افزودن قابلیت تحمل پذیری خطا به متدولوژی MaSE برای سیستم های چند عامله

محمد حسین داورپور[1*]

1- مربی، گروه مهندسی کامپیوتر، واحد سمنان، **دانشگاه آزاد اسلامی**، سمنان، ایران
* پست الکترونیک: davarpour@semnaniau.ac.ir

چکیده

برنامه های کاربردی زیادی امروزه بر مبنای مفهوم سیستم‌های چند عامله شکل گرفته اند و نیازمند این هستند که به طور پیوسته و بی وقفه کار کنند. سیستم‌های چند عامله نیز از بروز خطا مصون نیستند. به همین دلیل لازم است که تحمل پذیری خطا به عنوان یک نیاز غیر وظیفه مندی تا حد امکان برای آنها تامین گردد. روش های ارائه شده برای تحمل پذیری خطا تا به حال، بیشتر مبتنی بر تکثیر عامل ها بوده اند که خود باعث پیچیدگی بیش از حد سیستم می گردد. از همین رو در این مقاله سعی شده است تا با تکیه بر روش تکثیر وظیفه ها به جای فقط تکثیر عامل ها و اعمال این روش در متدولوژی MaSE، با توجه به مطلوبیت و کاربرد زیاد این متدولوژی در تحلیل و طراحی سیستمهای چند عامله، به یک طراحی کارا با قابلیت تحمل پذیری خطا برای سیستمهای چند عامله دست یابیم.

کلیدواژگان

سیستم های چند عامله، تحمل پذیری خطا، تکثیر عامل، تکثیر وظیفه، متدولوژی MaSE

Adding Fault Tolerance to MaSE Methodology for Multi-Agent Systems

Mohammad Hossein Davarpour [1*]

1- Department of Computer Engineering, Semnan Branch, Islamic Azad University, Semnan, Iran.
* davarpour@semnan.aiu.ac.ir

Abstract

Many software applications today are designed based on multi-agent concept. These applications which need to work non-stop and continuos for a long period are prone to errors. Therefore, they are required to be fault tolerant as much as possible. Most of fault-tolerant methods provided for multi-agent systems are based on replicating the number of agents which leads to more complexity in these systems. In this paper we used task replication along with increasing numbers of agents in MaSE methodology to have a reliable architecture with high fault tolerante capability for multi-agent systems.

Keywords

Multi-Agent Systems, MAS, Fault Tolerance, MaSE Methodology, Agents Replication, Tasks Replication

1- مقدمه

یک سیستم چند عامله ، مجموعه ای از موجودیت های خودکار و تعاملی را در بر می‌گیرد. برنامه های کاربردی زیادی (نظیر کنترل ترافیک هوایی، کنترل فرآیند و...) بر مبنای مفهوم سیستم های چند عامله شکل گرفته اند و نیازمند این هستند که به طور پیوسته و بدون وقفه اجرا شوند. همچنین گسترش روز افزون سیستم های توزیع شده و شبکه های کامپیوتری بستری مناسب را برای استقرار سیستم های چند عامله در سیستم های واقعی بیش از پیش فراهم کرده است به طوری که گاهی می‌توان سیستم توزیع شده را از منظر سیستمی چند عامله نگریست. [1، 11].

مانند سیستم های توزیع شده، سیستمهای چند عامله نیز در معرض بروز خطاهای نرم افزاری یا سخت افزاری قرار دارند. خرابی یک جزء اغلب می‌تواند به خرابی کل سیستم بینجامد. به همین دلیل نیاز است که تا حد امکان تحمل پذیری خطا را به عنوان یک نیاز غیر وظیفه مندی برای این سیستم‌ها تأمین کنیم. کارهای متعددی برای پشتیبانی از تحمل پذیری خطا در سیستم های چند عامله انجام گرفته که ایده بیشتر این کارها از تکنیک های تحمل پذیری خطا که در سیستم های توزیع شده مطرح می‌باشند اقتباس شده است.نمونه ای از تکنیکهای بکاررفته در نوع خاصی از شبکه ها به طور مفصل در [1] بررسی شده است.

کارهای متفاوت ارایه شده بر روی جنبه‌های مختلفی از تحمل پذیری خطا در سیستم های چند عامله تمرکز کرده‌اند. [12] روشی را برای تحمل پذیری خطا در عامل متحرک ارائه می‌دهد. این روش مبتنی بر مدل کردن اجرای عامل به عنوان یک توالی از مسائل توافق می‌باشد. متدولوژی FATMAS به عنوان متدولوژی تحمل پذیر خطا برای سیستم های چند عامله در [5] ارائه گردیده است. [14] به ارائه پروتکل SG-ARP پرداخته است که به عامل های متحرک اجازه می‌دهد که علی رغم خرابی ارتباط یا خرابی گره، اجرای خود را به درستی انجام دهند.

[3] نیز کوشیده تا تکنیک‌های سنتی تحمل پذیری خطا در سیستم های توزیع شده را با توجه به تئوری‌های مطرح در سیستم های چند عامله با آنها بیامیزد. مهمترین تکنیکی که برای تحمل پذیری خطا از سیستم های توزیع شده اقتباس شده است، تکثیر کردن عامل ها در سیستم برای افزایش قابلیت اطمینان می‌باشد. [15،4]. چارچوب DarX به عنوان یک روش بسیار مناسب که تحمل پذیری خطا را به صورت وقفی و بر مبنای تکثیر عامل ها برای سیستم های چند عامله فراهم می‌کند در [13،6،2] توصیف شده است. علی‌رغم توانایی روش تکثیر گره ها، تکثیر بی رویه، در ذات خود پیچیدگی و سربار زیادی را به سیستم اضافه خواهد کرد و در نتیجه به کاهش کارآیی آن منجر خواهد شد. بنابراین لازم است که ایده بسیار سودمند تکثیر، به نحوی

کاراتر مورد استفاده قرار گیرد. تکثیر وظیفه های یک عامل در عامل های دیگر، جایگزینی عملی برای تکثیر فقط عامل‌ها می باشد که در قسمت‌های بعد توضیح داده می شود.

