بسمه تعالی

درس مهندسی نرم‌افزار پیشرفته

فصل پانزدهم

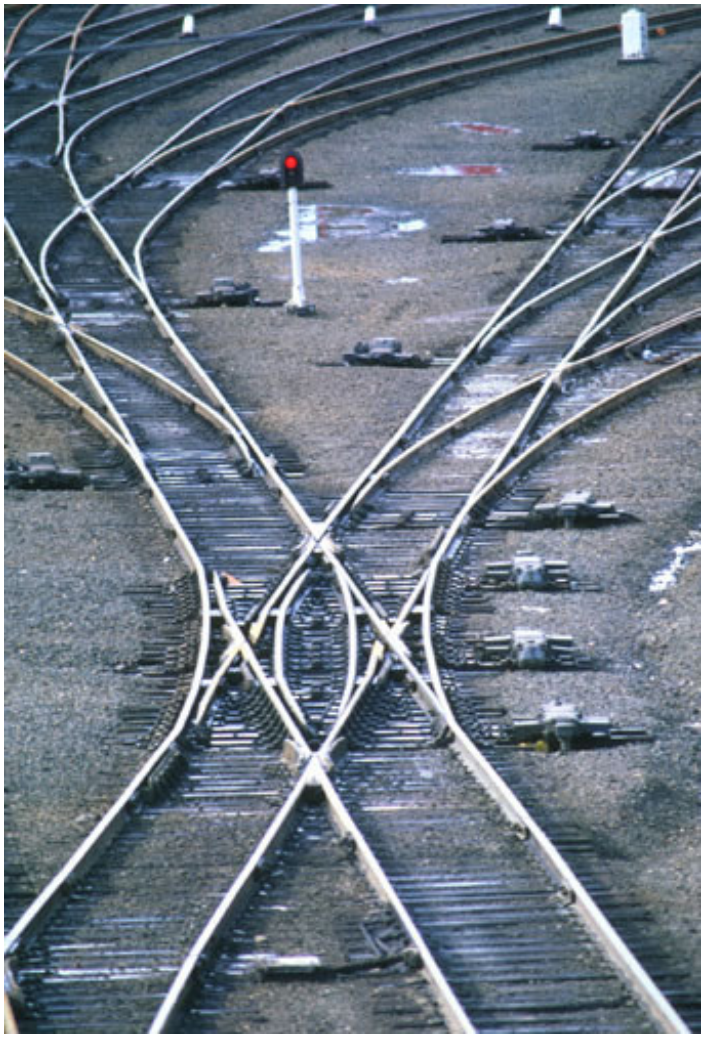
توسعه برپایهٔ عامل (*Agent*)

دکتر فریدون شمس

اهداف جلسه

- پیچیدگی نرم‌افزار و مفهوم عامل
- خصوصیات عامل
- آشنایی با سیستم‌های چندعامله
- آشنایی با انواع متدولوژی‌های عامل‌گرا (*Agent-oriented*)

پیچیدگی نرم‌افزار



- پیچیدگی به‌عنوان یکی از خصوصیات ذاتی نرم‌افزار مطرح است
- متدولوژی‌های بسیاری سعی بـه حـل پیچیدگی نموده‌اند
- روش‌های حل پیچیدگی
  - تجزیه (*Decomposition*)
  - تجرید (*Abstraction*)
  - سازماندهی (*Organisation*)

پیچیدگی نرم‌افزار (ادامه)

- در محیط‌های مدرن، پیچیدگی به شکل‌های دیگر ظهور نمـوده است
  - هوشمندی (*Intelligent*)
  - تعامل‌پذیری (*Interoperability*)
  - سازگاری (*Adaptive*)
  - ناهمگنی سکوها (*Heterogeneous*)
  - محیط توزیع‌شده (*Distributed*)

نرم‌افزارهای مدرن

- محیط‌های مدرن نیاز به نرم‌افزارهایی دارند که
  - نیاز به کاربر نداشته باشند و مستقل از او واکنش دهند
  - بتوانند به سود ما تصمیم‌گیری نمایند
  - با دیگر سیستم‌ها براحتی تعامل برقرار کند
  - در محیط‌های شبکه‌ای مختلف قابلیت اجرا داشته باشند
  - با شرایط جدید محیط سازگار شوند

هر یک از این ویژگی‌ها سبب توسعه علوم و دانش قبلی کامپیوتر شده‌اند، اما برآورده نمودن همه، سبب ایجاد زمینه جدیدی به‌نام «**سیستم‌های چند عامله**» شده است

عامل (*Agent*)

- موجودیتی نرم‌افزاری (یا سیستم کامپیوتری) است که مناسب محیط خاصی طراحی شده و قادر به انجام اعمال انعطاف‌پذیر و مستقل برای رسیدن به اهداف درنظر گرفته شده برای آن سیستم می‌باشد

عامل (ادامه)

- عامل‌ها، موجودیت‌هایی قابل تشخیص برای حل مسئله با محدوده و رابط خوش‌تعریف هستند
- برای محیط‌های خاصی مناسب هستند
- بسته به حالت‌شان ورودی‌هایی را از طریق سنسورها دریافت می‌کنند و را از طریق مجریان (*effectors*) می‌گذارند
- برای برآورده نمودن اهداف خاصی طراحی شده‌اند

خصوصیات عامل

- خودگردانی (*Autonomous*)
  - اعمال اصلی بدون دخالت انسان یا عامل دیگری انجام می‌شود
  - ارتباط میان عامل‌ها با دیگر اجزا به صورت سلسله مراتبی نیست
- پیش‌فعال بودن (*Pro-activeness*)
  - یک عامل نباید حتماً منتظر فراخوانی باشد بلکه باید تحت شرایط خاص محیط فعال شود
- واکنش‌دهی (*Reactivity*)
  - عامل باید در برخی سیستم‌ها محیط خود را حس کرده و به تغییراتی که در آن رخ می‌دهد پاسخ دهد

