دانشکده فنی و مهندسی

طراحی و پیاده‌سازی یک سامانه چند عاملی برای شرکت در مسابقات برنامه‌نویسی چند عاملی

پایان‌نامه‌ی دوره کارشناسی

در رشته مهندسی کامپیوتر - گرایش نرم افزار

دانشجویان:

مرتضی ذاکری نصرآبادی، علی صابری، محسن امیریان، نوید پارسا

و مهرداد تمیجی

استاد راهنما:

دکتر وحید رافع

تابستان ۱۳۹۴


بسم الله الرحمن الرحیم

تقدیم به

پویندگان راه علم و معرفت؛ تمام آزاد مردانی که نیک می اندیشند و عقل و منطق را پیشه خود نموده، جز پیشرفت و سعادت جامعه، هدفی ندارند.

قدردانی و سپاس

حمد و سپاس خدای را که توفیق کسب علم و معرفت به ما عطا فرمود. برخود لازم می دانیم از تمامی اساتید بزرگوار، به ویژه اساتید دوره کارشناسی که در طول چهار سال گذشته ما را در تحصیل علم و معرفت و فضایل اخلاقی یاری نموده اند، تقدیر و تشکر نماییم. از استاد گرامی جناب آقای دکتر وحید رافع که راهنمایی اعضای تیم را در انجام پژوهش و نگارش این پایان نامه تقبل نموده اند، نهایت تشکر و سپاسگزاری را داریم.

طراحی و پیاده سازی یک سامانه چند عاملی برای شرکت در مسابقات برنامه نویسی چند عاملی (MAPC)

چکیده

یک سامانه چند عاملی سامانه ای است که از چندین عامل هوشمند تعاملی تشکیل شده است. از سامانه های چند عامله می توان برای حل مسئله هایی استفاده کرد که حل آن برای یک عامل منفرد یا یک سامانه یکپارچه ی کلاسیک مشکل یا غیر ممکن می باشد. متدلوژی ها و ابزارهای گوناگونی برای توسعه و برنامه نویسی چند عاملی وجود دارد. هدف از این پروژه مطالعه و بررسی اینگونه سامانه‌ها، چالش های موجود در ساخت و بهره برداری از آن‌ها و در نهایت طراحی و پیاده سازی یک نمونه کاملا عملی از آن با استفاده از متدلوژی و چارچوب مناسب برای شرکت در MAPC می باشد.

برای تحلیل و طراحی برنامه از متدلوژی پرومتئوس بهره گرفته شده و پیاده سازی نیز با چارچوب عامل گرای Jason صورت پذیرفته است. در طراحی معماری داخلی هر عامل، رویکرد اصلی بر اساس مدل BDI بوده که با توجه به پشتیبانی Jason از این مدل، انتقال مفاهیم از فاز طراحی به پیاده سازی با سهولت انجام شده است. در ضمن ساختار اجرای عامل ها روی ماشین به صورت متمرکز است؛ بدین معنی که تمامی عامل ها بر روی یک ماشین اجرا می شوند. تمرکز اصلی در این قسمت، تحلیل سامانه، تشخیص و استخراج راه حل ها و در ادامه تقسیم آن ها به الگوریتم های قابل پیاده سازی و اجرا بر روی محیط بوده است.

با توجه به اقدامات صورت گرفته و پیاده سازی انجام شده می توان گفت در مبحث سامانه های هوشمند چند عاملی، همکاری و هماهنگی بین عامل ها و همگام سازی آن ها در انجام عمل بر روی محیط، موضوع حایز اهمیت می باشد. چالش دیگر پایداری سامانه در برابر محیط هایی با شرایط گوناگون و کنترل نحوه رفتار عامل ها در مقابل دریافت ادراک های متفاوت و چگونگی یادگیری ماشینی آن هاست.

کلمات کلیدی: عامل های هوشمند، سامانه های چند عاملی، عامل گرایی، مهندسی نرم افزار عامل گرا، متدلوژی های عامل گرا.

فهرست مطالب

عنوان صفحه

فصل ۱: مقدمه ۱

۱-۱- شرح مسأله ۲

۱-۲- اهداف و چشم‌اندازها ۴

۱-۳- ساختار پروژه ۴

فصل ۲: مروری بر مطالعات پیشین و ادبیات موضوع ۶

۲-۱- هوش مصنوعی- چشم اندازی تاریخی ۷

۲-۲- تعریف عامل ۹

۲-۳- چند عاملی ۱۱

۲-۴- مدل BDI ۱۲

۲-۴-۱- باورها ۱۲

۲-۴-۲- خواسته ها ۱۳

۲-۴-۳- قصد ۱۳

۲-۵- محیط ۱۳

۲-۶- ارتباطات ۱۴

۲-۷- متدلوژی های عامل گرا ۱۵

۲-۸- زبان های برنامه نویسی عامل گرا ۱۶

۲-۹- مسابقات برنامه نویسی چند عاملی ۱۶

۲-۹-۱- مروری بر سناریوهای پیشین ۱۷

۲-۹-۱-۱- جمع آوری غذا (۲۰۰۵م) ۱۷

۲-۹-۱-۲- معدن طلا (۲۰۰۶م و۲۰۰۷) ۱۸

۲-۹-۱-۳- گاوچران ها (۲۰۱۰-۲۰۰۸م) ۱۸

۲-۹-۱-۴- عامل ها بر روی مریخ (۲۰۱۴-۲۰۱۱م) ۱۸

۲-۹-۱-۵- عامل ها در شهر (۲۰۱۵م) ۱۹

فصل ۳: تحلیل و طراحی سامانه ۲۰

۳-۱- سناریو ۲۱

۳-۱-۱- مقدمه ۲۱

۳-۱-۲- مصور سازی ۲۳

۳-۱-۳- تیم ها و وسایل ۲۴
۳-۱-۴- آیتم ها و مراکز ۲۵
۳-۱-۵- پول و کار ۲۶
۳-۱-۶- اقدامات عامل ها ۲۷
۳-۱-۶-۱- پارامترهای اقدام ها ۳۰
۳-۱-۶-۲- کدهای شکست اقدام ها ۳۲
۳-۱-۷- ادراکات ۳۴
۳-۲- تجزیه و تحلیل سامانه ۳۴
۳-۲-۱- تحلیل و طراحی اولیه ۳۴
۳-۲-۲- استراتژی کلی ۳۵
۳-۲-۳- استراتژی جمع آوری اطلاعات ۳۵
۳-۲-۴- استراژی شارژ کردن ۳۷
۳-۲-۵- توصیف سناریو با استفاده از متدلوژی عامل گرای پرومتئوس ۳۸
۳-۲-۶- معماری سامانه چند عاملی ۳۹

فصل ۴: پیاده سازی ۴۱
۴-۱- چارچوب عامل گرای Jason ۴۲
۴-۱-۱- مفاهیم اولیه در Jason ۴۲
۴-۱-۱-۱- باورها و نقش ها ۴۳
۴-۱-۱-۲- اهداف و طرح ها (نقشه ها) ۴۴
۴-۱-۲- ارتباط Jason با Java ۴۶
۴-۱-۳- ارتباط عامل ها و ساز و کار پیام رسانی ۴۷
۴-۱-۴- مفسر Jason ۴۷
۴-۱-۴-۱- چرخه استدلال ۴۸
۴-۱-۴-۲- اصلاحات مفسر ۴۹
۴-۱-۵- JaCaMo ۵۱
۴-۱-۶- Jason IDE ۵۲
۴-۲- معماری و پیاده سازی مسابقات ۵۴
۴-۲-۱- بسته نرم افزاری مسابقه ۵۴
۴-۲-۲- سرویس دهنده MASSim ۵۵
۴-۲-۲-۱- آغاز به کار برنامه ۵۵
۴-۲-۲-۲- مانیتورMASSim ۵۵

۴-۲-۲-۳- سرور تحت وب MASSim ۵۵
۴-۲-۲-۴- پیکربندی MASSim ۵۶
۴-۲-۲-۵- پیکربندی شبیه سازی ۵۸
۴-۲-۲-۶- نقش ها (roles) ۶۲
۴-۲-۲-۷- محصولات ۶۳
۴-۲-۲-۸- مکان ها ۶۳
۴-۲-۲-۹- کارها ۶۵
۴-۲-۲-۱۰- عامل ها ۶۶
۴-۲-۲-۱۱- پیکربندی حساب های کاربری ۶۶
۴-۲-۳- ارتباط با MASSim ۶۷
۴-۲-۴- ارتباط در لایه پروتکل ۶۸
۴-۲-۴-۱- اصول کلی ارتباط عامل - سرور ۶۸
۴-۲-۴-۲- مروری بر پروتکل ارتباطی ۶۹
۴-۲-۴-۳- اتصال مجدد ۷۱
۴-۲-۴-۴- ساختار کلی پیام های xml ۷۲
۴-۲-۴-۵- AUTH-REQUEST (عامل به سرور) ۷۳
۴-۲-۴-۶- AUTH-RESPONSE (سرور به عامل) ۷۳
۴-۲-۴-۷- SIM-START (سرور به عامل) ۷۴
۴-۲-۴-۸- SIM-END (سرور به عامل) ۷۵
۴-۲-۴-۹- BYE (سرور به عامل) ۷۵
۴-۲-۴-۱۰- REQUEST-ACTION (سرور به عامل) ۷۶
۴-۲-۴-۱۱- ACTION (عامل به سرور) ۸۰
۴-۲-۴-۱۲- نتایج اقدامات ۸۱
۴-۲-۵- واسط استاندارد محیط ۸۲
۴-۲-۵-۱- درباره EISMASSim ۸۲
۴-۲-۵-۲- استفاده از EISMASSim ۸۳
۴-۲-۵-۳- پیکربندی EISMASSim ۸۵
۴-۲-۵-۴- زمان بندی ۸۷
۴-۲-۵-۵- اقدامات برای سناریوی ۲۰۱۵ ..... ۸۷
۴-۳- اتصال Jason به Massim-Server از طریق EIS ۸۸
۴-۳-۱- ایجاد اتصال به EIS ۸۸
۴-۳-۲- مبدل EIS-Jason ۹۲

۴-۴- ساختار کد برنامه ۹۳
۴-۴-۱- طرح کد ۹۳
۴-۴-۲- فایل های جاوا ۹۴
۴-۴-۳- فایل های Agent Speak (L) ۹۵
۴-۴-۴- اجرای برنامه ۹۵

مراجع **۹۸**

واژه نامه **۱۰۰**

فهرست شکل‌ها

عنوان **صفحه**

شکل (۱-۱) مراحل انجام پروژه ۵
شکل (۱-۲) سلسله مراتب موجودیت ها و جایگاه عامل در آن ۱۱
شکل (۲-۲) ارتباط عامل و محیط ۱۴
شکل (۲-۳) متدلوژی های عامل گرا و رابطه آن ها ۱۵
شکل (۱-۳) تصویر از عامل ها و مراکز موجود روی نقشه سناریو عامل ها در شهر ۲۳
شکل (۲-۳) معماری سامانه چند عاملی ۴۰
شکل (۱-۴) مثالی از قوانینی ساده در Jason ۴۳
شکل (۲-۴) مثالی از طرح ها در Jason ۴۴
شکل (۳-۴) چرخه استدلال Jason ۴۹
شکل (۴-۴)IDE مستقل Jason ۵۳
شکل(4-5) پنجره اشکال زدایی یا بازرس ذهن Jason ۵۳
شکل(4-6) ساختار کلی فایل پیکر بندی MASSim ۵۷
شکل(4-7) تنظیم در حالت دستی ۵۸
شکل(4-8) ساختار فایل XML شبیه سازی ۶۱
شکل (۹-۴) فاز آماده سازی ۷۰
شکل (۱۰-۴) فاز شبیه سازی ۷۰
شکل (۱۱-۴) فاز نهایی ۷۱
شکل (۱۲-۴) فرایند اتصال مجدد ۷۱
شکل (۱۳-۴) نمونه ای از پیام AUTH-REQUEST ۷۳
شکل (۱۴-۴) نمونه ای از پیام AUTH-RESPONSE ۷۴
شکل (۱۵-۴) نمونه ای از پیام SIM-START ۷۵
شکل (۱۶-۴) نمونه ای از پیام SIM-END ۷۵
شکل (۱۷-۴) پیام BYE ۷۶
شکل (۱۸-۴) بخشی از محتوای یک نمونه پیام REQUEST-ACTION ارسال شده توسط سرور برای عامل ها ۷۷
شکل (۱۹-۴) یک نمونه اقدام goto ۸۱
شکل (۲۰-۴) پیام ACTION و به همراه صفت id ۸۱

شکل (۲۱-۴) محاوره بین محیط شبیه سازی و معماری عامل ۸۹
شکل (۲۲-۴) پیاده سازی یک محیط توسط برنامه کاربر ۹۰
شکل (۲۳-۴) کد موجود در کلاس Environment سفارشی شده ۹۲
شکل (۲۴-۴) شماتیکی از نقاط ارتباطی سامانه و سرور مسابقات..... ۹۳
شکل (۲۵-۴) فایل های پیکربندی سامانه ۹۴

فصل ۱: مقدمه

در این فصل موضوع، هدف اصلی و ساختار پروژه توضیح داده می شود. علاوه بر آن ارتباط پروژه با حوزه های تخصصی موجود در علوم رایانه، به ویژه مهندسی نرم افزار و هوش مصنوعی، مورد بحث قرار خواهد گرفت.

۱-۱- شرح مسأله

مسائلی که امروز رایانه ها برای حل آن ها برنامه ریزی می شوند، بیش از گذشته پیچیده شده اند. نه تنها از لحاظ ساختار های نگه داشت داده بلکه در روندهای بازیابی اطلاعات، تحلیل و اجرای عملیات وابسته به آن ها نیازهای نو به میان آمده است، به گونه ای که شیوه های کلاسیک را برای پاسخ گویی به آن ها عملا ناکارآمد ساخته است.

این امر نیاز به سامانه های هوشمند در دهه اخیر را بیش از پیش نمایان کرده است. می توان چنین گفت که تقاضا برای **ارتقای بهره‌وری در حوزه های محاسباتی** افزایش چشم گیر یافته است و این افزایش کماکان ادامه خواهد داشت. افزایش حجم اطلاعات و بستر های تولید داده، عمومی شدن آن ها و در نتیجه رقابت میان سازمان ها و شرکت های مختلف را می توان از دلایل عدیده این موضوع برشمرد. این افزایش حجم اطلاعات خود سبب طولانی شدن زمان محاسبات و تصمیم گیری ها شده، به ویژه در مواردی که پاسخ های بهینه مد نظر است.

در حال حاضر استخراج دانش مورد نیاز از این اطلاعات، به یک منبع مهم راهبردی تبدیل شده است. وجود حوزه هایی نظیر داده کاوی، استخراج دانش و به طور کلی محاسبات نرم[۱]، خود موید این مطلب می باشد. ضرورت استفاده از سامانه های هوشمند در بسیاری از حوزه های امروزی به خصوص تجارت و صنعت احساس می شود.

بخش وسیعی از این نیازها با به کار بردن ماشین هایی که به صورت خودکار بسیاری از کارها را انجام می دهند، مرتفع می شود. چنین سامانه هایی را سامانه های هوشمند کاربردی می نامند. اما دسته ای دیگر از مشکلات به این سادگی مرتفع نمی شوند. در گستره ای از مسائل پیش رو، نیاز است تا ماشین با درک آن چه که در ذهن انسان می گذرد، بتواند کارها را یاد گرفته، پردازش کرده و انجام دهد. دست یافتن به چنین امری در وهله اول منوط به داشتن

---

۱ Soft Computing

آگاهی از علوم مختلف مانند روانشناسی و علوم مهندسی است که همگی به صورت مشترک بر روی پردازش اطلاعات، بازنمایی دانش و یادگیری اشتراک دارند.

ساخت و راه اندازی سامانه های هوشمندی از این دست، روش ها و راهبردهای قدرتمندتر و انعطاف پذیرتری را می طلبد تا منجر به تولید سامانه های واقعی تر و کاربردی تری شود. طراحی سامانه های هوشمند ترکیبی در واقع پاسخ علوم رایانه به چنین نیازهایی است. در این جا نوع خاصی که سامانه های چند عاملی[۱] هوشمند نامید می شود، بستر مناسبی را فراهم می سازد تا بتوان جنبه های تئوری موجود را آن چه در عمل نیاز است آمیخت و پاسخ مناسبی گرفت.

یادگیری روش های تحلیل مسائل در قالب سامانه های چند عاملی و سپس طراحی و ساخت یک سامانه کاربردی بر مبنای آن اقدامی ارزنده محسوب می شود. آن چه در این بین دارای اهمیت است، استفاده از مفاهیم هوش مصنوعی در مهندسی نرم افزار و بالعکس می باشد، که پنجره های نوینی از دانش رایانه، را به روی ما می گشاید و ابزار حل گستره وسیعی از مسائل پیچیده را با خود به ارمغان می آورد.

در این پروژه تمرکز ما بر روی مطالعه سامانه های چند عاملی هوشمند است که در بالا به آن اشاره شد. مطالعه متشکل از دو بخش اساسی است. بخشی پیرامون مفاهیم تئوری مربوط و بخش دیگر یک سامانه کاربردی به منظور نمایش آن چه در مطالعات بخش اول آمده است. از آن جایی که روش های توسعه این سامانه ها متنوع است و بسیاری از جنبه های آن از استاندارد خاصی پیروی نمی کند، در بخش عملی به دنبال تدوین و یا یافتن الگوهایی برای استفاده مجدد[۲]، در توسعه نرم افزار هستیم و لذا بیش تر بحث به قسمت عملی، تخصیص می یابد.

برای بخش عملی، مسابقات جهانی برنامه نویسی چند عاملی[۳] انتخاب شده است که به تفصیل بحث می شود. لذا پروژه با عنوان «طراحی و پیاده سازی یک سامانه چند عاملی برای شرکت در مسابقات برنامه نویسی چند عاملی» نام گذاری شده است که به اهمیت جنبه عملی کار می پردازد.

---

۱ Multi-Agent System

۲ Reusable Patterns

۳ Multi-Agent Programming Contest

۱-۲- اهداف و چشم‌اندازها

هدف اصلی مطالعه سامانه های چند عاملی در قالب مسابقات برنامه نویسی چند عاملی، شناخت متدلوژی ها و چارچوب های موجود در این زمینه، انتخاب ابزار های مناسب و کار با آن هاست. بررسی چگونگی استفاده از مفاهیم موجود در مهندسی نرم افزار در طراحی سامانه های عامل گرا که تحت عنوان مهندسی نرم افزار عامل گرا شناخته می شود و نیز طرح کد برنامه توسط زبان های عامل گرا که طراحی عامل گرا نامیده می شود، به کار گیری الگوریتم های مبنی بر هوش مصنوعی و یادگیری ماشین، تشکیل و استفاده از پایگاه های دانش در برنامه ها، معماری عامل و ارتباط بین عامل ها از اهداف جانبی و ثانویه پروژه هستند.

چشم انداز ما در پروژه، ایجاد یک روال برای توسعه سامانه های چند عاملی بر مبنای روش های موجود در مهندسی نرم افزار است که بتوان از آن در تولید برنامه های مشابه، نیز بهره گرفت. این روال باید شامل مراحل تحلیل، طراحی، پیاده سازی و آزمون سامانه های چند عاملی و در برگیرنده ابزارهای لازم هر قسمت باشد، به گونه که چرخه زندگی[۱] سامانه را به طور کامل پوشش دهد. متدلوژی های عامل گرای موجود، برخی از این جنبه ها را شامل می شوند ولی این روال شامل توصیه هایی برای انتخاب متدلوژی و چارچوب کاری مناسب نیز خواهد بود و عملا خواهد توانست سرعت فرایند تهیه یک سامانه چند عاملی را بهبود بخشد.

۱-۳- ساختار پروژه

پروژه حاوی برنامه کامل تولید شده به منظور شرکت در مسابقه و نیز پایان نامه پیش روست. پایان نامه مشتمل بر پنج فصل به شرح آمده در زیر است.

فصل جاری در برگیرنده موضوع و بیان مسئله، تعیین حوزه و مورد مطالعاتی انجام شده در پروژه و نیز ساختار تحقیق است.

فصل دوم به ادبیات موضوع اختصاص یافته و مفاهیم اصلی در مبحث سامانه های چند عاملی و عامل گرایی مطرح می گردد. در این فصل ابتدا مقدمه ای بر هوش مصنوعی و عامل های منطقی بیان می شود که حاوی تاریخچه ای بر موضوعات مورد بررسی در فصل های آتی و

---

۱ Life Cycle

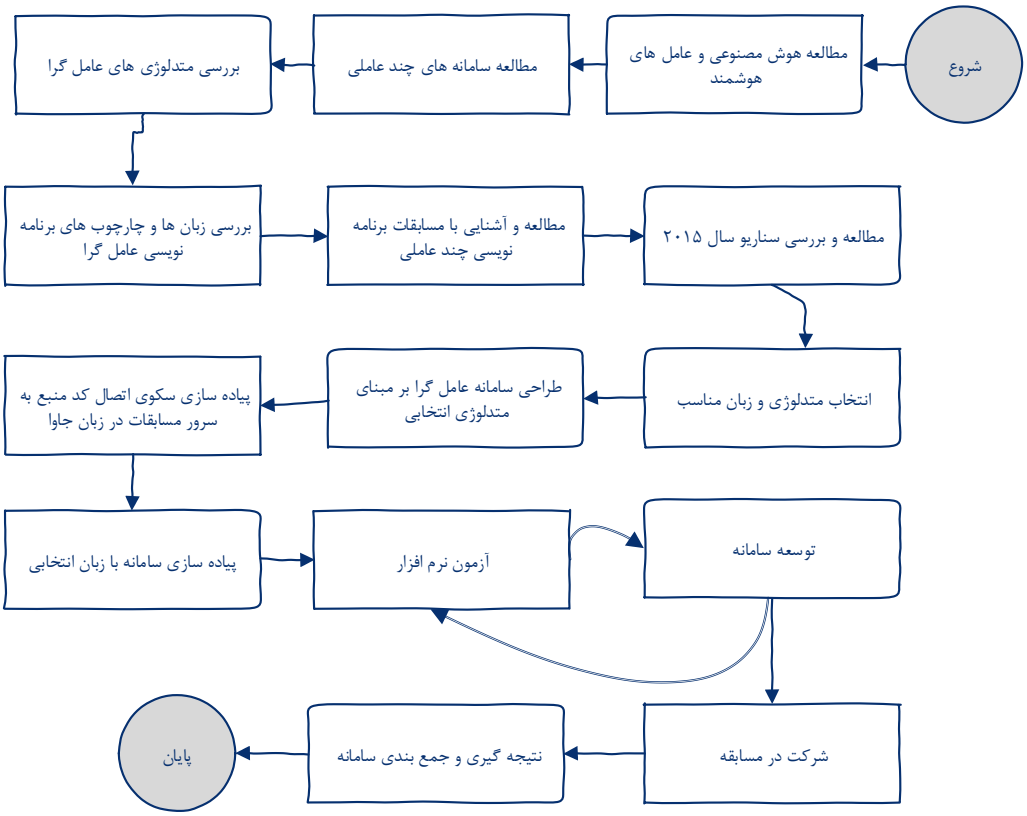
مطالعات پیشین انجام شده در این زمینه است. سپس به تعریف اصطلاحات تخصصی بحث پرداخته می شود و در پایان مسابقات برنامه نویسی چند عاملی، تاریخچه، اهداف و خلاصه ای از سناریوهای گذشته مطرح می شود.

فصل سوم با بیان کامل سناریوی مسابقات در سال ۲۰۱۵ آغاز می شود. در ادامه محتوای بسته نرم افزاری مسابقه نیز تشریح شده و وارد فرایند تولید سامانه ی چند عاملی مورد انتظار می شویم. بخش عمده این فصل به تحلیل و طراحی سامانه اختصاص یافته است.

فصل چهارم در مورد نحوه ی پیاده سازی سامانه است. برای این منظور از چارچوب برنامه نویسی عامل گرای Jason استفاده شده است که ابتدا مفاهیم موجود در Jason مطرح و سپس ساختار کد نوشته شده برای برنامه و بخش های اصلی موجود در کد شرح داده می شوند.

فصل پنجم پیرامون نتایج حاصله از پژوهش و مقایسه آن با پژوهش های مشابه است. در انتها نیز زمینه های مربوط به پژوهش های آتی و موضوعات مرتبط با پژوهش جاری آورده شده اند.

Error! Reference source not found. مراحل انجام پژوهش، مطالعات انجام شده و نتایج حاصله را از نظر ترتیب و توالی زمانی به تصویر می کشد.



شکل (۱-۱) مراحل انجام پروژه

فصل ۲: مروری بر مطالعات پیشین و ادبیات موضوع

در این فصل به بیان تاریخچه مختصری از مفاهیم پژوهش (هوش مصنوعی و عامل گرایی) و تعریف اصطلاحات کار می پردازیم و مسابقات برنامه نویسی چند عاملی را به عنوان تلاشی در این زمینه مورد مطالعه قرار خواهیم داد.

۲-۱- هوش مصنوعی- چشم اندازی تاریخی

اصطلاح *هوش مصنوعی*[۱] نخستین بار در سال ۱۹۵۰ میلادی برای توصیف «علم و مهندسی ساخت ماشین های هوشمند» به کار گرفته شد. از آن پس پژوهش ها در این حیطه از علوم رایانه به سرعت و با پیشرفت چشم گیری ادامه یافته است؛ این درحالی است که چندین رکود و سرخوردگی بزرگ نیز در این بین رخ داده، ولی مانع از کنار گذاشتن سرمایه گذاری روی این مجموعه از علم نشده است.

هنوز تعریف دقیق و جامع برای هوش مصنوعی که مورد توافق دانشمندان این علم باشد، ارایه نشده است و این به هیچ وجه مایه تعجب نیست؛ چرا که مقوله مادر و اساسی تر از آن، یعنی خود هوش هم هنوز به طور همه جانبه و فراگیر تن به تعریف نداده است. اما اکثر تعریف هایی که در این زمینه ارایه شده اند را می توان بر پایه یکی از چهار باور زیر طبقه بندی کرد [۱]:

۱. سامانه هایی که مانند انسان فکر می کنند.
۲. سامانه هایی که مانند انسان عمل می کنند.
۳. سامانه هایی که به طور منطقی فکر می کنند.
۴. سامانه هایی که به طور منطقی عمل می کنند.