از طرفی دیگر با پیشرفت مهندسی نرم افزار عامل گرا (AOSE) ، متدلوژیهای جدیدی با هدف بهبود مسئله تحلیل و طراحی سیستم های چند عامله پدید آمده اند. برخی از این متدلوژیها در [10،9] با هم مقایسه شده‌اند. در این بین، MaSE یکی از مهمترین وکاملترین این متدلوژیها برای استفاده در سیستم های چند عامله می‌باشد. اهمیت MaSE به این لحاظ است که اولاً دارای چرخه حیات کامل تولید بوده و ثانیاً ابزار agentTool به عنوان ابزاری برای استفاده از این متدلوژی در تحلیل، طراحی و پیاده سازی سیستم های چند عامله توسط ارائه دهندگان آن، عرضه شده است.[7 ، 8 ،16]

در این مقاله سعی شده است که با توجه به مقبولیت متدولوژی MaSE و کاربرد بسیار زیاد آن برای محیط های چند عامله بسته تحمل پذیری خطا به نحوی کارا برای طراحی سیستم های چند عامله در این متدولوژی گنجانده شود.

بدین منظور، ابتدا به بررسی مدل‌های خطا و خرابی در سیستم‌های چند عامله خواهیم پرداخت. سپس روش پروتکل تشخیص و ترمیم خطا با استفاده از تکثیر را بیان خواهیم کرد. بعد از آن نگاهی مختصر به متدولوژی MaSE خواهیم داشت و پس از آن به شرح نحوه اعمال روش بیان شده در این متدلوژی روی خواهیم اورد. در پایان نیز با بیان کارهای قابل انجام در زمینه تحمل پذیری خطا در سیستم های چند عامله یک نتیجه گیری کلی از بحث انجام خواهد شد.

2- سیستم‌های چند عامله، خطاها و خرابی‌ها

پیش از ارائه راه حل برای تحمل پذیری خطا در این بخش به طور مختصر به بیان تعریفی از انواع مدل های خطا و خرابی می پردازیم. به طور کلی می‌توان انواع خطا و خرابی موجود در سیستم های چند عامله را به صورت زیر بیان کرد.[15،13]

2-1- خطاهای برنامه

خطاهایی در برنامه نویسی که با آزمایش سیستم تشخیص داده نشده اند.

2-2- حالات پیش بینی نشده

خطاهای از قلم افتادگی مسائل در برنامه نویسی. به این منظور که برنامه نویس حالت خاصی را در نظر نگرفته است، به طوری که در یک نقطه مورد انتظار سرویس لازم ارایه نمی شود.این خطا معمولا آنی است و بر رفتار کل سیستم چندان اثر گذار نمی باشد.

2-3- حالات پیش بینی نشده

سیستم از کار بیفتد یا با کمبود منابع مواجه شود.

2-3- خطاهای ارتباطاتی

کند شدن، خرابی و یا سایر مشکلات موجود دراتصالات ارتباطی میان مولفه-های سیستم

2-5- پدیدار شدن رفتار ناخواسته

سیستم رفتاری از خود نشان بدهد که پیش بینی نشده است. رفتاری که ممکن است برای سیستم مفید یا مضر باشد.از آنجا که عامل رفتاری دلبخواه از خود نشان می دهد به این نوع خطا خرابی خود سرانگی نیز گفته شود.

2-6- خرابی مربوط به زمان

این خرابی هنگامی اتفاق می افتاد که یک پردازشگر یا یک سرویس نمی تواند در مدت زمان مقرر شده تحویل داده شود یا کامل گردد. مشخصاً در صورتی که برای ارائه سرویسها یا انجام پردازش ها، ضرب الاجلی وجود نداشته باشد. این خطاها اتفاق نمی افتند.

با این مقدمه و معرفی مختصر خطاهایی که باید مد نظر یک سیستم تحمل پذیر خطا قرا گیرند ، به ارایه روشی کارا برای تامین تحمل پذیری خطا در سیستم های چند عامله می پردازیم.

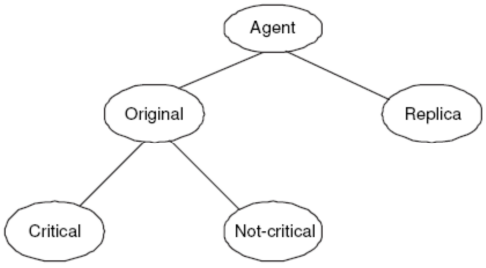
3- تحمل پذیری خطا مبتنی بر تکثیر وظیفه

روش‌های تحمل پذیری خطای موجود برای سیستمهای چند عامله عموماً مبتنی بر تکثیر عامل ها می‌باشند.در این روش‌ها طراحان سیستم با سربار کردن تعداد زیادی عامل جایگزین در سیستم چند عامله به منظور نیل به تحمل پذیری خطا پیچیدگی زیادی را به سیستم می افزایند. یک ایده جایگزین این است که ابتدا در صورت امکان وظایف را تکثیر کنیم وتنها در صورت لزوم به تکثیر خود عامل ها بپردازیم. می‌توان عامل‌ها را با توجه به تکثیر، در این روش به دو نوع اصلی و المثنی تقسیم کرد. عاملی المثنی است که به منظور افزودن قابلیت تحمل پذیری خطا از روی عامل اصلی تکثیر شده باشد. به طور کلی اگر یک عامل در خرابی باشد آن گاه باید عامل دیگری در سیستم وجود داشته باشد که بتواند به جای آن قرار بگیرد و کار آن را انجام دهد. روشی که برای تحمل پذیری خطا در این جا ارائه گردیده قصد دارد که تعداد نسخه های تکثیر شده از یک عامل را کمینه کند.

بنابراین در این روش فقط تعداد خاصی از عامل‌ها تکثیر می‌شوند. برای تعیین این که کدام یک از عامل‌ها باید تکثیر شوند و کدام یک نیاز نیست که تکثیر شوند، لازم است که به تعریف عامل های بحرانی و غیر بحرانی بپردازیم. عاملی را بحرانی می گوئیم که حداقل یکی از وظایفش نتواند به وسیله هیچ یک از دیگر عامل‌ها انجام شود. عامل غیر بحرانی عاملی است که هر کدام از وظایفش حداقل بوسیله یک عامل دیگر قابل انجام باشد . در صورتی که یک عامل غیربحرانی باشد نیاز به تکثیر ندارد چرا که در صورت بروز خطا عامل هایی هستند که بتوانند کار آن را انجام دهند. اما در مورد عامل های بحرانی ناچار از تکثیر آن‌ها هستیم. طبقه بندی ذکر شده برای عامل ها را می توان به صورت شکل 1 نشان داد.