خصوصیات عامل (ادامه)

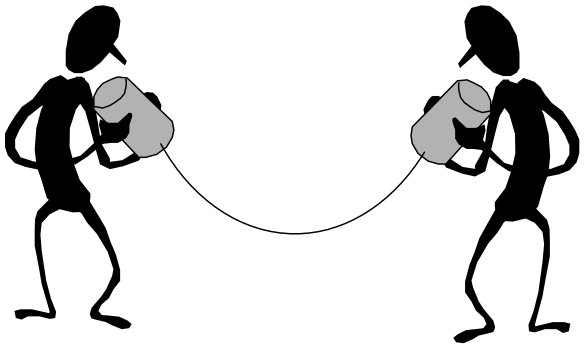
- ارتباط با دیگران (*Communicative*)
  - ممکن است یک عامل نرم‌افزاری عامل دیگری را فراخوانی کند، درباره آن استنتاج انجام دهد و یا با آن به مذاکره بپردازد
- یادگیری (*Learning*)
  - براساس تغییراتی که در محیط رخ می‌دهد رفتار خود را تغییر می‌دهد
- انعطاف‌پذیری (*Flexibility*)
  - رفتارهایی که عامل انجام می‌دهد، از پیش تعیین‌شده نیستند

خصوصیات عامل (ادامه)

- هدف‌گرا (*Goal-oriented*)
  - تنها به محیط خود پاسخ نمی‌گوید بلکه بدنبال دستیابی به اهداف خود است
- متحرک (*Mobile*)
  - توانایی جابه‌جا کردن خود از سیستمی به سیستم دیگر را دارد
- شخصیت (*Character*)
  - دارای حالت بوده و شخصیتی مستقل است

شباهت عامل با شی

- هر دو موجودیت‌هایی را تعریف می‌کنند
- عامل‌ها را اشیا فعال می‌نامند
- هر دو ساختار داخلی‌شان را بسته‌بندی می‌کنند
- به وسیله ارسال پیام با یکدیگر ارتباط برقرار می‌کنند



