یک حادثه یا یک عامل آسیب‌دیده باشد، در اثر انجام وظیفه، آن حادثه یا عامل از محیط حذف می‌شود و تصویر مربوط به وظیفه جایگزین آن‌ها خواهد شد. اما اگر عنصر Affect_on، یک شئ در محیط باشد، آن شئ از محیط حذف نمی‌شود بلکه فقط عامل‌ها به سمت مکان آن شئ حرکت می‌کنند و سپس تصویر مربوط به وظیفه، در کنار آن شئ قرار می‌گیرد. بدین ترتیب صفت done مربوط به آن وظیفه هم true می‌شود. این بخش، باید در متد action از کلاس Task پیاده‌سازی شود. کدهای مربوط به این بخش، در ضمایم آورده شده است.

۵. ارزیابی

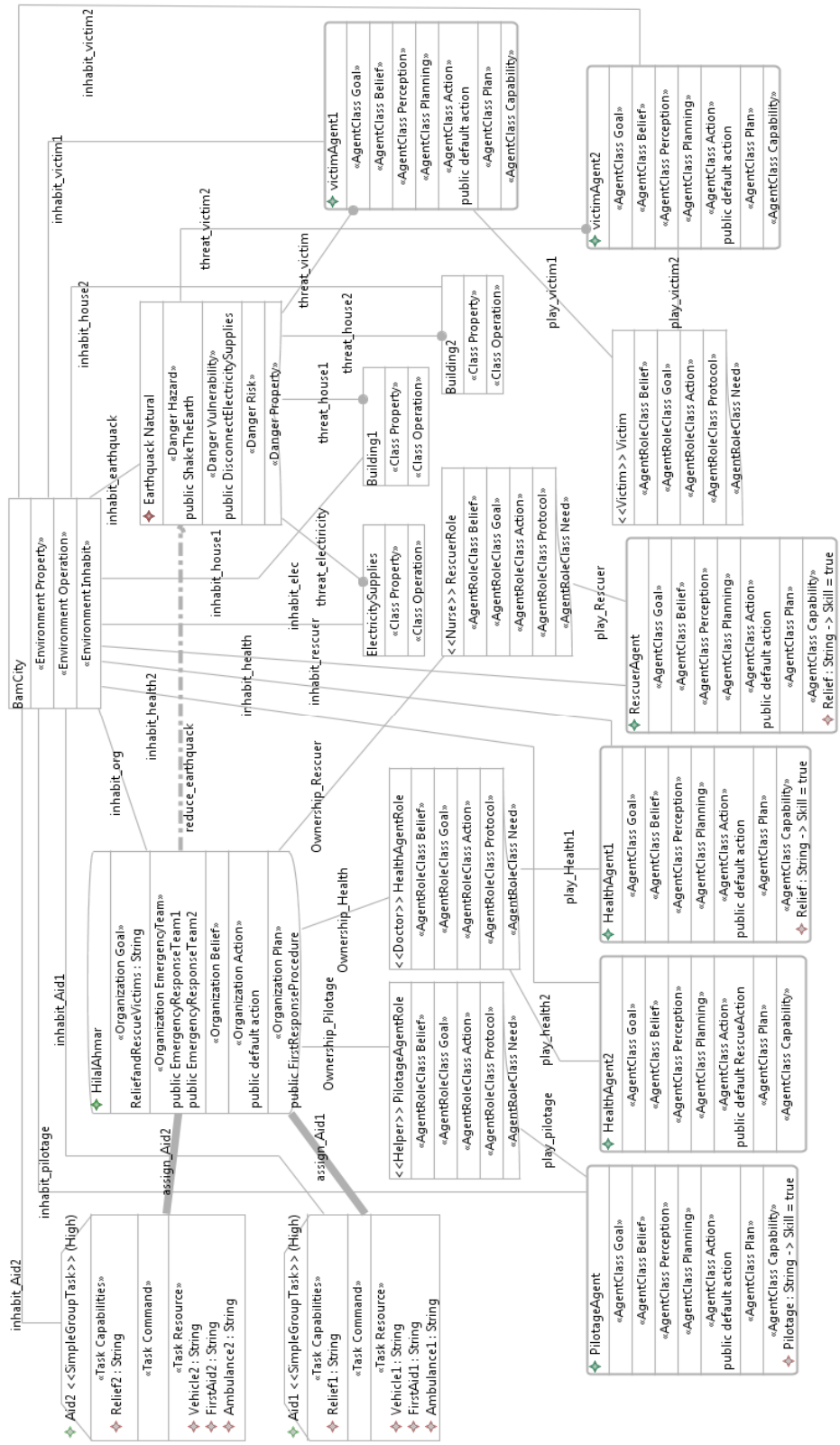
در این بخش به‌منظور ارزیابی، مطالعهٔ موردی زلزلهٔ بم به کمک ابزار ERE-ML مدل‌سازی می‌شود. سپس به کمک کدهای تبدیل نوشته‌شده، یک برنامهٔ قابل اجرا روی سکوی JADE، به‌صورت کاملاً خودکار تولید می‌شود.