لازم به ذکر است برای افزایش کارآیی، هر عامل فقط یکبار و هر وظیفه فقط در یک عامل تکثیر می‌شود. بدین ترتیب توانسته ایم از پیچیدگی سیستم تا حد امکان بکاهیم. حال هنگام کار سیستم لازم است که هر عامل، عامل های همکار (همتا) خود یعنی عاملهایی که با آن‌ها وظیفه مشترک دارد را بشناسد. مسئله مهم دیگر در تحمل پذیری خطا، نحوه ترمیم خطاست.

بنابراین سیستم لازم است بتواند اولا از بروز خطا باخبر شده به نحوی آن را تشخیص دهد و ثانیا به ترمیم آن بپردازد.



شکل 1. انواع مختلف عامل

در نوشتن فرمول‌ها رعایت نکات زیر الزامی است:

1- ارتباط با عامل از بین رفته است اما :

الف) عامل می‌تواند همه وظیفه هایش را انجام دهد

ب) عامل قادر است تنها برخی از وظیفه هایش را انجام دهد.

ج) عامل نمی تواند هیچ یک از وظیفه هایش را انجام دهد.

2- ارتباط با عامل برقرار است اما :

الف) عامل تنها برخی از وظیفه هایش را انجام می‌دهد.

ب) عامل نمی تواند هیچ یک از وظیفه هایش را انجام دهد.

در هر دو این حالات، عامل قادر نخواهد بود که سرویس مورد انتظار را ارائه دهد و در هر حال هر عامل باید قادر باشد که خرابی عامل همکارش را تشخیص دهد و به ترمیم آن بپردازد.

بدین منظور یک پروتکل دست دهی استفاده می‌شود تا تضمین شود که عامل های همکار از این که همکار آن ها از کار افتاده است یا برقرار است، با خبر باشند. بدین منظور در سطح پیاده سازی می‌توان متغیری با مقدار اولیه صفر در نظر گرفت که همه همکاران یک عامل قادر به تغییر آن هستند.

اگر عامل نتوانست به همکار خود در پروتکل دست دهی پاسخ دهد، آن گاه همکار آن عامل یک واحد به این متغیر که می توان آن را متغیر از کار افتادگی عامل دانست اضافه می‌کند. اگر پس از دست دهی همه همکاران، مقدار این متغیر با اندازه تعداد همکاران برابر باشد، هر یک از همکاران متوجه می‌شوند که عامل از کار افتاده است اما در غیر اینصورت عامل به طور کامل از کار نیافتاده است و فقط نمی تواند با یک یا چند همکار خود ارتباط داشته باشد.

از طرف دیگر اگر یک عامل در حالت خرابی باشد و نتواند وظیفه ای را انجام دهد باید همکار خود را با ارسال پیغامی برای جایگزین شدن با خبر کند. بدین صورت خرابی شناسایی می‌شود.

به منظور ترمیم خطا ، هر عاملی که در حالت خرابی قرار می‌گیرد، لازم است که همکارش یک نسخه از دانش آن را داشته باشد تا بتواند عملیات را از آخرین نقطه غیر خراب ادامه دهد. این مسأله مشابه عقب گرد کردن در تراکش ها به نظر می‌رسد.

داشتن یک نسخه از دانش یک عامل می‌تواند با استفاده از تبادل پیام بین عاملها و یا وجود یک حافظه مشترک بین عاملهای همکار انجام شود.البته به طور کلی وجود حافظه مشترک با توجه به سربار زیاد اجرای تبادل پیغام مناسبتر به نظر می رسد.

با توجه به آنچه در این قسمت گفته شد روش مبتنی بر تکثیر وظیفه ها می تواند باعث شود که افزودن قابلیت تحمل پذیری خطا سربار کمتری بر سیستم چند عامله تحمیل کند. در قسمت بعد MaSE را به عنوان یک متدولوژی مناسب برای اعمال این روش مد نظر قرار خواهیم داد.

4- تحلیل و طراحی سیستم های چند عامله با استفاده از MaSE

وجود سیستم های چند عامله منجر به ایجاد برنامه های توزیع شده هوشمند زیادی گردیده است که روز به روز بر تعداد آنها افزوده می شود . بسیاری از راههای سنتی طراحی سیستمهای نرم افزاری نمی تواند با طراحی سیستمهای چند عامله مطابقت کند، به همین دلیل تلاشهای زیادی برای دستیابی به روشهایی تحلیل ، طراحی و توسعه سیستمهای چند عامله انجام گرفته است . یکی از این تلاشها منجر به پیدایش متدولوژی MaSE گردیده است.[8،7]

این متدولوژی، برخلاف بیشتر متدولوژی های مطرح شده برای سیستم های چند عامله دارای چرخه حیات کامل می باشد و بیشتر در محیط های چند عامله بسته به کار می رود. متدلوژی MaSE دارای 2 فاز اصلی تحلیل و طراحی است که در زیر آنها را به طور مختصر توضیح می دهیم .

فازهای اصلی MaSE و مراحل مطرح در هر فاز در شکل 2 نشان داده شده است.

4-1- فاز تحلیل

فاز تحلیل مراحل زیر را در بر می گیرد .

1- Capturing Goals

2- Applying Use Cases

3- Refining Roles

4-1-1- Capturing Goals

اولین گام در متدولوژی MaSE به دست اوردن اهداف می باشد . در این مرحله مشخصات اولیه سیستم و نیاز مندیهای آن گرفته شده و به مجموعه ای ساخت یافته از اهداف سیستم تبدیل می گردد که در نمودار Goal Hierarchy نمایش داده می شود شکل3. این نمودار سلسله مراتبی از اهداف به هم مرتبط را نشان می دهد.

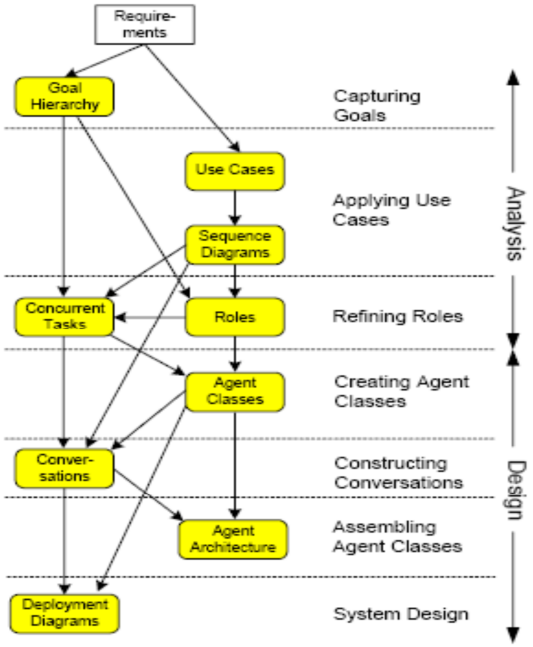
یک هدف، در MaSE همیشه در سطح سیستم تعریف می گردد.

