پایش رفتار نرم افزار و تطبیق آن با نیازها در سیستم های مبتنی بر هدف

عبداله آقایی[۱] ، عزیزاله رحمتی[۲]
[۱]دانشگاه آزاد اسلامی واحد تویسرکان aghaei_arak1384@yahoo.com
[۲]دانشگاه آزاد اسلامی واحد کنگاور، m_aziz_rahmati@yahoo.com

چکیده

در مهندسی نرم افزار، نیازها، ابزار لازم برای رسیدن به هدف کاربران هستند. در این ابزار، نوع نیازها، عامل های مسئول نیازها و محیط انجام نیازها تعیین می شود. عامل های مسئول نیازها متعهد به ارضاء نیازها در محیط سیستم هستند. پایش بر رفتار عامل نرم افزار یکی از مسائل مطرح در مهندسی نیازها است، ما در این مقاله از توصیف سیستم های هدف گرا به روش KAOS بهره میگیریم و یک مدل توصیفی برای پایش و تطبیق رفتار عامل نرم افزار با نیازهای هدف ارائه می دهیم، سپس این مدل را برای مسأله سیستم اعزام آمبولانس لندن به کار می بریم تا روش پیشنهادی را در عمل نشان دهیم.

کلمات کلیدی

KAO ، توسعه مبتنی بر هدف، پایش حین اجرا، تطبیق نیازها.

۱- مقدمه

دراین مقاله در مورد انحراف رفتار حین اجرای سیستم از نیازمندی های مشخص شده در زمان توسعه بحث می کنیم. چنین انحرافی ممکن است به دلیل، غیر قابل پیش بینی بودن، رفتارهایی که عوامل محیطی ممکن است بروز دهند، در هنگام تعیین مشخصات، یا شرایط نتیجه شده از محیط، باشد [۱].

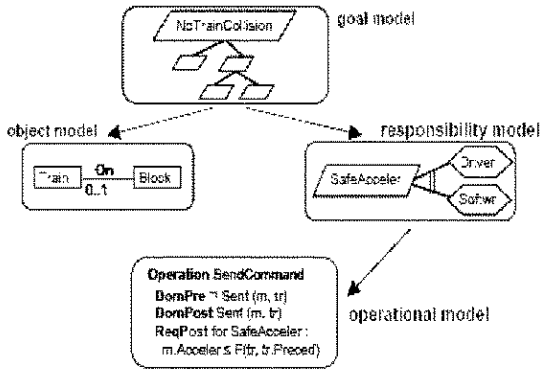
فرایند مهندسی نیازمندی ها اغلب براساس مشخصاتی است که به اندازه کافی مبتنی بر واقعیت نیستند، زیرا این مشخصات در حین اجرای سیستمی که آنها را پیاده سازی می کند، زمانی به زمان دیگر نقض می شوند[۱]. دلایل این امر عبارتند از: ۱- عامل های محیطی ممکن است به نحوی رفتار کنند که در زمان تعیین نیازمندی ها غیر قابل پیش بینی و مدل کردن آنها ناممکن باشد[۲]. ۲- تکامل آرایش شرایط محیطی بر اساس فرضیات ابتدایی می باشد، که این فرضیات، در ابتدا در محیط قابل پذیرش هستند، نه برای مدت طولانی[3]. دو روش مکمل برای مدیریت نقض حین اجرای نیازمندی ها می توانند به

صورت زیر مطرح شوند[۵,۴]: ۱- پیش بینی نقض ها در زمان توصیف تا حداکثر ممکن: موانع بر سر راه تحقق نیازمندی ها، اهداف، و فرضیات از طرح اولیه مشخصات به دست می آیند، بنابراین با شناسایی موانع، مشخصات مطمئن تری حاصل می شود[۶]. ۲- کشف و برطرف نمودن چنین نقض هایی در حین اجرا.

در هنگام تحلیل موانع هزینه زیادی برای دست یابی به سیستمی مطمئن تحمل می شود و شناسایی کامل تمام موانع، غیرممکن است. از این گذشته تعیین مشخصات مطمئن، ممکن است برای پیاده سازی پرهزینه بوده و برای نرم افزار پیچیدگی غیر ضروری نتیجه دهد [۳] پس، از تحلیل حین اجرا استفاده می کنیم. روش ما شامل سه مرحله مختلف است. در مرحله اول سیستم تحت توسعه را با روش مبتنی بر هدف KAOS توصیف می نماییم، سپس ادعاهایی که ممکن است نقض شوند شناسایی می شوند و از روی این ادعا ها کد پایشگر را مشخص می کنیم. کد پایشگر در واقع نقیض هدف مورد نظر است. بیان رسمی در هدف، به صورت منفی بیان می شود. یعنی در واقع تحقق این وضعیت برابر با شکست در هدف است و پایشگر باید رفتار حین اجرای سیستم را تحت نظر داشته باشد تا اگر این شرایط برقرار شد به تطبیق دهنده خبر دهد. در روش ما این کد به زبان FLEA ترجمه می شود. در هنگام اجرای سیستم، بخش پایشگر رفتار عامل ها را پایش می کند و پیغام مناسب را برای مسئول تطبیق سیستم تولید می کند. در مدل پیشنهادی برای تطبیق از روش تطبیق تک نقطه استفاده نموده ایم و این روش را در مدل پایش و تطبیق گنجانده ایم. مسئول تطبیق به پیام های تولید شده توسط پایشگر سرکشی می کند و با دریافت پیام درخواست تطبیق، سیستم را به حالت جدید یا برنامه مقصد تطبیق می دهد. در روش تطبیق تک نقطه، در نقطه خاصی از زمان، اجرای برنامه منبع متوقف و برنامه مقصد شروع به کار می کند. پس از تطبیق سیستم در صورت نیاز کد پایشگر برای پایش شرایط جدید تغییر می کند و این روند تکرار می شود.