شاید بتوان هوش مصنوعی را این گونه توصیف کرد: «هوش مصنوعی عبارت است از مطالعه این که چگونه رایانه ها را می توان وادار به کارهایی کرد که در حال حاضر انسان ها آن ها را صحیح یا بهتر انجام می دهند» (Harvnb,Poole,Mackworth,Goebel,1998). هوش مصنوعی به هوشی که یک ماشین از خود نشان می دهد و یا به دانشی در کامپیوتر که سعی در ایجاد آن دارد گفته می شود. جان مکارتی که واژه هوش مصنوعی را در سال ۱۹۵۶ استفاده نمود، آن را «دانش و مهندسی ساخت ماشین های هوشمند» تعریف کرده است [۲]. در زیر تعاریف مورد

---

۱ Artificial Intelligence (AI)

استناد دیگری از هوش مصنوعی که در طول زمان توسط دانشمندان مختلف مطرح شده، آمده است:

- هنر ایجاد ماشین هایی که وظایفی را انجام می دهند که انجام آنها توسط انسان ها نیاز به هوش دارد (کورزویل- ۱۹۹۰).
- مطالعه ی استعداد های ذهنی از طبق مدل های محاسباتی (کارنیاک و مک درموت - ۱۹۸۵).
- مطالعه این که چگونه رایانه ها را قادر به انجام اعمالی کنیم که در حال حاضر، انسان آن اعمال را بهتر انجام می دهد (ریچ و نایت -۱۹۹۱).
- خودکار سازی فعالیت هایی که ما آن ها را به تفکر انسانی نسبت می دهیم. فعالیت هایی مثل تصمیم گیری، حل مسئله، یادگیری و ... (بلمن -۱۹۷۸).
- تلاشی نو و مهیج برای اینکه کامپیوترها را قادر به فکر کردن کنیم. ماشین هایی با فکر و حس تشخیص واقعی (هاگلند-۱۹۸۵).
- یک زمینه تخصصی که به دنبال توضیح و شبیه سازی رفتار هوشمندانه بوسیله فرایندهای کامپیوتری است (شالکوف -۱۹۹۰).
- مطالعه محاسباتی که درک، استدلال و عمل کردن را توسط ماشین ها را ممکن می سازد. (وینستون - ۱۹۹۲).
- توانایی دست یافتن به کارایی در حد انسان در همه امور شناختی توسط رایانه (آلن تورینگ – ۱۹۵۰).
- هوش مصنوعی دانش و مهندسی ساخت ماشین های هوشمند و به خصوص برنامه های رایانه ای هوشمند است. هوش مصنوعی با وظیفه مشابه استفاده از رایانه ها برای فهم چگونگی هوش انسان مرتبط است، اما مجبور نیست خودش را به روش هایی محدود کند که بیولوژیکی باشند (جان مک کارتی – ۱۹۸۰).

به فرض اینکه تعاریف بالا را از هوشمندی بپذیریم، موارد زیر فهرستی است از وظایفی که از یک سامانه هوشمند انتظار می رود و تقریباً اکثر دانشمندان هوش مصنوعی بر آن توافق نظر دارند به شرح زیر است:

- تولید گفتار
- تشخیص و درک گفتار (پردازش زبان طبیعی انسان)

- دستور پذیری و قابلیت انجام اعمال فیزیکی در محیط طبیعی و مجازی
- استنتاج و استدلال
- تشخیص الگو و بازشناسی الگو برای پاسخ گویی به مسائل بر اساس دانش قبلی
- شمایل گرافیکی و یا فیزیکی جهت ابراز احساسات و واکنش های ظریف سرعت کنش بالا

بیش تر نوشته ها و مقاله های مربوط به هوش مصنوعی آن را «دانش شناخت و طراحی عامل های هوشمند» تعریف کرده اند. یک *عامل هوشمند*[۱] سامانه ای است که با شناخت محیط اطراف خود، شانس موفقیت خود را در محیط بالا می برد. اصطلاح *عامل خردمند*[۲] از تعریف سامانه هایی که به طور منطقی عمل می کنند، استخراج شده است و مدل مناسبی برای ساخت و توسعه سامانه های هوشمند امروزی می باشد. به همین جهت در این پژوهش ما تمرکز خود را به سوی طراحی چنین سامانه هایی معطوف می کنیم که کارگشای حل مسائل مختلف هستند.

پژوهش ما به شاخه ای ترکیبی در علوم رایانه ی حاصل از هوش مصنوعی و مهندسی نرم افزار با عنوان برنامه نویسی عامل گرا[۳] مربوط می شود. به طور خاص تر ما برنامه نویسی چند عاملی[۴] را دنبال می کنیم، که زیر مجموعه ای در همین شاخه می باشد. لذا در قسمت های آتی به بیان اصطلاحات مطرح در این شاخه می پردازیم.

۲-۲- تعریف عامل

برای اهداف این پایان نامه، تعریف مختصری از اصطلاح *عامل*[۵] اجتناب ناپذیر می باشد. استفاده مداوم از این اصطلاح در داخل و خارج از زمینه هوش مصنوعی سبب شده است که این اصطلاح حتی در بهترین حالت نیز دارای ابهام باشد. در چنین مواردی و برای اجتناب از ابهام یک عامل باید مبتنی بر نرم افزار بوده، در یک شبیه سازی گنجانده شده باشد، در یک محیط مجازی و ترجیحا همراه با عوامل دیگر باشد. یک عامل همچنین باید دارای خصوصیات زیر که توسط

---

۱ Intelligence Agent

۲ Rational Agent

۳ Agent-Oriented Programming (AOP)

۴ Multi-Agent Programing

۵ Agent

مایکل وودریج[۱] تعریف شده است، باشد [۲].

- **خودمختاری[۲]**

این ویژگی بیان می کند که ضرورتی ندارد که انسان ها بر عامل نظارت داشته باشند در واقع عامل اقدامات[۳] و تصمیم گیری های خود را به صورت مستقل انجام می دهد.

- **توانایی های اجتماعی**

این ویژگی بیان می کند که عامل ها قادر به ایجاد ارتباط و تعامل با سایر عامل ها یا انسان ها می باشند.

- **واکنش پذیری[۴]**

عامل قادر به درک محیط و واکنش به تغییرات و اتفاقاتی است که در آن محیط وجود دارد.

- **پیش فعال بودن[۵]**

به جای این که صرفا عامل به حوادث واکنش دهد (به ویژگی قبلی ذکر شده در این فهرست نگاه کنید) عامل می تواند خودش اقدامات را آغاز کند و در نتیجه به طور پویا، هر تعداد از اهداف را دنبال کند.

آخرین دو ویژگی به نوعی در تضاد با هم هستند. در واقع رسیدن به یک توازن مناسب از proactive بودن و reactive بودن در حال حاضر یک چالش مهم در راه طراحی سامانه های عامل گرا می باشد. البته در واقعیت، بسیاری از عامل ها به صورت ترکیبی ساخته می شوند. یعنی هم دارای رفتارهای واکنشی و هم ابتکاری هستند. مسئله ی اصلی در این حالت برای طراح، ایجاد توازن لازم بین این دو نوع رفتار است، به نحوی که در مجموع به یک رفتار بهینه دست یابد. افزون بر این خصوصیات دیگری را می توان برای عامل بر شمرد از جمله:

Situated Agent: به آن معنی است که عامل به طور کامل در یک محیط خاص قرار گرفته است.

Directed Agent: عاملی که برای خود هدف مشخصی را در نظر می گیرد و تمام رفتارش در

---

۱ Michael Wooldridge

۲ Autonomy

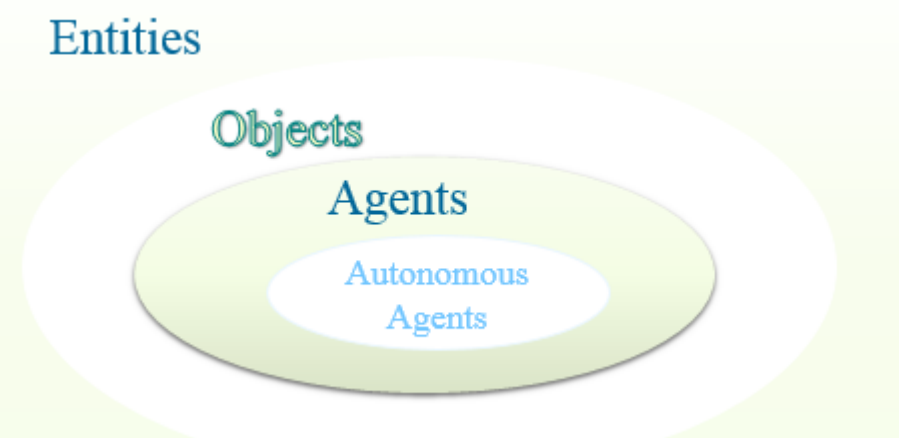
۳ Action

۴ Reactivity

۵ Pro-activeness

راستای رسیدن به آن هدف است.

در خیلی از موارد عامل ها با اشیا مقایسه می شوند. برخی عامل را شی هوشمند می دانند و برخی دیگر ترکیب شی و عامل را برای تولید سامانه های عامل گرا مناسب می دانند. برخی عامل را سطحی بالاتر از انتزاع نسبت به شی می دانند. اما تفاوت هایی ملموس بین عامل و شی وجود دارد که خارج از بحث ما است. **Error! Reference source not found.** سلسله مراتب موجودیت ها را نشان می دهد. همان طور که از شکل پیداست عامل در سطح پایین تر از انتزاع نسبت به شی و موجودیت قرار دارد. در واقع عامل مواردی فراتر از شی و موجودیت را پیاده می کند.



شکل (۲-۱) سلسله مراتب موجودیت ها و جایگاه عامل در آن

۲-۳- چند عاملی

سامانه هایی که از چندین عامل که یا با هم در رقابتند و یا با هم همکاری می کنند و در هر صورت با هم در تعامل هستند تا به یک هدف مشترک یا منحصر به فرد برسند را سامانه های چند عاملی می خوانیم.

از دیدگاه مهندسی نرم افزار، یک سامانه چند عاملی در زمان طراحی از یک مجموعه اولیه از عامل ها تشکیل شده اند و نه یک مجموعه نهایی؛ به عبارت دیگر، سامانه های چند عامل دارای یک معماری باز هستند و عامل ها در زمان اجرا می توانند به سامانه وارد و یا از آن خارج شوند. چنین معماری هایی را پویا می نامند.

معماری های پویا بر ای سامانه های شی گر و یا سایر انواع معماری ها مانند سرویس گرا نیز وجود دارد، اما تفاوت اصلی این است که عامل ها این کار را به صورت خود مختار انجام می دهند (نشان دهنده رفتار proactive آن ها می باشد) و به طور کامل از قبل قابل تشخیص نیست.

برای تکمیل تعریف عامل، این تعریف را باید با استفاده از یکی از مدل های موجود براساس مفاهیم ذهنی، گسترش داد. در این پروژه ما از مدل BDI در معماری عامل های خود استفاده می کنیم. مدل های معماری متنوعی در طول زمان مطرح شده اند که بیش تر ارزش تاریخی داشته و به بررسی آن ها نمی پردازیم.

۲-۴- مدل BDI

BDI[۱] در آغاز به عنوان مدلی برای توصیف بخشی از رفتار انسان گسترش پیدا کرد، هر چند که بعدا بخش عمده ای از آن به سرعت برای کار با عامل های خردمند درون نرم افزار ها به تصویب رسید. معماری مشتمل بر سه مفهوم اساسی است که عنوان معماری آن ها را به یدک می کشد.

۲-۴-۱- باورها

یک عامل دارای مجموعه از باورها یا داشته ها در مورد خود، محیطی که در آن قرار دارد، عامل های دیگر و هر چیزی که به نحوی با آن در ارتباط است، می باشد. مجموعه از باورهای اولیه می تواند توسط برنامه نویس در قالب کد به عامل داده شود. همچنین عامل برای بدست آوردن باورها در زمان اجرا از دو راه زیر می تواند استفاده کند:

- از طریق درک محیط اطراف
- توسط دریافت اطلاعات از عوامل دیگر (در سامانه های چند عاملی)

مهم است که توجه داشته باشیم، باور هایی که عامل نگهداری می کند لزوما درست نمی باشد. به عنوان مثال ممکن است ادراکات عامل ناقص بوده یا قسمتی از اطلاعاتی که عامل دیگری ارسال کرده است، اشتباه بوده که می تواند به صورت ارادی (عامل ارسال کننده یک عامل مخرب

---

۱ Belief-Desire-Intention

است) یا غیر ارادی باشد (عامل ارسال کننده اطلاعات، اطلاعات غلطی دارد اما اعتقاد دارد که این اطلاعات درست است).

اصطلاح باور در این معماری در واقع نشان دهنده *پایگاه دانش*[۱] در سطوح مختلف دانه بندی است.

۲-۴-۲- خواسته ها

خواسته[۲] یا آرزو هدفی را که عامل می خواهد به آن دست پیدا کند، بیان می دارد. خواسته ها لزوما دو به دو سازگار (انحصار متقابل) نمی باشند. بدین معنی که عامل ممکن است به تداخل یا نتایج آشکار متضاد برسد. در واقع داشتن یک خواسته تنها بدین معنی نمی باشد که بر تصمیم عامل تاثیر بگذارد و یا عامل تصمیم بگیرد که به طور فعال به سمت دستیابی به آن عمل کند. خواسته ها می تواند در ابتدا توسط برنامه نویس به عامل داده شود یا خود عامل به آن ها دست پیدا کند.

۲-۴-۳- قصد

مجموعه ای از مقاصد یا نیات شامل اهدافی است که در حال حاضر عامل، برای دستیابی به آن عمل می کند. به عبارت دیگر قصد، آن چه را که عامل تصمیم گرفته است به انجام آن مبادرت ورزد، بیان می کند. قصد توسط یک عامل که در حال تصمیم گیری به انجام خواسته های خود است، ایجاد می شود.

۲-۵- محیط

عامل ها معمولا در یک نوعی از *محیط*[۳] قرار دارند. این محیط می تواند یک جهان واقعی، یک برنامه فیزیکی از قبیل تولید خودکار، کارخانه بسته بندی مواد یا یک برنامه کمی بیش تر انتزاعی اما هنوز واقعی از قبیل اینترنت و یا سرانجام یک برنامه به طور کامل مجازی به سادگی یک شبیه

---

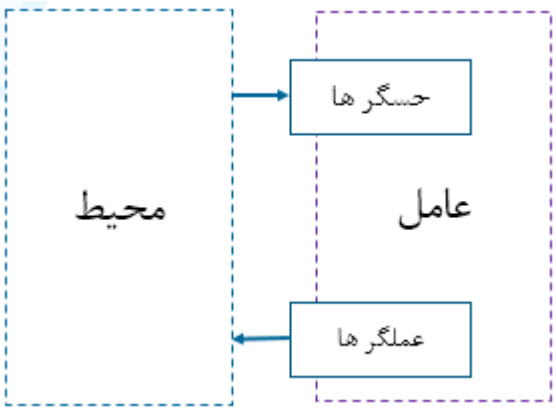
۱ knowledge base

۲ Desire

۳ Environment

سازی باشد.

با توجه به تنوع بسیار زیاد محیط های ممکن، چند شباهت کلیدی بین همه محیط ها وجود دارد که برای آن، روابط عامل – محیط قابل اعمال است. عامل ها از طریق عملگر هایشان قادر به انجام اقدامات و تغییر در وضعیت محیط به صورت محدود، می باشند. همچنین از طریق حسگر ها بازخورد و اطلاعات داخل محیط را، باز هم به صورت محدود، دریافت می کنند. **Error! Reference source not found.** ارتباط عامل با محیط را نشان می دهد [۱].



شکل (۲-۲) ارتباط عامل و محیط

خواسته های عامل ها معمولا با محیط در ارتباط است. بنابراین یک عامل معمولا مجموعه ای از اقدامات را با امید رساندن محیط به یک حالت مطلوب مورد نظر، روی آن انجام می دهد. از آن جا که در MAS، همان گونه که از نام آن پیداست، عوامل گوناگونی در یک محیط قرار دارند. اقدامات مربوطه این عامل ها ممکن است در مقابله یا در کمک به یکدیگر باشد. به همین دلیل است که همکاری و ارتباط بین عامل ها، جنبه بسیار مهمی در مطالعه ی سامانه های چند عاملی محسوب می شود.

۲-۶- ارتباطات

زبان های ارتباطی عامل از قبیل KQML[۱] و FIPA[۲]، عامل ها را قادر به ارتباط از طریق تبادل پیام با قالب های از پیش تعریف شده می کند. پیام ها می توانند برای اشتراک گذاری پیام، درخواست اقدامات، اعلان مقاصد و ارتباطات استفاده شود. پیام ها معمولا شامل المان های زیر

۱ Knowledge Query and Manipulation Language

۲ proposed by the Foundation for Intelligent Physical Agents

می باشد:

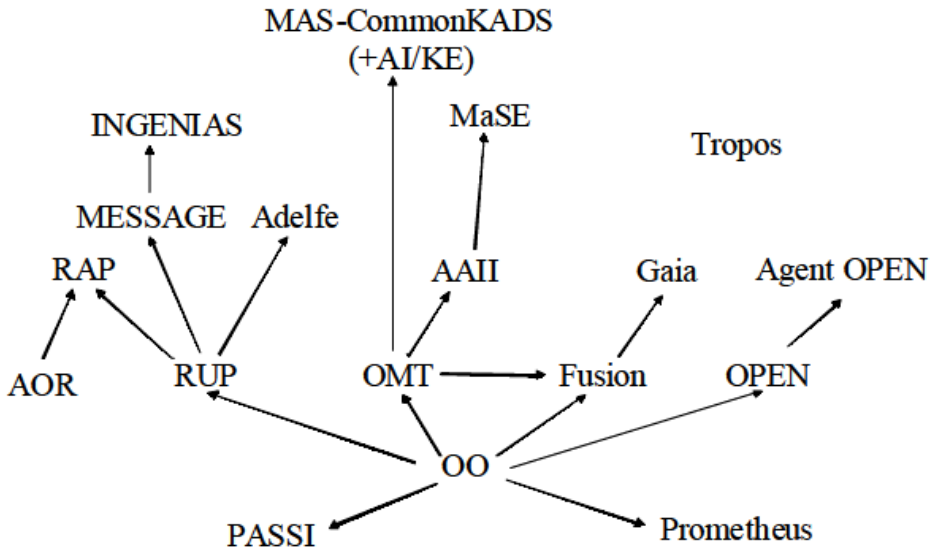
- گیرنده(های) مقصد
- نوع پیام (یک درخواست، اطلاعات و یا غیره)
- محتوای پیام

بخش ۴-۱-۳- چگونگی ارتباطات عامل هایی که در چارچوب Jason پیاده سازی شده اند را توضیح می دهد.

۲-۷- متدلوژی های عامل گرا

برای طراحی عامل گرای سامانه که در مبحث مهندسی نرم افزار عامل گرا مطرح می شود، نیاز به استفاده از متدولوژی های عامل گرا می باشد. یک متدولوژی در حالت کلی تمامی عناصر لازم را برای تولید یک سامانه ی نرم افزاری توصیف می کند. در میان متدولوژی های مختلف موجود، متدولوژی های عامل گرا بر طراحی و تولید سامانه های مبتنی بر عامل ها تکیه دارند. متدولوژی های عامل گرای مختلفی هستند که بیش تر آن ها در حال حاضر ماهیت آکادمیک دارند.

متدولوژی های عامل گرای موجود دارای پایه های متفاوتی اند. برخی ریشه در هوش مصنوعی دارند، برخی دیگر ریشه در شی گرایی و برخی ترکیبی از هر دو شاخه می باشند. **Error! Reference source not found.** متدولوژی های مختلف موجود و رابطه آن ها را با هم نشان می دهد.



شکل (۲-۳) متدلوژی های عامل گرا و رابطه آن ها

چنان که در **Error! Reference source not found.** دیده می شود از این ۱۰ متدولوژی، نه تای آن شی گرا و فقط Tropos است که اساسا عامل گراست.

بسیاری از متدلوژی های عامل گرا (مانند گایا و تروپوس) از خصوصیاتی نظیر ساختار و سازماندهی موجود در جوامع انسانی پیروی می کنند. در چنین ساختاری، هر عامل یک یا چند نقش خاص را با تعامل با سایر عامل ها انجام می دهند. در نتیجه بسیاری از ساختارها و سازماندهی هایی که در جوامع انسانی طرح ریزی شده است قابل استفاده برای طراحی سامانه های چند عاملی نیز می باشد. مفاهیمی چون نقش (Role)، وابستگی اجتماعی (Social dependency) و قواعد سازمانی (Organizational Rules) نه تنها برای مدل سازی محیطی استفاده می شود که عامل ها در آن فعالیت می کنند، بلکه برای مدل سازی خود سامانه چند عاملی هم استفاده می شوند.

۲-۸- زبان های برنامه نویسی عامل گرا

اگرچه اصول برنامه نویسی عامل گرا بیش از یک دهه است که معرفی شده اند، اما در حال حاضر زبان برنامه نویسی عامل گرایی که در عمل برای تولید سامانه های چند عاملی استفاده شود وجود ندارد. اخیرا برخی محیط ها و ابزارها برای پیاده سازی عامل ها و سامانه های عامل گرا

تولید شده اند، اما هیچ کدام بر اساس زبان عامل گرای مناسبی نیستند[۱].

بیش تر ابزارهای موجود برای تولید عامل ها مبتنی بر زبان جاوا است و از اصول شی گرایی برای تولید سامانه ها بهره می برند. در این پژوهش ما از چارچوب برنامه نویسی عامل گرای Jason برای پیاده سازی برنامه عامل گرای خود استفاده کرده ایم که در فصل آتی مفصل پیرامون آن صحبت می کنیم.

۲-۹- مسابقات برنامه نویسی چند عاملی

مسابقات برنامه نویسی چند عاملی، فرصتی را برای تیم های سراسر جهان فراهم کرده است تا سامانه های چند عاملی پیچیده را گسترش داده و عملکرد خود را در برابر تیم های دیگر در یک سناریوی رقابتی از پیش مشخص شده ارزیابی کنند.

این مسابقات توسط Jürgen Dix از دانشگاه Clausthal آلمان، مهدی دستانی از دانشگاه Utrecht و Peter Novák از دانشگاه چک در سال ۲۰۰۵ میلادی به وجود آمد. تمرکز این مسابقات بر روی برنامه نویسی منطقی در سامانه های چند عامله است. اهداف و حوزه مسابقات آن طور که در تارنمای رسمی[۲] آن آمده، عبارتند از [۳]:

این رقابت تلاشی است برای تحریک پژوهش در حوزه برنامه نویسی و توسعه سامانه های چند عاملی به وسیله‌ی

- شناسایی مسائل کلیدی (۲۰۰۵م)
- گردآوری معیار مناسب (۲۰۰۵م) و
- جمع آوری مواد آزمون مورد نیاز و اجرای اقدام هماهنگ با آن (۲۰۰۷م)

که می تواند به عنوان نقطه ی عطفی برای آزمودن زبان ها، چارچوب ها و ابزارهای برنامه نویسی عامل گرا به خدمت گرفته شود. ما همچنین انتظار داریم که شرکت در این مسابقه به اشکال زدایی سامانه های موجود کمک می کند و برای تشخیص نقاط ضعف و جنبه های قوت آن ها به کار می رود.

عملکرد یک سامانه خاص، در یک سری بازی که سامانه در رقابت با دیگر تیم ها انجام می

۱ لیست جامعی از این زبان ها در نشانی http://eprints.agentlink.org/view/type/software.html موجود است.

۲ https://multiagentcontest.org

دهد، تعیین خواهد شد. در حالی که پیروزی در مسابقه نقطه ی مهمی نیست، ما امیدواریم که این امر کاربرد چارچوب های معین را در حوزه ای خاص، روشن سازد.

۲-۹-۱- مروری بر سناریوهای پیشین

از بدو اجرای مسابقات تاکنون سناریوهای متنوعی طراحی و به اجرا گذاشته شده است. همکاری و هماهنگی بین عامل ها از خط مشی های رعایت شده در همه سناریوها می باشد. در ادامه شرح مختصری از عنوان و ماجرای هر سناریو، برای آشنایی بیش تر با مسابقات، ذکر گردیده است [۳].

۲-۹-۱-۱- جمع آوری غذا (۲۰۰۵م)

عامل ها باید دنبال غذا بگردند و آن را به یک انبار در شبکه جهانی دو بعدی برسانند. هر سلول می تواند یک عامل و یا مواد غذایی باشد. عوامل تنها می توانند بخش کوچکی از نقشه را ببینید.

در ابتدا هیچ غذایی در دسترس نیست. غذاها تصادفی در طول بازی ظاهر می شوند، به طوری که عامل ها به منظور برنده شدن نیاز به جست و جوی مداوم در نقشه دارند. این سناریو در سال ۲۰۰۵ مطرح شد.

۲-۹-۱-۲- معدن طلا (۲۰۰۶م و۲۰۰۷)

بر اساس یک نقشه شبکه محور، عامل ها دنبال طلا می گردند و آن را به انبار می برند. برخلاف سناریوی جمع آوری غذا، در این جا سلول ها می توانند هم از درخت هایی که عامل ها را بلوک می کند، باشند و هم می توانند از MAZE (شبکه ای نامنظم از مسیرها که پیدا کردن مسیر در بین آن ها دشوار است) هایی با پیچیدگی کم و زیاد تشکیل گردد. همچنین این بار دو تیم رقیب با هم سر پیدا کردن طلا رقابت می کنند.

این سناریو در سال های ۲۰۰۶ و ۲۰۰۷ بود. در سال ۲۰۰۷ سناریو تمدید شد تا به عامل ها اجازه دهد که بیش تر از یک تکه طلا حمل کنند تا عامل کناری فشار بیش تری را متحمل شود.

۲-۹-۱-۳- گاوچران ها (۲۰۱۰-۲۰۰۸م)

نقشه از درخت ها، حصارها، گاوها و عامل ها تشکیل می شود. دو تیم رقیب تلاش می کنند تا تعداد زیادی گاو را تا جای ممکن در یک حصار هدایت کنند. رفتار گاوها بر اساس هوش جمعی است و همچنین از گاوچران ها می ترسند و تلاش می کنند از دست آن ها فرار کنند. سناریوی گاوچران ها در سال های ۲۰۰۸، ۲۰۰۹ و ۲۰۱۰ میلادی استفاده می شد. در دو سال بعدی راه هایی اضافه شدند تا سناریو را به چالش بیش تری بکشند.

۲-۹-۱-۴- عامل ها بر روی مریخ (۲۰۱۴-۲۰۱۱م)

در سال ۲۰۱۱ این سناریو به منظور تسخیر هرچه بیش تر فضا در مریخ تا جای ممکن با استفاده از تیم همکاری کننده عامل ها مطرح شد. چالش در این جا دارای پیچیدگی بیش تری است؛ زیرا، پنج نقش با خواص و توانایی های مختلف معرفی می شوند که آن ها برای دیده بانی، فتح و نگه داشتن سرزمین های فتح شده گماشته می گردند. تیم HactarV2 در سال ۲۰۱۱ با استفاده از زبان برنامه نویسی GOAL پیروز شدند. در سال ۲۰۱۴ نیز به دلیل عقب ماندن کمیته اجرایی مسابقات از برنامه معرفی سناریوی جدید، برای چهارمین سال پیاپی این سناریو استفاده شد که البته مسابقات به صورت غیر رسمی در این سال برگزار گردید. تیم شرکت کننده از برزیل برای سومین بار پیروز مسابقات سناریوی عامل ها روی مارس شد.

۲-۹-۱-۵- عامل ها در شهر (۲۰۱۵م)

این سناریو آخرین و جدیدترین سناریو مسابقات است. همه چیز پیچیده تر شده و نیاز به مهندسی قدرتمند تر و هوشمندی بیش تری است. سناریو مورد مطالعاتی پژوهش جاری می باشد، لذا شرح مفصل آن در فصل آتی خواهد آمد.