دو گام اساسی در مرحله به دست آوردن اهداف وجود دارد:

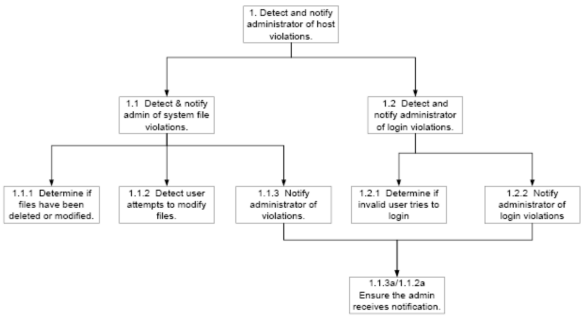
1- تشخیص و تعیین اهداف

2- سازماندهی اهداف

تحلیلگر با بررسی ماهیت نیازمندیها به تعیین اهداف می پردازد و سپس اهداف مشخص شده را چنان که نمایش داده شده در یک نمودار Goal Hierarchy سازماندهی می کند. در یک نمودار سلسله مراتب اهداف، اهداف بر اساس اهمیت سازماندهی می شوند هر سطح از سلسله مراتب باید شامل اهدافی شود که تقریبا از لحاظ منظور و اهمیت معادلند.



شکل 2. متدولوژی MaSE [7]



شکل 3. نمودار سلسله مراتبی اهداف [7]

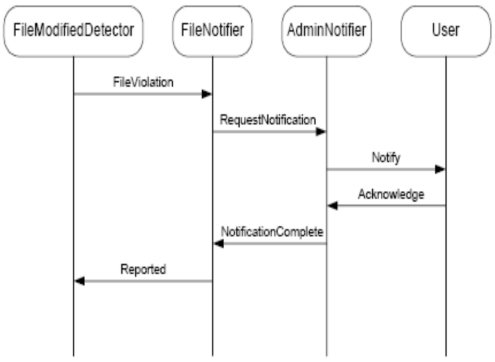
تحلیلگر همچنین باید زیر اهدافی را که برای تامین اهداف والد ضروری هستند بیابد. سرانجام، تحلیلگر هر هدف را با یک نقش و مجموعه ای از کلاس های عامل برای رسیدن به هدف مورد نظر مرتبط خواهد کرد.

4-1-2- Applying Use Cases

مرحله بکار بستن موارد کاربری یک مرحله بسیار مهم در تبدیل هدفها به نقش ها و وظیفه های مرتبط می باشد. تحلیلگر، مورد های کاربری را با توجه به کاربران و نیاز مندی های سیستم رسم می کند. هر مورد کاربری چنان که می دانیم توصیف یک توالی از رویدادها می باشد که رفتار مطلوب سیستم را مشخص می کند.

برای مشخص کردن بهتر ارتباطاتی که در یک سیستم چند عامله مورد نظر است تحلیلگر، موارد کاربری را به نمودارهای توالی(ترتیب) تبدیل می کند. شکل4 یک نمودار توالی رویداد ها را بین چند نقش، نمایش می دهد و در نتیجه ارتباطات کمینه ای که باید بین نقش ها اتفاق بیفتد را مشخص می کند.

نقش‌هایی که در این مرحله تشخیص داده می شوند برای تعیین کامل نقش‌های سیستم در مرحله بعد استفاده خواهند شد. در مراحل بعد، تحلیل-گر از رویدادهایی که در اینجا تشخیص داده شده اند برای تعریف وظیفه ها و مکالمه های بین عامل ها استفاده می کند.



شکل 4. نمودار ترتیب [7]

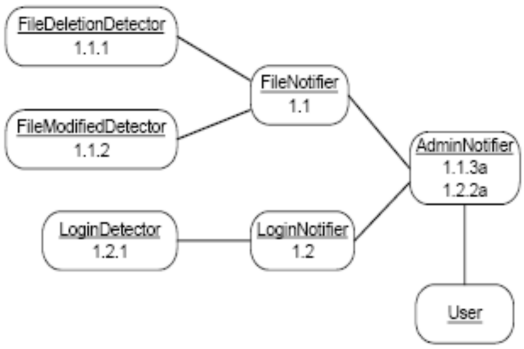
4-1-3- Refining List

سومین مرحله از MaSE در واقع تضمین این مطلب است که همه نقش‌های لازم شناخته شده اند. در این مرحله همچنین وظیفه هایی که رفتار نقش و الگوهای ارتباط را معین می کنند شناخته می شوند. با مرتبط کردن هر هدف با یک نقش بخصوص تضمین می کنیم که همه هدفهای سیستم لحاظ شده اند. هر نقش یک توصیف تجریدی از عمل مورد انتظار از یک موجودیت می باشد.

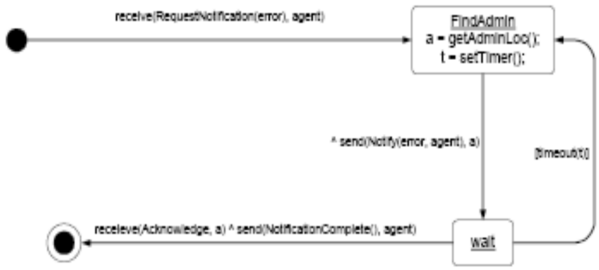
هر هدف به طور کلی به یک نقش نظیر می شود، با این حال مواقعی وجود دارد که چند هدف، برای به دست آوردن سهولت یا کارایی ، در یک نقش ترکیب می شوند.

نقش های به دست امده تا این مرحله ، مطابق شکل [5] در یک نمودار Role Model سازمان دهی می شوند. یکبار که نقش ها شناخته می شوند، وظیفه ها ایجاد می گردند. مهمترین و دشوارترین مرحله MaSE تبدیل نقش ها به Agent Class ها و تعیین مکالامات و رفتار داخلی آنها می باشد . برای نیل به این منظور ، نیاز داریم که وظیفه های سطح بالا را که می توانند به وظایف عامل تبدیل شوند، مشخص کنیم.