۲- مهندسی نیازمندی ها مبتنی بر هدف به روش KAOS

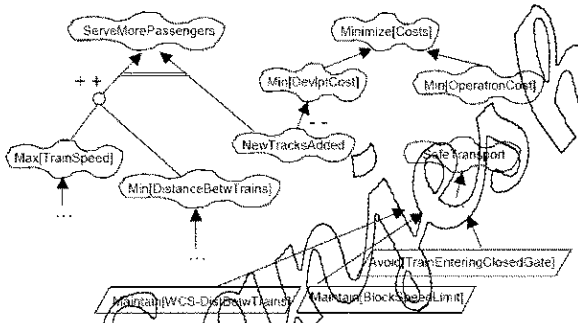
یک هدف مقصودی می باشد که سیستم باید با همکاری عامل ها در نرم افزار جدید و محیط به آن برسد. هدف نقشی برجسته در فرایند مهندسی نیازمندی ها دارد [۷و۸]. بنابراین روش های مبتنی بر هدف برای مهندسی نیازمندی ها مورد توجه بیشتری قرار گرفته- اند. دو چارچوب مکمل برای اجتماع اهداف و پالایش اهداف در مدل های نیازمندی ها پیشنهاد شده است. چارچوب رسمی و چارچوب کیفی. در چارچوب رسمی [۷] ، اهداف می توانند به طور شبه رسمی یا رسمی مشخص شوند. پالایش اهداف توسط گراف های AND/OR مشخص می شود. در پیوندهای پالایش AND دلیل کافی برای ارضاء هدف ارضاء همه زیر هدف ها در پالایش است. پیوندهای پالایش OR به هدفی با یک سری جایگزین از ارضاء ها مربوط است؛ ارضاء یکی از زیرشاخه ها شرط کافی برای ارضاء هدف است. اهداف با منطق زمانی بلادرنگ بیان می شوند درحالیکه کاربردها/ اشیاء بوسیله ثابت ها و پیش/پس شرط ها فرمولبندی می شوند. در زمینه مهندسی نیازمندی ها، طرح های رسمی برای بررسی ارضاء هدف ها [۹] ، کاربردی سازی[۱۰,۷] ، کشمکش [۸] ، کارشکنی [۱۱] ، انتصاب به عامل ها [۱۰] ، استنتاج از طرح ها [۱۲]، و یادگیری با مقایسه [۱۳] قابل بررسی هستند. در چارچوب کیفی [۱۴] ، انواع پیوند ضعیف تری برای برقرار کردن ارتباط هدف های نرم استفاده شده اند[۱۴]. شکل(۱) مراحل روش KAOS را با نمایش زیرمدل های آن نشان می دهد.



شکل (۱) مهندسی نیازمندیهای مبتنی بر هدف با KAOS [۱۵]

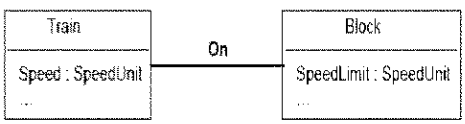
مدل شکل (۱) شامل چهار زیرمدل است. مدل هدف، مدل شیء و مدل مسئولیت که از مدل هدف به دست می آیند و مدل عملیاتی که از مدل مسئولیت به دست می آید. مراحل روش KAOS به صورت زیر می باشد:

۱- شناسایی هدف از اسناد اولیه(شکل۲)



شکل(۲) مدل اولیه اهداف برای سیستم BART [۱۵]

۲- فرموله کردن اهداف و استخراج مدل شئ (شکل ۳).



شکل(۳) بخشی از مدل شئ [۱۵]

۳- تشخیص انتصاب مسئولیت های جایگزین
۴- ایجاد واسط های عامل
۵- شناسایی عملیات.
۶- عملی کردن اهداف

۳- سیستم نظارتی FLEA

قسمتی از مدل که پایش رفتار حین اجرای سیستم را بر عهده دارد ، پس از اینکه به کمک روش KAOS توسعه یافت، به زبان رویدادگرای FLEA ترجمه می شود.

زبان FLEA ساختار هایی مشخص برای ترکیبات زمانی رویداد ها فراهم می کند. کد حین اجرای لازم برای پایش بر چنین سیستم هایی به صورت خودکار تولید می شود. سیستم حین اجرا شامل یک سیستم مدیریت پایگاه داده های سوابق، مجهز به ماشین واسط، مکانیسم ارتباطی برای گرد آوردن رویدادها و توزیع اختار های وقوع ترکیب رویدادها می باشد [۱۶].

FLEA رویدادها را به صورت ارتباط هایی خاص که اولین پارامتر آن زمان وقوع رخداد است، نمایش می دهد. دیگر پارامتر ها مشخصات رویداد هستند.