فصل ۳: تحلیل و طراحی سامانه

در این فصل ابتدا به شرح مفصلی از سناریو سال ۲۰۱۵ تحت عنوان «عامل ها در شهر» می پردازیم. سپس با معماری کلی مسابقات آشنا می شویم. ادامه فصل با تحلیل و طراحی سامانه پیشنهادی ما همراه خواهد بود. به طور مشخص ما از برخی مفاهیم موجود در متدلوژی عامل گرای پرومتئوس، که در فصل دوم مختصرا از آن نام برده شد، برای تحلیل و طراحی برنامه خود استفاده کرده ایم که در جای خود آن ها را توضیح خواهیم داد.

۳-۱- سناریو

سناریوی عامل ها در شهر برای نخستین بار امسال (۲۰۱۵م)، توسط کمیته مسابقات معرفی شده است. در زیر شرح سناریو که توسط تارنمای رسمی مسابقات منتشر شده به تفکیک قسمت های مختلف می آید [۴].

۳-۱-۱- مقدمه

سناریو شامل دو تیم از عامل هایی است که در خیابان های یک شهر حقیقی حرکت می کنند. هدف هر تیم بدست آوردن حداکثر مقدار پول ممکن است. پول به تکمیل برخی کارهای خاص اعطا می شود. کارها شامل به دست آوردن، مونتاژ و انتقال کالاها می باشد. این کارها می تواند توسط سامانه (محیط شبیه سازی) یا یکی از عامل های تیم ها ایجاد شود. دو نوع کار داریم: کارهای *قیمتی* (priced) و کارهای *مناقصه ای* (auctioned). هر تیم می تواند با شرکت در مناقصه کار مناقصه ای را بپذیرد. مقدار پولی که برای مناقصه توسط تیم پیشنهاد می شود بعد از تکمیل کار توسط تیم به تیم داده خواهد شد. اگر هر دو تیم در مناقصه شرکت کنند به طور طبیعی پایین ترین پیشنهاد برنده خواهد شد. اگر کار در زمان معین تکمیل نشود تیم مربوطه جریمه خواهد شد. پاداش کارهای قیمتی به اولین تیمی که کار را تکمیل می کند اعطا می شود. هر تیم باید تصمیم بگیرد که کدام کار و چگونه آن را انجام دهد، برای دریافت منابع به کجا برود و چگونه در نقشه با توجه به اهدافی مانند فروشگاه ها، ایسبرچسباه های شارژ و محل های ذخیره سازی مسیریابی کند.

هر تیم شامل انواع مختلفی از عامل ها می باشد. عامل ها در سرعت، چگونگی حرکت در شهر، میزان شارژ باتری، حجم کالاهایی است می توانند حمل کنند و ابزارهایی که می توانند

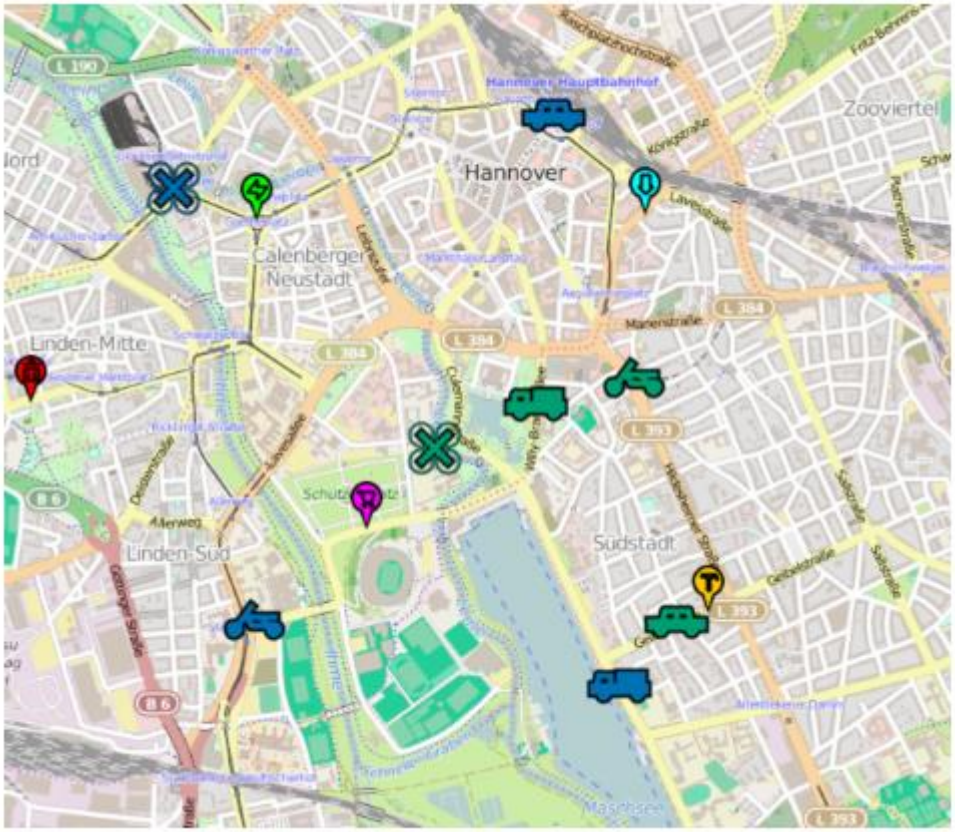
برای ساخت دیگر کالاها به کار گیرند، تفاوت دارند. در حال حاضر ما چهار نوع عامل داریم: اتومبیل ها، کامیون ها، موتور سیکلت ها و پهبادها.

کالاها می توانند خریداری، تولید و ذخیره شوند یا به عامل هم تیمی داده شوند یا به عنوان بخشی از تکمیل کار تحویل و یا از انبار بازیابی شوند و یا دور انداخته بشوند. این عملیات می توانند در محل ها (مراکز) مخصوص مربوط به خود اتفاق بیفتند. اگرچه ساخت یک آیتم به استفاده از دیگر محصولات، چه به عنوان ماده اصلی، چه به عنوان ابزار، نیازمند است و از آن جایی که هر نوع عامل تنها می تواند زیر مجموعه ای از ابزارها را داشته باشد، بنابراین ساخت برخی از کالاها نیاز به همکاری صریح و روشن دو یا بیش تر از دو عامل دارد.

هر عامل یک باتری قابل شارژ دارد که مقدار شارژ باتری با حرکت از یک مکان به مکان دیگر، کاهش می یابد. شارژ عامل ها نباید هیچ گاه تمام شوند و بنابراین باید طوری برنامه ریزی شود که عامل ها در مسیر خود ایسبرچسباه های شارژ را ملاقات کنند. حرکت از یک مکان به مکان دیگر و شارژ مجدد باتری در ایسبرچسباه های شارژ، اقداماتی هستند که تنها بخشی از آن در هر مرحله انجام می شوند و ممکن است به چندین مرحله برای کامل شدن نیاز داشته باشند.

امتیاز مسابقه بر اساس مقدار پولی که هر تیم در پایان مسابقه از آن خود کرده است، توزیع می شود. برای بدست آوردن بیشترین امتیاز، تیم باید بر تیم های دیگر غلبه کند و از یک آستانه خاص نیز پیش بیفتد. اگر تیم بدهی زیادی داشته باشد امتیازات کسر خواهد شد.

Error! Reference source not found. نمایی کلی از نقشه با انواع مختلفی از عامل ها و مراکز روی آن را نشان می دهد.



شکل (۳-۱) تصویر از عامل ها و مراکز موجود روی نقشه سناریو عامل ها در شهر

۳-۱-۲- مصور سازی

در این مسابقات ابزار مصور سازی تدارک دیده شده است و این امکان را می دهد که همه اطلاعات مرتبطی که در طول یک شبیه سازی اتفاق می افتد، به آسانی قابل دسترسی باشد. ابزار شبیه سازی در دو نوع ارایه شده است: اول می تواند از طریق RMI به massim متصل شده تا اطلاعات شبیه سازی را در آن واحد نمایش دهد و دوم امکان پخش آفلاین شبیه سازی را می دهد. این امکان توسط خواندن پوشه ای از فایل های Simulation-XML که قبلا توسط RMI-variant of the tool یا خود سرور MASSIM (که قابل اعتمادتر است) ثبت شده است، فراهم می آید.

- شبیه سازی زمان اجرا

```
$     ./startMapViewer.sh localhost
```

- شبیه سازی ضبط شده

```
$     ./startMapFileViewer.sh Logistics2015_AB_2015-
tournament-sim1
```

همچنین می توان با کلیک بر روی پانلی که در سمت راست تعبیه شده است جزییات بیش تری را درباره عامل یا مراکز بدست آورد. به علاوه می توان از طریق combo box که در بالای سمت راست قرار دارد یک عامل را انتخاب کرد. اگر در یک مثال در حال اجرا دکمه توقف را فشار داده شود، بازی در سرور هنوز در جریان است. که این امکان را به ما می دهد که زمان بیش تری برای مشاهده یک مرحله خاص داشته باشیم. پانل پایینی مستقل از نقشه بوده و این امکان را فراهم می کند که عامل ها، مراکز و کارها را با جزییات بیش تری مشاهده شوند.

۳-۱-۳- تیم ها و وسایل

در حال حاضر چهار نقش در این مسابقات تعریف شده است (به **Error! Reference source not found.** نگاه کنید)، که هر نقش مسیری که وسیله نقلیه اجازه دارد در آن ها حرکت کند، سرعت، ظرفیت باتری، ظرفیت بار و ابزارهایی که عامل ها را قادر به تولید آیتم های جدید می کند را توصیف می کند. هر تیم از ۱۶ عامل تشکیل شده است (۴ اتومبیل، ۴ کامیون، ۴ موتورسیکلت و ۴ پهباد).

جدول (۳-۱) نقش های مختلف و ویژگی های آن ها

Car	Speed:	3
	Routes:	`roads`
	Battery capacity:	500
	Load capacity:	550
	Tools:	`tool1, tool2`
Drone	Speed:	5
	Routes:	`air` (moves in straight line from one place to another).
	Battery capacity:	250
	Load capacity:	100
	Tools:	`tool1`
Motorcycle	Speed:	4
	Routes:	`roads`
	Battery capacity:	350
	Load capacity:	300
	Tools:	`tool1, tool3`
Truck	Speed:	1
	Routes:	`roads`
	Battery capacity:	3000
	Load capacity:	1000
	Tools:	`tool2, tool3`

۳-۱-۴- آیتم ها و مراکز

آیتم ها که با عنوان *محصول*[1] نیز شناخته می شوند، عناصری هستند که عامل ها می توانند به دست بیاورند، تغییر دهند و یا جابجا کنند. آیتم ها یک شناسه و حجم دارند که به طور مؤثر مقداری که هر عامل می تواند از آن آیتم حمل کند یا در مراکز ذخیره سازی، ذخیره گردد، را محدود کند. بعضی از آیتم ها ممکن است توسط عامل ها یا با استفاده از آیتم های دیگر چه به عنوان ماده اصلی و چه به عنوان ابزار، تولید شود. این که یک آیتم به عنوان ماده اصلی یا ابزار به کار گرفته شود، بستگی به آیتم ویژه ای که در حال ساخت است دارد (بدین معنی که یک آیتم ممکن است در برخی موارد به عنوان ماده اصلی و در برخی موارد به عنوان ابزار بکار گرفته شود).

تعداد زیادی از مراکز در نقشه توزیع شده است. هر مرکز به یکی از پنج نوع زیر تعلق دارد: **فروشگاه، انبار، ایسبرچسباه شارژ، کارگاه** و **محل تخلیه**. عامل ها می توانند در هر یک از این مکان ها اقدامات متفاوتی انجام دهند. جزییات بیش تر در زیر آورده شده است. مستقل از نوع مراکز، هر مرکز یک شناسه واحد و موقعیت مکانی مرتبط (که با طول و عرض جغرافیایی تعریف می شود) دارد.

---

۱ Product

فروشگاه‌ها آیتم های مختلفی را جهت فروش به عامل ها عرضه می کنند. نوع آیتمی که می تواند انتخاب شود، موجودی فعلی کالا و قیمت ها ممکن است در یک مغازه با مغازه دیگر متفاوت باشد. ایسبرچسباههای شارژ جایگاه(شیار) های محدودی دارند (زمانی که این جایگاه ها در استفاده باشند، عامل های دیگری که مراجعه می کنند در یک صف قرار می گیرند.) تعداد جایگاه ها، هزینه و سرعت شارژ شدن در یک ایسبرچسباه با ایسبرچسباه های دیگر متفاوت است. انبار به عامل ها امکان می دهد آیتم ها را به صورت موقتی ذخیره کنند، تا یا ظرفیت بار خود را خالی کنند یا اجازه بدهند به هم تیمی های خود تا آیتم ها را در مراحل بعدی بردارند. ظرفیت و هزینه ممکن است در یک انبار با انبار دیگر متفاوت باشد. محل های تخلیه به عامل این امکان را می دهد آیتم هایی که دیگر نیازی به آن ندارند را پاکسازی کنند، تا ظرفیت بارش افزایش یابد. کارگاه ها برای تولید آیتم ها استفاده می شوند.

۳-۱-۵- پول و کار

هدف بازی به دست آوردن بیش ترین مقدار پول ممکن در طی شبیه سازی و بیش تر از تیم های حریف است. سامانه همیشه مقدار پول باقیمانده را نگاهداری می کند. بدین معنی که عامل ها همیشه اجازه دارند برای همه اقدامات پولی هزینه پرداخت کنند (مستقل از مقدار پولی که دارند) که ممکن است یک مقدار پول باقیمانده منفی را ایجاد بکند. اطلاعات بیش تر در مورد هزینه های اعمال در زیر داده شده است.

راه بدست آوردن پول از طریق انجام کارها است. دو نوع کار وجود دارد: کارهای قیمتی و کارهای مناقصه ای. هر دو نوع کار با تحویل آیتم های مشخص شده توسط کار مورد نظر و در مکان مورد نظر که آن هم توسط کار مشخص شده است، کامل می شوند. تحویل آیتم ها می تواند در بخش ها و توسط عامل های مختلف، تا زمانی که کار باقی مانده باشد، ادامه پیدا کند. اما باید توجه داشت که به تحویل های جزئی پاداشی داده نمی شود.

کارهای به مناقصه ای با یک زمان مناقصه آغاز می شوند. تیم ها در طی این زمان برای بدست آوردن کار، پیشنهاد هایی را می دهند. در پایان زمان مناقصه، تیمی که کمترین پیشنهاد را بدهد، کار به او اختصاص پیدا می کند. پیشنهادی که برنده مناقصه می شود باید پایین تر از حد مشخصی باشد (که توسط کار تعریف شده اما برای تیم ها اعلام نمی شود). اگر کار تخصیص

داده شود، تیمی که برنده مناقصه شده است باید کار را در تعداد مشخصی مرحله تکمیل کند و هزینه ای را که برابر با مقدار مناقصه است را از آن خود می کند، در غیر این صورت جریمه ای که توسط کار معین شده است از باقیمانده ذخیره پول تیم برداشته خواهد شد.

کارهای قیمتی از سوی دیگر، پاداش ثابتی دارند که به اولین تیمی که آن را کامل کند پرداخت می شود. همچنین تعداد مراحل برای تکمیل کار محدود است. تحویل های جزئی توسط هر تیم به صورت جداگانه محاسبه می شود . اگر کاری در زمان مشخص شده توسط هیچ کدام از تیم ها تکمیل نشود به سادگی لغو می شود.

تعدادی زیادی کار در طول شبیه سازی توسط سامانه ارسال می شوند. تیم ها نیز اجازه دارند کارها را ارسال کنند. به نظر می رسد که این ها از سیستم ایجاد کار تیم های دیگر غیرقابل تشخیص می باشد، به طوری که تیم ها ممکن است بدون اینکه از آن اگاه شوند با دیگران همکاری کنند. البته در این چنین مواردی تیم ارسال کننده کار، هزینه کار را بعد از تکمیل و تحویل گرفتن آیتم ها می پردازد. همچنین در کارهای به مناقصه گذاشته شده ای که تیم های دیگر درآن شرکت کرده باشند و نتوانند آن را تکمیل کنند جریمه ای در نظر گرفته خواهد شد (این راه غیر مستقیمی است که تیم ها می توانند پول به دست بیاورند).

آیتم هایی که برای کارهایی که توسط تیم کامل نمی شوند تحویل داده می شوند ممکن است وقتی که توسط تیم دیگر تکمیل شده یا مهلت قانونی آن به پایان می رسد از انبار بازیابی شوند.

۳-۱-۶- اقدامات عامل ها

در این بخش ما همه اقداماتی که برای عامل ها در دسترس می باشد را توضیح می دهیم. در دسترس بودن یک اقدام برای یک عامل بخصوص وابسته به مکان و محتوای عامل دارد. جدول ۲ نسخه چکیده ای از ویژگی های عمومی عامل ها را توضیح می دهد. در زیر ما به توضیحی در مورد هریک از اقدامات عامل پرداخته ایم.

- goto: عامل به کمک این اقدام می تواند از یک مکان به مکان دیگر حرکت کند. مکان عامل می تواند به طور صریح (با دادن طول و عرض جغرافیایی) یا با ارجاع به شناسه یک مرکز مشخص شود. وابسته به فاصله تا مقصد و سرعت عامل، عامل به چند گام

شبیه سازی نیاز دارد تا مسیر را طی کند. در چنین مواردی برای عامل کافی و در واقع ترجیح داده می شود که مقصد را در ابتدای راه مشخص کند. اجراهای پی در پی این اقدام می تواند به آسانی برای پیشبرد مسیر از پیش تعیین شده ادامه پیدا کند.

- buy: استفاده از این اقدام تنها در فروشگاه امکان پذیر است. عامل به کمک این اقدام می تواند تعدادی از آیتم هایی که شناسه مشخصی دارند را خریداری کند. قابل دسترس بودن یک آیتم و هزینه پرداختی برای هر واحد وابسته فروشگاه خاص است. عامل خریدار باید ظرفیت خالی کافی برای دریافت آیتم هایی که قصد دارد خریداری کند را داشته باشد.
- give: عامل به کمک این اقدام می تواند تعدادی مشخصی از واحد هایی با شناسه مشخص را به عامل هم تیمی خود بدهد. این اقدام می تواند در هر نقطه از نقشه انجام شود، اما هر دو عامل دریافت کننده و ارسال کننده باید در یک محل باشند. این اقدام به همکاری صریح نیاز دارد؛ بدین معنی که عامل دریافت کننده باید عمل دریافت را در همان زمان انجام دهد تا انتقال موفقیت آمیز باشد.
- receive: عامل به کمک این اقدام می تواند تا زمانی که عامل ظرفیت خالی داشته باشد آیتم ها را از عامل هم تیمی دریافت کند. به اقدام give رجوع کنید.
- store: فقط در محل انبار امکان پذیر است. ذخیره تعدادی از واحدهای یک آیتم مشخص همراه با هزینه است. آیتم ها می توانند بعدا توسط هر عاملی از همان تیم دوباره بازیابی شوند. ظرفیت ذخیره سازی در هر مرکز محدود می باشد بنابراین برای ذخیره سازی، ظرفیت آزاد کافی در محل ذخیره سازی داده مورد نیاز است.
- retrieve: این اقدام فقط در محل انبار در دسترس است. عامل به کمک این اقدام می تواند تعدادی از واحدهای آیتم مشخص را که قبلا توسط عامل های هم تیمی در این انبار ذخیره شده اند را بازیابی کند.
- retrieve_delivered: این اقدام فقط در مکان انبار در دسترس می باشد. عامل به کمک این اقدام می تواند تعدادی از واحدهای یک آیتم مشخص را که قبلا بعنوان بخشی از رقابت کار تحویل داده شده اند را بازیابی کند. هرگاه کار تکمیل و هزینه پرداخت شد، تیم ارسال کننده کار می تواند از این اقدام استفاده کرده و آیتم هایی را که توسط تیم های دیگر تحویل داده شده است را بردارد. اگر کار توسط تیم های دیگر لغو یا تکمیل

شود عامل می تواند از این اقدام برای بازیابی آیتم هایی که برای تکمیل جزیی کار تحویل داده شده اند استفاده کند.

- dump: از این اقدام فقط در محل تخلیه می توان استفاده کرد. عامل به کمک این اقدام می تواند به منظور افزایش ظرفیت آزاد خود از تعدادی از واحدهای یک آیتم مشخص رهایی پیدا کند. بسته به نوع محل تخلیه، ممکن است هزینه ای در قبال این عمل دریافت شود.
- assemble: این اقدام فقط در مکان کارگاه در دسترس می باشد. عامل به کمک این اقدام می تواند یک واحد از آیتم مشخصی را تولید کرده و آن را بردارد. عامل ممکن است از هم تیمی هایش برای همکاری در مواد اصلی یا ابزارها کمک بگیرد.
- assist-assemble: عامل از این اقدام فقط در کارگاه می تواند استفاده کند. عامل به کمک این اقدام می تواند هم تیمی اش را در تولید یک آیتم پشتیبانی کند. مواد اصلی ممکن است هر جا که نیاز است به صورت خودکار از محصولاتی که در حال تولید می باشد گرفته شود. عامل ممکن است در بهره برداری از ابزارهایی که بوسیله آن ها می تواند کارهایی(که توسط نقش آن عامل تعریف می شود) را انجام دهد، نیز همکاری کند. در چنین مواردی نیاز است که عامل در عمل نیز چنین ابزارهایی را در اختیار داشته باشد.
- deliver_job: عامل به کمک این اقدام می تواند آیتم هایی را که برای تکمیل یک کار مشخص نیاز است را تحویل دهد. هر کار یک مکان ذخیره سازی مشخص دارد که تحویل آیتم ها تنها در این مکان ها پذیرفته می شود. تحویل های جزئی منظم که می تواند برای آزاد کردن ظرفیت قابل حمل یک عامل در طول فرایند تکمیل کار، مفید باشد، پذیرفته می شود.
- charge: این اقدام فقط در ایستگاه شارژ در دسترس می باشد. با ورود عامل به ایسبرچسباه شارژ، اگر جایگاه خالی در دسترس باشد، عامل به کمک این اقدام فورا شروع به شارژ کردن می کند در غیر این صورت در یک صف قرار گرفته و زمانی که جایگاه در دسترس قرار گرفت به صورت خودکار در گام های بعدی به جایگاه موجود حرکت داده می شود. حتی زمانی که عامل در جایگاه شارژ باشد، استفاده از این اقدام برای دستیابی به شارژ کامل باتری عامل می تواند به تعدادی از گام ها نیاز داشته باشد.

اگر اقدامی به جز شارژ یا ادامه توسط عامل اجرا شود، عامل مکانش را در جایگاه شارژ یا صف را از دست خواهد داد و ممکن است عامل قبل از این که قادر به ادامه شارژ شود مجبور باشد مدتی را منتظر بماند. زمانی که عامل به صورت کامل شارژ شد، عامل از جایگاه بیرون رانده می شود (بدین معنی که عامل نمی تواند به صورت نامعین جایگاه را مسدود کند)، هزینه و شارژ وابسته به ایسبرچسباه خاص می باشد.

- bid_for_job: عامل به کمک این اقدام می تواند یک پیشنهاد برای کارهای به مناقصه گذاشته شده که باید در مرحله عمل باشد، بدهد.
- post_job: عامل به کمک این اقدام می تواند یک کار پیشنهادی جدید را که تیم های دیگر ممکن است به منظور گرفتن پاداش آن را تکمیل کنند را ایجاد کند. این راه دیگری برای بدست آوردن آیتم ها علاوه بر خرید از فروشگاه یا تولید آن است.
- continue: عامل اجرای یک اقدام در حال انجام را ادامه خواهد داد (عمل goto یا charge).
- skip: عامل به کمک این اقدام هیچ کاری را انجام نمی دهد یا اجرای یک اقدام را همچنان ادامه می دهد.
- abort: عامل به کمک این اقدام اجرای اقدام های در حال اجرا را لغو می کند.
- نتیجه هر عمل صریحاً توسط عامل درک می شود. بدین معنی که اطلاعات در ادراک بعدی (که از سوی محیط برای آن عامل ارسال می شود) به عامل می رسد. نتیجه عمل آخر زمانی که عمل به طور موثر اجرا شود موفقیت آمیز خواهد بود. برای اطلاعات بیشتر در مورد شکست یک عمل به قسمت مربوطه مراجعه کنید.

۳-۱-۶-۱- پارامترهای اقدام ها

تعدادی از اقدام ها به استفاده از پارامترهایی که در فرم KEY=VALUE هستند و با یک فاصله از هم جدا می شوند، نیاز دارند. **Error! Reference source not found.** پارامترهای مرتبط با هر اقدام را نشان می دهد. اکثر پارامترهای اقدام ها، خود-توضیح (واضح) می باشند. اقدام post_job پیچیده تر از سایرین می‌باشد، بنابراین پارامترهای آن به تفصیل در زیر شرح داده شده اند:

- type: این پارامتر نوع کاری را که قرار است ارسال شود را مشخص می کند، این که کار می تواند از نوع قیمتی یا به مناقصه ای باشد.
- max_price: این پارامتر برای کارهای به حراج گذاشته شده، حداکثر قیمت پیشنهادی که در مناقصه پذیرفته خواهد شد را مشخص می کند. اگر همه قیمت های پیشنهادی برای مناقصه توسط تیم های شرکت کننده بیش تر از این مقدار باشد در پایان دوره مناقصه، کار لغو خواهد شد.
- fine: برای کارهای مناقصه ای، این پارامتر مقدار جریمه ای که برنده مناقصه در صورت عدم تکمیل کار مورد نظر، در زمان مشخص بایستی بپردازد را مشخص می کند.
- price: این پارامتر برای کارهای قیمت گذاری شده مقدار پولی که پس از اتمام کار پرداخت خواهد شد را مشخص می کند.
- active_steps: این پارامتر تعداد گام هایی را مشخص می کند که طی آن کار همچنان فعال باقی می ماند و در طی آن تحویل آیتم ها برای تکمیل کار پذیرفته می شود (باید توجه داشت که برای کارهای به مناقصه گذاشته شده، گام های فعال بعد از زمان مناقصه شروع به شمارش می کند).
- auction_steps: این پارامتر برای کارهای به مناقصه گذاشته شده تعداد گام های دوره مناقصه مشخص می کند.
- storage: شناسه انباری که آیتم ها باید در آن تحویل داده می شود.
- itemX and textttamountX (x یک شماره است): این فرمت برای مشخص کردن لیستی از آیتم های مورد نیاز(با شناسه آیتم و تعداد آیتم ها برای این شناسه) استفاده می شود. شماره ها باید متوالی بوده و از یک شروع شوند. ترتیب شماره ها اهمیتی ندارد اما شناسه همه آیتم ها باید از یکدیگر متمایز باشند.