این وظایف کمک می کند که اجزا ء داخلی عامل ها و جزئیات مکالماتی را که عامل در انها شرکت می کند مشخص کنیم. جزئیات یک Role Model در [7] آمده است. یک نمودار مدل نقش وظیفه‌هایی را که هر نقش باید برای رسیدن به هدف مورد نظر اجرا کند و پروتکل‌های بین وظیفه‌ها مشخص می شوند. این پروتکل‌ها در واقع مجموعه پیام‌هایی بین وظیفه‌ها را که به آنها اجازه می‌دهد تا با هم به صورت همکار، کار کنند تعریف می‌کند. جهت پروتکل‌ها از آغاز کننده به سمت پاسخ دهنده می-باشد . این وظیفه های همروند را می توان به صورت FSM (ماشینهای حالت محدود) نشان داد. با استفاده از وظیفه های همروند می توان، پروتکل‌های پیچیده‌تر و مفاهیمی سطح بالاتر را بین چند عامل تعریف کرد. مثالی از نمودارهای Concurrent Task در شکل6 نشان داده شده است.



شکل 5. نمودار مدل نقش [7]



شکل 6. نمودار وظیفه همروند [7]

صورت کلی یک انتقال بسیار مشابه انتقالهای نمودار حالت در UML می‌باشد.

Trigger(args1) [guard]/transmission (arg2)

به این ترتیب که اگر رویدادی با آرگومان‌های args1 روی داد و شرایط [guard] هم برقرار بود، انتقال با مجموعه آرگومان‌های args2 اتفاق می‌افتد.

2-4- فاز طراحی

در فاز طراحی، مدل های تحلیل به ساختارهایی که برای پیاده سازی حقیقی سیستم مفید خواهند بود، تبدیل می گردند. فاز طراحی شامل 4 مرحله زیر می باشد.

1- Creating agent classes
2- Constructing conversations
3- Assembling agent classes
4- System deployment

4-2-1- Creating Agent Classes

در این مرحله، کلاس های عامل از روی نقشها معین شده و در یک نمودار کلاس عامل قرار می گیرند. شکل7 مانند هدفها و نقش‌ها ، به طور کلی یک نگاشت یک به یک را بین نقش‌ها و کلاس‌های عامل نیز تعریف می‌کنیم. با این حال طراح ممکن است چندین نقش را در یک کلاس عامل ترکیب کند. یا یک نقش را به چندین کلاس عامل نگاشت دهد.

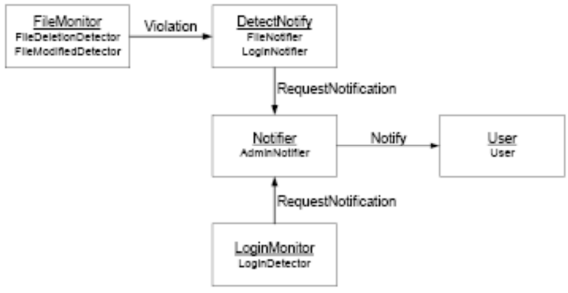
از آنجا که عاملها مسیرهای ارتباطی بین نقشها را به ارث می برند. هر مسیری بین دو نقش، مکالمات بین کلاس‌های مربوطشان را تشکیل می دهد. بنابراین، به محض این که طراح، نقش ها را به کلاس های عامل نسبت دهد، سازماندهی کلی سیستم معین می‌گردد. اغلب مطلوب است که دو نقش را که حجم زیادی از ترافیک پیام را به اشتراک گذاشته‌اند برای کارایی بیشتر با هم ترکیب می‌کنیم.

4-2-2- Constructing Conversations

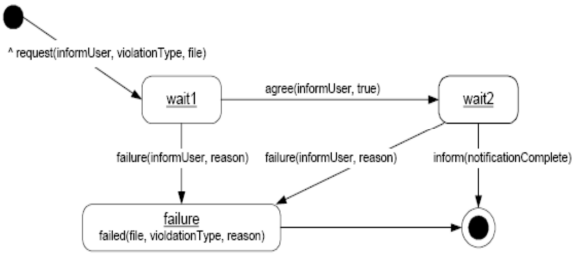
طراح ممکن است که دو فاز ایجاد مکالمات و سوار کردن عامل‌ها را به طور موازی انجام دهد. این دو مرحله به شدت به هم پیوند خورده‌اند. یک مکالمه در MaSE در واقع یک پروتکل هماهنگی را بین دو عامل تعریف می کند. به خصوص یک مکالمه دو نمودار کلاس ارتباط را در بر می گیرد. یکی برای آغاز کننده و یکی برای پاسخ دهنده.

یک نمودار کلاس ارتباط، یک جفت ماشین حالت محدود است که یک مکالمه را بین دو عامل شرکت کننده در آن نشان می دهد. یک طرف مکالمه در شکل8 نشان داده شده است. آغاز کننده همیشه مکالمه را با فرستادن اولین پیام آغاز می کند.

نمودار کلاس ارتباط بسیار شبیه نمودار Concurrent Tasks می باشد. تفاوت اصلی بین مکالمات و وظیفه های همروند این است که وظیفه های همروند ممکن است شامل چندین مکالمه بین نقش‌ها و وظیفه‌های متفاوت بسیاری شوند در حالی که مکالمات در واقع مبادلاتی دودویی بین دوتا عامل می باشند.



شکل 7. نمودار کلاس عامل [7]



شکل 8. نمودار کلاس عامل [7]

4-2-3- Constructing Conversations

در این مرحله از MaSE ،قسمت درونی کلاس های عامل ساخته می شود. می توان چندین قالب معماری مختلف برای این کار در نظر گرفت. BDI یکی از این قالب‌ها می باشد.

4-2-4- System Deployment

مرحله پایینی MASE ، پیکره بندی سیستمی را که باید به صورت واقعی پیاده سازی شود، معین می کند.

در MASE ها معماری کل سیستم را با استفاده از نمودارهای استقرار تعیین می کنیم. مطابق شکل 9 نمودارهای استقرار برای نشان دادن تعداد، نوع و موقعیت عامل‌ها در یک سیستم به کار می‌روند.

4-3- AgentTool

ابزاری است که MASE را پشتیبانی و اجرا می‌کند. در حال حاضر agentTool هر هفت مرحله MaSE را در کنار پشتیبانی خودکار تبدیل مدل‌های تحلیل به مدل‌های طراحی پیاده سازی کرده است. وجود agentTool به عنوان ابزاری قوی برای پشتیبانی از متدولوژی MaSE، این متدولوژی را نسبت به سایر متدولوژی‌های مطرح برای سیستم‌های چند عامله، قابل استفاده‌تر برای محیط‌های واقعی کرده است. جزئیات agentTool در [16] ذکر شده است.