جدول (۱) الگوهای زمانی گوناگونی که در FLEA قابل تعریف هستند را نشان می دهد.

جدول(۱) الگوی زمانی در FLEA [۱]

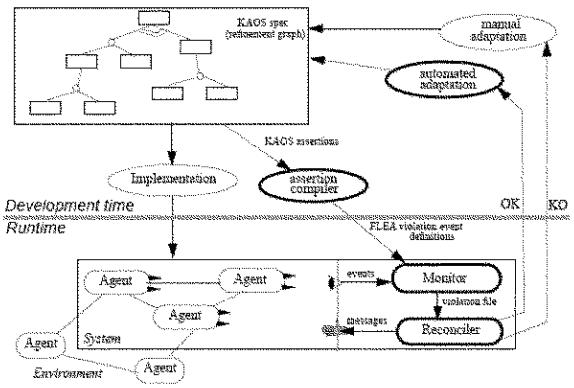
Syntax	Meaning
then P Q	an event *P* followed by an event *Q*
then-excluding P Q R	an event *P* followed by an event *Q*, without any event *R* in-between
in-time P Q d	an event *P* followed by an event *Q* within time delay *d*
too-late P Q d	an event *P* **not** followed by an event *Q* within time delay *d*

۳-۱- فرایند تطبیق حین اجرای نیازمندی ها

شکل (۴) دو لایه شامل فرایند تطبیق رفتار نیازمندی ها را نشان می دهد. در لایه توسعه، توصیف مبتنی بر هدف انجام می شود؛ ادعا- های KAOS که می توانند نقض شوند شناسایی شده و به طور خودکار به تعاریف رویداد FLEA ترجمه می شوند. برای شناسایی انحراف بین رفتار مشخص شده و رفتار حین اجرا، پایشگر باید سیستم در حال کار را پایش کند. دو روش می تواند مطرح شود: (۱) محدود نمودن مشاهده به قسمت های خودکار سیستم، یا (۲) مشاهده تا نهایت ممکن شامل مقادیری از محیط. این دو امکان به ترتیب برابر با تصور پایشگر داخلی و خارجی، می باشد [۱۷]. روش دوم به واسط اختصاصی که برای دستیابی به وضعیت عوامل محیطی طراحی شده است، نیاز دارد.

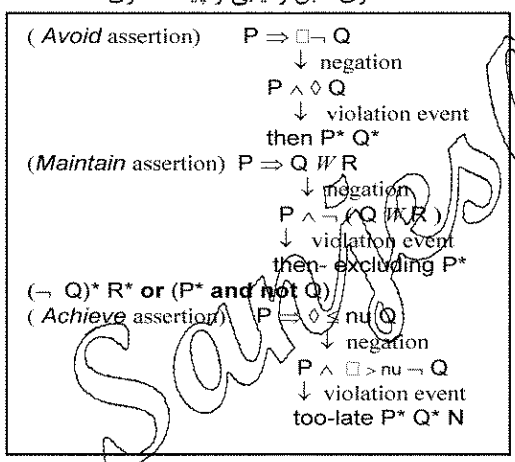
۳-۲- لایه توسعه

D1 . توسعه گراف پالایش/ عملی کردن هدف، شناسایی ادعا های شکننده در مشخصات، و فرموله نمودن آنها [۵].



شکل (۴) معماری کلی سیستم خود تطبیقی [۱]

D2 . بررسی قابلیت پایش و شناسایی پارامترهای نظارتی.
D3 . شناسایی روش های تطبیق.
D4 . ترجمه ادعاهای شکننده به FLEA (شکل۵) .
D5 . ساخت معماری قابل ردیابی و پیاده سازی.



شکل (۵) بیان اثبات های KAOS با FLEA [۱]

۳-۳- سطح حین اجرا

R1 . قابل دسترس نمودن اطلاعات حالت برای پایشگر.
R2 . به روز رسانی رویدادهای FLEA در صورت لزوم.
R3. تطبیق نیازمندی های سیستم و رفتار حین اجرا.

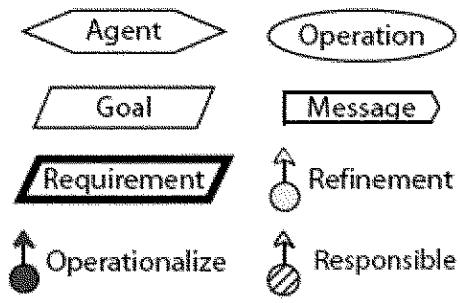
ما در این مقاله برای بخش تطبیق دهنده از تطبیق تک نقطه استفاده نموده ایم، زیرا سیستم ما با انتخاب جایگزین کار می کند که ماهیت تطبیق تک نقطه براساس انتخاب جایگزین است.

۴- مفهوم تطبیق

در این بخش روش تطبیق سیستم را به طور مختصر با استفاده از مثال بیان نموده و روش تطبیق تک نقطه [۱۸] را شرح می دهیم.