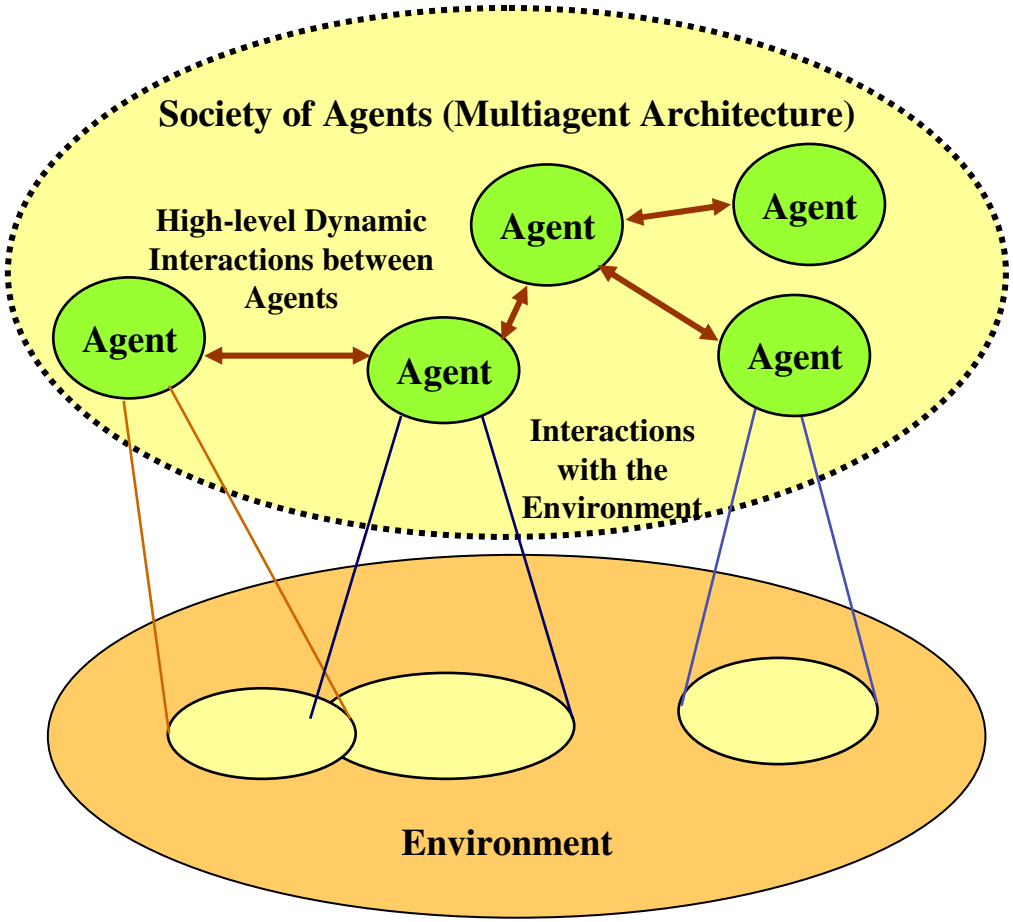
تفاوت عامل با شی

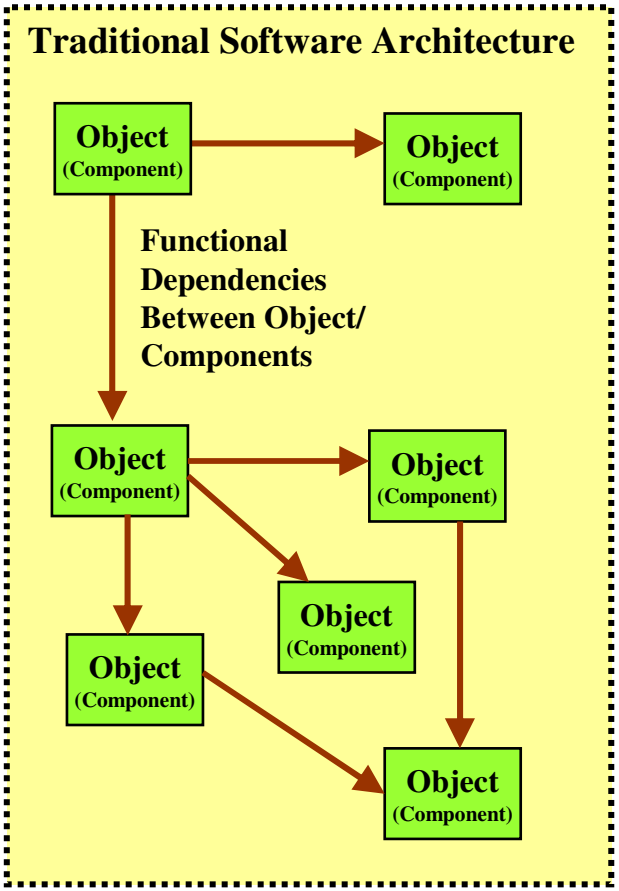
- عامل‌ها **پیش‌فعال** هستند
  - اشیا قبل از آنکه فعال شوند باید درخواست دریافت کنند
- عامل‌ها از نظر **ساختار درونی انعطاف‌پذیرند**
  - اشیا در واقع متدها و خواص را بسته‌بندی می‌کنند
  - اگر یک تابع به صورت خصوصی در شی تعریف شده باشد اشیا دیگر امکان دسترسی به آن را ندارند
- هر عامل یک **کنترل مخصوص به خود** دارد
  - کنترل در سیستم‌های شی‌گرا به صورت مرکزی انجام می‌شود

تفاوت عامل با شی (ادامه)

- هر عامل می‌تواند در هر زمان که تشخیص بدهد و با توجه به شرایط با عامل دیگر ارتباط برقرار کند
  - در روش شی‌گرا ارتباط‌ها به صورت مقطعی و از پیش تعیین شده در سیستم وجود دارند که به وسیله کلاس‌های سلسله مراتبی تعریف می‌شوند

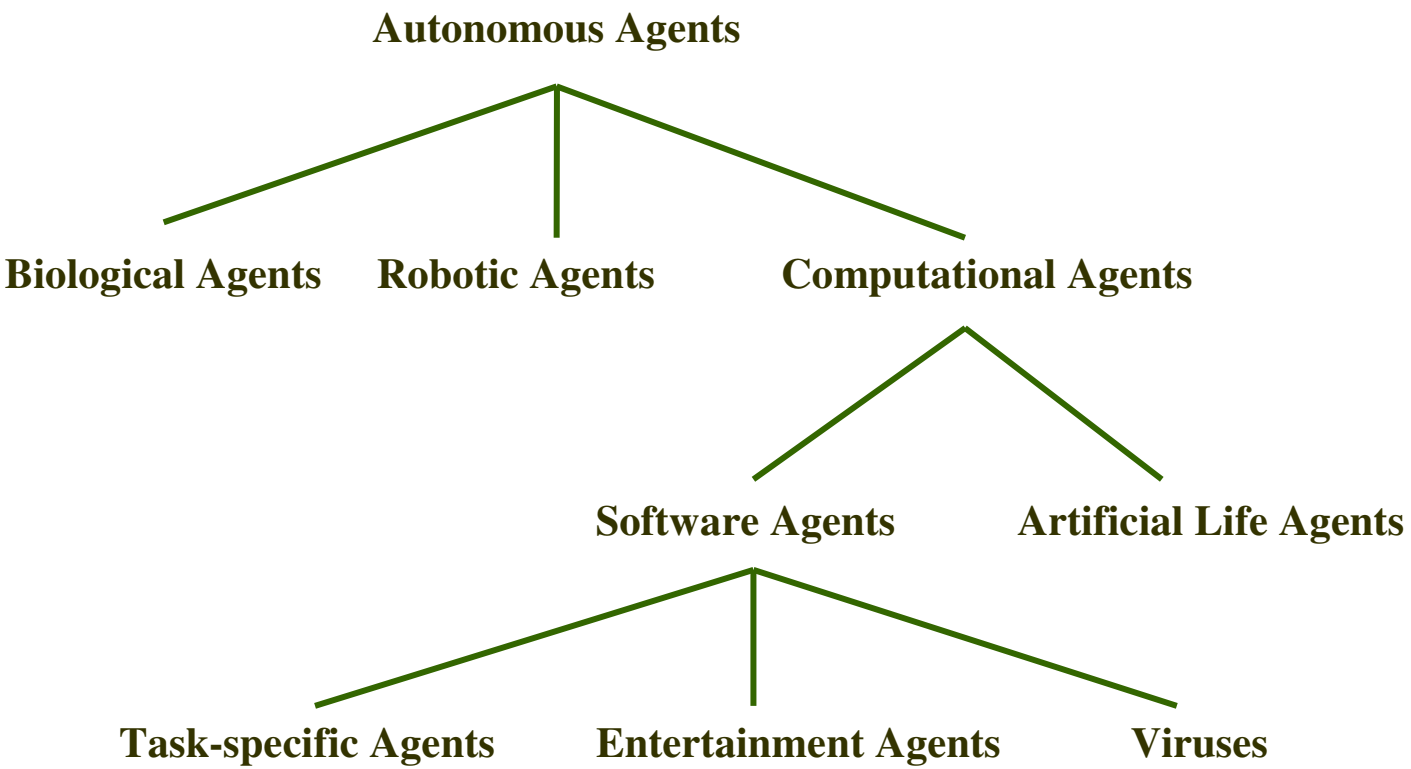
مقایسه عامل با شی (ادامه)