جدول (۳-۲) اقدامات متفاوت عامل ها

goto	Parameters: - option 1: - option 2: Current location: Cost:	(optional if agent is already en-route) facility=fac_id lat=51.805 lon=10.3355 Any Battery charge
buy	Parameters: Current location: Cost:	item=item_id amount=10 Shop Money according to item price at that shop (and amount of items bought).
give	Parameters: Current location: Cost:	agent=id1 item=item_id amount=10 Any (receiver agent at same location) -
receive	Parameters: Current location: Cost:	- Any (giver agent at same location) -
store	Parameters: Current location: Cost:	item=item_id amount=10 Storage Money according to storage location's price
retrieve	Parameters: Current location: Cost:	item=item_id amount=10 Storage -
retrieve_delivered	Parameters: Current location: Cost:	item=item_id amount=10 Storage -
dump	Parameters: Current location: Cost:	item=item_id amount=10 Dump location Money according to dump location's price
assemble	Parameters: Current location: Cost:	item=item_id Workshop Items needed to build the product will be taken from the agent
assist_assemble	Parameters: Current location: Cost:	assembler=agentId1 Workshop (assembler agent at same location) Some items needed to build the product may be taken from the agent
deliver_job	Parameters: Current location: Cost:	job=job_id Storage -
charge	Parameters: Current location: Cost:	- Charging Station Money according to charging station's price (if charge was effectively increased)
bid_for_job	Parameters: Current location: Cost:	job=job_id price=100 Any (If bid is won and job is not completed, the fine is deduced from the teams' money).
post_job	Parameters: - option 1: - option 2: Current location: Cost:	 type=auction max_price=110 fine=50 active_steps=30 auction_steps=10 storage=storage_id item1=item_id1 amount1=10 item2=item_id2 amount2=5 ... ttype=priced price=110 active_steps=30 storage=id_storage1 item1=item_id1 amount1=10 item2=item_id2 amount2=5 ... Any -
continue	Parameters: Current location: Cost:	- Any (effect varies according to context) Depending on context
skip	Parameters: Current location: Cost:	- Any (effect varies according to context) Depending on context
abort	Parameters: Current location: Cost:	- - -

۳-۱-۶-۲- کدهای شکست اقدام ها

اقدام ها به دلایل مختلفی می توانند شکست بخورند. دلایل ممکن برای شکست هر اقدام در **Error! Reference source not found.** توضیح داده شده است. در صورت شکست یک اقدام، نتیجه آخرین اقدام دریافت شده (که در ادراک بعدی به عامل می رسد)، با پیشوند failed_ همراه خواهد بود که واژه بعد از این پیشوند دلیل شکست را اعلام می دارد. در این جا توضیح خواهیم داد که هر شکست ممکن، به چه معنایی هست.

- failed_location: عامل در محل/ مرکز مناسبی برای اجرای این اقدام قرار ندارد.
- failed_unknown_item: شناسه آیتمی که به عنوان پارامتر داده شده است با آیتم های درون شبیه سازی ارتباطی ندارد.
- failed_unknown_agent: شناسه عاملی که به عنوان پارامتر داده شده است با عامل های درون شبیه سازی ارتباطی ندارد.
- failed_unknown_job: شناسه کاری که به عنوان پارامتر داده شده است با کارهای درون شبیه سازی ارتباطی ندارد.
- failed_unknown_facility: شناسه مرکزی که به عنوان پارامتر داده شده است با مراکز درون شبیه سازی ارتباطی ندارد.
- failed_item_amount: تعداد آیتم هایی که به عنوان پارامتر برای این اقدام مشخص شده است، بزرگتر از تعداد آیتم های موجود می باشد.
- failed_capacity: برای انجام این اقدام ظرفیت موجود کافی (برای عامل یا انبار) در دسترس نمی باشد.
- failed_wrong_facility: عامل در مرکزی متفاوت از نوعی که اقدام مورد نظر قابلیت اجرا دارد، قرار گرفته است.
- failed_tools: عامل (یا مجموعه ای از عامل ها) دارای ابزارهای لازم (که عامل ها می توانند بپذیرند)، برای ساخت محصول نیستند.
- failed_item_type: عامل در تلاش برای تولید آیتمی است که نمی تواند توسط عامل ها تولید شود.
- failed_job_status: وضعیت این کار برای اجرای این اقدام صحیح نمی باشد (به عنوان مثال تلاش برای تکمیل کاری را در نظر بگیرید که قبلا تکمیل یا لغو شده است یا دادن پیشنهاد برای کاری بعد از سپری شدن زمان مناقصه و ...).
- failed_job_type: عامل برای کاری که از نوع مناقصه ای نیست، پیشنهاد مناقصه دهد یا برای ارسال کاری که نوع آن مشخص نمی باشد، تلاش کند.
- failed_counterpart: عامل همکار در اجرای درست سهم خود برای عملی که نیاز به هماهنگی دارد شکست بخورد (در اقدام هایی از قبیل assist assemble و give).
- failed_wrong_param: برخی از پارامترها در قالب های اشتباه می باشند (به عنوان

مثال پارامترهای عددی).

- failed_unknown_error: یک خطای ناشناخته در طی اجرای این اقدام رخ داده است.
- failed: اقدام نامعلوم است بنابراین نمی تواند اجرا شود.
- Successful_partial: برخی از آیتم ها جهت تکمیل کار تحویل داده شده اند، اما کار هنوز تکمیل نشده است.
- useless: هیچ آیتمی جهت تکمیل کار تحویل داده نشده است.

علاوه بر امکان شکست ذاتی هر اقدام یک فاکتور غیر قطعی نیز معرفی شده است که به اقدام ها اجازه می دهد تا با احتمال یک درصد شکست بخورند. در چنین مواردی اقدام به عنوان اقدام skip در نظر گرفته می شود (کد شکست اقدام در این حالت failed_random می باشد).

۳-۱-۷- ادراکات

در هر گام، عامل این ادراکات را دریافت می کند.

- وضعیت شبیه سازی به عنوان مثال گام فعلی
- وضعیت تیم به عنوان مثال مقدار پول و کارهای تیم
- وضعیت وسیله های نقلیه به عنوان مثال درون وسایل نقلیه با توجه به توضیحات بالا
- اطلاعاتی درباره وسیله های نقلیه دیگر به عنوان مثال شناسه، محل، تیم و نقش وسیله
- مراکز به عنوان مثال اطلاعات عمومی و جزییات مراکز اگر مراکزی در نزدیکی باشد.
- کارها به عنوان مثال نوع کار، نیازمندی ها و سایر مشخصات

برای اطلاعات بیش تر درباره ادراکات به توضیحات پروتکل ارتباطی در بخش ۴-۲-۴- مراجعه کنید.

۳-۲- تجزیه و تحلیل سامانه

۳-۲-۱- تحلیل و طراحی اولیه

برای شروع طراحی یک سامانه با غایت کسب بیش ترین مقدار پول ممکن در طول بازی، بایستی برخی از جنبه ها را با دقت بیش تری تحلیل کرد و سپس وارد فرایند طراحی شد. این در حالی

است که سناریو خود شامل تحلیل کلی سامانه ای که باید طراحی شود، می باشد. ما چند زیر سناریو را برای سامانه در نظر می گیریم:

- سناریو گرفتن job
- سناریو تهیه محصولات
- سناریو شارژ

۳-۲-۲- استراتژی کلی

در این بخش قصد داریم استراتژی کلی تیم برای ساخت کالاها و شیوه ذخیره سازی آن ها را شرح دهیم. در این طرح از کامیون ها به عنوان انبار استفاده خواهیم کرد. پس از اینکه ارزان ترین کارگاه شناسایی شد، تمام عامل ها در آن کارگاه جمع خواهند شد. در واقع آن کارگاه به عنوان مرکز تجمع و فعالیت تیم خواهد بود و تمام رفت و آمد ها به آن نقطه از محیط منتهی خواهد شد.

با توجه به اطلاعات محصولات و همچنین کارهایی که در محیط اعلام می شود، یکی از کالاها و یکی از مواد اولیه پرکاربرد انتخاب می شوند و شروع به ساخت و ذخیره سازی آنها در کامیون ها می کنیم. پس از آن با توجه به محتویات ذخیره شده در کامیون ها، تصمیم به گرفتن و یا رد کردن کارهای اعلام شده می کنیم. در ادامه، استراتژی های اتخاذ شده برای جمع آوری اطلاعات، شارژ کردن عامل ها، خرید، ساخت کالاها، گرفتن کارها و تحویل کار را به تفضیل شرح خواهیم داد.

۳-۲-۳- استراتژی جمع آوری اطلاعات

در فاز جمع آوری اطلاعات، تصمیم داریم اطلاعاتی مربوط به محیط به دست آوریم که مهم ترین آن ها عبارتند از:

- تعداد و محل قرارگیری (مختصات) مکان[۱]ها و همچنین اطلاعات مربوط به هرکدام
- انواع محصولات[۲] و اطلاعات مربوط به هر نوع محصول.

---

۱ Facility

۲ Products

همان طور که انتظار می رود، در هر گام[۱] از مسابقه، اطلاعات محیط در قالب ادراکات[۲] از سمت سرور برای تمام عامل ها ارسال می شود. می توان به عنوان مهم ترین این اطلاعات، میزان شارژ عامل، مکان قرار گیری عامل، کارهایی که در محیط اعلام شده اند و اطلاعات مربوط به مکان های موجود در محیط (فروشگاه ها، ایستگاه های شارژ و ...) را نام برد. اما این ادراکات شامل تمام اطلاعات موجود در محیط نمی باشد. به عنوان مثال مطلبی که راجع به فروشگاه ها نیاز به دانستن آن داریم و در قالب ادراکات به عامل ها ابلاغ نمی شود، قیمت هر کدام از مواد و ابزارهای داخل یک فروشگاه می باشد. این نوع اطلاعات فقط زمانی دریافت می شود که یک عامل درون فروشگاه باشد.

با توجه به مطالب بالا، طرحی که برای جمع آوری اطلاعات در نظر گرفته شده است، به شرح زیر است:

در گام اول، پس از شناسایی ارزان ترین کارگاه (که در ادامه برای اختصار آن را "کارگاه" می نامیم) در شهر، تمام کامیون ها و موتورها به سمت آن حرکت خواهند کرد (لازم به ذکر است هزینه مربوط به ساخت کالا در هر کارگاه از همان گام ابتدایی برای عامل ها ارسال شده و به سادگی می توان ارزان ترین کارگاه را شناسایی کرد).

هواپیماها به سمت فروشگاه های موجود حرکت کرده تا اطلاعات مورد نیاز در مورد محصولات موجود در فروشگاه ها را شناسایی کنند. پس از شناسایی فروشگاه های مورد نظر تیم، هواپیماها در صورت نیاز خریدهایی انجام داده و به سمت کارگاه حرکت خواهند کرد.

ماشین ها تا شناسایی بهترین فروشگاه ها از نظر تیم، صبر می کنند. سپس برای انجام خرید محصولات مورد نیاز به سمت فروشگاه های مدنظر حرکت کرده و پس از خرید، به سوی کارگاه روانه خواهند شد.

با توجه به اینکه سرعت موتورها نسبت به کامیون ها بیشتر است، به احتمال زیاد زودتر از کامیون ها به کارگاه خواهند رسید. بعد از آن، موتورها وظیفه ی بررسی فاصله ی کارگاه تا سایر مکان های مد نظر تیم در نقشه (براساس تعداد گام) را خواهند داشت. برای این منظور هر کدام از موتورها به سمت یکی از مکان های موجود در نقشه عمل goto انجام خواهند داد. به عنوان مثال:

---

۱ Step

۲ Percepts

```
goto ("facility= shop2")
```

در این صورت در گام بعدی، عامل ادراکی با نام routeLength دریافت خواهد کرد که بیان کننده تعداد نقاطی است عامل در مسیر رسیدن به shop2 (فروشگاه شماره ۲) خواهد پیمود. هر عامل با توجه به سرعت خود، در هر گام تعدادی مشخصی از نقاط را خواهد پیمود. به عنوان مثال موتور سیکلت که دارای سرعت ۴ است، در هر گام ۴ نقطه را خواهد پیمود، در حالی که کامیون با توجه سرعتش، فقط یک نقطه را در هر گام می پیماید. با این اوصاف، عامل (موتورسیکلتی که عمل goto را انجام داده بود) پس از دریافت ادراک routeLength برای آن مکان، آن را به کل تیم اعلام کرده و خود به سمت کارگاه بازخواهد گشت. ازین پس عامل های دیگر از هر نوعی که باشند می توانند قبل از شروع حرکت از کارگاه به shop2، تعداد گام هایی که طول می کشد و همچنین مقدار شارژی که مصرف خواهند کرد را محاسبه کنند. به عنوان مثال در صورتی که برای shop2 مقدار routeLength=30 به تیم اعلام شده باشد و یک ماشین قصد داشته باشد این مسیر را طی کند، به کمک محاسبات زیر قادر به تخمین تعداد گام ها و شارژ مصرفی خود خواهد بود:

SC=3 (سرعت ماشین)

تعداد گام هایی که ماشین در حرکت خواهد بود $= \frac{routeLength}{SC} = \frac{30}{3} = 10$

میزان شارژ مصرفی ماشین = تعداد گام ها × شارژ مصرفی در هر گام $= 10 \times 10 = 100$

۳-۲-۴- استراژی شارژ کردن

از آن جا که میزان شارژ هر عامل یک فاکتور بسیار مهم برای هر عامل محسوب شده و در صورت اتمام شارژ، آن عامل دیگر قادر به ادامه بازی نخواهد بود، پس داشتن یک طرح مناسب برای شارژ کردن عامل ها یک امر بسیار مهم خواهد بود.

هر کدام از عامل ها که در کارگاه هستند و مشغول به کاری نمی باشند در شروع هر گام موظف هستند یک لیست از تمام مکان هایی که می توانند بدون مشکل در میزان شارژ به آن ها بروند اعلام کنند (با استفاده از روشی که در استراتژی جمع آوری اطلاعات گفته شد)، تا در صورت نیاز به مکان های مورد نظر اعزام شوند. مکان هایی که در لیست هر عامل وجود دارند می بایست به گونه ای باشند که عامل با توجه به شارژ خود، بتواند به آن مکان رفته، بازگردد و

سپس بتواند به یک ایستگاه شارژ (ترجیحاً ارزان ترین) برود. در صورتی که لیست اعلام شده ی یک عامل تهی باشد، می بایست قطعاً عامل به سمت ایستگاه شارژ رفته و شارژ کند. همچنین اگر عاملی مدت زمان زیادی (مثلاً ۲۰ گام) در کارگاه بوده و میزان شارژ آن کم تر از مقداری خاص (با توجه به نوع عامل) بود، این بدین معنی است که شارژ این عامل کم بوده و به مکان های کمی قابل اعزام است و آن مکان ها هم مد نظر ما نیستند، پس می بایست اقدام به شارژ کردن کند.

۳-۲-۵- توصیف سناریو با استفاده از متدلوژی عامل گرای پرومتئوس

ابزار طراحی پرومتئوس (PDT) جهت طراحی و پیاده سازی سامانه های چند عاملی که از متدلوژی پرومتئوس استفاده می کنند، ساخته شده است. هدف این ابزار کمک کردن به برنامه نویسان در راستای توسعه و مستند سازی جنبه های مختلف تشخیص و طراحی سامانه های مبتنی بر عامل، می باشد. در ادامه سه فاز اصلی در متدلوژی پرومتئوس که توسط ابزار PDT نیز پشتیبانی می شوند را به صورت مختصر معرفی می کنیم:

- **خصیصه های سامانه:** در این فاز اهداف سامانه تعریف می شوند، ارتباط بین عامل ها و محیط آن ها در قالب ادراکات و اعمال بیان می شود، ویژگی ها شرح داده می شود، و سناریوهای متشکل از گام های متوالی توسعه می یابند.
- **طراحی (معماری) سطح بالا:** در این فاز به وسیله ترکیب قابلیت ها، انواع عامل هایی که در سامانه وجود خواهند داشت تعریف می شوند، ساختار کلی سامانه با استفاده از یک نمودار بررسی اجمالی تعریف می شود، و پروتکل های ارتباطی جهت برقراری پویایی سامانه در قالب توالی پیام ها، استفاده می شوند.
- **طراحی با جزئیات کامل**: این فاز باطن هر یک از عامل ها را در قالب قابلیت ها، رویدادها، طرح ها و اطلاعات توسعه می دهد. نمودار فرآیندها به عنوان پایه و اساس پروتکل های ارتباطی بین عامل ها، استفاده می شوند.

همانند اکثر متدلوژی های نوین مهندسی نرم افزار، پرومتئوس از روشی تکرار شونده بهره می گیرد. مسئله ای که رخ دادن آن اجتناب ناپذیر است، این است که هنگام اعمال کردن یک تغییر

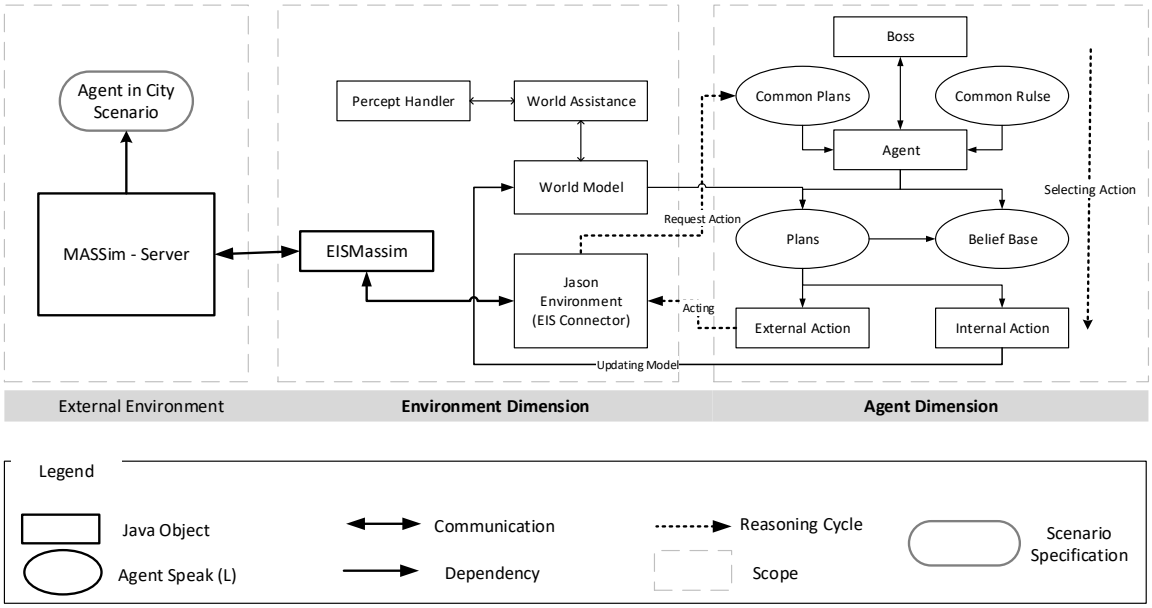
در طراحی، تقریباً غیر ممکن است بیاد داشته باشیم تمام بخش هایی که تأثیر پذیرفته اند را تغییر دهیم. همچنین تضمین کردن اینکه طراحی هنوز هم استوار و نامتناقض است، دشوار است. بنابراین نیاز به شیوه هایی جهت بررسی استواری و عدم تناقض در طرح، وجود دارد. این مسئله می تواند به صورت دستی انجام شود، اما به شدت ملامت آور بوده و مستعد اشتباه می باشد. بنابراین نیاز به این که این ابزار (PDT) به صورت خودکار بررسی استواری طرح را انجام دهد، مشهود است. این مشکل توسط دانشجویان و افرادی که از پرومتئوس استفاده کرده بودند، قبل از این که ابزار PDT استفاده شود، تایید شده است.

ابزار PDT توسط زبان جاوا نوشته شده است و برروی هر سکویی که از جاوا پشتیبانی می کند، اجرا می شود. این ابزار به صورت گسترده تحت ویندوز و یونیکس استفاده می شود و توسط لینک زیر قابل دریافت است:

- http://www.cs.rmit.edu.au/agents/pdt

۳-۲-۶- معماری سامانه چند عاملی

معماری سامانه چند عاملی تیم، در **Error! Reference source not found.** نشان داده است. در طراحی درونی عامل ها ما از مدل BDI که مفاهیم آن در بخش **Error! Reference source not found.** صحبت شد، استفاده کرده ایم. هر عامل دارای یک مجموعه باور مختص به خود است. همچنین کلیه عامل ها دارای مجموعه مشترکی از طرح ها و قوانین هستند. در هر گام محیط پیامی تحت عنوان request action به سامانه می فرستد که محتوی ادراک های محیط برای هر عامل است و منتظر دریافت یک اقدام مناسب از سوی عامل می گردد. عامل با استفاده از این ادراک ها World Model خود را بروز رسانی می کند. همه عامل ها یک World Model مشترک دارند و نقشه ای از اجزای موجود می سازند. با وقوع رویداد request action هر عامل مخزن طرح های خود را بررسی می کند تا طرح مناسب اجرا را پیدا کند. هنگام بررسی برای انتخاب طرح مناسب اجرا، عامل علاوه بر بهره گیری از مجموعه باور داخلی خود، از مدل دنیای مشترک نیز استفاده می کند.



شکل (۳-۲) معماری سامانه چند عاملی

World Model مجموعه ای از کلاس ها و اشیای تعریف شده در جاوا است که یک مدل انتزاعی از آن چه واقعاً در محیط وجود دارد، به دست می دهد. محاسبات روی آن توسط عامل ها انجام می شود و برخی اطلاعات اضافی ناشی از این محاسبات نیز در مدل ذخیره می شود تا برای سایر عامل ها در دسترس باشد.

پس از آن که عامل طرح مناسب را انتخاب کرد، برای اجرای آن روی محیط اقدام می کند و منتظر request action بعدی از سمت سرور می ماند. اقدامات داخلی واسط بین World Model و عامل های Jason هستند. محاسبات عامل ها روی World Model از این طریق انجام می شود.

بدنه طرح ها، قوانین داخل هر عامل فایل های asl ((Agent Speak (L)) هستند و اقدامات داخلی کدهای جاوا می باشند. جزئیات بیش تر پیاده سازی این معماری در فصل چهارم بیان می شود.

.

فصل ۴: پیاده سازی

فصل پیش رو شرح پیاده سازی سامانه چند عاملی ما را نشان می دهد. فصل از سه بخش کلی تشکیل شده است. ابتدا معرفی چارچوب عامل گرای Jason می پردازیم. نشان خواهیم داد چگونه یک مجموعه عامل با معماری داخلی BDI توسط Jason ساخته می شوند و مفاهیم متناظر آن ها در Jason چگونه است. بخش دوم به توصیف بسته نرم افزاری MASSim، و روش در نظر شده برای اتصال عامل ها به سرور مسابقات اختصاص یافته است. در بخش آخر سامانه چند عاملی طراحی شده در فصل سوم را به کمک محیط Jason پیاده سازی خواهیم کرد و چالش های اشکال زدایی و زمان اجرا را تجربه می کنیم.

۴-۱- چارچوب عامل گرای Jason

Jason برای برنامه نویسی عامل های منطقی و سامانه های چند عاملی استفاده می گردد. سامانه های توسعه داده شده با چارچوب Jason، چندین بار در MAPC شرکت کرده و دوبار موفق به کسب مقام قهرمانی این مسابقات شده اند (در سناریوی عامل ها در مریخ طی سال های ۲۰۱۳ و ۲۰۱۴م).

Jason از یک مفسر مبتنی بر زبان جاوا برای اجرا و گسترش AgentSpeak، زبان انتزاعی برنامه نویسی عامل ها، استفاده می کند که در سال ۱۹۹۶ توسط Anand S. Raoمعرفی شد. Agent Speak براساس برنامه نویسی منطقی و معماری BDI است [۲].

۴-۱-۱- مفاهیم اولیه در Jason

پایه و بنیان هر پروژه در Jason، یک فایل پیکر بندی با پسوند mas2j. است که زیرساخت، محیط، عامل های شرکت کننده، واسط کاربر گرافیکی و سایر تنظیمات مربوط به پروژه را مشخص می کند. با این که برخی از اجزا مانند واسط کاربر اختیاری هستند، یک پروژه می بایست شامل عامل هایی باشد که کد آن ها بر اساس زبان برنامه نویسی Jason نوشته شده است که این کد

معمولاً توسط کد نوشته شده در جاوا هم پشتیبانی می شود. کد Jason شامل **باورها**[۱]**ی اولیه عامل ها، قوانین**[۲]**، اهداف**[۳] **و طرح ها**[۴] می باشد. پسوند فایل های کد در Jason به صورت asl. بوده که از Agent Speak Language ریشه می گیرد [۵].

۴-۱-۱-۱- باورها و نقش ها

باورها به وسیله چیزی شبیه به *حقایق*[۵] در زبان برنامه نویسی Prolog نمایش داده می شوند. به عنوان مثال capital(Denmark, Copenhagen) یا animal(cat) نمونه ای از حقایق هستند.

قوانین عبارات خبری هستند که با استفاده از عملگر منطقی :- برای توصیف یک رابطه میان حقایق استفاده می شوند[۶]. کد موجود در **Error! Reference source not found.**مثالی از دو قانون خیلی ساده را نشان می دهد.

```
1 student(N) :- attends_university(N) & owns(N,books) &
2               has(N,beer).
3 affordable(I,P) :- P < 50.
```

شکل (۴-۱) مثالی از قوانینی ساده در Jason

به عنوان مثال N یک دانش آموز (student) است، اگر او در دانشگاه حضور داشته باشد (attends_university) و صاحب کتاب هایی باشد (owns) و دارای نوشیدنی هم باشد. آیتم I مقرون به صرفه است، اگر قیمت (P) آن کمتر از ۵۰ باشد.

---

۱ Beliefs

۲ Rules

۳ Goals

۴ Plans

۵ Facts

۶ http://www.cs.trincoll.edu/~ram/cpsc352/notes/prolog/factsrules.html

۴-۱-۱-۲- اهداف و طرح ها (نقشه ها)

اهدافی که در ابتدا توسط برنامه نویس به عامل داده می شوند، در کد نوشته می شوند. عامل ممکن است در آینده اهداف جدیدی را پذیرفته و یا به او محول گردد، اما این ها در کد نوشته نمی شوند.

طرح ها اعمال[۱] امکان پذیری را نمایش می دهند که یک عامل ممکن است جهت پاسخ دادن به یک رویداد و یا در جهت رسیدن به یک هدف اتخاذ کند. طرح ها برای آماده کردن عامل جهت قرار گرفتن در وضعیت هایی است که عامل خود را در آن ها می یابد. در یک سناریوی شبیه سازی شده، مانند مسابقه سامانه های چند عاملی، تعداد رویداد های ممکن محدود بوده و بنابراین عامل می تواند به طور دقیق برنامه نویسی شود. یک عامل تنها می تواند با توجه به توانایی هایش که در طرح ها تعریف شده اند، عمل کرده یا مسائل را حل کند. عامل ها در واقع می توانند دانش خود را در مورد طرح ها با یکدیگر تبادل کنند. به هر حال در نهایت دامنه ی طرح های موجود برای یک مجموعه از عامل ها در یک محیط، شامل طرح هایی است که برنامه نویس ارائه کرده است. همین مسئله، محدودیت اصلی و مشکل عمده در ساخت سامانه های چند عاملی برای برنامه های کاربردی پیچیده واقعاً عملی که ذاتاً غیر قطعی هستند و تعداد رویدادهای ممکن و محتوای آن ها نامحدود است، می باشد. **Error! Reference source not found.** نمونه ای از طرح ها در Jason را نشان می دهد.

```
1 +!no_money : has(job,Me) <- +work_more.
2 +!no_money : not has(job,Me) <- !!find_job ; !!do_budget.
3 -!no_money : true <- +panic ; !call(parents).
4 +!find_job : true <- !!search_jobs ; !write_CV ; !apply.
5 +!do_budget : true <- ?spendings(Me,S) ; +reduce(S).
```

شکل (۴-۲) مثالی از طرح ها در Jason

یک طرح شامل اجزای زیر است:

۱ Actions

- رویداد محرک[۱] (طرح در چه زمانی مناسب است؟)
- زمینه[۲] (در چه شرایطی طرح قابل اجرا است؟)
- محتویات[۳] (زمانی که طرح برای اجرا انتخاب می گردد، چه باید انجام شود؟)

در **Error! Reference source not found.**، سه طرح برای وضعیت فقدان پول (no_money) وجود دارد، یک طرح برای نحوه ی یافتن یک شغل و یک طرح برای چگونگی ایجاد بودجه. فرض کنید یک رویداد فقدان پول رخ می دهد: no_money+. دو طرح **مربوط با** اداره کردن این رویداد وجود دارد (دو طرح ابتدایی در شکل). بخش زمینه مشخص می کند که در هر وضعیت یک طرح قابل اجرا می باشد، به طوری که طرح اول فقط مربوط به زمانی است که عامل دارای یک شغل است، در حالی که طرح دوم برای زمانی است که هیچ شغلی نداشته باشد. طرح هایی که قسمت زمینه آن ها true می باشد در هر وضعیتی قابل اجرا هستند.