۴-۱- عناصر KAOS به کار رفته در مدل پیشنهادی

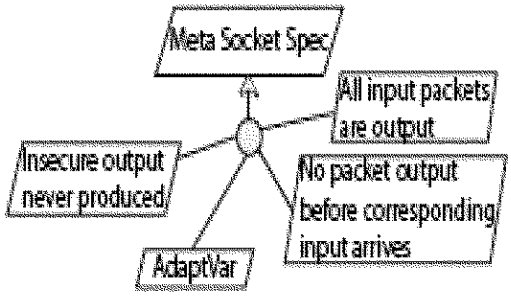
در شکل (۶) عناصر گرافیکی به کار رفته در مدل های KAOS را نمایش داده ایم. جزئیات در مورد KAOS در [۱۹و۲۰] یافت می شود.



شکل(۶) عناصر KAOS به کار رفته در مقاله ما

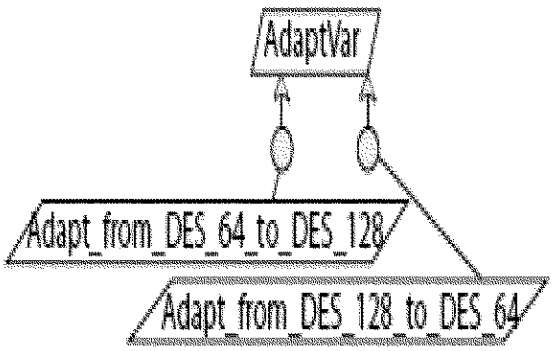
۴-۲- تطبیق تک نقطه

تطبیق تک نقطه به این مفهوم است که سیستم تطبیقی در حال کار کردن با برنامه فعلی است که به آن برنامه منبع می گوییم. سیستم دارای این قابلیت است که با روش جایگزین دیگری هم می تواند کار کند که به آن برنامه مقصد می گوییم. حال اگر در صورت نیاز در نقطه خاصی از زمان اجرا، روند برنامه منبع متوقف شود و سیستم با برنامه جدید یا برنامه مقصد شروع به کار کند، می گوییم تطبیق تک نقطه صورت گرفته است. این روش تطبیق مناسب سیستم هایی است که قابلیت کار به حداقل دو روش جایگزین را دارند. ژانگ و چنگ [۲۱] مفهوم تطبیق را برای برنامه ای که متا سوکت را برای انجام کدگذاری و کدگشایی DES 64 و DES 128 به کار می گیرد، ارائه داده اند. DES 64 نشان دهنده کدکننده و کد گشای ۶۴ بیتی و DES 128 نشان دهنده کدکننده و کد گشای ۱۲۸ بیتی است. متا سوکت نوع خاصی از سوکت است که از کلاس سوکت موجود در جاوا ساخته شده است، اما معماری و رفتار آن می تواند در تطبیق به منظور پاسخ به تحریک خارجی در حین اجرا، تغییر کند[۲۲]. متا سوکت می تواند بنا بر شرایط محیط اجرا در یکی از حالت های مشخص شده DES 64 و یا DES 128 کار کند. در شکل(۷) مدل هدف برای متاسوکت ارائه شده است.



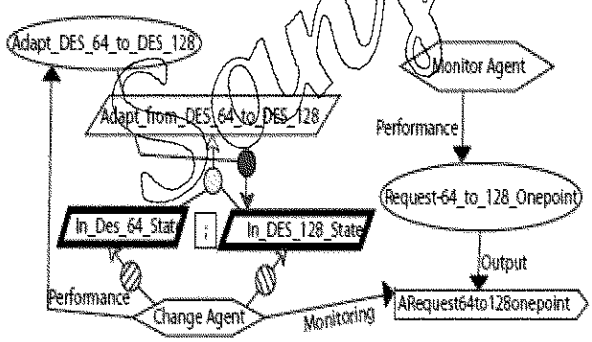
شکل (۷) درخت هدف برای مثال متاسوکت [۱۸]

هدف Achieve[AdaptVar] در شکل (۸) یک پالایش OR است. یعنی هدف AdaptVar که به معنی تغییر روش تطبیق است به دو زیرشاخه که می توانند جایگزین دیگری شوند تصفیه شده است یکی از این زیرهدف ها تغییر روش تطبیق از کد/دیکد ۶۴ بیتی به کد/دیکد ۱۲۸ بیتی و دیگری عکس این تطبیق را نشان می دهد. پالایشOR به پالایش یک هدف به زیرشاخه های جایگزین گفته می شود. یعنی با ارضاء هر زیر شاخه هدف ریشه قابل تحقق است.



شکل(۸) پالایشOR برای هدف **AdaptVar**[۱۸]

تطبیق تک نقطه با تطبیق سیستم از برنامه منبع به برنامه مقصد در یک نقطه خاص در حال اجرا مشخص می شود[۲۱]. شکل (۹) بیان KAOS مثال تطبیق DES 64 to DES 128 را نشان می دهد.