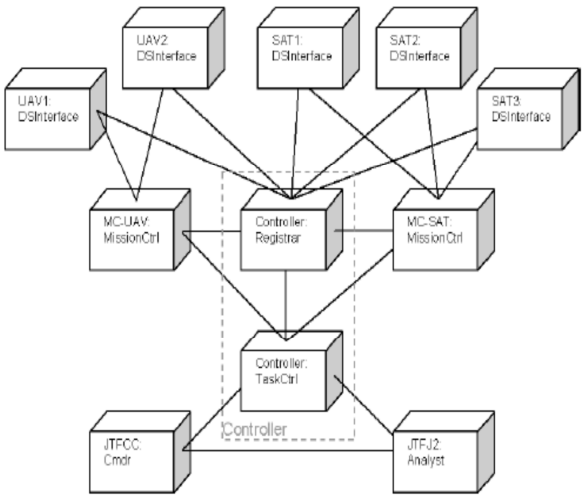
5- اعمال روش تکثیر وظایف در متدولوژی MaSE

یک سیستم چند عامله، در واقع، از چند عامل که با یکدیگر تبادل دارند تشکیل شده است. هر عامل چنان که گفته شد می تواند چندین وظیفه را انجام دهد و هر وظیفه نیز شرایط و منابعی را نیاز دارد تا بتواند به درستی انجام شود. چنان که گفته شد برای انجام روش تکثیر وظیفه ها باید مراحل زیر انجام گرفته باشد:

1- شناسایی ورودی‌ها و خروجی‌های محیط
2- شناسایی وظیفه‌هایی که باید به وسیله عامل‌ها انجام گیرند.
3- شناسایی عامل‌هایی که وظیفه ها را انجام خواهند داد.
4- شناسایی تعاملات بین عامل‌ها

هر وظیفه تحت شرایط خاصی انجام می شود و ممکن است به منابعی خاص نیاز داشته باشد. علاوه بر این ممکن است بخواهد که با محیط ارتباط برقرار کند.

به طور کلی اگر قرار باشد سیستم چند عامله مورد نظر، باز در نظر گرفته شود، محیط و ورودی و خروجی های آن باید در فاز تحلیل شناخته شود. بنابراین برای به کاربردن روش تکثیر وظیفه ها در هر متدولوژی سیستم های چند عامله، نخستین گام باید شناسایی محیط،ورودی ها و خروجی های آن و شرایط لازم برای شروع به کار صحیح سیستم باشد.



شکل 9. نمودار استقرار [7]

این کار باید در ابتدای فاز تحلیل متدولوژی انجام گیرد.

در مورد سیستم های چند عامله بسته، مثل تیم های شبیه سازی شده فوتبال (RoboSoccer) عامل های ما –که در اینجا بازیکنان هستند- عموماً با خارج از محیط تبادل ندارند. در واقع در محیط های چند عامله بسته شناسایی ورودی و خروجی های محیط چندان مصداق نخواهد داشت و تعاملات موجود در سیستم محدود به تعاملات بین عاملها و احیاناً تعاملات بین عاملها و خود محیط می باشد.MaSE چنان که گفته شد ، یک متدولوژی کامل از نظر چرخه حیات و مناسب برای محیط های چند عامله بسته با تعداد عامل محدود می باشد.

با توجه به این مساله، برای پشتیبانی از تحمل پذیری خطا ، ابتدای فاز تحلیل MaSE دستخوش تغییرات زیادی نخواهد شد. با این حال برای اینکه MaSE ، و یا هر متدولوژی دیگر، بتواند سیستمهای تحمل پذیر خطا ایجاد کند، لازم است که در ابتدای کار به بررسی کلیه خطاها و خرابی های ممکن، چنانکه در قسمتهای قبل ذکر شد بپردازد.

این کار در MaSE به علت بسته بودن محیط و محدود تر بودن خطاهای ممکن، دارای پیچیدگی کمتری می باشد و صرف نظر از روش به کار رفته برای تحمل پذیری خطا، لازم است که برای نیل به آن در ابتدای فاز تحلیل به انجام برسد.

پرداختن به شناسایی خطاهای ممکن در ابتدای کار و با توجه به محیط- گرچه بسیاری از خطاها را نمی توان با آن کنترل کرد- اما سبب خواهد شد که بتوان علاوه بر شناسایی و ترمیم خطا که به نوع روش بستگی دارد، به صورت کلی از پیشگیری خطا هم برای افزایش قابلیت تحمل پذیری و قابلیت اطمینان سیستم استفاده کرد. به طور کلی توانایی یک سیستم در مورد پیش گیری از وقوع خرابی های شناخته شده و حتی خطاهای ناشناخته یکی از پارامترهای مهم برای سنجش میزان تحمل پذیری خطای سیستمها و قابلیت اطمینان آنها به شمار می آید.

پس از بررسی خطاها و خرابی های ممکن و نیز تعیین ورودی ها و خروجی های احتمالی در ابتدای فاز تحلیل MaSE دومین مرحله شناسایی وظیفه هایی است که باید بوسیله عامل ها انجام گیرد.

هر وظیفه با پیش شرط ها و منابع مورد نیازش توصیف می شود. شناسایی وظیفه ها در MaSE چنان که گفته شد در مرحله Refining Roles از فاز تحلیل انجام می گیرد.

برای اعمال روش تکثیر وظیفه ها با توجه به این که هر وظیفه شرط هایی را به همراه خود دارد ، نیاز است که در همان مرحله از فاز تحلیل کار شناسایی این پیش شرط ها و همچنین تعیین منابعی که برای انجام شدن وظیفه ضروری خواهند بود، انجام گیرد.

پس به طور کلی، کار شناسایی وظیفه ها ، شناسایی پیش شرط های آن و منابع مورد نیاز باید برای پشتیبانی MaSE از این روش تحمل پذیری خطا به فاز تحلیل و خصوصاً به مرحله Refining Roles آن اختصاص یابد.

لازم به یاداوری است که در مرحله Refining Roles در MaSE با مرتبط کردن وظیفه ها به نقشها و پس از آن نقشها به هدفها در واقع تعیین می کردیم که همه هدف های مورد انتظار سیستم لحاظ شده اند و در نتیجه سیستم ، تا بدینجا برای عملکرد مورد انتظار خود ، همه ملزومات را فراهم آورده است.