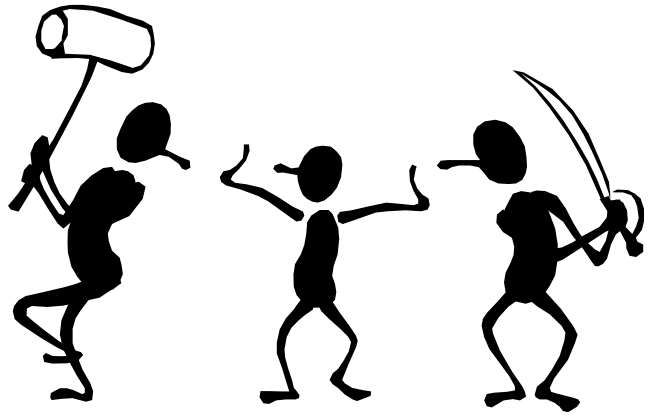
طبقه‌بندی عامل‌ها

- عامل‌ها از نقطه نظرات متفاوت، طبقه‌بندی می‌شوند
  - محیط، نوع عامل، هوشمندی، نوع کار و ...



سیستم‌های چندعامله

- سیستمی نرم‌افزاری که از مجموعه‌ای از عامل‌ها که با یکدیگر تعامل دارند، رقابت می‌کنند و همکاری دارند، تشکیل شده است
  - عامل‌ها می‌توانند دارای اهداف متفاوتی باشند
  - عامل‌ها همانند انسان‌ها با دیگر عامل‌ها تعامل دارند



مهندسی نرم‌افزار عامل‌گرا

- برای استفاده از عامل‌ها نمی‌توان از روش‌های متداول تحلیـل و طراحی استفاده نمود
- روش‌های متداول نمی‌توانند به سوالات ذیل پاسخ گویند:
  - چگونگی تجزیه مسئله
  - چگونگی انتساب وظایف به عامل‌ها
  - چگونگی هماهنگی، کنترل و ارتباط عامل‌ها
  - چگونگی حل تعارض بین اهداف عامل‌ها
  - چگونگی تصمیم‌گیری عامل در مورد سایر عامل‌ها و حالتشان

مهندسی نرم‌افزار عامل‌گرا (ادامه)

- متدولوژی عامل‌گرا
  - متدولوژی که برای توسعه یک سیستم برپایه عامل مورد استفاده قرار می‌گیرد
- متدولوژی عامل‌گرا علاوه بر پاسخ به سوالات قبلی، باید روشی شفاف و منظم برای تحلیل، طراحی و توسعه سیستم‌های چندعامله ارائه دهند

انواع متدولوژی‌های عامل‌گرا

- متدولوژی‌های عامل‌گرا (*agent-oriented*)
- متدولوژی‌هایی که از روش‌های شی‌گرا ناشی شده‌اند
  - AUML، GAIA، ADEPT، Prometheus و ...
- متدولوژی‌هایی که از مهندسی دانش ناشی شده‌اند
  - DESIRE، MAS-CommonKADS، Tropos و ...

متدولوژی‌های عامل‌گرا

- **مزایایی** متدولوژی‌هایی که از روش‌های شی‌گرا ناشی شده‌اند
  - برخی عامل را شی فعال می‌نامند و ادعا می‌کنند که روش‌های شی‌گرایی به‌تنهایی برای توسعه عامل‌ها کافی هستند
  - روش‌های شی‌گرا متداول و شناخته شده هستند
  - یادگیری این متدلوژی‌ها توسط مهندسان نرم‌افزار سریعتر انجام شده و هزینه کلی تولید سیستم کاهش می‌یابد
  - امکان استفاده از برخی مدل‌های موجود در روش‌های شی‌گرا مانند موارد کاربری در عامل‌ها وجود دارد

متدولوژی‌های عامل‌گرا (ادامه)

- معایب متدولوژی‌هایی که از روش‌های شی‌گرا ناشی شده‌اند
  - عامل‌ها تجرید بالاتری نسبت به شی دارند، بنابراین نحوه شکستن مسئله در روش‌های مبتنی بر عامل نسبت به شی‌گرا متفاوت می‌باشد
  - در روش‌های شی‌گرا تکنیکی برای مدل کردن حالت‌هایی مانند هدف‌گرایی و پیش‌فعال بودن وجود ندارد
  - مدل کردن نحوهٔ ارتباط میان عامل‌ها با استفاده از متدلوژی‌های شی‌گرا امکان‌پذیر نیست