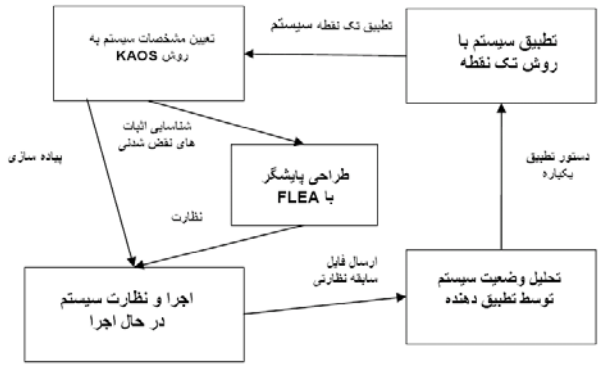


شکل(۹) مدل KAOS برای مفهوم تطبیق تک نقطه [۱۸]

در شکل (۹) عامل تغییر مسئول تحقق برنامه منبع یعنی In_Des_64_State و تحقق برنامه مقصد یعنی In_DES_128_State است. پایشگر دستور تطبیق را صادر می کند . پیام تطبیق ساخته شده و مسئول تغییر آنرا می خواند. عامل تغییر با عمل Adapt_DES_64_to_DES_128 برنامه منبع را متوقف و برنامه مقصد را اجرا می کند. در نتیجه هدف تطبیق یعنی Adapt_from_DES_64_to_DES_128 ارضاء می شود.

۵- مدل پیشنهادی

در اینجا مدل ترکیبی خود (شکل ۱۰) را رائه نموده و تشریح می کنیم. در مدل شکل (۴) روش مناسبی برای پایش حین اجرای سیستم مبتنی بر هدف ارائه شده است اما برای تطبیق سیستم روش خاصی پیشنهاد نشده است. در مدل شکل (۹) تطبیق تک نقطه ارائه شده است. این مدل برای سیستم هایی که حداقل دو روش کارکرد متفاوت ارائه شده است، و برای تحقق یک هدف از روش های جایگزین متفاوت استفاده نشده است. مدل ما این مزیت را دارد که هم از پایش حین اجرای پایشگر رویدادگرای FLEA بهره می برد و هم برای تطبیق سیستم با وضعیت جدید از روش تطبیق تک نقطه استفاده می کند. ما سیستم را به روش مدل شکل(۴) پایش می کنیم و سپس با روش تطبیق تک نقطه تطبیق می دهیم، اما برخلاف مدل شکل (۹)، ما یک هدف واحد را از روشی جایگزین در درخت هدف AND/OR تحقق می بخشیم. در واقع مدل پیشنهادی بخش پایش را از مدل شکل (۴) و بخش تطبیق را از مدل شکل (۹) با اندکی تغییرات، به همراه هماهنگ سازی اجزاء مدل در بر گرفته است. مدل ما مزیت هر دو مدل را در یک مدل جمع بندی کرده است. قسمت اول مدل ما، تعیین مشخصات سیستم به روش KAOS می باشد. در این قسمت که در زمان توسعه سیستم انجام می شود، اهداف و نیازمندی های سیستم با گراف های AND/OR مشخص می شوند. بعد از اینکه مسئولیت تحقق اهداف به عامل ها واگذار شود سیستم می تواند اجرا شود. کار دیگر در هنگام توسعه، وارسی سیستم به منظور پیش بینی و رفع خطاهای ممکن است. از آنجا که پیش بینی همه حالات مستعد خطا امکان پذیر نیست و هزینه زیادی دارد، ما اهدافی را که ممکن است در حین اجرای سیستم نقض شوند مشخص می کنیم و با شناسایی این اهداف، پایشگر را طراحی می کنیم. پایشگر یک عامل است. وظیفه پایشگر پایش رفتار سیستم و مقایسه رفتار اجزا تحت پایش با وضعیت تعیین شده در هنگام تعیین مشخصات سیستم می باشد. پایشگر بر اساس مشاهده رویداد ها عمل می کند، پس ما از ابزار زبان FLEA که زبانی مبتنی بر رویداد است استفاده نموده و مشخصات پایشگر را که از درخت هدف استخراج شده است به این زبان ترجمه می نماییم. پس از این مرحله سیستم وارد محیط اجرا می شود. در این مرحله عامل ها سیستم را اجرا می کنند. پایشگر با پایش رفتار بخش های قابل پایش سیستم، فایل سابقه نقض های رفتاری را تهیه نموده و این فایل را برای تحلیل و تصمیم گیری به عامل تطبیق دهنده ارسال می کند. تطبیق دهنده پس از تحلیل نقض ها در صورت لزوم فرمان تطبیق سیستم به وضعیت جدید را صادر می نماید. اگر عامل تطبیق دهنده دستور تطبیق را صادر کند روش قدیمی اجرا یا برنامه منبع متوقف شده و برنامه مقصد یا روش جدید اجرای سیستم شروع می شود، یعنی بعد از تطبیق یک زیر شاخه جایگزین از درخت پالایش AND/OR مربوط به هدف سطح بالای سیستم انتخاب می شود.



شکل (۱۰) چارچوب مدل پیشنهادی

سپس در صورت لزوم، ساختار پایشگر بازسازی شده و سیستم اجرایی شده و این چرخه ادامه می یابد. روش KAOS برای توسعه سیستم های بسیاری به کار گرفته شده است، بخش پایشگر هم در مدل ارائه شده در بخش۳ ساخته شده و برای اینکه بتواند رویداد های سیستم را پایش نماید ، به زبان رویدادگرای FLEA ترجمه شده است. قسمت تطبیق دهنده، می تواند نرم افزار و یا عامل انسانی باشد. این بخش بر اساس سیاست های سیستم، وظیفه تصمیم گیری را بر عهده دارد. مثلاً می توانیم به بخش تطبیق دهنده اجازه دهیم که اگر نقضی برای بار اول به طور تصادفی ایجاد شد، دستور تطبیق را به تأخیر انداخته و در صورت تکرار نقض دستور تطبیق را صادر نماید.

