بررسی یک پروتکل تجارت الکترونیکی به کمک ابزار بازبینی[1] ربکا[2]

کامیار رفعتی ۸٤۲۰۳٦۹٤
rafati@ce.sharif.edu
دانشگاه صنعتی شریف

نعیم اصفهانی ۸٤۲۰۱۰۰۳
esfahani@ce.sharif.edu
دانشگاه صنعتی شریف

مهندسی نرم‌افزار پیشرفته
مدرس: دکتر میریان

کلمات کلیدی:
روش‌های رسمی، درستی‌یابی، ربکا، اکتور، تجارت الکترونیکی

چکیده:

با توجه به پیشرفت دنیای رایانه و اینترنت، تقریبا تمام امکانات و فعالیت‌های دنیای واقعی به گونه‌ای به این دنیا نگاشته شده‌اند؛ یکی از جنبه‌های دنیای واقعی که به دلیل درآمد فراوان بسیار مهم می‌باشد، داد و ستد است. امروزه شاهد گسترش روز افزون پروتکل‌هایی برای پشتیبانی از این امکان در دنیای الکترونیکی هستیم. در تجارت الکترونیکی علاوه بر نیازمندی‌های معمول در ارتباطات نیازهای دیگری هم مطرح است؛ به طور مثال پرداخت تنها در صورت اخذ کالای مبادله شده، از این دسته نیازهاست. برای آزمودن برآورده شدن این نیازها از ابزارهای درستی‌یابی استفاده شده است. ماهیت تجارت الکترونیکی یک ماهیت غیر همگام و مبتنی بر عامل است این در حالی است که زبان‌ها و ابزارهای درستی‌یابی قدیمی ذاتا این خاصیت را ندارند و به گونه‌ای باید با استفاده از امکانات دیگر این خاصیت را شبیه‌سازی و مدل نمود. یکی از زبان‌های مدل‌سازی مطرح شده که ماهیت مدل آن غیرهمگام و برپایه‌ی مدل شی‌گرایی است ربکا [4] نام دارد. مدل کردن عامل‌ها با استفاده خاصیت این مدل بسیار راحت‌تر از مدل‌های قدیمی‌تر است. در این مقاله با استفاده از این زبان، یک پروتکل تجارت الکترونیکی را مدل کرده و نشان داده‌ایم که این کار به مراتب راحت‌تر از مدل کردن این پروتکل با زبان‌های قدیمی است.

[1] Verification
[2] Rebeca

۱- مقدمه:

در سال‌های اخیر شاهد گرویدن روزافزون در سیستم‌های با اهمیت بالا به روش‌های رسمی هستیم. دیگر برای مدیران قابل قبول نیست که سیستمی که با پول زیاد و یا جان انسان‌ها سروکار دارد دچار شکست بشود. با آزمون این سیستم‌ها نمی‌توان مطمئن شد که سیستم کاملا درست کار می‌کند؛ آزمون تنها باعث بالا بردن اطمینان می‌شود. در زمینه‌ی درستی یابی سیستم‌های سخت افزاری به بلوغ بیشتری رسیده‌اند چون این سیستم‌ها پیچیدگی کم‌تری از سیستم‌های نرم‌افزاری دارند و قابلیت تغییر کم‌تری بعد از تولید دارند در نتیجه سادگی و نیاز دست به دست هم داده‌اند تا این حوزه پیشرفت سریع‌تری داشته باشد. پروتکل‌های تجارت الکترونیکی که با مبادله‌ی پول و کالا سروکار دارند بسیار حساس هستند و اگر در آن‌ها مشکل و خطایی پیدا شود منشا از دست رفتن سرمایه‌ی فراوان و مشکلات برای کاربران می‌شوند؛ گسترش بیش‌تر این پروتکل‌ها و اعتماد بیش‌تر به آن‌ها این مشکل را مشهودتر کرده است. برای درک آن که آیا این سیستم‌ها نیازهای عمومی سیستم‌های نرم‌افزاری و نیازهای خاص حوزه‌ی تجارت را برآورده می‌کنند از روش‌های درستی‌یابی استفاده‌ی فراوانی شده است. علاوه بر بررسی خصوصیات مشترک با سیستم‌های دیگر نرم‌افزاری از قبیل امنیت و ...، خصوصیات دیگری هم مخصوص این سیستم‌ها توسط محققان تشخیص داده شده است؛ صحت پرداخت پول و دریافت کالا از این دست هستند. در ادامه توضیح این خصوصیات آمده است و سعی در آزمودن این خصوصیات در پروتکلی است که در بخش معرفی مشکل شرح داده می‌شود. عمل اعتبارسنجی به این ترتیب است که سیستم را مدل کرده و سپس خاصیت‌های مورد نظر را نیز مدل می‌کنیم؛ سپس این دو را به یک ابزار می‌دهیم تا آن را اعتبار سنجی کند. ابزار همه‌ی حالات ممکن را برای ما می‌سازد و خاصیت گفته شده در آن‌ها می‌آزماید. در صورت درست بودن خاصیت‌های گفته شده سیستم به ما آن را اطلاع می‌دهد و در صورت وجود مشکل سیستم با ارائه‌ی مثال نقض مشکل را مشخص می‌کند.