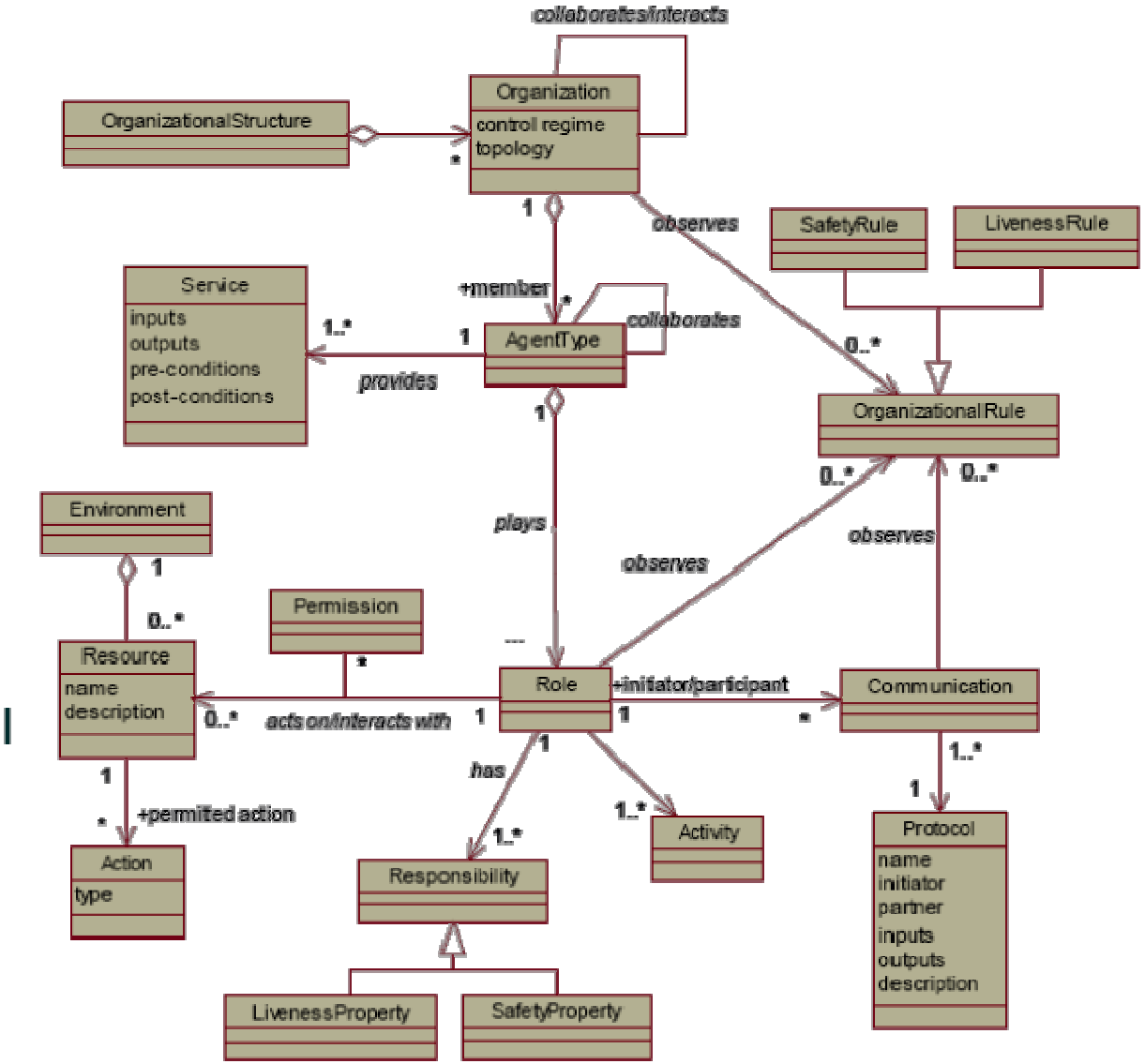
متدولوژی‌های عامل‌گرا (ادامه)

- متدولوژی‌هایی که از مهندسی دانش ناشی شده‌اند
  - فرآیند استخراج (*Eliciting*)، ساختاربندی (*Structuring*)، شکل‌دهی (*Formalizing*) و عملیاتی نمودن (*Operationalizing*) دانش
- مزایا
  - امکان استفاده از ابزارها و کتابخانه‌های هستان‌شناسی و روش‌های با قابلیت استفاده مجدد در حل مسائل
- معایب
  - ویژگیهای توزیعی عامل در این روش‌ها دیده نمی‌شود

متدولوژی توسعه GAIA

- معروفترین متدولوژی توسعه عامل
  - اولین بار توسط *Jennings* و *Wooldridge* در سال ۱۹۹۹
  - نسخه نهایی توسط *Zambonelli*، *Jennings* و *Wooldridge* در سال ۲۰۰۳ ارائه شد
- اهداف
  - شروع از نیازمندی‌ها (بعد از تشخیص نیازمندی‌ها)
  - هدایت توسعه‌دهنده برای طراحی یک سیستم چندعامله خوش-تعریف
  - توانایی مدلسازی و حل پیچیدگی سیستم‌های چندعامله را دارد

متدولوژی توسعه GAIA – متامدل



متدولوژی توسعه GAIA – مراحل

- فاز تحلیل
  - سازماندهی (*Sub-organisation*)
    - تعیین سازماندهی‌های چندگانه در سیستم (در صورت وجود)
    - آیا می‌توان سیستم را به چند مجموعه از مسائل با اتصال سست تقسیم نمود؟
  - مدل محیط (*Environmental Model*)
    - تحلیل محیط عملیاتی
    - چگونه می‌توان محیط را با استفاده از عامل‌ها مدل نمود
    - منابعی که مورد استفاده قرار می‌گیرند و چگونگی استفاده از آنها

متدولوژی توسعه GAIA – مراحل

- فاز تحلیل (ادامه)
  - مدل نقش‌های اولیه (*Preliminary Role Model*)
    - یافتن نقش‌هایی که در سازمان‌ها (*organisations*) وجود دارند
    - هر نقش، مجموعه‌ای از وظایف و رفتارهای مورد انتظار را در سازمان تعریف می‌نماید
  - مدل تعامل اولیه (*Preliminary Interaction Model*)
    - چگونگی تعامل نقش‌ها با یکدیگر برای رسیدن به هدف
    - تعریف پروتکلی برای هر تعامل بین نقش‌ها