۵-۱- طرح مسأله

به منظور نشان دادن عملکرد مدل پیشنهادی درخت هدف مربوط به مثال سیستم اعزام آمبولانس لندن[۱۳] را استفاده نموده ایم. شکل کلی مدل هدف این سیستم در شکل (۱۱) نمایش داده شده است. هدف ما استفاده از بخشی از این مثال و قرار دادن آن در مدل پیشنهادی، به منظور نشان دادن صحت مدل می باشد (شکل ۱۲). هدف ریشه در خت شکل (۱۱) به معنی ورود آمبولانس به صحنه حادثه است (AmbulanceIntervention). این هدف دو زیر هدف دارد که تحقق هدف ریشه وابسته به تحقق هر دو زیر هدف است. یک زیر هدف می گوید که آمبولانس اعزام شده (AmbulanceMobilized) و زیرهدف دیگر می گوید که آمبولانس اعزام شده به صحنه حادثه وارد شده

(MobilizedAmbulanceIntervention). در سطح بعدی زیرهدف اعزام آمبولانس (AmbulanceMobilized) خود به دو زیرهدف پالایش می شود. یک زیرهدف بیان می کند که نوع حادثه مشخص شده (IncidentFormEncoded) و زیر هدف دیگر می گوید که آمبولانس بر اساس نوع حادثه اعزام شده است (AmbulanceMobilizedBasedOnIncForm). در سطح بعد این هدف به دو زیرهدف دیگر پالایش می شود. زیر هدف (AmbulanceAllocated BasedOnIncForm) بیانگر این است که آمبولانس بر اساس نوع حادثه اختصاص یافته است و زیرهدف (AllocatedAmbulance Mobilized) بیان می کند که آمبولانس اختصاص یافته اعزام شده است. تا این سطح همه پالایش ها، پالایش AND هستند و در مدل، زیرشاخه جایگزینی برای تحقق هدف ها وجود ندارد. اما هدف Allocated Ambulance Mobilized دارای دو زیر شاخه با پالایش OR می باشد. این هدف می تواند به دو طریق متفاوت محقق شود. در زیرشاخه چپ اعزام از طریق ارتباط رادیویی و در زیرشاخه سمت راست اعزام از طریق پایانه داده های سیار (MDT) مورد بررسی قرار گرفته است. در زیر شاخه اول دو زیر هدف داریم. زیرهدف Ambulance mobilization communicated on radio به معنی اعزام آمبولانس بر اساس ارتباط رادیویی و زیرهدف Ambulance mobilized from radio mobilization به معنی این که آمبولانس یا اعزام رادیویی اعزام شده است. در زیرشاخه سمت راست زیر هدف Ambulance mobilization communicated on MDT به معنی اعزام آمبولانس بر اساس ارتباط با پایانه داده های سیار و Ambulance mobilized from MDT mobilization به معنی اینکه آمبولانس با اعزام از طریق پایانه داده های سیار اعزام شده است، می باشد. ما این دو زیرشاخه را به عنوان دو روش جایگزین که هر یک می توانند هدف والد را تحقق بخشند، مد نظر قرار داده ایم. در مثال شکل (۱۲) به منظور صرفه جویی در فضا، ما تنها قسمتی از توصیف سیستم که دارای ادعای نقض شدنی و پالایش جایگزین است را در قسمت توصیف سیستم به روش KAOS مطرح نموده ایم. دو هدف برای تطبیق سیستم در نظر گرفته ایم. یکی adapt from radio mob to MDT mob به معنی تطبیق از حالت ارتباط رادیویی به حالت ارتباط از طریق پایانه داده های سیار. این تطبیق زمانی انجام می شود که سیستم در حالت استفاده از ارتباط رادیویی باشد و به هر دلیلی ارتباط قطع شود. در این حالت باید از ارتباط رادیویی به ارتباط از طریق MDT روی بیاوریم. زیر هدف دیگر یعنی adapt from MDT mob to radio mob دقیقا عکس زیرهدف اول است. ما برای مثال زیرهدف اول را توسعه داده ایم و زیرهدف دوم هم به همین ترتیب قابل پیاده سازی در مدل است. بعد از پالایش هدف ها به زیراهداف، هر زیرهدف به عنوان مسئولیت به یک عامل واگذار می شود. ما زیرهدف های Ambulance mobilization communicated on MDT و Ambulance Mobilization Communicated OnRadio را به عامل ارتباط (communication agent)، زیرهدف دیگر یعنی Ambulance mobilized from radio mobilization را به عامل رادیو (Radio Agent) و زیرهدف Ambulance mobilized from MDT mobilization را به عامل MDT واگذار نموده ایم. برای ساخت پایشگر از تبدیل زیر استفاده می کنیم:

(*Achieve* assertion) $P \Rightarrow \Diamond \leq nu\ Q$

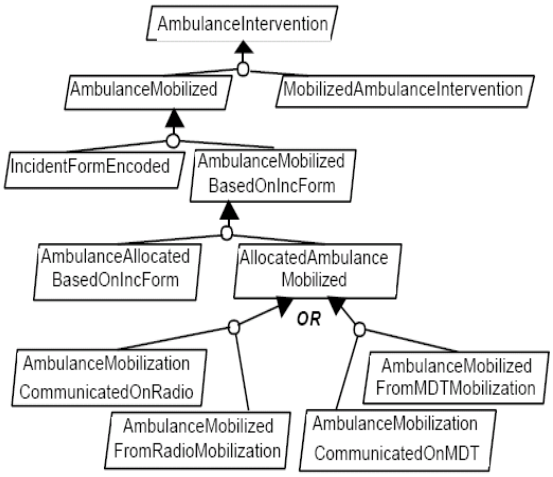
$\downarrow$ negation

$P \wedge \Box > nu \neg Q$

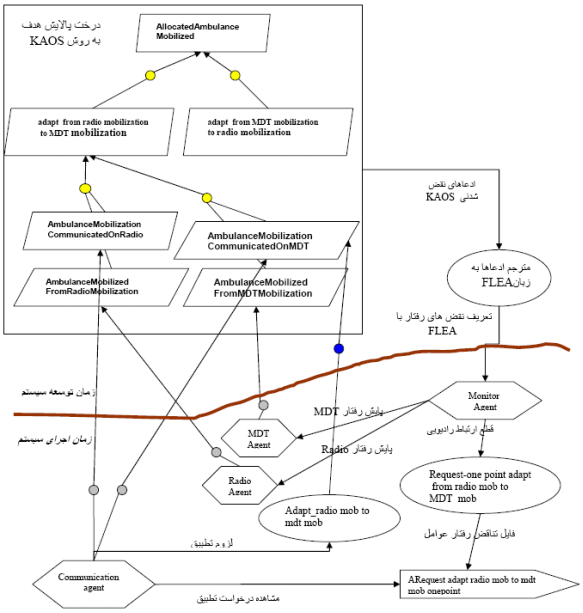
$\downarrow$ violation event

too-late P* Q* N

یعنی هدف ما این است که بعد از N واحد زمانی از برنامه منبع به برنامه مقصد برسیم، نقیض این هدف را تولید می کنیم و معادلش را در زبان FLEA یعنی too-late P* Q* N به عنوان کد پایشگر در نظر می گیریم.



شکل(۱۱) مدل اهداف برای سیستم اعزام آمبولانس[۱۳]



شکل(۱۲) جزئیات مدل پیشنهادی ما برای سیستم اعزام آمبولانس

در مدل ما P معادل هدف Ambulance mobilization communicated from radio و Q معادل هدف Ambulance mobilization communicated from MDT هستند. در هنگام اجرا عامل پایشگر رفتار عامل های رادیو و MDT را پایش می کند. در مثال ما رفتار فرستنده رادیویی مبنای کار است زیرا فرض می کنیم که ارتباط فعلی سیستم از طریق رادیو برقرار است. در صورت قطع ارتباط رادیویی، عامل پایشگر عمل درخواست Request-one point adapt from radio mob to MDT mob را انجام می دهد که سبب ایجاد یک پیام با همین نام می شود. عامل ارتباط که نوع ارتباط را مشخص نموده و ارتباط لازم در سیستم را برقرار می کند، مرتباً پیام را ملاحظه و تحلیل می کند و زمانی که درخواست تطبیق را مشاهده کند، عمل تطبیق Adapt_radio mob to mdt mob را انجام می دهد. در تکرار بعدی دیگر زیرسیستم adapt from MDT mob to radio mob باید توسعه یابد زیرا سیستم در حال استفاده از ارتباط از طریق MDT است و همچنین کد پایشگر نیز باید بر مبنای این نیاز طراحی شود.

۶- نتیجه گیری، پیشنهادات و کارهای آینده

مدل تطبیق تک نقطه برای سیستم هایی پیشنهاد شده است که ماهیت تطبیقی دارند، یعنی حداقل به دو روش خاص می توانند کار کنند. ما مدل تطبیق تک نقطه را در بدنه سیستم پایش حین اجرا برای جایگزین نمودن زیرشاخه جایگزین از درخت پالایش هدف، استفاده نمودیم. در این مقاله ما با ارائه مثالی ساده درستی مدل خود را نشان دادیم. در [۱] مدلی کلی برای پایش سیستم در حین اجرا ارائه شد اما مزیتی که مدل ما دارد تاکید بر بخش تطبیق دهنده و به طور خاص در این مثال استفاده از تطبیق تک نقطه است. همچنین در [۲۱] مدل تطبیق ارائه شده که ما ازاین مدل استفاده نمودیم ولی در مدل ما پایشگر مبتنی بر رویداد است. مدل تطبیق تک نقطه خاص سیستم های تطبیقی است که ذاتاً قابل تطبیق هستند اما ما این مدل را برای بالا بردن تحمل عیب سیستم و استفاده از زیرشاخه جایگزین به کار برده ایم که دلیل مزیت مدل ما است. در واقع ما مدلی ارائه نمودیم که سیستم را به صورت مبتنی بر هدف توسعه داده و مبتنی بر رویداد پایش می کند و در حین اجرا به وضعیت جدید تطبیق می دهد، که مزایای هر دو مدل را با هم همراه نموده است. در [۱۸] سه روش تطبیق مختلف ارائه شده است که می توان هریک از آنها را در بخش تطبیق دهنده مدل ما به کار گرفت. انتخاب تطبیق دهنده بستگی به نوع کاربرد مورد نظر دارد. همچنین به جای استفاده از زبان رویدادگرای FLEA می توان با زبان های جنبه گرا مانند ASPECTJ کد پایشگر را در مکان مورد نظر نگاشت. ما مدل خود را برای مثال ساده اعزام آمبولانس ارائه نمودیم که استفاده از مدل کلی برای کاربردهای دیگر می تواند موضوع کارهای جدید باشد.