محتویات یا بدنه طرح (بخشی که بعد از -> می آید) شامل مراحل یا گام هایی است که برای تکمیل شدن آن طرح نیاز هستند. بنابراین در مثال اول، دستورالعمل حل مسئله فقدان پول، اضافه کردن یک *یادداشت ذهنی*[۴] برای انجام کار بیش تر می باشد، در حالی که طرح دوم خواستار به کارگیری طرح های اضافی، برای یافتن کار و ایجاد یک بودجه، است. یافتن کار و ایجاد کردن یک بودجه ممکن است به نوبه خود نیازمند بکارگیری اعمال و یا طرح های اضافی باشند. به عنوان نمونه، پیدا کردن کار نیازمند مرور کارهای موجود و نوشتن رزومه می باشد (که نیازی نیست به صورت ترتیبی و یکی پس از دیگری انجام شوند، چون همانند مقاصد جانبی، وظایفی سازگار در کنار یکدیگر می باشند). تنها زمانی فرآیند برنامه ممکن است آغاز شود که این وظایف به طور کامل انجام شوند.

?spendings(Me,S) یک هدف آزمایشی است که اطلاعات مورد نظر را از مجموعه باورها، در

---

۱ Triggering event

۲ Context

۳ Content

۴ Mental Note

این مورد S یا همان رقم خرج کردن، بازیابی می کند. پس از آن یک یادداشت ذهنی درمورد کم شدن مقدار S به وسیله رویداد reduce(S)+ به مجموعه باورها افزوده شده است.

در پایان، طرح no_money!- یک کنترل کننده شکست[۱] است. درصورتی که طرح های دیگر برای کنترل کردن رویداد no_money شکست بخورند، این طرح استفاده می شود. در این مثال، کنترل کننده شکست به عامل می گوید یک راهبرد ناگهانی را به کار گیرد و والدین خود را برای حل کردن مشکل مالی خودش صدا بزند.

۴-۱-۲- ارتباط Jason با Java

چارچوب Jason با زبان جاوا ارتباط تنگاتنگ دارند. این بدین معناست که بخش های معینی از برنامه کاربر، می تواند توسط کد جاوا پیاده سازی شده و از طریق Jason قابل دستیابی خواهد بود. مورد دیگر ارتباط Jason و جاوا پیاده سازی مفسر این زبان با جاواست. در بخش به این موضوع خواهیم پرداخت.

مزیت پیاده سازی با جاوا در این است که وظایف معینی که قرار نیست با محوریت معماری Speak Agent کنترل شوند، می توانند برای حل به کد جاوا موکول شوند. این ویژگی اجازه می دهد تا از امکانات جاوا و برنامه سازی شی گرا جهت پیاده کردن قسمت هایی از سامانه که Agent Speak خوب برای آن تجهیز نیست، بهره گرفته شد. این قسمت ها شامل پیاده سازی مفسر، محیط، واسط گرافیکی کاربر و الگوریتم های پیچیده تر نظیر الگوریتم دایجسترا و همچنین اقدامات داخلی عامل ها می شوند.

Jason شامل تعدادی اقدام درون-ساخت استاندارد برای اهداف مشترکی چون ارتباط بین عامل ها، کار با لیست ها و رشته ها، اعمال محاسباتی و دیگر اعمال است. این ویژگی که *اقدام داخلی*[۲] نامیده می شود، توسط برنامه نویس هم قابل استفاده است؛ یعنی برنامه نویس هم می

۱ failure handler

۲ Internal Action

تواند اقدام داخلی خود را تعریف و پیاده سازی کند.

۴-۱-۳- ارتباط عامل ها و ساز و کار پیام رسانی

عامل ها به کمک اقدامات درونی (که در بخش ۴-۱-۲-، معرفی شدند)، send. broadcast. با یکدیگر ارتباط برقرار می کنند. هر دوی این اعمال شامل پارامترهای *اجبار غیر بیانی*[۱] و تعدادی محتوا از قبیل یک رشته یا یک لیترال هستند. عمل send. می بایست شامل دریافت کننده مورد نظر نیز باشد، در حالی که عمل broadcast. به سادگی پیامی را به تمام عامل های موجود در محیط ارسال می کند. اجبار غیر بیانی مقصود پیام را مشخص می کند. Jason مواردی که در زیر لیست شده اند را فراهم می کند (اجبارهای غیر بیانی با قلم درشت نوشته شده اند).

- **Tell** و **Untell** به ترتیب برای اشتراک گذاری و پاک کردن باورها استفاده می شوند.
- **Achieve** و **Unachieve** به ترتیب برای وکالت دادن و درخواست کردن اهداف لغو شده استفاده می شوند.
- **AskOne** یک جمله را در متن پیام فراهم می کند و انتظار دارد دریافت کننده براساس مجموعه باورهایش پاسخ دهد که آیا آن جمله درست است یا خیر.
- **AskAll** تمام پاسخ هایی را که یک دریافت کننده می تواند برای جمله داده شده فراهم کند، درخواست می کند.
- **TellHow** یک طرح را با گیرنده به اشتراک می گذارد، در حالی که **UntellHow** از گیرنده درخواست می کند که یک طرح را در نظر نگیرد.
- در پایان، **AskHow** طرح هایی مناسب را برای کنترل کردن رویداد داده شده درخواست می کند.

---

۱ illocutionary force

۴-۱-۴- مفسر Jason

مفسر Jason همان طور که قبلا هم گفته شد، با زبان جاوا پیاده سازی شده است. مزیت استفاده از زبان جاوا برای پیاده سازی مفسر احتمالا دو مورد مهم است: اول آن که استفاده کردن از Jason بر روی هر سیستم عامل یا سکویی را ممکن می سازد و دوم آن که فهم و اصلاح کردن کد مفسر را (که ممکن است به آن نیاز باشد) آسان تر می کند. در بخش راجع به آن صحبت خواهیم کرد.

۴-۱-۴-۱- چرخه استدلال

قبلا گفتیم که Jason از معماری BDI برای عامل ها پیروی می کند. بدین صورت که عامل ها بارها و بارها تحت یک *چرخه استدلال*[۱] قرار می گیرند که در آن باورهایشان بروز رسانی می شود و همچنین ملاحظاتی برای اقدامات بعدی ایجاد می کنند. این چرخه به چند گام اصلی تقسیم می شود که در زیر می بینیم:

۱. **ادراک**: فرآیند جمع آوری اطلاعات موجود در مورد محیط
۲. **بروز رسانی باورها**: اطلاعات ادراکی قدیمی حذف شده و اطلاعات جدید (از مرحله ۱) به مجموعه دانش اضافه می شود.
۳. **دریافت کردن ارتباطات**: چک کردن صندوق پیام عامل ها برای پیام های دریافتی
۴. **انتخاب کردن پیام های قابل قبول**: فرآیند تصمیم گیری بر روی این که آیا پیام ها «مقبولیت اجتماعی(دسته جمعی)» دارند یا خیر. برای مثال براساس سطح اعتماد فرستنده به صورت پیش فرض تمام پیام ها پذیرفته می شوند.
۵. **انتخاب یک رویداد**: فرآیند انتخاب یک رویداد، مثلا یک تغییر در محیط، که در چرخه استدلال جاری بایستی رسیدگی شود. در هر چرخه یک رویداد انتخاب می شود.

---

۱ Reasoning cycle

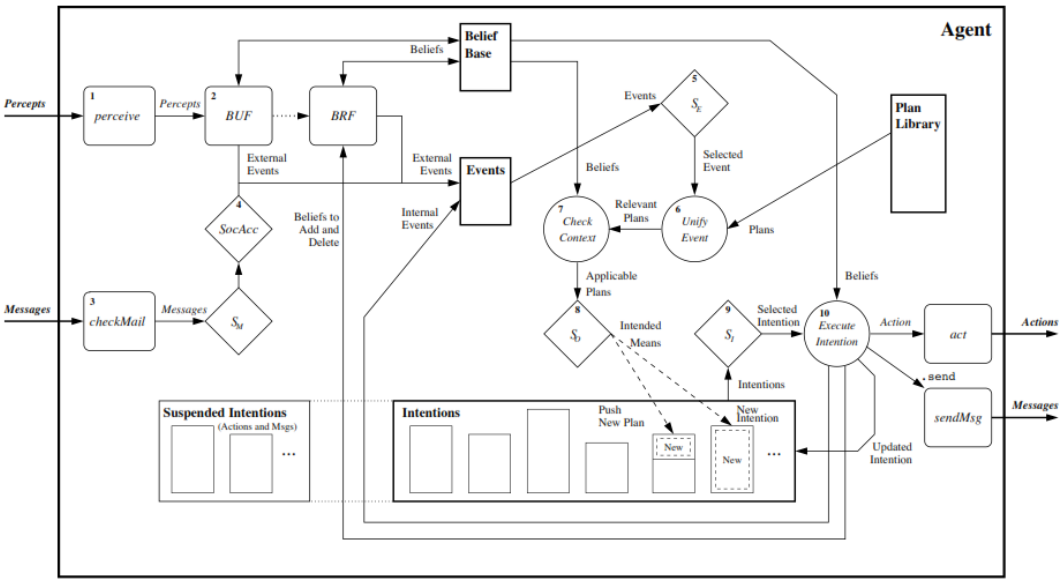
۶. **بازیابی طرح های مناسب**: فرآیند بررسی مجموعه ی طرح ها جهت یافتن طرح های متناسب با رویداد انتخاب شده در مرحله ۵.

۷. **تعیین کردن طرح های قابل انجام**: از مجموعه طرح های مربوطه که در مرحله ۶ یافته شدند، فقط آن طرح هایی که در وضعیت فعلی و یا حالت کنونی محیط قابل اجرا هستند، برگزیده می شوند.

۸. **انتخاب کردن یک طرح قابل اجرا**: یکی از طرح های یافته شده در مرحله ۷ با استفاده از تابع انتخاب کننده، انتخاب می شوند (توضیحات بیش تر در بخش ۴-۱-۴-۲-).

۹. **انتخاب کردن یک قصد برای اجرای بعدی:** از مجموعه ی قصدها، که صرفاً یک پشته از طرح های جزیی است، یک قصد برای چرخه فعلی انتخاب می شود.

۱۰. **اجرای یک گام از یک قصد**: یک عمل از قصد انتخاب شده اجرا می شود.



شکل (۴-۳) چرخه استدلال Jason

۴-۱-۴-۲- اصلاحات مفسر

Error! Reference source not found. مراحل تشریح شده در بخش ۴-۱-۴-۱- را مجسم

می کند و یک چشم انداز تصویری مختصر از مفسر فراهم می کند. توجه به این مطلب مهم است که مفسر به گونه ای پیاده سازی شده است که به برنامه نویس اجازه انجام سفارشی سازی زیادی را می دهد. در ادامه یک لیست از بخش هایی از مفسر که قابل تغییر هستند به اضافه چند مثال از تغییرات ممکن و رایج آورده شده است:

- **ادراک:** شیوه ای که یک عامل محیط خود را درک می کند قابل تغییر است. برای مثال، با شبیه سازی کردن ادراکات ناقص و یا محدود کردن تعدادی از عناصر یا بخش های درک شده ی محیط.
- **بروز رسانی و اصلاح باورها:** چگونگی افزوده شدن و اصلاح کردن باورها در مجموعه باورها قابل تغییر هستند. برای مثال، برخی از باورها ممکن است بر اساس معیارهای خاصی کنار گذاشته شوند، امکان نگهداری باورهای متناقض در مجموعه باورها ممکن است وجود داشته باشد و یا اینکه وجود نداشته باشد، و یا دوره تناوب اصلاح کردن مجموعه باورها ممکن است تغییر کند.
- **ارتباط:** تغییر دادن تمام توابع مربوط به ارتباط عامل امکان پذیر است. به عنوان مثال، پیام ها ممکن است براساس اهمیت آنها اولویت بندی شوند، پیام های دریافتی از برخی از عامل های خاص می تواند کنار گذاشته شوند، پیام ها می توانند از لحاظ اجتماعی براساس فاکتورهای قراردادی خاص، قابل پذیرش یا غیرقابل پذیرش پنداشته شوند، به عامل اجازه برقراری ارتباط با برخی از عامل ها داده نشود، یا اینکه گزینه های ارتباطی آن می توانند دچار مشکل شوند، به عنوان مثال داشتن اجازه فقط برای ارسال نوع خاصی از پیام ها.
- **انتخاب رویدادها، طرح ها و اهداف:** تغییر دادن توابع مربوط به انتخاب رویدادها و اهداف امکان پذیر است. برای مثال، رویداد ها می توانند براساس قرارداد هایی اولویت بندی شوند و یا حتی به طور کامل نادیده گرفته شوند. به طور پیش فرض، اگر چندین طرح قابل اجرا موجود باشد، مفسر یکی را براساس مکانش در متن کد انتخاب می کند

(مثلاً یکی از ابتدا انتخاب می شود). برای برخی از برنامه ها، به کار گرفتن برخی ملاحظات اضافی می تواند سودمند باشد. اهداف نیز همانند رویدادها می توانند اولویت بندی شوند. همچنین تعهد به اهداف می تواند شبیه سازی شود. برای مثال، این امکان وجود دارد که در هر چرخه، عامل را مجبور به اتخاذ کردن یک هدف کرد، تا زمانی که آن هدف به سمت کامل شدن پیش برود.

این مسئله اکنون واضح است که مفسر Jason امکانات وسیعی برای سفارشی سازی و اختصاصی کردن هرچه بیش تر عامل ها در سامانه های چند عاملی ارائه می دهد. ایجاد تغییرات در بخش های خاص مفسر نسبتاً ساده است. کل فرآیند، **شامل ساختن یک کلاس جاوا است که از نسخه پیش فرض ارث بری می کند** و سپس بازنویسی رویه های استاندارد و غیر ضروری و یا پیاده سازی توابع اضافی، می باشد.

JaCaMo -۵-۱-۴

الگوی برنامه نویسی چند عاملی بر اساس بیش تر تعریف های آن شامل چهار بعد مجزا است:

- زبان های برنامه نویسی عامل گرا
- پروتکل ها و زبان های تعاملی
- معماری ها، چارچوب ها و زیرساخت های محیط
- سامانه های مدیریت سازمان

سکوهای زیادی بر اساس ابعاد ذکر شده وجود دارند، اما سکویی که شامل تعدادی یا همه آن ها باشند، کمیاب هستند. JaCaMo سه پروژه را در کنار یکدیگر قرار می دهد: Jason، CArtAgo و $Moise^{+}$، که سه بعد از ابعاد چهارگانه بالا را پوشش می دهد (عامل ها، محیط ها و سازمان ها). ترکیب کردن ابعاد تعاملی که اخیرا توسط ویژگی های ارتباطی Jason صورت گرفته است (رجوع شود به ۴-۱-۳-)، توسط توسعه دهندگان سکوی JaCaMo کار شده است.

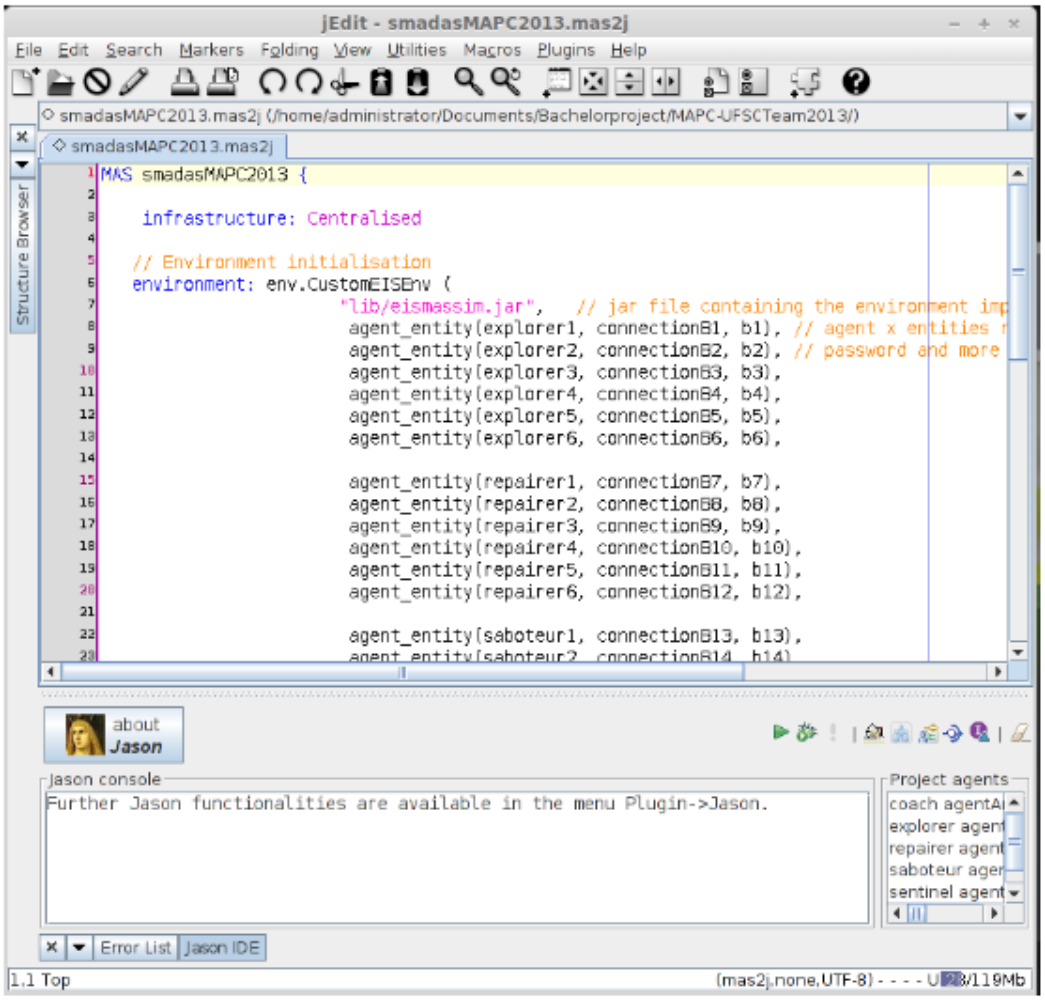
CArtAgo برای برنامه نویسی محیط ها و مصنوعات طراحی شده است، درحالی که $Moise^{+}$ برای برنامه نویسی سازمان ها استفاده می شود. از آنجا که عامل ها در MAS تقریباً همیشه در

انواع محیط های مختلف قرار می گیرند و اغلب به شیوه های مختلف سازماندهی می شوند(تیم ها، زیر تیم ها، ماموریت ها / اهداف)، این سکو منابع فوق العاده ارزشمندی فراهم می کند. موضوع اصلی این پایان نامه، استفاده از Jason به تنهایی است؛ لذا از ورود به جزئیات بیش تر مربوط به JaCaMo در این جا خودداری می کنیم.

Jason IDE -۶-۱-۴

توزیع Jason به همراه محیط توسعه یکپارچه[۱] مستقل خود می آید (به **Error! Reference source not found.** نگاه کنید)، که در jEdit پیاده سازی شده است. همچنین برای Eclipse نیز یک افزونه وجود دارد. این IDE ها دارای تمام ابزار های ضروری برای ساختن سامانه های چند عاملی، خود عامل ها و نیز محیط های ساده، می باشند. ما از افزونه Jason در Eclipse استفاده می کنیم. این به ما اجازه خواهد داد تا از ویژگی های کد جاوا راحت تر استفاده کنیم. از جمله در ارتباط با سرور مسابقه و مدل کردن دنیای بازی، ما مجموعه ای از کلاس های جاوا را نوشته ایم.

۱ (IDE) Integrated Development Environment

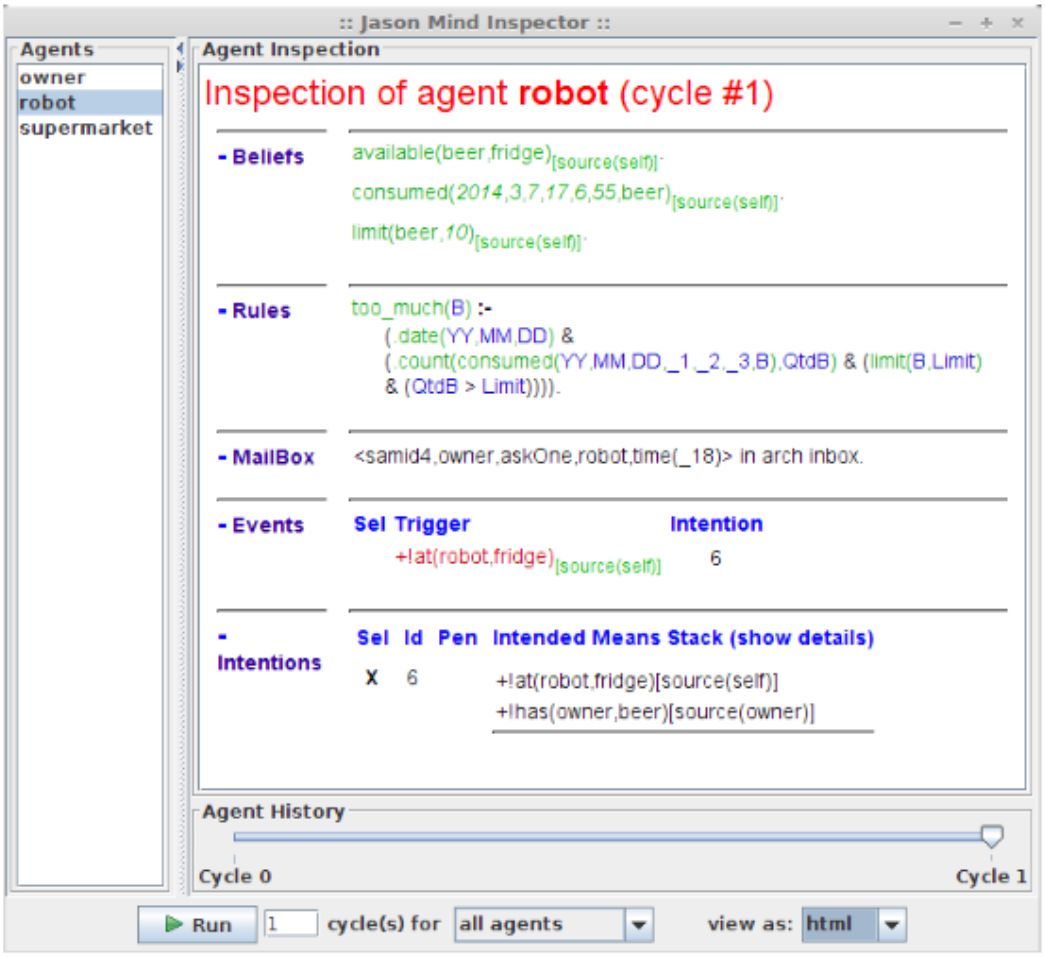


شکل (۴-۴) IDE مستقل Jason

یکی از ابزارهای فوق العاده پر کاربرد، *بازرس ذهن*[۱] است (**Error! Reference source not found.** را ببینید)، که برای اشکال زدایی استفاده می شود. بازرس ذهن این امکان را می دهد که عامل را چرخه به چرخه اجرا کنیم و حالات ذهنی آن را در هر زمان داده شده طی اجرا، بررسی کنیم. حالت ذهنی عامل شامل این موارد می شود: باور های فعلی، خواسته ها، اهداف، نقش ها، طرح ها و صندوق پیام. در استفاده از بازرس ذهن Jason باید دقت کافی به خرج داد. همگام نبودن چرخه حالت داخلی عامل در Jason با گام های محیط در محیط های گسسته در

۱ mind inspector

برخی موارد ایجاد اشکال می کند.



شکل (۴-۵) پنجره اشکال زدایی یا بازرس ذهن Jason

۴-۲- معماری و پیاده سازی مسابقات

برای داشتن یک مجموعه کامل چند عاملی افزون بر برنامه عامل گرای موجود، محیطی که عامل ها در آن اثر گذارند (اقدام) و تاثیر پذیرند (ادراک) نیز لازم است. در این مسابقات محیط و کلیه اعمال مربوط به آن که سناریو را مدل می کنند، در سمت سرور اجرا شده و از طریق اتصال TCP با سامانه عامل گرای هریک از تیم ها که همزمان در سمت مشتری فعال است، ارتباط برقرار شده

و مسابقه انجام می شود. بنابراین مسابقات از نوعی معماری سرویس دهنده – سرویس گیرنده مبتنی بر پشته پروتکلی TCP/IP بهره می برد.

کلیه پیام ها، چه از سمت سرور به مشتری (که حاوی ادراک های فرستاده شده از محیط به عامل های سامانه چند عاملی هستند) و چه از سمت مشتری به سرور (که حاوی اقدامات متنوع عامل ها روی محیط می باشند)، در قالب بسته های XML مبادله می شوند. بنابراین نیاز به ساز و کار ی قانونمند و دقیق برای ایجاد چنین ارتباطی خواهیم داشت. نحوه برقراری این ارتباط را در بخش ۴-۲-۳- مورد مطالعه قرار خواهیم داد، اما پیش از آن لازم است مروری بر بسته نرم افزاری مسابقات و آن چه در سمت سرور مسابقه اتفاق می افتد، داشته باشیم.

۴-۲-۱- بسته نرم افزاری مسابقه

MASSim مخفف عبارت Multi-Agent System Simulation، به معنای شبیه ساز سامانه های چند عاملی، عنوان بسته نرم افزاری مسابقات است که هر سال مطابق با سناریو و تغییرات اعمال شده از سوی کمیته برگزاری مسابقه ارایه می شود. بسته محتوی نرم افزار های لازم برای اجرای مسابقه، کد منبع هریک از نرم افزارها و API متناظر با آن، سناریو و دیگر اسناد لازم برای مطالعه به علاوه یک سامانه چند عاملی نمونه جهت ارزیابی و توسعه (که برای کمک به شرکت کنندگان در مسابقات در نظر گرفته شده) می باشد. در طول یک دوره از مسابقات ممکن است نسخه های مختلفی از بسته MASSim ارایه شود، که با رفع ایرادات بهبود قسمت های گوناگون بسته همراه خواهد. نام بسته معمولا شامل عبارت MASSim به همراه سال ارایه آن و شماره نسخه است. به عنوان مثال برای مسابقات این دوره بسته massim-2015-0.3-bin می باشد.