پس از تعیین وظیفه ها، باید به شناسایی عاملهایی که وظیفه ها را انجام می دهند بپردازیم.

وظیفه های متفاوت شناخته شده تا بدین جا، قرار است که توسط عاملها به اجرا در آیند و هر عامل باید به منابعی که مورد نیاز وظیفه هایش است، دسترسی داشته باشد. طبق روش ارائه شده برای تحمل پذیری خطا، لازم است که عاملهای بحرانی و غیر بحرانی مشخص گردند. این کار با توجه به نقش عامل و ماهیت وظیفه های فعلی هر عامل می تواند با شناخته شدن عاملها و در فاز طراحی انجام گیرد. چنانکه پیشتر ذکر شد ، چنان چه حداقل یکی از وظیفه های یک عامل نتواند توسط عامل دیگری انجام گیرد ، عامل ، بحرانی خواهد بود . در این مورد گزیری از تکثیر کردن خود عامل وجود ندارد . اما چنان چه با توجه به نقش عامل های دیگر بتوان وظیفه های یک عامل را بر عهده آنها نیز قرار داد، این عامل غیر بحرانی خواهد بود و تکثیر وظیفه های این عامل در عامل های دیگر ، ما را از تکثیر خود عامل بی نیاز خواهد کرد .

برای مجهز شدن MaSE به تحمل پذیری خطا لازم است که کار تکثیر با شروع فاز طراحی MaSE و پیش از مرحله ایجاد مکالمات بین کلاسهای عامل انجام گیرد.

مساله ای که در این جا پیش می اید این است که ممکن است عامل های متعددی وجود داشته باشند که قادر به بر عهده گیری یک وظیفه یک عامل باشند . این که کدام عامل به عنوان همکار عامل مورد نظر انتخاب شود (یعنی یک وظیفه خاص را بر عهده بگیرد) خود مساله ای است که می تواند به طور مفصل به ان پرداخته شود. هر انتخابی که انجام گیرد، لازم است که برای بهینه کردن پارامتری در سیستم موثر باشد.

اما به عنوان یک راه حل مناسب می توان پیشنهاد کرد که به منظور ایجاد تعادل بین عامل ها و برقراری توازن بین وظیفه های آنها، عامل ای را انتخاب کنیم که خود، وظیفه های کمتری داشته باشند . این روال هر بار می تواند برای انتخاب یک عامل از بین عامل های کاندید برای پذیرش یک وظیفه تکرار گرددو سبب می شود که وظیفه ها تا حد امکان بین عامل ها توزیع شوند.توزیع وظیفه ها همچنین سبب می شود که طراحی سیستم وابسته به عنصری خاص نباشد.خود این امر به قابلیت اطمینان سیستم خواهد افزود.

مکانیزم انتخاب یک عامل به عنوان عامل جایگزین کاملا به طراح سیستم بستگی دارد که این مساله از یک سیستم به سیستم دیگر و از یک طراح به طراح دیگر متفاوت خواهد بود و نباید به عنوان جزیی از متدولوژی بر طراح تحمیل شود.

مساله دیگری که لازم به ذکر است این است که توصیه می شود که به منظور تجزیه مناسب نقش ها به طور کلی تا حد امکان از به اشتراک گذاردن یا تکثیر کردن وظیفه ها خودداری کنیم . در حالی که این مساله در جای خود کاملا درست است ، اما ما ناچاریم که برای پشتیبانی از تحمل پذیری خطا به تکثیر وظیفه ها روی اوریم .در واقع مصالحه ای بین قابلیت تحمل پذیری خطا و تجزیه پذیری نقش ها وجود دارد و به طور کلی این که کدام یک از این نیازمندیها مهمتر است بیشتر به نوع سیستم بستگی خواهد داشت .

پس از تعیین بحرانی بودن یا نبودن عامل ها و نیز تعیین نوع تکثیر برای عامل‌ها (تکثیر عامل یا وظیفه) لازم است که به شناسایی مکالمات بین عامل‌ها بپردازیم.

مرحله ایجاد مکالمات در فاز طراحی MaSE در کنار مرحله بعدی یعنی سوار کردن کلاسهای عامل به طور کامل به ایجاد و سازماندهی مکالمات و تعاملات بین عامل ها و نیز بر پا کردن کامل کلاسهای عامل خواهد پرداخت .

در همین مرحله است که باید پروتکلهای مورد نیاز برای تحمل پذیری خطا و بخصوص ترمیم خطا چنان که در قسمتهای قبل ذکر شد ، ایجاد شده و به عامل های مربوط ، مرتبط شوند .

با توجه به این که هر عاملی مستعد خطاست، چنان چه خطا واقعا بروز کند ، عاملی که در حالت خطا قرار گرفته ، می تواند با توجه به همین پروتکلهای ایجاد شده در این مرحله به شناسایی خطا و بر عهده گرفتن وظیفه هایی که در خطا قرار گرفته اند بپردازند. به طور کلی می توان گفت که توالی پیامهای مبادله شده و مکالمات بین عامل ها با بروز خطا احتمالا تغییر خواهد کرد و با بروز خطا سیستم چند عامله با توجه به عامل ها و وظیفه ها و مکالمات بین آنها مجددا پیکره بندی خواهد شد.

بدین ترتیب تکنیک بکار گرفته شده برای شناسایی و ترمیم خطا در پروتکل‌هایی تعبیه می شود که در مرحله سازماندهی مکالمات بین عامل ها تعیین می گردند. با انجام این کار آنچه برای اعمال روش تکثیر در متدولوژی لازم بوده است تکمیل شده است.

با توجه به مطالب گفته شده در این قسمت تکنیک سازمان دهی مجدد در قسمت‌های قبل به طور کامل با متدولوژی MaSE منطبق می شود .

بدین ترتیب و با توجه به کامل بودن متدولوژی MaSE از نظر چرخه حیات بخصوص با وجود ابزار agentTool، که خود را به عنوان ابزاری مناسب در طراحی و پیاده سازی سیستم های چند عامله به طراحان سیستم های چند عامله قبولانده است، می توان تحمل پذیری خطا را بیش از پیش و به شکلی عملی تر و کارا تر برای سیستمهای چند عامله به کار برد .