۱.۵. نمودار کلاس برای زلزلهٔ بم

مهم‌ترین عناصری که در طراحی نمودار کلاس برای زلزلهٔ بم در نظر گرفته می‌شوند، عبارت‌اند از:

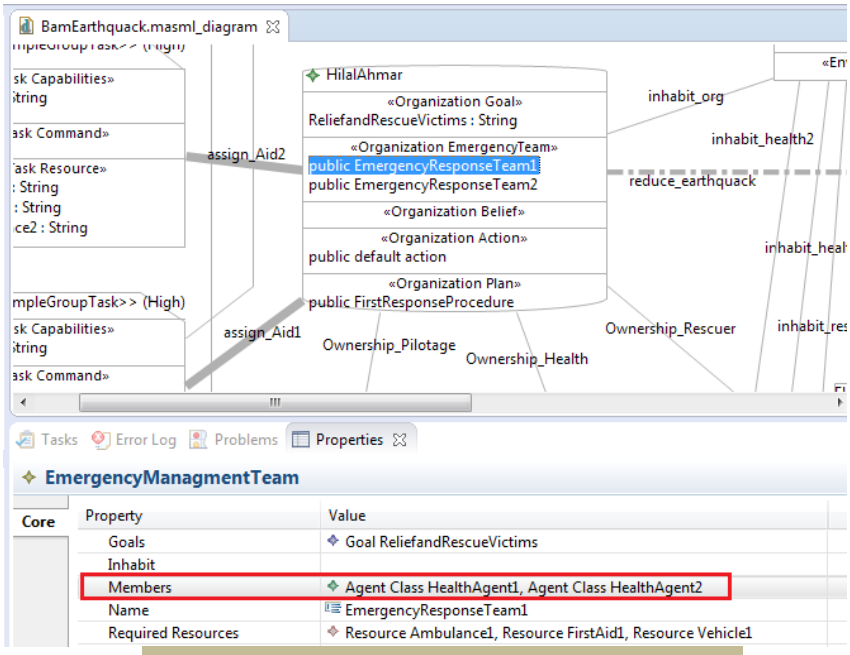
- محیط شهر بم (BamCity)
- حادثهٔ زلزله (Earthquack)
- عناصر تحت‌تأثیر حادثه مثل خانه‌ها و دکل‌های برق (Buildings- ElectricitySupplies)
- عوامل آسیب‌دیده (victim Agents)
- سازمان هلال احمر (HilalAhmar)
- وظایف امدادرسانی (Aids)
- عوامل پزشک (Health Agents)
- امدادگر (Rescuer)
- خلبان (Pilotage)

در شکل (۶)، نمودار کلاس ترسیم‌شده در ابزار ERE-ML، برای زلزلهٔ بم نشان داده شده است. برای این مطالعهٔ موردی، یک حادثهٔ زلزله در شهر بم اتفاق می‌افتد. این حادثه ساختمان‌ها، دکل‌های برق و افراد را تهدید می‌کند (در مدل با رابطهٔ Threat نشان داده شده است). سازمان هلال احمر مسئول به حداقل رساندن آسیب‌های ناشی از این حادثه است (رابطهٔ Reduce). همچنین فرض شده است که دو عامل پزشک، یک عامل خلبان، یک عامل امدادگر و دو مصدوم در محیط وجود دارند. سازمان هلال احمر دو تیم اورژانس تشکیل می‌دهد و وظایف امدادرسانی ۱ و ۲ را به آن‌ها اختصاص می‌دهد (نشان داده شده با رابطهٔ Assign).