متدولوژی توسعه GAIA – مراحل

- فاز تحلیل (ادامه)
  - قواعد سازمانی (*Organisational Rules*)
    - تحلیل و یافتن قواعدی که در کل سیستم وجود دارند و بر رفتار و تعامل بین نقش‌ها تاثیر می‌گذارند (این قواعد به قواعد اجتماعی یا قانون معروفند)

متدولوژی توسعه GAIA – مراحل

- فاز تحلیل (ادامه)
  - قواعد سازمانی (*Organisational Rules*)
    - تحلیل و یافتن قواعدی که در کل سیستم وجود دارند و بر رفتار و تعامل بین نقش‌ها تاثیر می‌گذارند (این قواعد به قواعد اجتماعی یا قانون (*law*) معروفند)

در انتهای فاز تحلیل شناخت کاملی از سازماندهی مسئله، ساختار سازمان‌ها و ارتباطات حاصل می‌شود

متدولوژی توسعه GAIA – مراحل

- فاز طراحی معماری

هدف از این فاز تعیین معماری نهایی سیستم است

- انتخاب ساختار سازماندهی
  - نکاتی چون سادگی، واقعی بودن، پیچیدگی مسئله و ... در ایـن انتخـاب نقـش دارند
- تکمیل مدل نقش‌ها
- تکمیل مدل تعامل
- انتقال نقش‌ها و پروتکل‌های تعامل به مولفه‌هایی که قابل پیاده‌سـازی باشند

متدولوژی توسعه GAIA – مراحل

- فاز طراحی معماری (ادامه)
  - تبدیل نقش‌ها به عامل‌ها
    - در برخی موارد، چند نقش در یک عامل قرار می‌گیرند
    - ابتدا «کلاس عامل» طراحی می‌شود
    - هر عامل نیاز به دانش داخلی، زمینه، فعالیت‌های داخلی و فراهم نمودن سرویس‌هایی برای دیگران دارد
  - پروتکل‌های تعامل تبدیل به دنباله‌ای از پیام‌ها می‌شوند
    - پیام‌ها بین دو عامل خاص رد و بدل می‌شوند

متدولوژی توسعه GAIA – ضعف‌ها

- مسائل و مشکلات پیاده‌سازی در نظر گرفته نمی‌شوند
- تعیین نیازمندی‌ها و مدل‌سازی آنها مسکوت مانده است
- توسعه با این روش، ترتیبی است

متدولوژی توسعه Troops

- این متدولوژی اساساً عامل‌گرا طراحی شده است
- نمادگذاری (*notions*) عامل‌ها، اهداف و مسائلی از این قبیل در فرآیند توسعه دیده شده است
- با تعداد کمی از عناصر شروع می‌کند و تدریجاً آنها را افزایش می‌دهد
  - جزئیات تدریجاً افزوده می‌شوند
  - تعامل و وابستگی‌ها بررسی و بروز می‌شوند
  - هر گام می‌تواند منجر به معرفی/حذف عناصر یا ارتباطات شود

متدولوژی توسعه Troops – مراحل

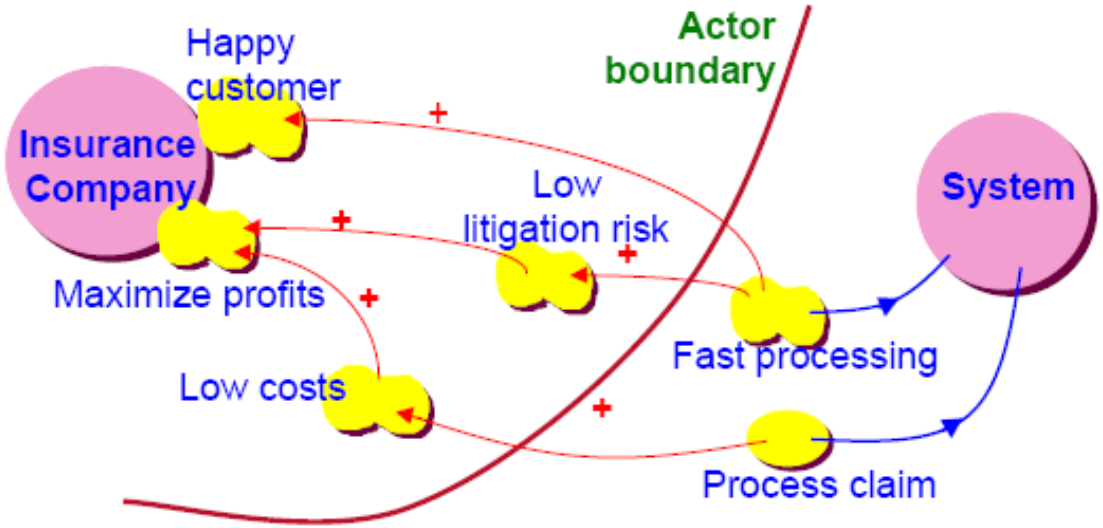
- تحلیل نیازمندی‌های اولیه (*Early Requirements Analysis*)
  - تمرکز روی منظور ذینفعان
  - با استفاده از تحلیل هدف‌گرا (*Goal-oriented*)، اهداف اولیه به مجموعه‌ای از نیازهای وظیفه‌مندی و غیروظیفه‌مندی تبدیل می‌شوند
  - ذینفعان به‌عنوان مجموعه‌ای از عامل‌ها (*Actors*) نشان داده می‌شوند که برای دستیابی به اهداف با یکدیگر وابسته شده‌اند. این امر منجر به تولید *Actor diagram* و *Rationale diagram* می‌شود