منابع

[1] M. Feather, S. Fickas, A. van Lamsweerde, and C. Ponsard. Reconciling system requirements and runtime behavior. In IWSSD:Proceedings of the 9th international workshop on Software specification and design, page 50, 1998.

[2] C. Potts, “Using Schematic Scenarios to Understand User Needs”, Proc. DIS’95 - ACM Symposium on Designing interactive Systems: Processes, Practices and Techniques , University of Michigan, August 1995.

[3] S. Fickas and M. Feather, “Requirements Monitoring in Dynamic Environments”, Proc. RE’95 - 2nd International Symposium on Requirements Engineering , York, IEEE, 1995.

[4] A. van Lamsweerde, “Divergent Views in Goal-Driven Requirements Engineering”, Proc. Viewpoints’96 - ACM SIGSOFT Workshop on Viewpoints in Software Development , October 1996.

[5] A. van Lamsweerde, E. Letier, C. Ponsard. “Leaving Inconsistency”, Proc. ICSE’97 workshop on “Living with Inconsistency” , May 17, 1997

[6] A. van Lamsweerde, E. Letier. “Integrating Obstacles in Goal-Driven Requirements Engineering”, Proc. ICSE’98 - 20th International Conference on Software Engineering, Kyoto, April 1998.

[7] A. Dardenne, A. van Lamsweerde and S. Fickas, “Goal-Directed Requirements Acquisition”, Science of Computer Programming, Vol. 20, 1993, 3-50.

[8] A. van Lamsweerde, R. Darimont and E. Letier, “Managing Conflicts in Goal Driven Requirements Engineering”, IEEE Trans. on Sofware. Engineering, Special Issue on Inconsistency Management in Software Development, November 1998.

[9] R. Darimont and A. van Lamsweerde, “Formal Refinement Patterns for Goal-Driven Requirements Elaboration”, Proc. FSE’4 - Fourth ACM SIGSOFT Symposium on the Foundations of Software Engineering, San Francisco, October 1996, 179-190.

[10] E. Letier, Reasoning About Agents in Goal-Oriented Requirements Engineering. PhD Thesis, University of Louvain, 2001.

[11] A. van Lamsweerde and E. Letier, “Handling Obstacles in Goal-Oriented Requirements Engineering”, IEEE Transactions on Software Engineering, Special Issue on Exception Handling, October 2000.

[12] A. van Lamsweerde and L. Willemet, "Inferring Declarative Requirements Specifications from Operational Scenarios", IEEE Trans. on Sofware. Engineering, Special Issue on Scenario Management, December 1998, 1089-1114.

[13] P. Massonet and A. van Lamsweerde, “Analogical Reuse of Requirements Frameworks”, Proc. RE-97 - 3rd Int. Symp. on Requirements Engineering, Annapolis, 1997, 26-37.

[14] Mylopoulos, J., Chung, L., Nixon, B., “Representing and Using Nonfunctional Requirements: A Process-Oriented Approach”, IEEE Trans. on Sofware. Engineering, Vol. 18 No. 6, June 1992, pp. 483-497.

[15] A.Van.Lamsweerde. Building Formal Requirements Models for Reliable Software. Reliable Software Technologies, Ada-Europe’2001,Springer-Verlag Lecture Notes in Computer Science, LNCS 2043, May 2001

[16] D. Cohen, M. S. Feather, K. Narayanaswamy and S. Fickas, “Automatic Monitoring of Software Requirements”, Proc. 19th International conference on Software Engineering , Boston, May 1997.

[17] Bay Area Rapid Transit District, Advance Automated Train Control System, Case Study Description. Sandia NationalLabs,http://www.hcecs.sandia.gov/bart.htm .

[18] G.Brown, B. Cheng,, H. Goldsby, J. Zhang. Goaloriented Specification of Adaptation Requirements Engineering in Adaptive Systems. Internatinal conference on software engineering proceedings of the 2006. international workshot on self adaptation and self managing coy stems. Pages 23-29. year 2006.

[19] A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-directed requirements acquisition. Science of computer Programming, 20:3–50, 1993.

[20] E. Letier. Reasoning about Agents in Goal-Oriented Requirements Engineering. PhD thesis, Louvain-la-Neuve, Belgium, 2001.

[21] J. Zhang and B. H. C. Cheng. Using temporal logic to specify adaptive program semantics. Journal of Systems and Software Special Issue on Architecting Dependable Systems, 2006.

[22] S. M. Sadjadi, P. K. McKinley, and E. P. Kasten. Architecture and operation of an adaptable communication substrate. In Proceedings of the Ninth IEEE International Workshop on Future Trends of Distrib uted Computing Systems (FTDCS’03), pages 46–55, San Juan, Puerto Rico, May 2003.