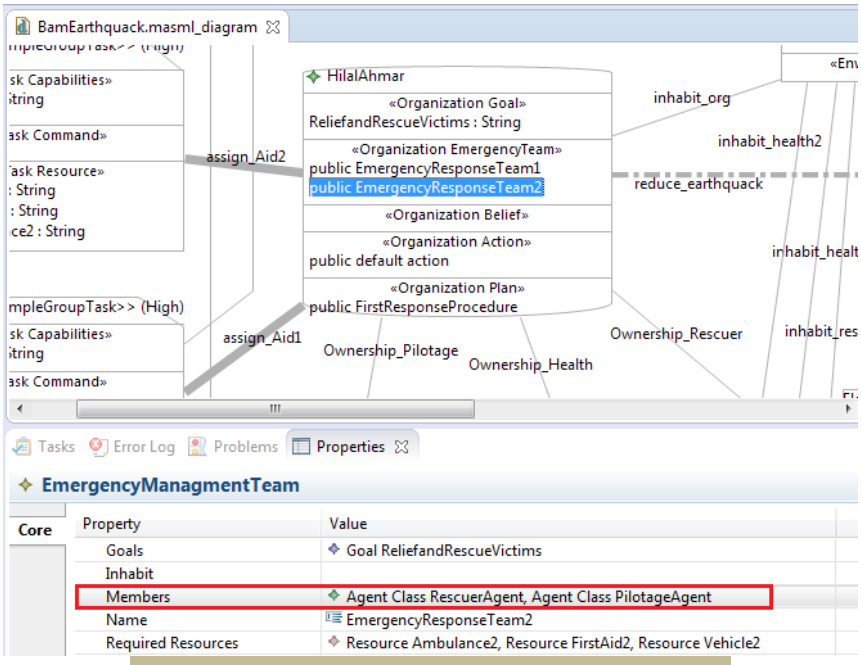


شکل (۶): نمودار کلاس طراحی‌شده در ابزار ERE-ML برای زلزلهٔ بم

هریک از تیم‌های موجود در سازمان، یک ویژگی به نام Members دارد که باید هنگام مدل‌سازی با ابزار ERE-ML از بین عامل‌های موجود در محیط انتخاب شوند. برای مثال، همان طور که در شکل‌های ۷ و ۸ نشان داده شده است، برای تیم اورژانس ۱، عوامل پزشک ۱ و ۲ انتخاب می‌شوند و برای تیم اورژانس ۲، عامل خلبان و عامل امدادگر.



شکل (۷): عامل‌های انتخاب‌شده برای تیم اورژانس ۱ در سازمان هلال احمر

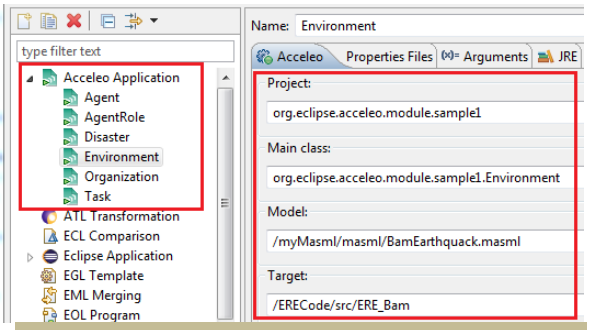


شکل (۸): عامل‌های انتخاب‌شده برای تیم اورژانس ۲ در سازمان هلال احمر

شایان ذکر است که پس از طراحی نمودار کلاس، مدل باید اعتبارسنجی شود. این امکان با نوشتن قیود OCL در ابزار ERE-ML فراهم شده است [۴]. با اعتبارسنجی مدل، در صورت وجود خطا، باید خطاها از مدل برطرف شوند. در نهایت، مدلی که بدون خطاست، باید به‌عنوان ورودی به کدهای تبدیل داده شود، که در ادامه به آن پرداخته می‌شود.

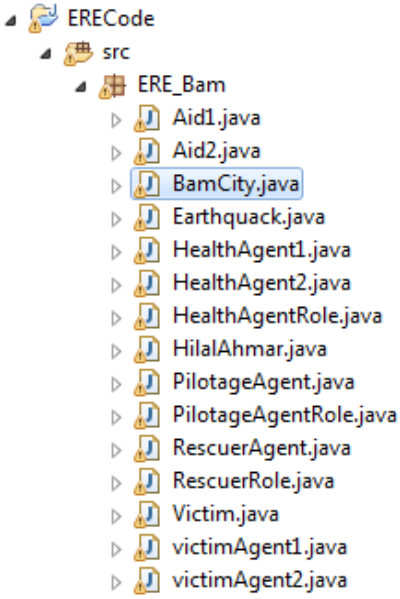
۲.۵. اجرای کدهای تبدیل

پس از مرحلهٔ مدل‌سازی، باید کدهای تبدیل اجرا شوند. برای این منظور، مطابق شکل (۹)، مدل طراحی‌شده در ابزار ERE-ML، به‌عنوان ورودی به فایل‌های تبدیل mtl موجود در پروژهٔ Acceleo داده می‌شود و یک پروژهٔ جاوا به‌عنوان مقصد انتخاب می‌گردد. در نهایت با زدن کلید Run، کد تبدیل اجرا شده، و فایل‌های جاوا در مقصد انتخاب‌شده تولید می‌شوند.