متدولوژی توسعه Troops – مراحل

- تحلیل نیازمندی‌های اولیه (ادامه)
  - *Actor Diagram*
    - یک گراف از عامل‌هاست که وابستگی استراتژیک عامل‌ها را نشان می‌دهد
    - وابستگی، توافق بین دو عامل است
  - *Rationale diagram*
    - نشان‌دهنده و توصیف‌کننده دلایلی است که یک عامل را به عامل دیگر مرتبط می‌سازد
    - در این نمودار دلایل تعامل عامل بیان می‌شود

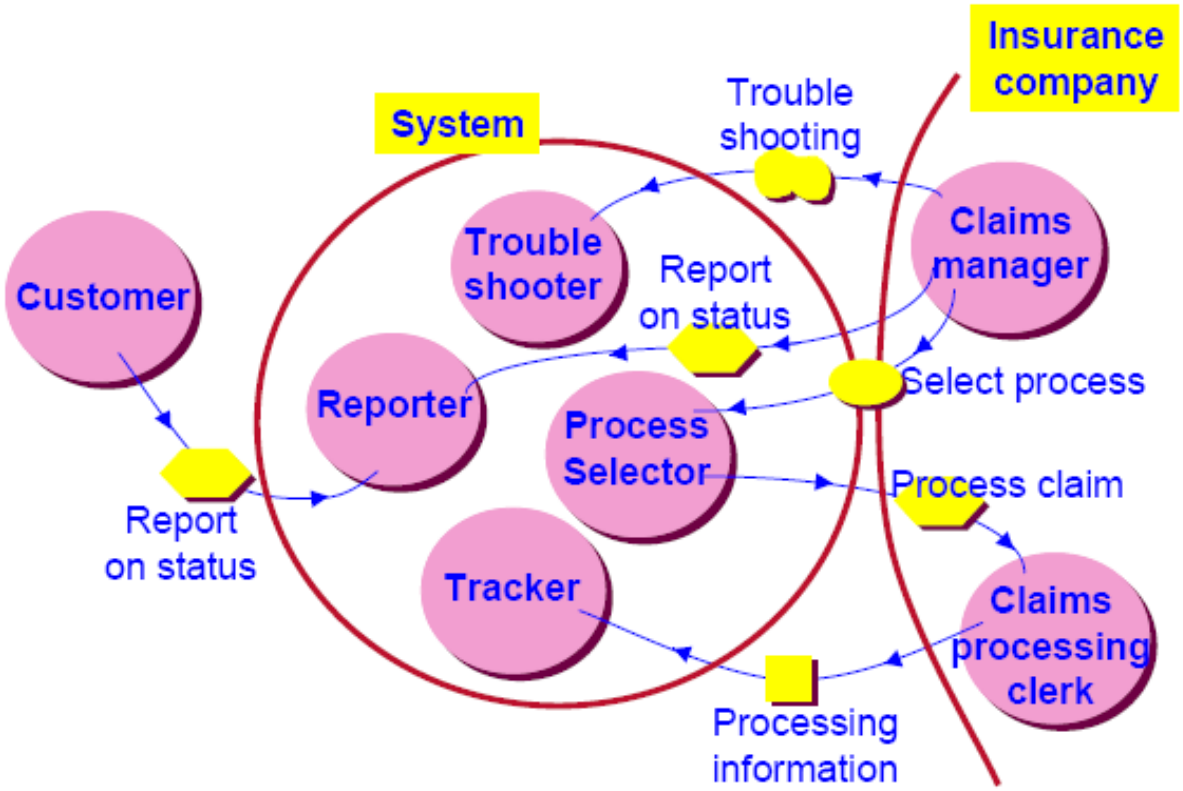
متدولوژی توسعه Troops – مراحل

- تحلیل نیازمندی‌های نهایی (*Late Requirements Analysis*)
  - مدل مفهومی «تحلیل نیازمندی‌های اولیه» توسعه داده می‌شود
    - سیستم به عنوان عامل (*Actor*) جدید توسعه داده می‌شود و ارتباطات آن با محیط در نظر گرفته می‌شود