۴-۲-۲- سرویس دهنده MASSim

۴-۲-۲-۱- آغاز به کار برنامه

برای سیستم عامل ویندوز، پیشنهاد می شود که MSYS1 یا Cygwin2 را به منظور اجرای نرم

افزار MASSim، با استفاده از اسکریپت های shell، نصب کرد. می توان با استفاده از ساختار زیر، MASSim را راه اندازی کرد:

```
$ ./startServer.sh
```

سپس به سرعت می توان یک شبیه سازی را انتخاب کرد. در ادمه به صراحت فایل ها را توضیح می دهیم. سرور، فایل های XML، ارقام، داده ها و غیره را تولید می کند. به پوشه هایی که در طول اجرا تولید می شوند توجه کنید (output وbackup).

۴-۲-۲-۲- مانیتور MASSim

به موازات انجام دادن کار بالا، می توان کار مشاهده شبیه سازی را آغاز کرد. توجه داشته باشید که این مانیتور جهت نمایش کامل جزئیات برای ناظر بیرون بازی فراهم شده است. عامل ها به جزئیات کامل دسترسی ندارد (دید عامل ها محدود است). مانیتور بازی را بر روی دیسک سخت ذخیره می کند. می توان با استفاده از دستور زیر، این فایل ها را مشاهده کرد:

```
$ ./startMapFileViewer.sh /path/where/the/files/are
```

۴-۲-۲-۳- سرور تحت وب MASSim

برای این مسابقات یک وب سرور ارائه شده است که با استفاده از Apache Tomcat، RMI، XML و XSLT بر روی Apache اجرا می شود. این سرور برای توسعه سامانه چند عامله مورد نیاز نیست، ولی از جهت کامل بودن، اطلاعاتی در مورد چگونگی نصب آن را فراهم شده است. یک اسکریپت نصب که فرآیند را توضیح می دهد در محل scripts/tools/ بسته نرم افزاری مسابقه قرار دارد.

۴-۲-۲-۴- پیکربندی MASSim

هنگامی که MASSim آغاز به کار می کند، شما باید یک فایل پیکربندی برای سرور فراهم کنید. فایل های پیکربندی مبتنی بر XML هستند و مجموعه ای از فایل های پیکربندی که درمسیر scripts/conf نصبِ MASSim قرار دارند. شرح مختصری از فایل پیکربندی در ادامه داده شده است.

ساختار کلی فایل پیکربندی در **Error! Reference source not found.** موجود است. توجه داشته باشید که از برخی از ویژگی های اضافی XML که اجازه استفاده بیش از یک فایل و استفاده مجدد از قطعات XML در بخش های مختلف پیکربندی را می دهد، در این ساختار به کار بسته شده است. بنابراین، باید به فایل اصلی و همچنین به فایل config.dtd توجه شود. می توان زمان شروع را با هم اکنون تنظیم کرد. همچنین، یک سطح اشکال زدایی معرفی شده است. علاوه بر این، برخی از پارامترها با جزئیات بیشتری توضیح داده شده است.

```
<?xml version="1.0" encoding="UTF-8"?>
<conf   backuppath="backup"
        launch-sync-type="key"
        reportpath="./backup/"
        time="14:05"
        time-to-launch="10000"
        tournamentmode="0"
        tournamentname="City2015 "
        debug-level="normal" >
    <simulation-server>
        <network-agent backlog="10" port="12300"/>
    </simulation-server>
    <match>
        <simulation ...> ... </simulation>
        ...
        <simulation ...> ... </simulation>
    </match>
    ...
    <match>
        ...
    </match>
    <accounts>
        ...
    </accounts>
</conf>
```

شکل (۴-۶) ساختار کلی فایل پیکر بندی MASSim

برچسب conf: ویژگی های این برچسب به صورت زیر است:

- backuppath: مسیر محلی که اطلاعات مهم هر گام شبیه سازی آنجا ذخیره می شود.
- launch-sync-type: تعیین اینکه سرور توسط فشردن ENTER و یا پس از گذشتن یک زمان خاص تعریف شده در time-to-launch و یا در زمان خاصی (تعریف شده توسط time) آغاز به کار کند. مقدارش می تواند key ، timer یا time باشد.
- reportpath: مسیری که در آن نتایج کلی مسابقات ذخیره می شود.
- time: نقطه ی زمان برای گزینه time.
- time-to-launch: زمان برای گزینه timer.
- tournamentmode: ساختار مسابقات را تعریف می کند. مقدار 0 مسابقات را در حالت نوبت گردشی[1] تنظیم می کند. مقدار 1 برای زمانی استفاده می شود که تنها یک تیم می بایست در مقابل سایرین بازی کند. و در آخر مقدار 2 به یک شخص اجازه می دهد تمام مسابقات را به صورت دستی راه اندازی کند. به منظور کار کردن با آن، می بایست کد هایی شبیه به تصویر 2 پس از </match> اضافه شود.

```
<manual-mode>
  <match team1="A" team2="B"/>
  <match team1="A" team2="C"/>
  <match team1="B" team2="D"/>
</manual-mode>
```

شکل (۴-۷) تنظیم در حالت دستی

- tournamentname: نام مسابقات را برابر با مقدار آن قرار می دهد.
- debug-level: تغییر محتوای خروجی روی پوسته. مقادیر مجاز : debug ، normal ، error ، critical. .

برچسب simulation-server : این برچسب دارای یک فرزند بوده که شامل دو ویژگی می باشد. ویژگی backlog بازه های زمانی (به میلی ثانیه) جهت چاپ کردن پیام های اشکال زدایی

۱ Round robin

به stdout و stderr را تعیین می کند. ویژگی port، پورت سرور را تنظیم می کند.

برچسب match : یک پیکربندی که می تواند یکی یا تعداد بیشتری برچسب match داشته باشد و می تواند به ویژگی های tournamentmode وابسته باشد.

برچسب accounts : و در آخر برچسب accounts شامل جزئیات مربوط به عامل هایی است که اجازه شرکت در مسابقات به آن ها داده شده است.

۴-۲-۲-۵- پیکربندی شبیه سازی

برچسب simulation برای به اجرا درآوردن سناریو به همراه تمام پارامترهایی که بر روی شبیه سازی اثر گذار هستند، استفاده می شود.

برچسب simulation : ویژگی های این برچسب را در ادامه بیان می کنیم:

- id: یک شناسه برای شبیه سازی می باشد. جهت ایجاد تمایز بین نمونه شبیه سازی هایی که در طول مسابقات اجرا می شوند. این شناسه به نام تیم های شرکت کننده در یک نمونه شبیه سازی اضافه خواهد شد.
- simulationclass: نام کلاس اصلی جاوا که سناریو را اجرا می کند. برای سناریوی جاری کلاسی که می بایست استفاده شود massim.competition2015.MapSimulation است.
- configurationclass: نام کلاس جاوا که اطلاعات پیکربندی مشخص شده در برچسب configuration فرزند را نگهداری می کند. برای سناریوی سال 2015، کلاسی که استفاده می شود massim.competition2015.configuration.MapSimulationConfiguration می باشد.
- rmixmlobsserverhost: میزبانی که نمایشگر سناریو باید به آن متصل گردد.
- rmixmlobsserverport: پورتی که نمایشگر سناریو باید به آن وصل شود.

- rmixmlobserver: نام کلاس جاوا که حالت جاری سناریو را بصورت داده های XML ترجمه کرده و آن را زمانی که اتصال صورت گرفت توسط RMI به نمایشگر سناریو ارسال می کند.
- xmlstatisticsobserver: این ویژگی برای وضعیت اتصال Apache Tomcat و صفحه نتایج مورد نیاز است.
- rmixmlobserverweb: این ویژگی برای شبیه سازی جاری و صفحه نتایج مورد نیاز است.
- visualisationobserver: این ویژگی یک کلاس برای مصورسازی تعریف می کند.
- visualisationobserver-outputpath: این ویژگی مسیر خروجی برای فایل های مصورسازی را تعریف می کند.
- xmlobserver: این ویژگی امکان ذخیره کردن نتیجه شبیه سازی را به عنوان فایل XML می دهد.
- xmlobserverpath: مسیر را برای xmlobserver مشخص می کند.

یک قالب کلی از فایل XML برای برچسب simulation در **Error! Reference source not found.**نمایش داده شده است. این برچسب دو فرزند دارد، configuration و agents. بخش configuration منطبق با سناریو است و می بایست مطابق با configurationclass که در ویژگی های simulation است، باشد. برای سناریوی سال 2015 ویژگی های configuration عبارتند از:

- maxNumberOfSteps: تعداد گام هایی که شبیه سازی می بایست اجرا گردد تا برنده تعیین شود.

```
<simulation ...>
    <configuration ...>
        <roles>
            <role ...>
                <roads>
                    <road .../>
                    <road .../>
                    ...
                </roads>
                <tools>
                    <tool .../>
                    <tool .../>
                    ...
                </tools>
            </role>
        </roles>
        <facilities>
            <facility ...>
                <location .../>
                ...
            </facility>
        </facilities>
        <jobs>
            <job ...>
                <products>
                    <product ...>
                    ...
                </products>
            </job>
        </jobs>
        <products>
            <product ...>
                <requirements>
                    <product ...>
                    ...
                </requirements>
            </product>
        </products>
    </configuration>
    <agents>
        <agent ...>
            <configuration .../>
        </agent>
        <agent ...>
            <configuration .../>
        </agent>
        ...
    </agents>
</simulation>
```

شکل (۴-۸) ساختار فایل XML شبیه سازی

- numberOfAgents: تعداد تمام عامل های شرکت کننده در شبیه سازی.
- numberOfTeams: تعداد تیم های شرکت کننده در شبیه سازی.
- minLon, minLat, maxLon, maxLat: بیشترین و کمترین طول و عرض جغرافیایی که در نقشه استفاده شده است که تنها بر روی ابزار مصورسازی اثر می گذارد.

- proximity: تعیین حداکثر فاصله افقی و عمودی که دو عنصر که در یک مکان هستند، می توانند از یکدیگر داشته باشند. می توان مورد ذکر شده را به عنوان طول و عرض جغرافیایی مشخص کرد.
- cellSize: تعیین اندازه واحدی از فاصله که هنگام حرکت عامل محاسبه می شود (سرعت عامل تعیین میکند که چه فاصله ای را در مدت زمان یک گام طی کند). می توان مورد ذکر شده را به عنوان طول و عرض جغرافیایی مشخص کرد.

۴-۲-۲-۶- نقش ها (roles)

بخش roles، نقش های مختلف که عامل های شرکت کننده در شبیه سازی می توانند بپذیرند، تعریف می کند. یک role شامل تمام ویژگی های درونی عامل است. ویژگی های زیر باید برای هر role مشخص شود:

- name: نامی که به عنوان مرجع برای این نقش استفاده می شود.
- speed: سرعتی که به وسیله آن، عامل در یک گام حرکت می کند (یک عدد صحیح مثبت). واحد فاصله توسط ویژگی پیکربندی cellSize تعیین می شود.
- loadCapacity: تعیین حداکثر حجم که عامل قادر به حمل است (یک عدد صحیح مثبت).
- batteryCapacity: تعیین حداکثر شارژ باتری ممکن برای یک وسیله (یک عدد صحیح مثبت).

بخش roads نوع مسیری که عوامل دارای یک نقش خاص می توانند برای حرکت از یک مکان به مکان دیگر انتخاب کنند، تعیین می کند. هر یک از آن ها توسط یک برچسب road با یک ویژگی name منحصر به فرد تعریف می شود (به عنوان مثال: <roads><road name="road"/></roads>). در حال حاضر تنها دو نوع مسیر را تعریف می کنیم: جاده ها، که به عامل ها اجازه می دهد تا از خیابان های منظم استفاده کنند و هوا، که اجازه می دهد یک عامل با حرکت در خط مستقیم به مقصد خود برسد.

بخش tools تعیین می کند که هنگام ساخت کالای جدید، کدام کالاها می تواند به عنوان ابزار توسط عامل های این نقش استفاده شود. هر یک از آنها توسط یک برچسب tool با ویژگی

id منحصر به فرد تعریف می شود (مثال: <tools><tool id="tool1"/><tool id="tool2"/></tools>) کالاهای موجود در بخش محصولات در زیر توضیح داده شده است.

۴-۲-۲-۷- محصولات

بخش products مشخصات تمام کالاهایی که در شبیه سازی استفاده می شود را تعریف می کند. ویژگی های زیر باید برای هر یک از کالاها مشخص شود:

- id: شناسه ای است که توسط آن، کالا شناخته می شود.
- volume: یک عدد مثبت است که حجم کالایی را که در نهایت از جمع کلی چندین کالا ساخته شده است و ممکن است در انبار ذخیره شده و یا در جاهای دیگر استفاده شود را نشان می دهد.
- userAssembled: می تواند دو مقدار true یا false داشته باشد، تعیین اینکه آیا عوامل می توانند این کالا را از کالاهای دیگر بسازند یا خیر.

محصولاتی که لازم است توسط عوامل ساخته شوند، یک لیست از نیازمندی ها (محصولات) دارند. در ادامه یک مثال آورده شده است. هر محصول موجود در لیست دارای سه ویژگی است:

- id: شناسه مربوط به کالا.
- amount: میزان حجم کالایی که لازم است تا بتوان یک واحد از محصول پایه را ساخت.
- consumed: می تواند دو مقدار true یا false داشته باشد، مشخص کننده این است که این کالا در طول ساخت مصرف می شود (همانند یک ماده اولیه) یا اینکه مصرف نمی شود (مانند ابزار).

```
<product id="material1" volume="10" userAssembled="true">
  <requirements>
    <product id="base1" amount="5" consumed="true"/>
    <product id="tool1" amount="1" consumed="false"/>
  </requirements>
</product>
```

۴-۲-۲-۸- مکان ها

بخش facilities تمام مکانهای مختلف موجود در شبیه سازی را نشان می دهد. بعضی از ویژگی ها به نوع مکان بستگی دارد. اما تمام مکان ها دارای id و type هستند که مکان های مختلف را

از یکدیگر متمایز می کنند. علاوه بر این برای تمام مکانها در نقشه به وسیله زیر برچسب location، مختصاتی تعریف شده است که دارای دو ویژگی lat و lon بوده که اعداد حقیقی هستند. ویژگی های بیشتر برای هر مکان در زیر شرح داده شده است:

- workshop
  - cost: یک عدد صحیح که نشان دهنده هزینه مربوط به استفاده از کارگاه در یک گام است.
- storage
  - cost: یک عدد صحیح که نشان دهنده هزینه مربوط به ذخیره کالا در انبار است.

capacity: یک عدد صحیح که نشان دهنده حداکثر حجم ممکن، حاصل از جمع کالاهای ذخیره شده در انبار است.

- Dump
  - cost: یک عدد صحیح که نشان دهنده هزینه مربوط به استفاده از این مکان در یک گام است.
- Charging
  - cost: یک عدد صحیح که نشان دهنده هزینه هر واحد شارژ در این جایگاه شارژ است.
  - rate: یک عدد صحیح که مشخص کننده نرخ شارژ در هر گام است، به عبارت بهتر، تعداد واحدهایی از شارژ که عامل در یک گام می تواند شارژ کند.
  - concurrent: یک عدد صحیح که مشخص کننده حداکثر تعداد عامل هایی است که می توانند باتری هایشان را همزمان در این ایسبرچسباه شارژ کنند (زمانی که این تعداد پر شود، عوامل به طور موقت در یک صف قرار می گیرد).
- shop: همانطور که در مثال زیر نشان داده شده shop ها ویژگی اضافه ای تعریف نمی کنند. در عوض، یک لیست از محصولات (موجود برای خرید در فروشگاه) تعریف می کنند. ویژگی های تعریف شده برای هر محصول در این لیست عبارتند از:
  - id: شناسه کالا است.
  - amount: مقدار یک محصول که فروشگاه در ابتدای شبیه سازی در خود ذخیره دارد.
  - cost: یک عدد صحیح که نشان دهنده هزینه یک واحد از کالا در فروشگاه است.
  - restock: تعداد گام های شبیه سازی که پس از آن یک واحد جدید از این کالا به فروشگاه اضافه می شود. (۰ به این معنی است که از این کالا هرگز به فروشگاه اضافه

نمی شود).

```
<facility type="shop" id="shop1">
  <location lat="52.3619" lon="9.7299"/>
  <products>
    <product id="item1" cost="5" amount="500" restock="2" />
    <product id="item2" cost="17" amount="50" restock="3"/>
  </products>
</facility>
```

۴-۲-۲-۹- کارها

بخش job، سامانه ایجاد کار برای شبیه سازی است. برخی از ویژگی ها به نوع کار بستگی دارد، اما بسیاری از آن ها مشترک است. همه ویژگی ها در زیر توضیح داده شده است:

- id: شناسه ای که کار با آن شناخته میشود.
- type: نوع کارکه یا auction یا price است.
- firstStepAuction: شماره گام شبیه سازی که در این کار باید دوره مزایده خود را آغاز کند.
- firstStepActive: شماره گام شبیه سازی که بعد از آن گام، کار باید فعال شود (برای کار auction این به این معنی پایان دوره مزایده است).
- lastStepActive: شماره گام شبیه سازی که در آن، کار باید کامل شود.
- reward: میزان پولی که در صورت کامل کردن کارِ price به ما داده میشود.
- maxReward: بیشترین مقدار پول برای یک کار که در ابتدا اعلام می شود. این مقدار پول در صورتی پرداخت می شود که درخواست بهتری (کمتری) برای آن کار نباشد.
- fine: مقداری که در صورت انجام ندادن کارِ auction باید به عنوان جریمه پرداخت کرد.
- storageId: شناسه انباری که کالا ها باید در جهت تکمیل کار، در آن ذخیره شوند.

هدف برچسب کار(job) تعریف یک لیست از محصولات (درون زیر برچسب <products>) است که در آن هر محصول (زیر برچسب <product>) دو ویژگی دارد: شناسه محصول و مقدار مورد نیاز از آن است. در اینجا یک مثال از یک کار auction آورده شده است:

```
<job id="job1" type="auction" firstStepAuction="50" firstStepActive="60"
  <products>
    <product id="material1" amount="3"/>
    <product id="material2" amount="3"/>
  </products>
</job>
```

۴-۲-۲-۱۰- عامل ها

بخش عامل ها در شبیه سازی، جایی است که تیم ها در سمت سرور تعریف شده اند و در شبیه سازی شرکت می کنند. عواملی که در اینجا تعریف شده اند با عوامل تعریف شده در سمت سرویس گیرنده (client) که توسط شرکت کنندگان ساخته شده، مطابقت دارند. این تطبیق عامل ها در پارامترهای تابع tournamentmode در بخش ۲,۱ توضیح داده شده است.

ویژگی های برچسب agent به صورت زیر است:

- team: نام تیم در سمت سرور است.
- agentclass: نام کلاس اصلی پیاده کننده عامل هاست. برای سناریوی سال ۲۰۱۵ کلاسی که استفاده می شود،massim.competition2015.GraphSimulationAgent می باشد.
- agentcreationclass: نام کلاس جاوا که پیکربندی تجزیه شده (parse) توسط زیر برچسب configuration را در خود دارد. برای سناریوی سال ۲۰۱۵ کلاس مورد استفاده massim.competition2015.GraphSimulationAgentParameter است.

زیر برچسب configuration برای سناریوی ۲۰۱۵ دارای سه ویژگی است:

- roleName: که اشاره به نام یکی از نقش های تعریف شده دارد.
- lat و lon: که محل ابتدایی عامل است.

۴-۲-۲-۱۱- پیکربندی حساب های کاربری[۱]

بخش accounts در فایل پیکربندی، تیم های شرکت کننده در مسابقه را پیکربندی می کند و اعتبارنامه هر تیم، اجازه اتصال تیم را به MASSim می دهد.

- actionclassmap دارای یک ویژگی نام است و تمام action های موجود را برای

۱ Accounts

account عامل ها تعریف می کند. هر actionclass دارای یک کلاس ویژگی و یک شناسه است. ساختار account به صورت زیر است:

- actionclassmap: ارجاع به نام actionclassmap دارد که برای این account استفاده می شود.
- auxtimeout: timeout اضافی برای پیام است. هدف از این پارامتر دادن زمان اضافه است تا به سرور اجازه دهد که پیام را پردازش کند.
- defaultactionclass: برای تنظیم کردن کلاس پیش فرض action.
- maxpacketlength: تعیین طول بیشینه یک پیام.
- password: رمز عبور برای یک عامل.
- team: نام تیم عامل.
- timeout: timeout مربوط به پیام ها.
- username: نام کاربری عامل.

۴-۲-۳- ارتباط با MASSim

ارتباط با سرویس دهنده MASSim که در بخش ۴-۲-۳- معرفی شد، به سه وسیله متفاوت که هر کدام در سطح مجزایی از انتزاع هستند، امکان پذیر است. این سه سطح عبارتند از [۶]:

- **ارتباط در لایه protocol**: این ارتباط که در بخش ۴-۲-۴- صحبت می شود، چگونگی ارتباط با عامل ها را در پایین ترین سطح نشان می دهد که با استفاده از آن ها می توان پروتکل ارتباطی client-server را پیاده ساخت. این بدان معناست است که از شما انتظار می رود که یک اتصال تصدیق شده را بسازید که با MASSim-server ارتباط برقرار کند و با آن از طریق پیام های XML در ارتباط باشد.
- **ارتباط توسط واسط استاندارد محیط (eismassim)**: یک واسط سازگار با محیط EIS[۱] که پروتکل ارتباطی ذکر شده در بالا را پیاده سازی می کند. این رابط یک ارتباط تصدیق شده با سرور را ایجاد می نماید که دریافت ادراکات و انجام اقدامات در محیط

۱ http://sourceforge.net/projects/apleis/

را به فراخوانی متدهای جاوا و ارزیابی پاسخ آن ها، تقلیل می دهد. این روشی است که ما برای اتصال سامانه ی عامل گرای خود در محیط Jason، به سرویس دهنده MASSim استفاده کرده ایم. این ارتباط در بخش ۴-۲-۵- بحث می شود.

- **ارتباط توسط عامل های اولیه جاوا (javaagents)**: یک تیم ساده از عامل ها که برای تست محیط پیاده سازی شده است. اگرچه می توان آن ها را با هوش مصنوعی ارتقا داد، ولی ما آن را توصیه نمی کنیم. توجه کنید که dummy agent ها شما را ملزم به استفاده از واسط محیط ذکر شده در قسمت قبل می کنند.

۴-۲-۴- ارتباط در لایه پروتکل

در این قسمت به توضیح فایل پروتکل موجود در بسته نرم افزاری مسابقات می پردازیم. هدف استفاده از این قسمت برای درک محتوا و سازمان پیام هایی است که بین عامل ها و سرور رد و بدل می شوند [۷].

۴-۲-۴-۱- اصول کلی ارتباط عامل - سرور

عامل های هر تیم شرکت کننده به صورت محلی اجرا می شود (روی سخت افزار شرکت کننده). محیط به گونه ای شبیه سازی شده که تمام عامل های تیم های رقیب action هایشان را با اجرا روی سرور شبیه ساز از راه دور انجام می دهند.

عامل ها با سرور مسابقه از طریق استاندارد TCP/IP stack باsocket session interface در ارتباط است. اینترنت (آدرسIP و port) با سرور بازی (و سرور اختصاصی تست) که بعدا از طریق mailing list رسمی مسابقه اعلام خواهد شد، هماهنگ می شود.

عامل ها از طریق تبادل پیام XML با سرور در ارتباط هستند. پیام ها به شکل XML است که جلوتر توضیح می دهیم. ما توصیه می کنیم برای تولید و پردازش پیام های XML از جداسازهای استاندارد XML موجود برای زبان های برنامه نویسی استفاده کنید. توجه شود که پیام هایی با قالب نادرست، پیام هایی که طبق قواعد پیام نباشد، نادیده گرفته خواهند شد.

۴-۲-۴-۲- مروری بر پروتکل ارتباطی

هر دوره مسابقات شامل تعدادی بازی است. بازی دنباله ای از شبیه سازی هاست که در طی آن چندین تیم از عامل ها در تنظیمات مختلف روی محیط، رقابت می کنند. هرچند، از دید عامل، این مسابقات شامل تعدادی شبیه سازی در تنظیمات مختلف محیط و در برابر حریف های مختلف است.

مسابقه به سه مرحله یا فاز تقسیم می شود:

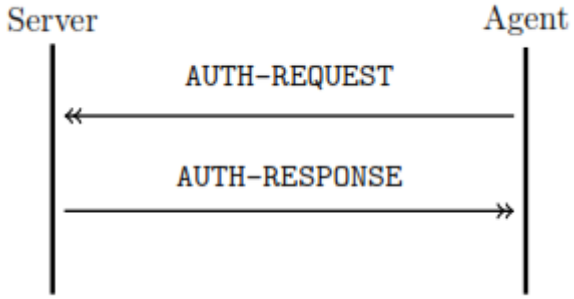
۱. فاز آماده سازی،
۲. فاز شبیه سازی و
۳. فاز نهایی.

در مرحله اول، عامل ها به سرور شبیه ساز متصل شده و با نام کاربری و رمز عبور، خود را معرفی می کنند (پیام AUTH-REQUEST). اعتبار سنجی برای هر عامل به صورت توزیع شده و پیشرفته ای با ایمیل پیش می رود. عامل نتیجه درخواست احراز هویت خود را (پیام AUTH-REQUEST) که می تواند موفقیت آمیز (succeed) یا شکست (fail) باشد، دریافت می کند. بعد از احراز هویت موفقیت آمیز عامل باید صبر کند تا اولین شبیه سازی مسابقه آغاز شود. **Error! Reference source not found.** فاز آماده سازی را نشان می دهد.

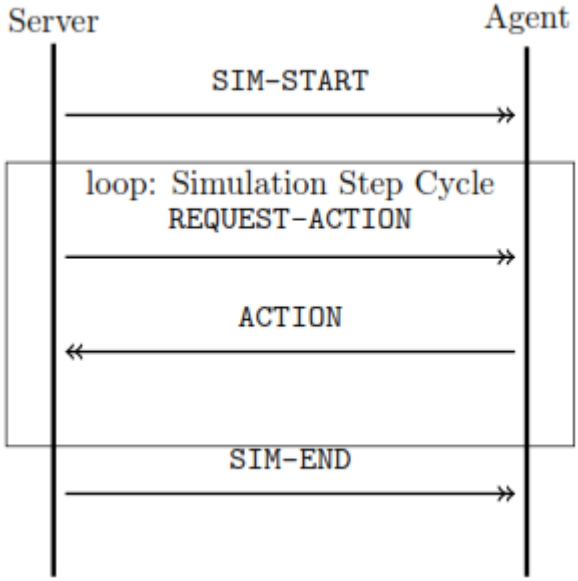
در آغاز هر شبیه سازی، عامل های دو تیم شرکت کننده مطلع می شوند (پیام SIM-START) و اطلاعات مخصوص شبیه سازی را دریافت می کنند.