شکل (۹): ورودی‌های مورد نیاز برای تولید کد در ابزار Acceleo

در شکل (۱۰)، فایل‌های تولیدشده برای مطالعهٔ موردی زلزلهٔ بم نشان داده شده است. این فایل‌ها حاوی کدهای جاوا برای هریک از کلاس‌های طراحی‌شده در نمودار کلاس (شکل ۶) هستند. برای مثال کلاس‌های Aid1 و Aid2 کدهای جاوا مربوط به وظایف امدادرسانی ۱ و ۲ هستند. کلاس‌های HealthAgent1، HealthAgent2، PilotageAgent، RescuerAgent، victimAgent1 و victiomAgent2 به‌ترتیب عامل‌های پزشک، خلبان، امدادگر و عامل‌های آسیب‌دیده را نشان می‌دهد. Earthqueck، HilalAhmar و BamCity نیز به‌ترتیب فایل‌های جاوای مربوط به حادثهٔ زلزله، سازمان هلال احمر و محیط شهر بم هستند.

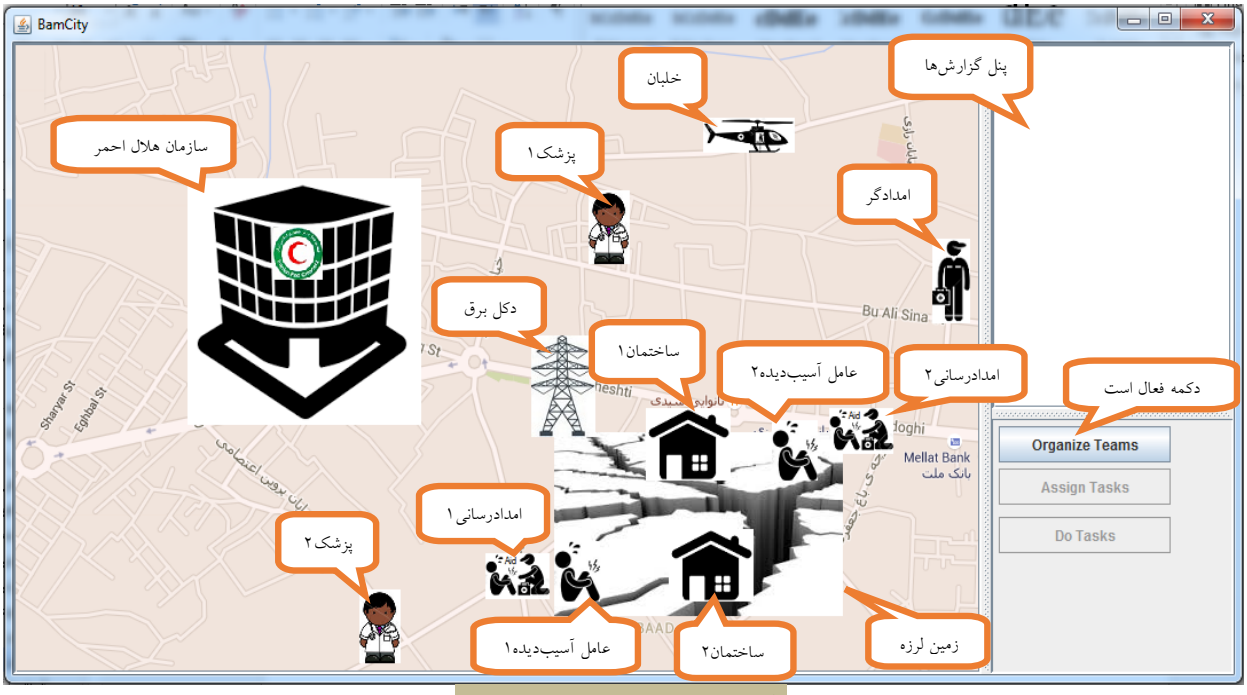


شکل (۱۰): فایل‌های جاوای تولیدشده برای مطالعهٔ موردی زلزلهٔ بم

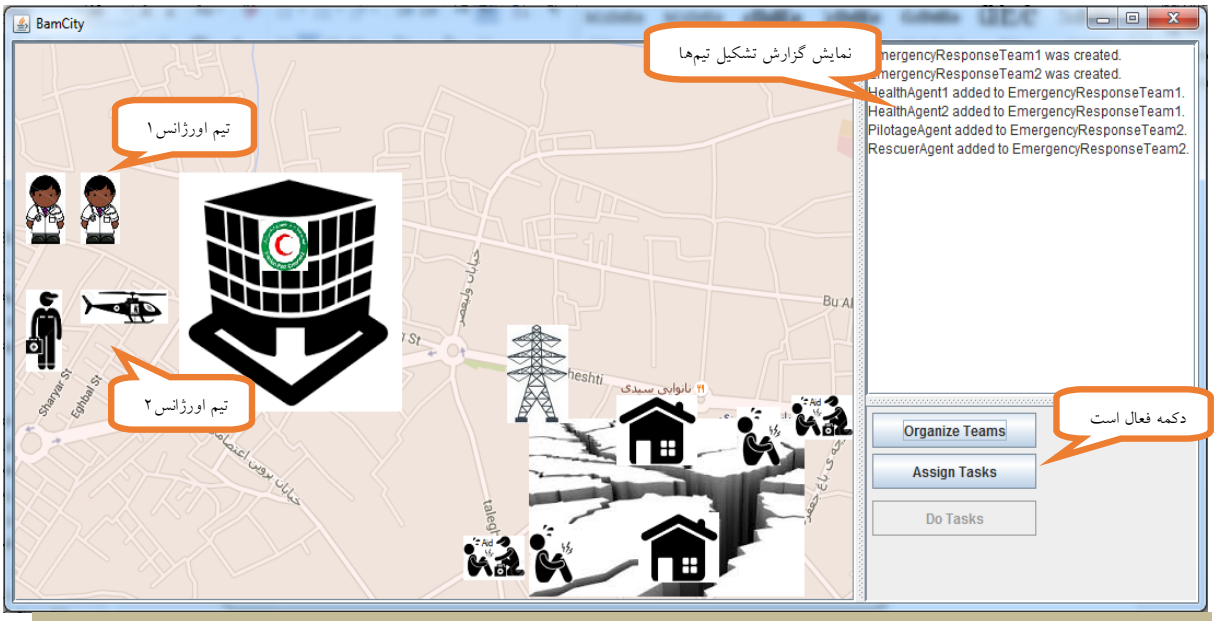
۳.۵. اجرای برنامهٔ کاربردی جاوا

همان طور که در بخش ۲.۵ بیان شد، با اجرای کدهای تبدیل، یک برنامهٔ کاربردی جاوا تولید می‌شود. کلاس BamCity.java فایل اصلی این برنامه است که با اجرای آن، نمونه‌ای از همهٔ عناصر مستقر در محیط ایجاد می‌شود. با اجرای این فایل، نمایه‌ای از محیط، عامل‌ها، وظایف، سازمان‌ها