آنچه می ماند، بهینه کردن روش اعمال شده برای تحمل پذیری خطا در متدولوژی به گونه ایست که بتوان بدان وسیله تحمل پذیری خطا را به نحوی برای سیستمهای خاص سفارشی کرد .به عنوان مثال گاهی لازم است که شناسایی خطا به صورت کاملا بلادرنگ انجام گیرد و یا حداکثر زمان مورد نیاز برای ترمیم خطا لازم باشد که از مقداری خاص تجاوز نکند. این مسایل می توانند مبنای تحقیقات و کارهای آینده قرار گیرند.

6- جمع‌بندی و کارهای آینده

مرحله ایجاد مکالمات در فاز طراحی MaSE در کنار مرحله بعدی یعنی سوار کردن کلاسهای عامل به طور کامل به ایجاد و سازماندهی مکالمات و تعاملات بین عامل ها و نیز بر پا کردن کامل کلاسهای عامل خواهد پرداخت .

در همین مرحله است که باید پروتکلهای مورد نیاز برای تحمل پذیری خطا و بخصوص ترمیم خطا چنان که در قسمتهای قبل ذکر شد ، ایجاد شده و به عامل های مربوط ، مرتبط شوند .

با توجه به این که هر عاملی مستعد خطاست، چنان چه خطا واقعا بروز کند ، عاملی که در حالت خطا قرار گرفته ، می تواند با توجه به همین پروتکلهای ایجاد شده در این مرحله به شناسایی خطا و بر عهده گرفتن وظیفه هایی که در خطا قرار گرفته اند بپردازند. به طور کلی می توان گفت که توالی پیامهای مبادله شده و مکالمات بین عامل ها با بروز خطا احتمالا تغییر خواهد کرد و با بروز خطا سیستم چند عامله با توجه به عامل ها و وظیفه ها و مکالمات بین آنها مجددا پیکره بندی خواهد شد.

بدین ترتیب تکنیک بکار گرفته شده برای شناسایی و ترمیم خطا در پروتکل‌هایی تعبیه می شود که در مرحله سازماندهی مکالمات بین عامل ها تعیین می گردند. با انجام این کار آنچه برای اعمال روش تکثیر در متدولوژی لازم بوده است تکمیل شده است.

با توجه به مطالب گفته شده در این قسمت تکنیک سازمان دهی مجدد در قسمت‌های قبل به طور کامل با متدولوژی MaSE منطبق می شود .

بدین ترتیب و با توجه به کامل بودن متدولوژی MaSE از نظر چرخه حیات بخصوص با وجود ابزار agentTool، که خود را به عنوان ابزاری مناسب در طراحی و پیاده سازی سیستم های چند عامله به طراحان سیستم های چند عامله قبولانده است، می توان تحمل پذیری خطا را بیش از پیش و به شکلی عملی تر و کارا تر برای سیستمهای چند عامله به کار برد .

آنچه می ماند، بهینه کردن روش اعمال شده برای تحمل پذیری خطا در متدولوژی به گونه ایست که بتوان بدان وسیله تحمل پذیری خطا را به نحوی برای سیستمهای خاص سفارشی کرد .به عنوان مثال گاهی لازم است که شناسایی خطا به صورت کاملا بلادرنگ انجام گیرد و یا حداکثر زمان مورد نیاز برای ترمیم خطا لازم باشد که از مقداری خاص تجاوز نکند. این مسایل می توانند مبنای تحقیقات و کارهای آینده قرار گیرند.

7- مراجع

[1] S. Paydar, M. H. Davarpour, Sensor Networks: Introduction, usags, of intelligent methods, safety and fault tolerance techniques, B.Sc. Thesis, Computer Engineering Department, Ferdowsi University of Mashhad, 2005. (In Persian)

[2] Marin, Olivier.Sens Pierre," Towards Adaptive Fault Tolerance For Distributed Multi-Agent Systems" 2003.

[3] S. Kumar, Philip .R Cohen, Towards a Fault-Tolerant Multi-Agent System Architecture, *Proceedings of the fourth international conference on Autonomous agents*, pp. 459-466, Barcelona, Spain, 2000

[4] Z. Guessoum, J. Briot, P. Sens, O. Marin, Toward Fault-Tolerant Multi-agent Systems, *Proceedings of ERSADS*, pp. 195-201, 2001.

[5] S. Mellouli, FATMAS: A Methodology to Design Fault-tolerant Multi-agent Systems, PhD thesis, Université Laval 2005.

[6] Z. Guessoum, J.-P. Briot, S. Charpentier, A Fault-Tolerant Multi-Agent Framework, P*roceedings of the first international joint conference on Autonomous agents and multiagent systems*, pp. 672-673, 2002.

[7] S. A. DeLoach, Analysis and Design using MaSE and gentTool, *presented at 12th Midwest Artificial Intelligence and Cognitive Science Conference*, 2001.

[8] M. F. Wood, S. A. DeLoach, An Overview of the Multiagent Systems Engineering Methodology, *Lecture Notes in Computer Science*, voluom 1957, pp 207-221, 2000

[9] KH. H. Dam, M. Winikoff, Comparing Agent Oriented Methodologies, *Agent-Oriented Information Systems, 5th International Bi-Conference Workshop*, Melborn, Australia, 2003.

[10] M. Kumar, Contrast and comparison of five major Agent Oriented Software Engineering (AOSE) methodologies, 2002; http://students.jmc.ksu.edu/grad/madhukar/www/professional /aosepaper.pdf

[11] V. Lesser, C. Ortiz, M. Tambe, Distributed Sensor Networks: A Multiagent Perspective, First Edition, Springer Publication, 2003.
[12] S. Pleisch, A. Schiper, Modeling Fault-Tolerant Mobile Agent Execution as a Sequence of Agreement Problems, *19th IEEE Symposium on Reliable Distributed Sustems*, pp. 11-20, 2000.
[13] O. Marin, The DARX Framework:Adapting Fault Tolerance For Agent Systems, PhD thesis, Université duHavre, France, 2003.
[14] M. Shivakant, Agent fault tolerance using group communication, *In the Proceedings of the 2001 International Conference on Parallel and istributed Processing Techniques and Applications (PDPTA 2001)*, Las Vegas, USA, 2001.
[15] A. Fedoruk, R. Deters, Improving Fault Tolerance by Replicating Agents, In Proceedings of The 1st International Joint Conference on Autonomous Agents & Multiagent Systems (AAMAS 2002), July 15-19, Bologna, Italy,2002.
[16] agentTool 1.8.3 User's manual, Graduate School of Engineering and Management, Air Force Institute of Technology, Ohio, 2001.