در هر مرحله از شبیه سازی هر عامل ادراک در مورد محیط خود را دریافت می کند (REQUEST-ACTION message) و باید با انجام یک اقدام (پیام ACTION) بدان پاسخ گوید.

عامل باید پاسخ خود را در مهلت تعیین شده (deadline) بدهد. پیام action باید شامل شناسه عمل مورد نظر و در صورت نیاز پارامترهای آن باشد.



شکل (۴-۹) فاز آماده سازی

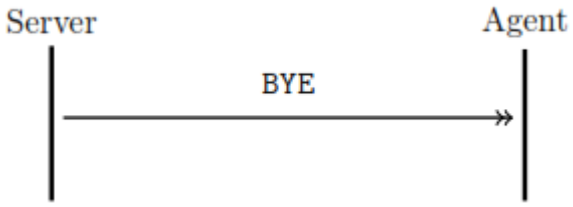


شکل (۴-۱۰) فاز شبیه سازی

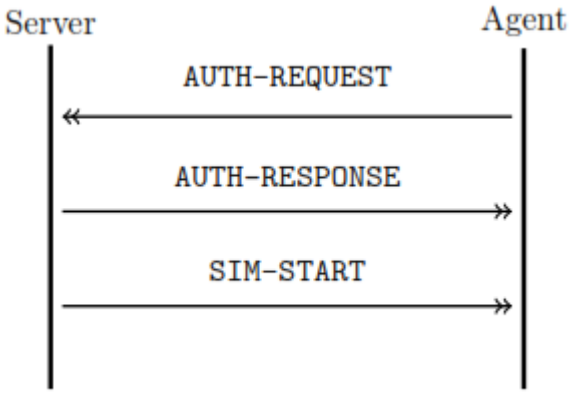
Error! Reference source not found. فاز شبیه سازی را نشان می دهد. زمانی که شبیه سازی به پایان رسید، عامل های شرکت کنندگان در مورد پایان مطلع می شوند (دریافت پیام SIM-END) که شامل نتیجه شبیه سازی است.

تمام عامل های که در حال حاضر در مسابقه شرکت داده نشده اند، باید منتظر باشند تا زمانی که سرور یا به آن ها پیام دهد که مسابقه را آغاز کن و به شرکت کنندگان بپیوند و یا پیام پایان مسابقه را بدهد.

در پایان مسابقه، تمام عوامل مطلع می شوند (پیام BYE). پس از آن سرور شبیه ساز به ارتباط با عامل ها پایان می دهد. **Error! Reference source not found.** فاز نهایی را نشان می دهد.



شکل (۴-۱۱) فاز نهایی



شکل (۴-۱۲) فرایند اتصال مجدد

۴-۲-۴-۳- اتصال مجدد

زمانی که یک عامل ارتباطش را با سرور شبیه سازی از دست می دهد، نتایج حاصل از مسابقه از بین نمی رود، فقط تمام اقدامات جاری عامل قطع شده، خالی(skip) در نظر گرفته می شود. عامل ها خود مسئول حفظ اتصال به سرور شبیه سازی هستند و در مواردی که اتصال قطع شود، به آن ها اجازه داده می شود تا دوباره وصل شوند.

عامل ها با انجام همان دنباله گام آغاز مسابقه می توانند مجددا به سرور شبیه سازی متصل شوند. پس از ایجاد اتصال به سرور شبیه سازی، سرور پیام AUTH-REQUEST می فرستد و AUTH-RESPONSE دریافت می کند. پس از احراز هویت موفق، سرور پیام SIM-START را به یک عامل می فرستد. اگر یک عامل شرکت کننده در شبیه سازی در حال حاضر در حال اجرا باشد، پیام SIM-START بلافاصله پس AUTH-RESPONSE تحویل داده شود. در غیر این صورت عامل تا زمان رسیدن شبیه ساز بعدی که بتواند در آن مشارکت کند، صبر می کند. در گام بعدی زمانی که عامل برای انجام یک اقدام انتخاب می شود، آن یک پیام REQUEST-ACTION استاندارد شامل درک عامل از گام فعلی محیط شبیه سازی و نتایج شبیه سازی تا به الان دریافت می کند. **Error! Reference source not found.** نشان دهنده یک تصویر از

فرایند اتصال مجدد است.

۴-۲-۴-۴- ساختار کلی پیام های xml

پیام های XML ای که بین سرور و عامل ها رد و بدل می شود از نوع رشته های zero terminated UTF-8 هستند. هر پیام XML که بین سرور شبیه سازی و عامل رد و بدل می شود شامل سه بخش است:

- **سرآیند استاندارد XML**: شامل سرآیند استاندارد به کار رفته در اسناد XML

```
<? xml version=”1.0” encoding=”UTF-8” ?>
```

- **بسته پیام:** عنصر ریشه[۱] ای تمام پیام های XML، <message> است که دارای دو صفت[۲] timestamp و message type identifier می باشد.
- **جدا کننده پیام:** توجه داشته باشید که به دلیل اینکه نوع هر پیام رشته zero terminated UTF-8 است، پیام ها توسط nullbyte از یکدیگر جدا شده اند.

timestamp یک رشته عددی حاوی وضعیت ساعت جهانی سرور شبیه سازی در زمان ایجاد پیام است. واحد ساعت جهانی میلی ثانیه است که نتیجه فراخوانی سیستمی استاندارد time در Unix موجود روی سرور شبیه سازی (اندازه گیری تعداد میلی ثانیه از ۱ ژانویه ۱۹۷۰ UTC) است. نوع شناسه پیام نیز یکی از مقادیر هفبرچسبانه ی زیر است:

```
auth-request, auth-response, sim-start, sim-end, bye,
request-action, action.
```

پیام هایی که از سرور به یک عامل ارسال می شوند شامل همه صفات عنصر ریشه است. با این حال، صفت timestamp را می تواند در پیام هایی که از یک عامل به سرور ارسال می شود حذف کرد. در چنین مواردی سرور هم به راحتی آن را نادیده می گیرد. مثالی از یک پیام **سرور به عامل** در زیر آمده است:

```
<message timestamp=”10001980000000” type=”request-action”>
<!--optional data-->
```

۱ Root element
۲ Attribute

```
</message>
```

نمونه ای از پیام **عامل به سرور**:

```
<message type=”auth-request”>
<!--optional data-->
</message>
```

بسته به نوع پیام، عنصر ریشه یعنی <message> می تواند شامل داده های شبیه سازی خاصی باشد.

۴-۲-۴-۵- AUTH-REQUEST (عامل به سرور)

زمانی که یک عامل به سرور متصل می شود، باید خود را با استفاده از نام کاربری و رمز عبور تهیه شده توسط سازمان مسابقه احراز هویت کند. با این روش ما از اتصالات غیر مجاز متعلق به شرکت کنندگان جلوگیری می کنیم. AUTH-REQUEST اولین پیامی است که یک عامل برای سرور مسابقه می فرستد.

بسته پیام حاوی یک عنصر<authentication> و بدون زیر عنصر است. این عنصر خود دارای دو صفت username و password است. **Error! Reference source not found.** نمونه ای از این پیام را نشان داده است.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<message type="auth-request">
  <authentication password="1" username="a1"/>
</message>
```

شکل (۴-۱۳) نمونه ای از پیام AUTH-REQUEST

۴-۲-۴-۶- AUTH-RESPONSE (سرور به عامل)

پس از دریافت پیام AUTH-REQUEST، سرور اعتبار نامه ارائه شده را بررسی می کند و توسط یک پیام AUTH-RESPONSE که نشان دهنده موفقیت یا شکست در احراز هویت است، پاسخ می دهد. این پیام دارای یک صفت timestamp هم می باشد که نشان دهنده زمانی ارسال پیام است.

بسته پیام شامل یک عنصر<authentication> و بدون زیر عنصر است. این عنصر دارای یک

صفت به نام result از نوع رشته است که می تواند دو مقدار"ok" و "fail" را بپذیرد. **Error! Reference source not found.** نمونه ای از این پیام را نشان داده است.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<message timestamp="1297263037617" type="auth-response">
  <authentication result="ok"/>
</message>
```

شکل (۴-۱۴) نمونه ای از پیام AUTH-RESPONSE

۴-۲-۴-۷- SIM-START (سرور به عامل)

شبیه سازی با آگاه ساختن عامل های مربوطه از جزئیات آغاز شبیه سازی، شروع می شود. این اطلاع رسانی با ارسال پیام SIM-START انجام می شود.

اطلاعات مربوط به آغاز شبیه سازی در یک عنصر <simulation> با صفات زیر است:

- نقش عامل مربوطه،
- شناسه ی شبیه سازی،
- تعداد کل مراحل شبیه سازی و
- محصولات موجود در آن شبیه سازی

یک گام[۱] شامل اجرای اقدامات تمام عامل هاست. بنابراین اگر یک شبیه سازی دارای n مرحله باشد، این بدان معناست که هر عامل n پیغام REQUEST-ACTION در طول شبیه سازی (با فرض یک اتصال پایدار به سرور) دریافت می کنند. **Error! Reference source not found.** نمونه ای از این پیام را نشان داده است.

۱ Step

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<message timestamp="1297263004607" type="sim-start">
  <simulation id="0" steps="500" team="A">
   <role name="Car" speed="3" maxLoad="550" maxBattery="500">
           <tool name="tool1"/>
           <tool name="tool2/>
   </role>
   <products>
    <product name="product1" volume="50" assembled="true">
     <consumed>
      <item name="it1" amount="5"/>
      ...
     </consumed>
     <tools>
      <item name="it2" amount="1"/>
      ...
     </tools>
    </product>
   </products>
  </simulation>
</message>
```

شکل (۴-۱۵) نمونه ای از پیام SIM-START

۴-۲-۴-۸- SIM-END (سرور به عامل)

هر شبیه سازی یک تعداد معینی از گام طول می کشد. در پایان هر شبیه سازی سرور در مورد پایان آن و نتیجه کارشان در شبیه سازی به عامل ها اطلاع رسانی می کند.

عنصر <sim-result> دارای دو صفت است. ranking رتبه بندی تیم و score نمره نهایی است. **Error! Reference source not found.** نمونه ای از این پیام را نشان داده است.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<message timestamp="1297269179279" type="sim-end">
  <sim-result ranking="2" score="9"/>
</message>
```

شکل (۴-۱۶) نمونه ای از پیام SIM-END

۴-۲-۴-۹- BYE (سرور به عامل)

در پایان مسابقه سرور به هر عامل اطلاع می دهد که آخرین شبیه سازی به پایان رسید و پس از آن اتصالات را قطع می کند. هیچ داده ای در ساختار این پیام وجود نخواهد داشت. **Error!**

Reference source not found. این پیام را نشان داده است.

```
<?xml version="1.0" encoding="UTF-8"?>
<message timestamp="1204978760555" type="bye"/>
```

شکل (۴-۱۷) پیام BYE

۴-۲-۴-۱۰- REQUEST-ACTION (سرور به عامل)

در هر مرحله شبیه سازی، سرور به عامل ها درخواست انجام یک اقدام را می دهد و برای آن ها ادراک های مربوطه را می فرستد.

این پیام، با توجه به پیچیدگی آن، بهتر است با استفاده از یک مثال توضیح داده شود. در **Error! Reference source not found.** بخشی از این نوع پیام نشان داده شده است. توجه داشته باشید که پیام **Error! Reference source not found.**یک نوع ساخبرچسبی است، که هرگز توسط سرور ارسال نمی شود.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<message timestamp="1297263230578" type="request-action">
<perception deadline="1297263232578" id="201">
 <simulation step="200"/>
 <self
  charge="19"
  load="9"
  lastAction="skip"
  lastActionParam=""
  lastActionResult="successful"
  lat="44"
  lon="44"
  inFacility="none"
  fPosition="-1"
  routeLength="2">
  <items>
   <item name="it1" amount="2"/>
   ...
  </items>
  <route>
   <n i="0" lat="10" lon="11"/>
   <n i="1" lat="10" lon="12"/>
   ...
  </route>
 </self>

<team money="1">
 <jobs-taken>
  <job id="job_id1"/>
  ...
```

شکل (۴-۱۸) بخشی از محتوای یک نمونه پیام REQUEST-ACTION ارسال شده توسط سرور برای عامل ها

در حال حاضر، لازم نیست به شرح عناصر یا به اصطلاح برچسب های تودرتو که در مثال واضح است، پرداخت. ما فقط بر روی عنصرهای مهم تمرکز خواهیم کرد.

- <perception> دارای دو صفت است.
  - deadline: اشاره دارد به آخرین لحظه ای که سرور یک اقدام را می پذیرد.
  - id: نمایش id مربوط به action که فرض شده به پیام action اضافه شده است.
- <simulation> دارای صفت step است که گام جاری در شبیه سازی را نشان می دهد.
- <self> نشان دهنده وضعیت وسیله (vehicle) با صفات زیر:
  - charge: میزان شارژ فعلی است.
  - load: میزان بار حال حاضر است.

- lastAction: آخرین اقدامی است که عامل در محیط انجام داده است.
- lastActionParam: پارامترهای آخرین اقدامی است که صورت پذیرفته است.
- lastActionResult: نتیجه دقیق آخرین اقدامی است که انجام شده است.
- batteryCapacity: حداکثر ظرفیت شارژ است.
- loadCapacity: حداکثر ظرفیت بار است.
- lat: عرض جغرافیایی عامل است.
- lon: طول جغرافیایی عامل است.
- inFacility: مکانی است که عامل در حال حاضر در آن واقع شده است (و یا "none")،
- fPosition : موقعیت قرار گیری عامل در مکان فوق را مشخص می کند (یا ۱-)
- routeLength: طول مسیری است که عامل آن را دنبال می کند.

- برچسب <items> شرح کالاهایی است که عامل حمل می کند. درون این برچسب زیر برچسب های <item> قرار می گیرد که هر کدام دارای دو صفت است:
  - name: نام کالا.
  - amount: مقداری که حمل می کند.
- برچسب route که با استفاده از تعدادی برچسب <n> مسیر عامل را مشخص می کند و دارای صفت های زیر است:
  - i: شماره در لیست مرتب شده اشاره گرهای مسیر.
  - lat: عرض جغرافیایی اشاره گر مسیر.
  - lon: طول جغرافیایی اشاره گر مسیر.
- <team> نشان دهنده وضعیت وسیله های تیم با صفت می باشد:
  - money: نشان دهنده مقدار پول موجود تیم و سپس زیر برچسب های
  - <jobs-taken> و <jobs-posted>: که هر کدام می توانند شامل

چندین برچسب <job> با صفت id باشند.

- <entities> شامل یک برچسب برای هر عامل در شبیه سازی. صفت های آن نیاز به توضیح نداشته و واضح است.
- <facilities> نشان دهنده تمام مکان های موجود در شبیه سازی (برای خوانایی بیش تر، ما صفات بدیهی که در زیر آمده است را از مثال فوق حذف کردیم):
  - chargingStation: نمایش یک ایسبرچسباه شارژ است؛ که در آن:
    - rate: به معنی نرخ شارژ است،
    - slots: نشان دهنده جایگاه های شارژ موجود در آن ایسبرچسباه است و
    - برچسب <info> حاوی اطلاعاتی (مانند تعداد وسایل نقلیه در صف انتظار برای شارژ) است که فقط در صورتی که عامل نزدیک مکان شارژ باشد نمایش داده می شود.
  - dumpLocation: نشان دهنده مکان زباله است.
  - shop: نشان دهنده یک فروشگاه و لیست تمام آیتم های موجود در آن است. همچنین شامل قیمت ها و مقادیر هر آیتم، در صورتی که عامل نزدیک فروشگاه باشد، نیز می شود.
  - storage: نمایش یک مکان ذخیره سازی با تمام محصولاتی که ذخیره یا تحویل داده شده اند.
  - workshop: نمایش مکان کارگاه.
- <jobs> شامل تمام کارها که در حال حاضر فعالند و یا در مرحله مزایده اند می شود، به عنوان مثال:
  - <auctionJob>
  - <pricedJob>

در نهایت، برای به دست آوردن درک بهتر از معنی همه این اطلاعات، می توانید توضیحات سناریو در بخش ۳-۱-را بخوانید.

۴-۲-۴-۱۱- ACTION (عامل به سرور)

عامل باید به پیام REQUEST-ACTION توسط یک اقدام که می خواهد انجام دهد، پاسخ دهد. ساختار پیام ACTION از یک عنصر <action> تشکیل شده که شامل صفت های type و id است. صفت type نشان دهنده اقدامی است که باید انجام شود و می تواند یکی از رشته های زیر باشد:

- "goto": با صفت اختیاری param ، وسیله ها را به یک موقعیت یا مکان دیگر می برد.
- "buy": با صفت اجباری param، اقدام به خرید تعدادی کالا از فروشگاه موجود در مکان فعلی می کند.
- "give": با صفت اجباری param، تعدادی از یک کالا را به هم تیمی می دهد.
- "receive": کالا از هم تیمی دریافت می کند.
- "store": با صفت اجباری param، تعدادی کالا را در مکان ذخیره سازی که در مکان فعلی است، ذخیره می کند.
- "retrieve": با صفت اجباری param، تعدادی از کالاها (که قبلا ذخیره شده) از مکان ذخیره سازی بازیابی می کند.
- "retrieve_delivered": با صفت اجباری param، تعدادی از یک کالا (که قبلا به عنوان بخشی از یک کار تحویل داده شده) از مکان ذخیره سازی بازیابی می کند.
- "dump": با صفت اجباری param، تعدادی از کالاها در مکان زباله را از بین می برد.
- "assemble": با صفت اجباری param، ساخت یک چیز با کالاها.
- "assist_assemble": با صفت اجباری param، ساخت یک چیز از کالاها با کمک عامل های دیگر.
- "deliver_job": با پارامتر اجباری، تحویل دادن تمام کالاها به مکان ذخیره که عامل در جهت تکمیل کردن کار حمل میکند.
- "charge": افزایش شارژ یک عامل با توجه به این که عامل در مکان شارژ است.
- "bid_for_job": با صفت اجباری param، برای یک کار مناقصه ای یک پیشنهاد ارائه می دهد.
- "post_job": با صفت اجباری param، ساختن یک کار برای ارسال.

- "continue": اگر یک عمل در حال انجام وجود دارد (goto یا charge)، آن را ادامه بده، در غیر این صورت هیچ کاری نمی کند.
- "skip": همانند continue
- "abort": هیچ کاری نمی کند، کارهای در حال انجام متوقف می شود.

با این که توضیحات سناریو شامل معانی دقیق اقدامات و پارامتر های آن است (بخش ۳-۱-۶-)، **Error! Reference source not found.** یک نمونه اقدام goto را نشان داده است.

```
<?xml version="1.0" encoding="UTF-8"?>
<message type="action">
  <action type="goto" param="at=51.805 lon=10.3355">
</message>
```

شکل (۴-۱۹) یک نمونه اقدام goto

صفت id رشته ای است که باید شامل شناسه پیام REQUEST-ACTION باشد. عامل باید بتواند به سادگی مقدار صفت id درون پیام REQUEST-ACTION را به درون صفت id پیام ACTION بریزد. در غیر این صورت این پیام دور انداخته خواهد شد.

توجه شود که پیام ACTION باید در مدت زمان تعیین شده در صفت deadline پیام REQUEST-ACTION تحویل داده شود. عامل ها باید پیام ACTION را قبل از این که مدت زمان deadline سر برسد بفرستند، تا سرور آن را بپذیرد. شکل این مطلب را نشان می دهد.

```
<?xml version="1.0" encoding="UTF-8"?>
<message type="action">
  <action id="70" type="skip"/>
</message>
```

شکل (۴-۲۰) پیام ACTION و به همراه صفت id

۴-۲-۴-۱۲- نتایج اقدامات

بخشی از پیام REQUEST-ACTION، نتیجه اقدام انجام شده قبلی است. معمولا سه ویژگی ارایه می گردد: lastAction که نام اقدام فرستاده شده توسط عامل است، lastActionParam پارامتر(های) اقدام فرستاده شده توسط عامل است و lastActionResult نتیجه آخرین اقدام است.

برای فهم تفاسیر مقادیر ممکن برای lastActionResult، به توضیحات مندرج در بخش

۳-۱-۶-۲- مراجعه کنید. در ضمن همه مقادیر ممکن یک بار دیگر در اینجا لیست شده اند.

- successful
- failed_location
- failed_unknown_item
- failed_unknown_agent
- failed_unknown_job
- failed_unknown_facility
- failed_item_amount
- failed_capacity
- failed_wrong_facility
- failed_tools
- failed_item_type
- failed_job_status
- failed_job_type
- failed_counterpat
- failed_wrong_param
- failed_unknown_error
- failed
- successful_partial
- useless

۴-۲-۵- واسط استاندارد محیط

۴-۲-۵-۱- درباره EISMASSim

EISMASSim بر اساس EIS پایه گذاری شده است که یک استاندارد پیش فرض برای تعامل عامل با محیط است. EISMASSim ارتباط بین MASSim-server و عامل ها را برقرار می کند که این کار با ارسال و دریافت پیام های XML به روش Java-method-calls و call-backs صورت می پذیرد. در ابتدا EISMASSim به صورت خودکار یک اتصال به MASSim-server ایجاد کرده و در حالت اتصال باقی می ماند. همچنین به صورت پیش فرض اطلاعات اجرای عامل ها را جمع آوری می کند. EISMASSim و EIS هر دو در فایل jar درون بسته گنجانده شده اند.

۴-۲-۵-۲- استفاده از EISMASSim

به منظور استفاده از EISMASSim در پروژه باید مجموعه ای از گام ها که در زیر آمده است، انجام داده شود:

۱. **راه اندازی class-path**: در مرحله اول باید EISMASSim و EIS را به پروژه اضافه شود. بهتر است از eis-0.3.jar و eismassim-2.2.jar استفاده شود.

۲. **ساخت یک نمونه از واسط محیط**: هدف این نیست که به طور مستقیم یک نمونه از کلاس واسط استاندارد ساخته شود. به جای آن توصیه می شود که از eis.EILoader استفاده شود. در اینجا یک مثال برای معرفی کلاس واسط محیط یا همان class-loader ذکر شده است.

```
EnvironmentInterfaceStandard ei = null;
try {
String cn = "massim.eismassim.EnvironmentInterface"; ei =
EILoader.fromClassName(cn);
}
catch (IOException e)
{
// TODO handle the exception
}
```

۳. **معرفی عامل ها**: اکنون که محیط آماده است، عامل ها باید به محیط معرفی شوند. برای این کار باید هر عاملی که از طریق واسط محیط با محیط در تعامل است به صورت جداگانه از طریق نام یا کد منحصر به فرد معرفی شود. برای هر عامل باید اقدامات زیر انجام شود:

```
try {
ei.registerAgent(agentName); } catch (AgentException e1)
{
// TODO handle the exception
}
```

۴. **ارتباط عامل ها با وسایل**: در این مرحله باید عامل ها با موجودیت ها مرتبط شوند. یک موجودیت یک اتصال به یک وسیله است، که بخشی از شبیه سازی اجرا شده توسط

MASSim-server است. می توان یک عامل را از طریق نام موجودیت، با موجودیت ارتباط داد. نام یک موجودیت در پیکربندی فایل های XML، مخصوص و منحصر به فرد است (به قسمت زیر توجه کنید). پس از این که یک عامل به یک موجودیت متصل شود، یک اتصال بهMASSim-server ایجاد می شود. در اینجا یک مثال از چگونگی ارتباط یک عامل با یک موجودیت را می توانید مشاهده کنید:

```
try {
ei.associateEntity(agentName,entityName); } catch
(RelationException e)
{
// TODO handle the exception
}
```

۵. **آغاز اجرا:** گام بعدی آغاز تمام اجراها است. در زیر چگونگی انجام این کار را می توانید ببینید:

```
try {
ei.start();
} catch (ManagementException e)
{
// TODO handle the exception
}
```

۶. **درک محیط:** درک محیط از دو طریق صورت می گیرد: ۱. دریافت تمام ادراک ها با فراخوانی تابع یا ۲.مدیریت ادراک همانند یک پیام بدین معنی که هر زمان یک درک می آید یک تابع شنونده به منظور دادن پاسخ فراخوانی می شود .توجه شود که این سیاست معمول EIS درباره ادراکات است. در اینجا یک مثال برای بازیابی تمام ادراکات بیان شده است:

```
try {
Collection<Percept> ret = getAllPercepts(getName());
// TODO interpret the percepts
} catch (PerceiveException e) {
// TODO handle the exception
} catch (NoEnvironmentException e)
{
// TODO handle the exception
}
```

۷. انجام یک کار بر روی محیط: اجرای یک عمل به معنی فراخوانی تابع performAction و ارسال نام عاملی که می خواهد کار را انجام دهد و شی action که نشان دهنده نوع عملی است که قرار است انجام شود. در زیر یک نمونه از اجرای یک عمل بیان شده است:

```
Action = new Action(...); try {
ei.performAction(agentName, action); } catch (ActException
e)
{
// handle the exception
}
```

۴-۲-۵-۳- پیکربندی EISMASSim

واسط محیط EISMASSim می تواند با استفاده از فایل eismassimconfig.xml پیکربندی شود. توجه شود که EISMASSim زمانی که واسط محیط آماده است به صورت خودکار بار گذاری می شود. کد زیر مثالی از تنظیمات پیکربندی EISMASSim را نشان می دهد:

```
<?xml version="1.0" encoding="UTF-8"?>
<interfaceConfig scenario="city2015" host="localhost"
port="12300" scheduling="yes" times="no"
notifications="no" timeout="5000" statisticsFile="no"
statisticsShell="no">
<entities>
<entity name="vehicle1" username="a1" password="1"
xml="yes" iilang="yes"/>
<entity name="vehicle2" username="a2" password="1"
xml="yes" iilang="yes"/>
<entity name="vehicle3" username="a3" password="1"
xml="yes" iilang="yes"/> <entity name="vehicle4"
username="a4" password="1" xml="yes" iilang="yes"/> ...
</entities>
</interfaceConfig>
```

ویژگی های برچسب <interfaceConfig> به شرح زیر است:

- **Scenario:** مشخص کننده سناریوی مسابقه است که به صورت پیش فرض مقدار گذاری شده است. برای سال ۲۰۱۵ این مقدار برابر با city2015 تنظیم شده است.
- **Host:** این مقدار برابر با آدرس URL مربوط به MASSim-server است که شبیه سازی را اجرا می کند. این مقدار بعنوان مثال می تواند localhost، یا یک IP صحیح یا نام یک

hostname واجد شرایط از سرور تیم شرکت کننده باشد.

- **Port**: مشخص کننده شماره پورت مربوط به MASSim-server است.
- **Scheduling**: از این مقدار برای فعال/غیرفعال کردن زمان بندی استفاده می شود. فعال بودن زمان بندی بدین معنی است که پیام اقدام، ارسال نمی شود مگر یک action-id معتبر وجود داشته باشد (شرح فایل پروتکل را برای جزئیات بیش تر action-id ببینید). این سازوکار اطمینان حاصل می کند که یک action-id فقط یک بار استفاده شود. توجه شود که مبادرت به ارسال پیام اقدام پس از ۵ ثانیه امکان پذیر است (که در timeout تعریف شده است). مقدار پیش فرض برای فعال بودن زمانبندی yes است.

اخطار: باید توجه داشت که اگرچه غیرفعال کردن زمان بندی، مسئولیت زمان بندی را به عهده کاربر می گذارد، اما باید دانست که سرور با بیش از یک اقدام در هر اتصال و simulation-step هماهنگ نیست.

- **Times**: از این ویژگی برای فعال/غیرفعال کردن ثبت کننده زمان[۱] استفاده می شود. اگر این ویژگی فعال باشد هر درک با یک time-stamp ثبت می شود. این اتفاق زمانی می افتد که هر درک توسط سرور تولید می شود (برای جزئیات time-stamps به توضیحات پروتکل مراجعه شود.)
- **Notifications**: این ویژگی اشاره می کند که آیا ادراکات به عنوان پیام بیایند یا نه؟ مقدار پیش فرض برای این ویژگی no است.
- **Timeout**: این ویژگی مشخص کننده یک عدد بر حسب میلی ثانیه برایtimeout های پروسه زمان بندی است. این ویژگی زمانی قابل استفاده است که ویژگی Scheduling فعال باشد. مقدار پیش فرض ۵۰۰۰ است.
- **statisticsFile:** این ویژگی تابع statistics را فعال می کند. کار این تابع محاسبه میانگین زمان پاسخ عامل ها و درصد تکرار هر نوع اقدامی است. مقداری که این تابع محاسبه می کند در فایل نشان داده می شود.
- **statisticsShell:** شبیه statisticsFile است و تفاوت آن در این است که نتایج در پوسته نشان داده می شود. مقدار پیش فرض هردو ویژگی no است.

---

۱ time-annotations

هر برچسب <entity> مشخص کننده یک اتصال منحصر به فرد به MASSim-server است. ویژگی های این برچسب به شرح زیر است:

- **Name:** این ویژگی مشخص کننده نام اتصال است. این پارامتر برای انجام اقدام و دریافت درک لازم است و نیاز است که منحصر به فرد باشد.
- **Username و password:** مشخص کننده اعتبارنامه ای است که مورد نیاز سازکار احراز هویت MASSim است (یا توسط سازمان و یا فایل پیکربندی سرور خود تیم تهیه می شود).
- **Xml**: از این ویژگی برای فعال/غیرفعال کردن چاپ پیام های xml ورودی/خروجی به کنسول که برای نمایش نتایج اشکال زدایی و اجرا مفید است، استفاده می شود. این ویژگی به طور پیش فرض غیرفعال است.
- **Iilang**: از این ویژگی برای فعال/غیرفعال کردن چاپ ادراکات در کنسول که برای نمایش نتایج اشکال زدایی و اجرا مفید است، استفاده می شود. این ویژگی نیز به طور پیش فرض غیرفعال است.

۴-۲-۵-۴- زمان بندی[۱]

اگر زمان بند فعال باشد واسط محیط اطمینان پیدا می کند که عامل ها به درستی با MASSim-server هماهنگ هستند. این امکانی است که این اطمینان را ایجاد می کند که بتوان تابع getAllPercepts را فراخوانی کرد و تابع performAction را فقط یکبار در طول هر step صدا زد. در هر گام سرور، یک شناسه اقدام تهیه می شود، که فقط یکبار می تواند استفاده شود. تا زمانی که یک شناسه اقدام جدید از سرور دریافت نشده، getAllPercepts و performAction تا وقتی که timeout تعریف شده توسط کاربر تمام شود، مسدود می شوند.

۴-۲-۵-۵- اقدامات برای سناریوی ۲۰۱۵

برای سناریو اقدامات و ادراکات ماهرانه ای طراحی شده است. هر اقدام و درک شامل یک نام است که از یک لیست اختیاری از پارامترها پیروی می کند. پارامترها باید همانند یک شناسه

۱ scheduling

تهیه شوند تا نشان دهنده لیستی از کلید-مقدار های ساده به فرم:

```
<key1>=<value1> <key2>=<value2>
```

باشد که هر جفت با یک فاصله[1] از هم جدا شده اند. ایجاد یک شی action که به تابع performAction پارامتر ارسال کند، بسیار ساده است.