و عناصر تحت‌تأثیر حادثه نمایش داده می‌شود و مدیر بحران می‌تواند با انتخاب گزینه‌های Organize Team، Assign Tasks و Do Tasks که در برنامه تعبیه شده است، عملیات طراحی‌شده در مدل را مشاهده کند. در این بخش، اجرای برنامهٔ کاربردی تولیدشده برای زلزلهٔ بم، نمایش داده شده است.

با اجرای برنامه، ابتدا نقشهٔ جغرافیایی شهر بم، و سایر عواملی که در بخش ۱.۵ نام برده شد بر اساس مکان و تصویری که مدل‌ساز در مدل برای آن‌ها انتخاب کرده است، روی نقشه نمایش داده می‌شوند. همان طور که در شکل (۱۱) نشان داده شده، در پنل سمت راست، ابتدا فقط کلید Organize Team فعال است و کاربر (مدیر بحران) می‌تواند با فشردن آن نحوهٔ تشکیل تیم‌ها را در سازمان مربوط مشاهده کند. تیم‌ها، سازمان مربوط و اعضای هر تیم بر اساس آنچه در زمان مدل‌سازی طراحی شده است، مشخص می‌شوند.



شکل (۱۱): نمایی از برنامه در شروع اجرا



شکل (۱۲): نمایی از برنامه پس از تشکیل تیم‌ها

در شکل (۱۲)، دو تیم اورژانس در سازمان هلال احمر تشکیل می‌شود. تیم ۱ شامل دو پزشک و تیم ۲ شامل یک امدادگر و یک خلبان است. با تشکیل هر تیم و اضافه‌شدن اعضا به آن، پیغام‌های مربوط در پنل سمت راست نمایش داده می‌شود.