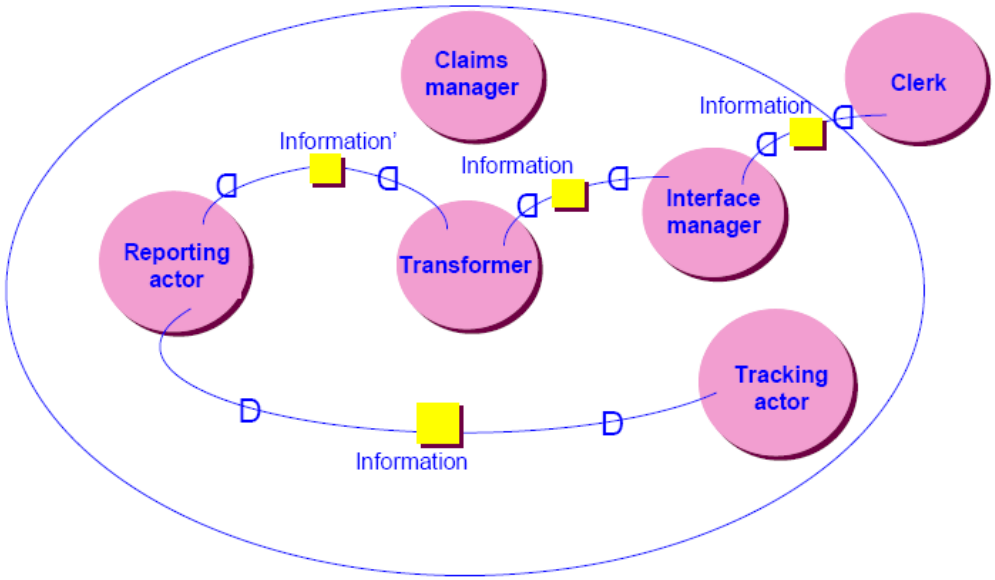
متدولوژی توسعه Troops – مراحل

- تحلیل نیازمندی‌های نهایی
  - عامل‌های درون سیستم نشان داده می‌شوند



متدولوژی توسعه Troops – مراحل

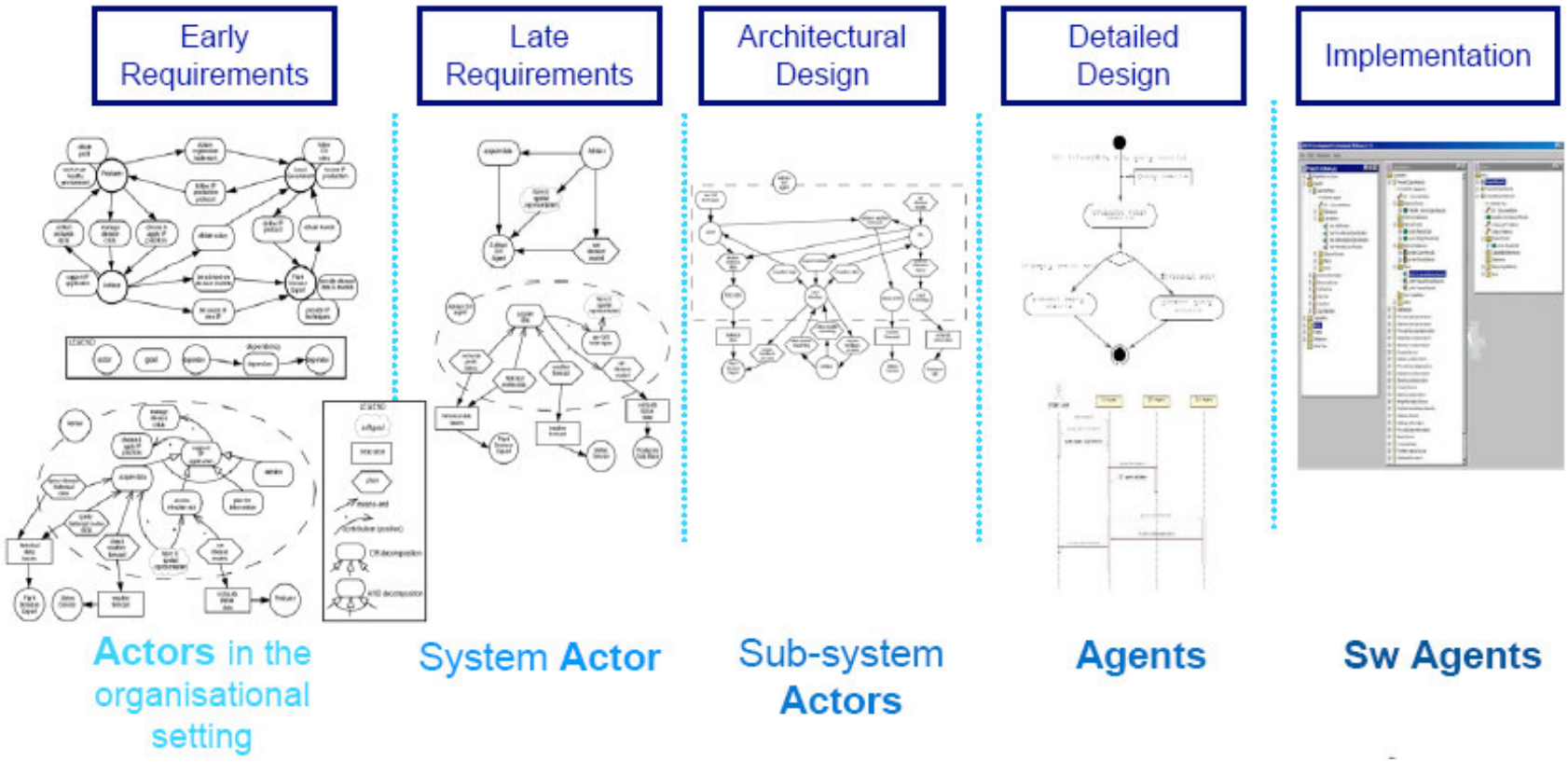
- طراحی معماری (*Architectural Design*)
  - عامل‌های مورد نیاز برای دستیابی به اهداف افزوده می‌شوند
  - افزودن عامل (*Actor*) می‌تواند براساس سبک یا الگوی معماری خاصی می‌باشد (سبک یا الگوی عامل‌گرا)



متدولوژی توسعه Troops – مراحل

- طراحی تفصیلی (*Detailed Design*)
  - جزئیات بیشتری به عامل‌ها نظیر اهدافی که برآورده می‌کننـــد، نحـــوهٔ انجام آن، اجبارهای زمانی و ... افزده می‌شوند
  - در این مرحله عامل‌ها (*Actor*) به عامل (*Agent*) تبدیل می‌شوند!
  - استفاده از الگوهای طراحی کمک شایانی به طراحی تفصیلی می‌کند

متدولوژی توسعه Troops – مراحل



متدولوژی توسعه Troops – ضعف‌ها

- برای طراحی نرم‌افزارهای عامل پیچیده که ارتباطات بین ذینفعان پیچیده است، مناسب نیست
- تضمین کیفیت تنها با استفاده از الگوهای طراحی!
- راهنمایی‌هایی بسیار کمی در مورد گام‌ها و چگونگی انجام آن می‌توان یافت نمود
- تحویل‌دادنی‌های آن خوش-تعریف نیستند