```
Action a = new
Action("goto",newIdentifier("facility=shop1"));
```

از new Identifier("...") برای مشخص کردن لیستی از پارامترها برای یک اقدام استفاده می شود. راه درست مفهومی دیگر برای مشخص کردن پارامترها زمان بندی برای بروزرسانی در آینده است.

۴-۳- اتصال Jason به Massim-Server از طریق EIS

برای اتصال برنامه عامل ها در Jason به سرور مسابقات، از سطح دوم ارتباط یعنی ارتباط توسط واسط استاندارد محیط، که در بخش ۴-۲-۵- بحث شد، استفاده شده است. پیاده سازی این ارتباط شامل دو بخش کلی است:

۱. ایجاد یک اتصال به EIS،
۲. ترجمه مفاهیم EIS به Jason و بالعکس.

هرچند این دو بخش در پیاده سازی به هم گره می خورند، تفکیک آن ها موجب افزایش ماژولاریتی سامانه می شود. در ادامه صحبت خود را معطوف به توضیح این قسمت ها می نماییم. لازم به ذکر است کلیه پیاده سازی های این بخش با زبان جاوا انجام می شود.

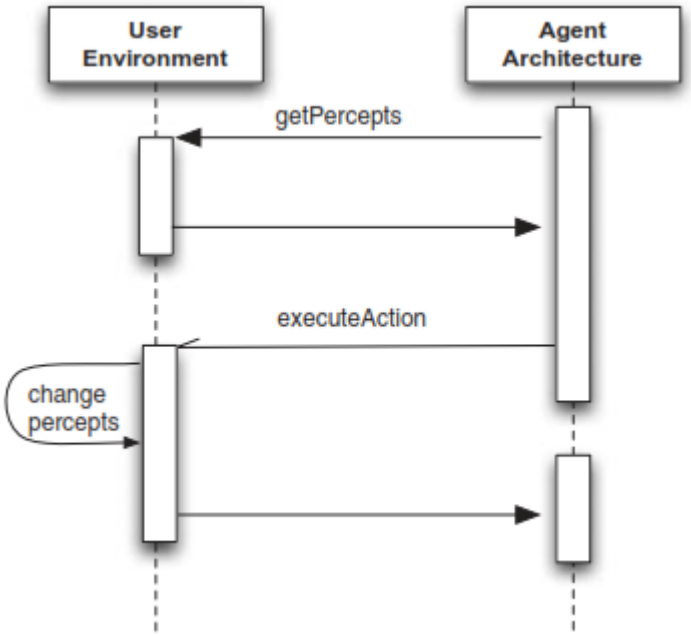
۴-۳-۱- ایجاد اتصال به EIS

نحوه ساخت یک نمونه از واسط استاندارد محیط و معرفی عامل ها به آن در بخش قبلی نشان داده شد. Jason در معماری داخلی خود یک کلاس جاوا تحت عنوان Environment پیشنهاد

---

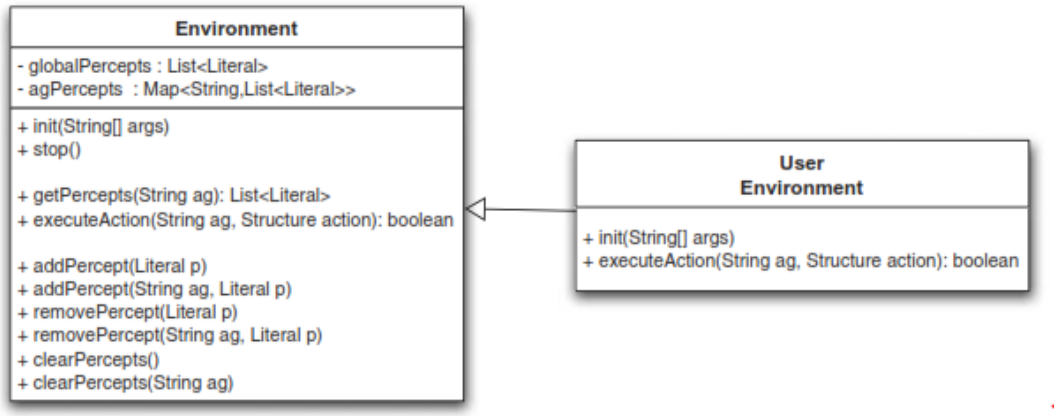
۱ space

می دهد که برای اتصال به محیط بیرونی در نظر گرفته شده است. تغییر در متدهای این کلاس به کمک فرایند ارث بری و ایجاد یک کلاس جدید توسط کاربر امکان پذیر خواهد بود. تنها بایستی متدهای مورد نظر برای ارتباط با محیط شبیه سازی باز نویسی شوند. نمودار ترتیب UML نشان داده شده در **Error! Reference source not found.** چگونگی دریافت ادراک ها توسط معماری عامل ها از محیط تعریف شده کاربر در Jason را نشان می دهد. متد ()getPercepts در ابتدای هر چرخه منطقی عامل صدا زده می شود تا ادراک ها را به عامل ها برساند. همچنین ()excuteAction یک اقدام در راستای قصد در نظر گرفته شده توسط عامل را به محیط می رساند [۵].



شکل (۴-۲۱) محاوره بین محیط شبیه سازی و معماری عامل

Error! Reference source not found. نحوه ارث بری برنامه کاربر از کلاس Environment و متدهای عمومی این کلاس را نشان می دهد. همان طور که مشاهده می شود در صورتی که محیط هم در زبان جاوا پیاده سازی شده باشد، ارتباط عامل و محیط بسیار ساده انجام خواهد شد.



شکل (۴-۲۲) پیاده سازی یک محیط توسط برنامه کاربر

برخی از متدهای جاوای این کلاس که برای برنامه نویسی محیط در Jason به کار می روند عبارتند از:

- addPercept(L): این متد لیترال L را به فهرست مشترک ادراک ها اضافه می کند؛ یعنی، همه عامل ها این ادراک را در صورتی که محیط را قبل حذف آن از فهرست مشترک ادراک ها، لمس کنند، خواهند داشت.
- addPercept(A,L): لیترال L را به فهرست ادراک های انحصاری عامل A اضافه می کند؛ یعنی، تنها در عامل A این ادراک را دریافت می کند.
- removePercept(L): لیترال L را از فهرست ادراک های مشترک تمامی عامل ها حذف می کند.
- removePercept(A,L): لیترال L را از فهرست ادراک های انحصاری عامل A حذف می کند.
- clearPercepts(): همه ادراک های موجود در فهرست ادراکات مشترک را پاک می کند.
- clearPercepts(A): همه ادراک های منحصر به عامل A را پاک می کند.

تنها نمونه های کلاس Literal که بخشی از بسته Jason هستند، بایستی به فهرست ادراک هایی که محیط نگهداری می کند، اضافه شوند.

پیاده سازی قسمت اصلی محیط منوط به بازنویسی دو متد init و executeAction در بدنه کلاس Environment کاربر است. متد init می تواند برای دریافت پارامترهای اولیه محیط از فایل پیکربندی برنامه به کار رود. این متد یک بار در آغاز اجرای برنامه صدا زده می شود. ما در این متد کدهای لازم را برای ایجاد یک نمونه از واسط استاندارد محیط، معرفی عامل ها به سرور

بازی و سایر تنظیمات اولیه، جاسازی کرده ایم.

متد executeAction را می توان هسته اصلی ارتباط با محیط دانست. این متد یک اقدام را روی محیط خارجی عامل انجام می دهد. مفهومی که در Jason، اقدام خارجی (External Action) نامیده می شود. این متد دو آرگومان را دریافت می کند، آرگومان اول رشته محتوی نام عامل است که به طور خودکار به عامل مجری اقدام مقید می شود. آرگومان دوم از جنس Structure (معادل structure موجود در prolog) داده ساختاری است که اقدام و پارامتر های آن را حمل می کند. در این متد ما کدهای انجام یک اقدام برروی محیط توسط EIS را جاسازی کرده ایم و البته قسمت های دیگری هستند که برای کنترل و اشکال زدایی به کار می روند. **Error! Reference source not found.** نمونه ای از کد نوشته شده برای توصیف این کلاس را نشان می دهد.

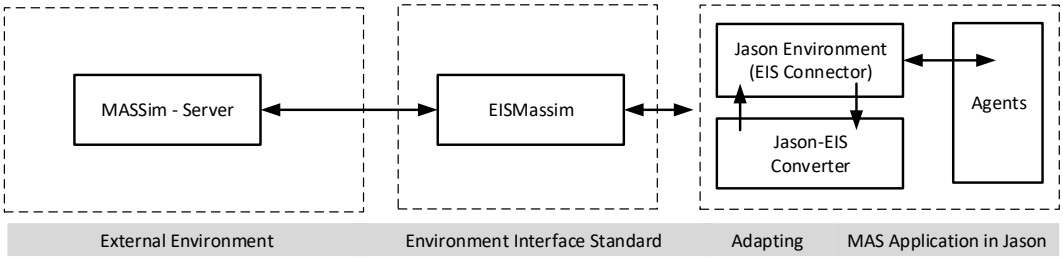
```
import jason.asSyntax.*;
import jason.environment.*;

public class <EnvironmentName> extends Environment {

    // any class members needed...

    @Override
    public void init(String[] args) {
        // setting initial (global) percepts ...
            addPercept(Literal.parseLiteral("p(a)"));

        // if open-world is begin used, there can be
        // negated literals such as ...
            addPercept(Literal.parseLiteral("~q(b)"));

        // if this is to be perceived only by agent ag1
            addPercept("ag1", Literal.parseLiteral("p(a)"));
    }

    @Override
    public void stop() {
        // anything else to be done by the environment when
        // the system is stopped...
    }

    @Override
    public boolean executeAction(String ag, Structure act) {
        // this is the most important method, where the
        // effects of agent actions on perceptible properties
        // of the environment is defined
    }
}
```

شکل (۴-۲۳) کد موجود در کلاس Environment سفارشی شده

۴-۳-۲- مبدل EIS-Jason

برای آن که ارتباط ما با محیط مسابقه برقرار شود، بایستی داده ساختارهای موجود در Jason که از خانواده زبان Prolog هستند را با داده ساختارهای EIS تطبیق دهیم. خوشبختانه در این مورد چون هر دو چارچوب از زبان جاوا بهره برده اند، نوشتن یک مبدل مفاهیم چندان پیچیده نخواهد بود. کلاس Converter که ما آن را توسعه داده ایم برای این منظور کافی است. کارکرد متدهای این کلاس روشن است و نیاز به توضیح بیش تر ندارد. شکل یک شماتیک ساده از نحوه اتصال برنامه به سرور مسابقات را مطابق آن چه گذشت، نشان می دهد. بدیهی است این شکل

حاوی جزئیات معماری برنامه چند عاملی ما نیست.



شکل (۴-۲۴) شماتیکی از نقاط ارتباطی سامانه و سرور مسابقات

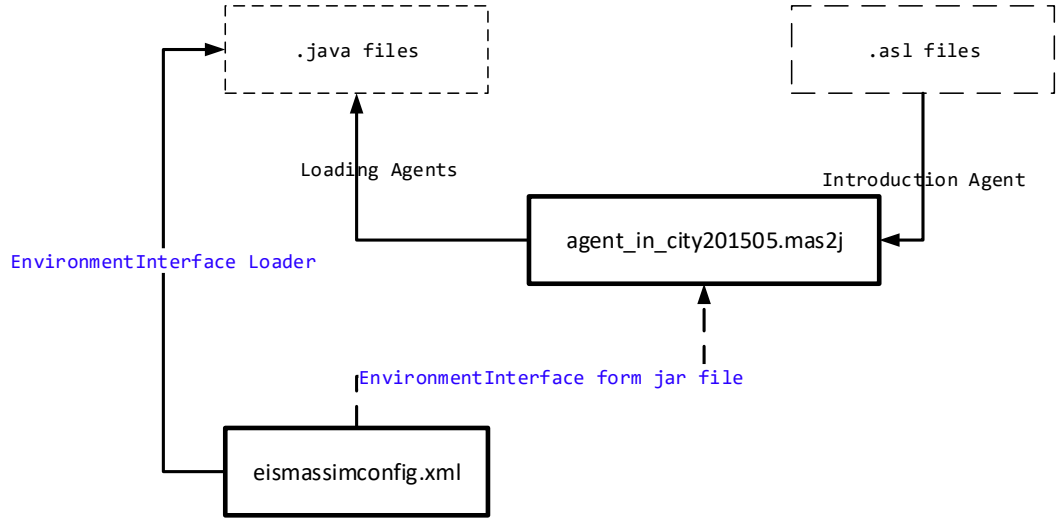
۴-۴- ساختار کد برنامه

در بخش قبل ما نحوه پیاده سازی یک اتصال از Jason به MASSim-Server را از طریق واسط استاندارد محیط (EIS) توضیح دادیم. در این قسمت به ساختار کد برنامه و فایل های تشکیل دهنده آن می پردازیم. معماری کلی سامانه در بخش معماری سامانه چند عاملی۳-۲-۶-، مطرح شد. نشان خواهیم دادید چگونه این معماری را پیاده سازی کرده ایم.

۴-۴-۱- طرح کد

به طور کلی کد تشکیل دهنده برنامه را می توان به سه بخش مجزا تفکیک کرد: بخش اول کدهای نوشته شده در چارچوب Jason و بخش دوم کدهای نوشته شده در زبان جاوا و بخش سوم فایل های پیکربندی. هر آن چه مربوط به برنامه عامل، باور ها، اهداف، طرح ها و قوانین آن است، در چارچوب Jason (فایل هایی با پسوند asl.) کد شده است.

کدهای جاوا اقدامات داخلی، World Model (که در فصل قبل بحث شد) و نیز کلاس های ارتباط را پیاده سازی کرده اند. در نهایت ارتباط بین این دو قسمت از طریق فایل های پیکربندی حاصل می شود. فایل های پیکربندی افزون بر این شامل اطلاعات لازم برای اتصال عامل ها به وسایل سرور بازی نیز هستند. **Error! Reference source not found.** یک شمای کلی از جایگاه فایل های پیکربندی و ارتباطی که برقرار می کند را به تصویر کشیده است. فایل های پیکربندی در این تصویر مربع های با خطوط پر رنگ هستند.



شکل (۴-۲۵) فایل های پیکربندی سامانه

فایل *mas2j*. همان طور که قبلا گفته شد، محتوی تعریف سامانه ی چند عاملی است. نام عامل ها، تعداد نمونه از هر عامل در سامانه، آدرس کد منبع عامل ها و اطلاعات محیط در این فایل است. در آغاز اجرای برنامه محیط پارامتر های اولیه خود را از اطلاعات موجود در این فایل می گیرد.

فایل eismassimconfig.xml محتوی تنظیمات لازم برای اتصال عامل های برنامه به سرور مسابقات است. این فایل هنگامی که یک نمونه از واسط استاندارد محیط بارگذاری می شود، فراخوانی گردیده و اطلاعات موجود در آن eismassim را پیکربندی می کنند. مهمترین پارامتر قابل تنظیم توسط این فایل نحوه دریافت ادراک ها از محیط است، که ما این پارامتر را به صورت notifications=”yes” تنظیم می کنیم (رجوع شود به بخش . . .).

نحوه بارگذاری یک نمونه از واسط استاندارد محیط به دو صورت است. حالت اول با استفاده از متد fromClassName است که آدرس یک کلاس جاوای محتوی این شی را به عنوان آرگومان می پذیرد و حالت دوم با استفاده از متد fromJarFile که از یک فایل jar شی مربوط را می خواند. آدرس این jar فایل هنگام اجرا از طریق یکی از پارامترهای فایل mas2j. به محیط ارسال می شود؛ که البته ما از شیوه اول استفاده می کنیم. خط نقطه چین موجود در **Error! Reference source not found.** موکد این موضوع است.

۴-۴-۲- فایل های جاوا

قسمت کدهای جاوا شامل چهار package به شرح آمده در زیر است که کلاس های موجود در

هریک از این بسته ها از کتابخانه های موجود در دایرکتوری lib استفاده می کنند. مهم ترین فایل این دایرکتوری jason.jar است.

- بسته env: محتوی کلاس های لازم برای اتصال یعنی EISConnector و Converter.
- بسته ia: اقدامات داخلی نوشته شده برای عامل ها.
- بسته competition2015: محتوی کلاس های لازم برای ایجاد یک مدل مفهومی از برنامه (Model World).
- بسته competition2015.scenario: ساختمان داده هایی جهت نگهداری مفاهیم سناریو از قبیل مکان ها، کارها، نقش ها و عامل ها را برای استفاده در World Model پیاده سازی می کند.

۴-۴-۳- فایل های Agent Speak (L)

قسمت کدهای Agent Speak (L) دارای دو package اصلی agent و asl است. بسته agent شامل کد منبع عامل ها و بسته asl شامل کدهای مشترک استفاده شده در همه عامل هاست. در قسمت عامل ها ما یک عامل با نام boss علاوه بر عامل های شرکت کننده در مسابقه داریم که در مواقعی تصمیماتی را برای اجرا به دیگر عامل ها می فرستد.

۴-۴-۴- اجرای برنامه

برای توسعه و کد نویسی برنامه ما از محیط توسعه مجتمع eclipse java ee mars استفاده کرده ایم. کنترل کننده پیش فرض وابستگی های برنامه در پروژه های این محیط ant است. به همین جهت برنامه را هم می توان در محیط eclipse اجرا کرد و هم توسط فایل ant موجود در پروژه و از طریق وارد کردن دستورات مربوط در خط فرمان سیستم عامل.

در آغاز اجرای برنامه آرگومان های کد زیر از فایل mas2j به متد init در کلاس EISConnector (رجوع شود به بخش ۴-۳-)، پاس داده می شود:

```
MAS agent_in_city201504
{

     infrastructure: Centralised

//EISConnector.init(String args[]) method args is here:
```

```
      environment: env.EISConnector
      (
            "lib/eismassim-2.4.jar",   // jar file
containing the environment implementation
          agent_entity(car1, connectionA1, a1),  //
agent[x] entities relation
          agent_entity(car2, connectionA2, a2),  //password
and more details in eismassimconfig.xml
          agent_entity(car3, connectionA3, a3),
          agent_entity(car4, connectionA4, a4),

          agent_entity(drone1, connectionA5, a5),
          agent_entity(drone2, connectionA6, a6),
          agent_entity(drone3, connectionA7, a7),
          agent_entity(drone4, connectionA8, a8),

          agent_entity(motorcycle1, connectionA9, a9),
          agent_entity(motorcycle2, connectionA10, a10),
          agent_entity(motorcycle3, connectionA11, a11),
          agent_entity(motorcycle4, connectionA12, a12),

          agent_entity(truck1, connectionA13, a13),
          agent_entity(truck2, connectionA14, a14),
          agent_entity(truck3, connectionA15, a15),
          agent_entity(truck4, connectionA16, a16)
      )
```

متد init اطلاعات مورد نیاز برای اتصال عامل ها به سرور مسابقه را از قبیل نام عامل ها توسط پارامتر String args[] دریافت می کند. سپس برنامه منتظر وقوع اولین در خواست سرور برای ارسال اقدام می ماند . یک طرح برای اداره کردن این درخواست نوشته شده است که رویداد محرک [۱] آن با نام request action در فایل common.asl موجود است. یعنی هر عامل به این رویداد با این طرح پاسخ می گوید.

```
+requestAction: true <-
      .abolish(requestAction);
      !nowStartNewStep.
```

بدنه طرح که در کد فوق آمده است، به عامل یک هدف با نام nowStartNewStep اضافه می کند که اکنون باید اجرا شود زیرا طرح اول به پایان رسیده است. اما پیش از آن کد .abolish(requestAction); یک اقدام داخلی Jason را فراخوانی می کند که وظیفه دارد

۱ Event Triggering

درخواست اقدام برای گام فعلی سرور را از فهرست ادراک های عامل پاک کند؛ زیرا، عامل دیگر به آن نیازی ندارد و گام بعدی را مجددا با همین رویداد آغاز می کند.

بدنه طرح nowStartNewStep در زیر آمده است. همان طور که دیده می شود این طرح هم به اقدام روی محیط منجر نمی شوند، تنها رویداد محرک طرح های اصلی هر عامل برای پاسخ به گام فعلی هستند. طرح هایی که مناسب بودن آن ها در context طرح بررسی می شوند.

```
+!nowStartNewStep: .my_name(Name) <-
      ia.getAllNewPercepts(Name);
      !start.
```

.my_name(Name) یک اقدام داخلی است که نام عامل را باز می گرداند و ia.getAllNewPercepts(Name); اقدام داخلی ماست که برای هر عامل تمام ادراک های فرستاده شده در هر گام را می گیرد. پیش از آن ادراک های قبلی را پاک کرده و مدل را بروز رسانی می کند. طرح start پس از این کار را ادامه خواهد داد.

.

مراجع

[١] S. Russell and P. Norvig, *Artificial-Intelligence-A-Modern-Approach*-, 3rd-Edition ed. Upper Saddle River, New Jersey 07458, United States of America.: Pearson Education, Inc., 2010.

[٢] P. Drozdowski and N. Beuschau, "Multi-Agent Programming in Jason ", Compute-B.Sc., Department of Applied Mathematics and Computer Science, Technical University of Denmark, 2800 Kongens Lyngby, Denmark, 2014.

[٣] T. Ahlbrecht, J. Dix, and F. Schlesinger. (2015). *Multi-Agent Programming Contest Official Website*. Available :https://multiagentcontest.org/aims-a-scope

[٤] T. Ahlbrecht, J. Dix, and F. Schlesinger, "Multi-Agent Programming Contest Scenario Description (2015 Edition)," ed: multiagentcontest.org, 2015.

[٥] R. H. Bordini, J. F. Hübner, and M. Wooldridge, *Programming Multi-Agent Systems in AgentSpeak using Jason*, 1 ed. John Wiley & Sons Ltd, The Atrium, Southern Gate, Chichester, West Sussex PO19 8SQ, England: John Wiley & Sons Ltd., 2007.

[٦] T. Ahlbrecht, J. Dix, and F. Schlesinger, "Multi-Agent Programming Contest Read Me First (2015 Edition)," 2015 ed: multiagentcontest.org, 2015.

[٧] T. Ahlbrecht, J. Dix, and F. Schlesinger, "Multi-Agent Programming Contest Protocol Description (2015 Edition)," 2015 ed: multiagentcontest.org, 2015.

واژه نامه

بخش الف: واژه نامه فارسی به انگلیسی

بخش ب: واژه‌نامه انگلیسی به فارسی

Abstract

A multi-agent system is a system that consists of several interactive intelligent agents. Multi-agent systems can be used to solve problems that are difficult or impossible to solve for a single agent or a classical integrated system. There are various methodologies and tools for multi-agent programming and development. The goal of this project is to study such systems, the challenges in development and oprating them, and finally design and implement a fully functional sample of them, using an applicable methodology and framework for participation in MAPC.

Keywords: Intelligent agents, multi-agent systems, agent-oriented, agent-oriented software engineering, agent-oriented methodologies.

Arak University
School of Engineering

Design and implementation of a multi-agent system for participating in multi-agent programming contest (MAPC)

A Thesis Submitted in Partial Fulfillment of the Requirement for the Degree of Bachelor of Science in Computer Engineering – Software Engineering

By:
Morteza Zakeri Nasrabadi, Ali Saberi, Mohsen Amirian, Navid Parsa and Mehrdad Tamiji

Supervisor:
Dr. Vahid Rafeh

September 2015