پس از تشکیل تیم‌ها، کلید Assign Task فعال می‌شود که با انتخاب آن، وظایف موجود در محیط به هریک از تیم‌های مربوط (بر اساس آنچه مدل‌ساز هنگام مدل‌سازی، طراحی کرده است) تخصیص می‌یابد. در اینجا دو وظیفهٔ امدادرسانی در

محیط تعریف شده که یکی از آن‌ها به تیم ۱ و دیگری به تیم ۲ اختصاص می‌یابد (با قرار گرفتن تصویر مربوط به وظایف در مجاورت تیم‌ها). تخصیص وظایف در شکل (۱۳) نشان داده شده است. پس از اختصاص وظایف به هریک از تیم‌ها، کلید Do Tasks فعال می‌شود. با فشردن این کلید، عامل‌ها به‌سمت هدف (عنصری که وظیفه روی آن اثر می‌گذارد) حرکت می‌کنند. بنا بر آنچه هنگام مدل‌سازی مشخص شده است، هر وظیفهٔ امدادرسانی روی یک عامل آسیب‌دیده اثر خواهد گذاشت. اینکه کدام وظیفه روی کدام عامل باید اثر بگذارد، در زمان مدل‌سازی با تعیین ویژگی Affect-on برای هر وظیفه، مشخص شده است. لذا اعضای هر تیم به‌سمت عامل آسیب‌دیدهٔ مربوط حرکت می‌کنند. همان طور که در بخش ۳.۴ بیان شد، با توجه به اینکه در اینجا عامل تحت‌تأثیر وظایف، عامل‌های آسیب‌دیده هستند، برای نشان‌دادن انجام وظایف امدادرسانی، عامل آسیب‌دیده از محیط حذف و تصویر مربوط به وظیفهٔ امدادرسانی، جایگزین آن‌ها خواهد شد. این بدین معناست که تیم‌ها وظیفهٔ امدادرسانی خود را با موفقیت انجام داده‌اند. همچنین پیغامی مبنی بر این امر، در پنل سمت راست نشان داده می‌شود. در شکل (۱۴)، نمایی از برنامه پس از انجام وظایف در محیط نشان داده شده است.

شکل (۱۳): نمایی از برنامه پس از تخصیص وظایف به تیم‌ها

شکل (۱۴): نمایی از برنامه پس از انجام وظایف

۶. نتیجه‌گیری

در این مقاله به ارائهٔ راهکاری برای بصری‌سازی سناریوی طراحی‌شده در مدل پرداخته شد. برای این منظور از زبان مدل‌سازی ERE-ML، چارچوب JAMDER و کدهای تبدیل برای تبدیل خودکار مدل به کد استفاده شد. از مزایای کار پیشنهادی می‌توان به مزایای استفاده از رویکرد مدل‌رانده به‌منظور مشاهدهٔ نتایج برای ذی‌نفعی که الزاماً دانش توسعهٔ نرم‌افزار ندارد ولی می‌خواهد از نتایج حاصل از مدل‌سازی یک سناریوی واکنش اضطراری آگاهی یابد، اشاره کرد.

گفتنی است که در این پژوهش، فقط جنبه‌های ایستای سیستم در قالب یک نمودار کلاس طراحی شدند. یک سیستم چندعاملی جنبه‌های پویایی نیز دارد که از جملهٔ آن‌ها می‌توان به چگونگی برقراری ارتباط بین عامل‌ها، ایجاد هماهنگی بین آن‌ها و موارد دیگر اشاره کرد. در نظر گرفتن جنبه‌های پویای یک سیستم چندعاملی، برای شبیه‌سازی محیط‌های واکنش اضطراری، به‌عنوان کارهای آتی این پژوهش پیشنهاد می‌شود.

ضمایم

کدهای تبدیل نوشته‌شده برای نحوهٔ انجام وظایف در یک محیط واکنش اضطراری، به زبان MTL به‌صورت نشان داده‌شده در شکل‌های (۱۵) و (۱۶) است.

Task.mtl

```
@Override
public void action() {
    [if (t.affect_on.oclIsTypeOf(DangerClass))]
          Point target =
environment.getDanger("[t.affect_on.name/]").Location;

     environment.mainframe.remove(environment.mainframe.getComponentAt(target)
);
          environment.getDanger("[t.affect_on.name/]").Location=null;
     [elseif (t.affect_on.oclIsTypeOf(AgentClass))]
          Point target = environment.getAgent("[t.affect_on.name/]").Location;

     environment.mainframe.remove(environment.mainframe.getComponentAt(target)
);
          environment.getAgent("[t.affect_on.name/]").Location=null;
     [elseif (t.affect_on.oclIsTypeOf(Class))]
          Point target =
environment.getObject("[t.affect_on.name/]").Location;
     [/if]
          Component c =
environment.mainframe.getComponentAt(environment.getTask(name).Location);
          c.setBounds(target.x, target.y , c.getWidth(), c.getHeight());
          environment.getTask(name).Location = target;
          done = true;
      }
[/if]
}
```

شکل (۱۵): قطعه کد تبدیل مربوط به انجام وظایف توسط تیم‌ها

Organization.mtl

```
env.DoTaskButton.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
    [for (a:Assign | org.ownedAssign)]
      [for (t:EmergencyManagmentTeam  | a.OwnedTeam )]

        [for (m:AgentClass | t.Members)]
          env.getAgent("[m.name/]").addBehaviour(env.getTask("[a.AssignSourc
    e.name/]"));
        [/for]
        ActionListener taskPerformer[org.ownedTeam->indexOf(t)/] = new
ActionListener() {

          [for (m:AgentClass | t.Members)]
            [let i : Integer = (org.ownedTeam->indexOf(t)-1)*2 + (t.Members-
>indexOf(m)-1)]
              Component C[i/] =
env.mainframe.getComponentAt(env.getAgent("[m.name/]").Location);
              Point P[i/] = env.getAgent("[m.name/]").Location;
              boolean flag[i/] = false;
            [/let]
          [/for]
          [let i : Integer = org.ownedTeam-> indexOf(t)-1]
            [if (a.AssignSource.affect_on.oclIsTypeOf(DangerClass))]
              Point target[i/] =
env.getDanger("[a.AssignSource.affect_on.name/]").Location;
            [elseif (a.AssignSource.affect_on.oclIsTypeOf(AgentClass))]
              Point target[i/] =
env.getAgent("[a.AssignSource.affect_on.name/]").Location;
            [elseif (a.AssignSource.affect_on.oclIsTypeOf(Class))]
              Point target[i/] =
env.getObject("[a.AssignSource.affect_on.name/]").Location;
            [/if]
          [/let]

        public void actionPerformed(ActionEvent evt) {
          [for (m:AgentClass | t.Members)]
            [let i : Integer = (org.ownedTeam->indexOf(t)-1)]
             [let j : Integer = i*2 + (t.Members->indexOf(m)-1)]
             if (P[j/].x == (target[i/].x-30 + [(t.Members->indexOf(m)-
1)/]*50) && P[j/].y == (target[i/].y-70))
                flag[j/] =  true;

              else if (P[j/].x < (target[i/].x-30 + [(t.Members->indexOf(m)-
1)/]*50))
                P[j/].x++;
              else if (P[j/].x > (target[i/].x-30 + [(t.Members->indexOf(m)-
1)/]*50))
                P[j/].x--;

              if (P[j/].y < (target[i/].y-70))
                P[j/].y++;
              else if (P[j/].y > (target[i/].y-70))
                P[j/].y--;
                C[j/].setBounds(P[j/].x, P[j/].y, C[j/].getWidth(),
C[j/].getHeight());
                env.getAgent("[m.name/]").Location = P[j/];
             [/let]
            [/let]
          [/for]
          [for (m:AgentClass | t.Members)]
            if(flag[(org.ownedTeam->indexOf(t)-1)*2 + (t.Members-
>indexOf(m)-1)/]){
          [/for]
// do Tasks
     env.getTask("[a.AssignSource.name/]").action();
          if(env.getTask("[a.AssignSource.name/]").done())
            env.reportText.append("[a.AssignSource.name/] is done by
[t.name/].\n");
          else
            env.reportText.append("[a.AssignSource.name/] is not done by
[t.name/].\n");
            ((Timer) evt.getSource()).stop();
            [for (m:AgentClass | t.Members)]
        }
      [/for]
    }
};
          Timer t[org.ownedTeam->indexOf(t)/] = new Timer(1,
taskPerformer[org.ownedTeam->indexOf(t)/]);

          t[org.ownedTeam->indexOf(t)/].start();

      [/for]
   [/for]

    }
});
}
```

شکل (۱۶): کد تبدیل برای متد action از کلاس Task

مراجع

[1] Mustapha, K., Mcheick, H. and Mellouli, S., "Modeling and simulation agent-based of natural disaster complex systems", in *Proceedings of the 4th International Conference on Emerging Ubiquitous Systems and Pervasive Networks*, pp. 148–155, 2013.

[2] Thabet, I., Chaawa, M. and Ben Said, L., "A multi-agent organizational model for a snow storm crisis management", in *Proceedings of the International Conference on Information Systems for Crisis Response and Management in Mediterranean Countries*, pp. 143–156, 2014.

[3] Gascueña, J. M., Navarro, E. and Fernández-Caballero, A., "Model-driven engineering techniques for the development of multi-agent systems", *Engineering Applications of Artificial Intelligence*, vol. 25, no. 1, pp. 159–173, 2012.

[۴] آدم‌زاده، طاهره، **یک زبان مدل‌سازی به‌منظور توسعهٔ سیستم‌های چندعاملی برای محیط‌های واکنش اضطراری**، دانشکده مهندسی کامپیوتر، دانشگاه اصفهان، پایان‌نامه کارشناسی ارشد، ۱۳۹۳.

[5] Lopes, Y. S., "Desenvolvimento orientado a modelos em sistemas multi-agentes com diferentes arquiteturas internas de agente", M.Sc. thesis, Ceará State University, 2012.

[6] Bauer, B., Müller, J. P. and Odell, J., "Agent UML: A Formalism for Specifying Multiagent Interaction", *International Journal of Software Engineering and Knowledge Engineering*, Vol. 11, No. 3, pp. 207–230, 2001.

[7] OMG, "Unified Modeling Language (UML) - Version 2.5." [Online]. Available: http://www.omg.org/spec/UML/2.5/. [Accessed: 26-Mar-2017].

[8] Winikoff, M., "Towards making agent UML practical: A textual notation and a tool", in *Proceedings of the International Conference on Quality Software*, pp. 401–406, 2005.

[9] Hahn, C., "A Domain Specific Modeling Language for Multiagent Systems", in *Proceedings of 7th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2008)*, pp. 233–240, 2008.

[10] Jorge, R. F. and Gómez-Sanz, J., "Agent Oriented Software Engineering with INGENIAS", in *Proceedings of the International Central and Eastern European Conference on Multi-Agent Systems*, pp. 394–403, 2003.

[11] Pavón, J., Gómez-Sanz, J. and Fuentes, R., "The INGENIAS Methodology and Tools", in *Agent-oriented methodologies*, B. Henderson-Sellers and P. Giorgini, Ed. IGI Global, 2005, pp. 236–276.

[12] Gómez-Sanz, J. J., Fernández, C. R. and Arroyo, J., "Model driven development and simulations with the INGENIAS agent framework", *Simulation Modelling Practice and Theory*, Vol. 18, No. 10, pp. 1468–1482, 2010.

[13] Da Silva, V. T., Choren, R. and De Lucena, C. J. P., "MAS-ML: a multiagent system modelling language", *International Journal of Agent-Oriented Software Engineering*, Vol. 2, No. 4, pp. 382–421, 2008.

[14] Silva, V. A., Garcia, A., Brandao, C., Chavez, Lucena, C. and Alencar, P., "Taming agents and objects in software engineering", *International Workshop on Software Engineering for Large-scale Multi-agent Systems*, pp. 1–26, 2002.

[15] Gonçalves, E. J. T., Farias, K., Cortés, M. I., Feijó, A. R., Oliveira, F. R. and Da Silva, V. T., "MAS-ML tool: A modeling environment for multi-agent systems", in *Proceedings of the 13th International Conference on Enterprise Information Systems (ICEIS 2011)*, pp. 192–197, 2011.

[16] Lopes, Y. S., Goncalves, E. J. T., Cortés, M. I. and Freire, E. S. S., "A MDA Approach Using MAS-ML 2.0 and JAMDER", in *13th International Workshop on Agent-Oriented Software Engineering*, 2012.

[17] Othman, S. H., Beydoun, G. and Sugumaran, V., "Development and validation of a disaster management metamodel", *Information Processing and Management*, Vol. 50, pp. 235–271, 2014.

[18] Othman, S. H. and Beydoun, G., "A Metamodel-based Knowledge Sharing System for Disaster Management", *Expert Systems with Applications*, Vol. 63, pp. 49–65, 2016.

[19] Betke, H., "Structure and Elements of Disaster Response Processes – A General Meta-Model", in *Proceedings of the Information Systems for Crisis Response and Management Conference (ISCRAM 2015)*, pp. 1-5, 2015.

[20] Othman, S. H. and Beydoun, G., "Metamodelling Approach To Support Disaster Management Knowledge Sharing", in *21st Australasian Conference on Information Systems (ACIS 2010)*, pp. 1–10, 2010.

[21] Adamzadeh, T., Zamani, B. and Fatemi, A., "A modeling language to model mitigation in emergency response environments", in *Proceedings of the 4th International Conference on Computer and Knowledge Engineering (ICCKE 2014)*, pp. 302–307, 2014.