همان‌طور که اشاره شد این زمینه با توجه به استفاده‌ی آن در سخت افزار یکی از قدیمی‌ترین زمینه‌های مهندسی نرم‌افزار است؛ بنابراین ابزارها و زبان‌هایی برای مدل‌سازی وجود دارند ولی مشکل اساسی آن‌ها منطبق نبودن بر نیازهای جدید و ساختارهای جدید است. به طور مثال ماهیت سیستم‌ها در تجارت الکترونیکی بر پایه‌ی ناهمگامی و مستقل بودن است که برای مدل کردن سیستمی با این خصوصیات باید خصوصیت را نیز مدل کرد. این کار باعث پیچیدگی مضاعف برای مدل کننده می‌شود؛ بهتر است زبان مدل‌سازی خود از این خصوصیات پشتیبانی کند تا بر سادگی مدل‌سازی بیافزاید. در این مقاله پروتکلی که قبلا توسط یک زبان مدل‌سازی بررسی شده را به زبان ربکا [4] که ماهیتا غیرهمگام و دارای قسمت‌های مستقل است مدل کرده و نشان می‌دهیم که این مدل‌سازی به مراتب راحت‌تر از مدل قبلی است. راحتی در مدل سازی فواید بسیاری دارد، از جمله این که در صورت پیدا کردن مثال نقض توسط ابزار راحت‌تر می‌توان آن را یافت و رفع کرد؛ همچنین تولید اولیه‌ی سیستم هم بسیار راحت‌تر و سریع‌تر می‌باشد.

در ادامه ابتدا در بخش بعد معرفی مختصری از زبان مدل‌سازی ربکا ارائه می‌دهیم، سپس صورت مساله را تعریف می‌کنیم، مدل سازی خود از سیستم را ارائه داده، سیستم را با وجود خطا مورد درستی‌یابی قرار می‌دهیم و مشکل به وجود آمده را پیدا می‌کنیم و در نهایت نتایج بدست آمده را ارائه می‌دهیم.

۲- معرفی زبان ربکا:

در این بخش به معرفی مدل ربکا[3] می پردازیم. ربکا به مخفف زبان مدل‌سازی اشیا واکنشی می باشد. این زبان یک زبان عامل گرا با ویژگی‌های صوری است. می‌توان آن را به عنوان یک مدل مرجع برای مدل‌سازی سیستم‌های همروند ، بر مبنای تعامل عوامل در نظر گرفت. همچنین چهارچوبی برای توسعه سیستم‌های شی‌گرای همروند در عمل در اختیار قرار می‌دهد. این زبان علاوه بر اینکه یک مدل مناسب و کارا برای مدل‌سازی سیستم‌های همروند و توزیع شده در اختیار ما قرار می‌دهد ، همچنین امکان سنجش درستی آنها را به کمک وارسی صوری فراهم می‌آورد. ربکا به کمک یک ابزار بازبینی پشتیبانی می‌شود که می‌توان با استفاده از آن زبان ربکا را به زبان سایر ابزارهای بازبینی تبدیل نمود تا آن را وارسی نمود. همچنین این امکان ، به صورت محدود ، وجود دارد که مستقیما آن را وارسی نمود. از تکنیک‌های وارسی پیمانه‌ای و تجرید در این مدل استفاده شده است تا اینکه فضای حالت کاهش یابد و امکان وارسی سیستم‌های واکنشی بزرگ فراهم آید.

مدل ربکا متشکل است از تعدادی شی واکنشی یا ربک که به صورت همروند وجود دارند و با یکدیگر در تعامل هستند. ربک‌ها اشیا کپسوله شده‌ای هستند که متغیر اشتراکی ندارند و به از طریق پیغام‌های ناهمگام با یکدیگر در ارتباطند. ربک‌ها از یک کلاس واکنشی[4] معرفی می‌شوند که یک فرآیند اجرای مستقل دارند و به وسیله خواندن یک پیغام از سر یک صف نامحدود برانگیخته می‌شود. هر پیغام مشخص کننده تابع خاصی است که هنگام سرویس دادن به یک پیغام فراخوانده می‌شود. هر گاه که یک پیغام از صف خوانده می‌شود ، تابع سرویس دهنده آن فرا خوانده می‌شود و آن پیغام از سر صف حذف می‌گردد. ربک‌ها کنترل صریح بر صف خود ندارند. همچنین به علت ماهیت ناهمگام مکانیزم ارتباطی ، که تنها فرستادن ناهمگام را در بر دارد و اینکه دریافت علنی ندارد ، توابع می‌توانند به صورت خودکار عمل نمایند. در شکل ۱ نمونه‌ای از یک کلاس ربک را می‌بینید.

```
Reactiveclass Rebec1(2)
{
        knownobjects
        {
                Rebec2 d;
        }
        statevars
        {
                Boolean flag;
        }
        msgsrv initial()
        {
                self.msg1();
        }
        msgsrv msg1()  {/*Handling messge 1*/}
        msgsrv msg2()  {/*Handling messge 2*/}
}
```

شکل ۱ : نمونه‌ای از یک کلاس ربک

[3] **Re**active O**bje**ct L**a**nguage
[4] Reactive Class

همان طور که گفته شد امکان وارسی مستقیم مدل ربکا هنوز به صورت کامل مهیا نیست لذا در این تحقیق از ابزار ترجمه به زبان پرومﻻ[5] استفاده شده است. این ابزار مدل ربکا را دریافت می‌نماید و مدل معادل را در زبان پرومﻻ تهیه می‌نماید. سپس ویژگی‌ها در زبان پرومﻻ به مدل اضافه شده و وارسی انجام شده است.

مدل پرومﻻ ، به صورت مختصر ، متشکل است از تعدادی پروسه ، تعدادی متغیر مشترک ، تعدادی کانال ارتباطی و یک موتور معنایی که مشخص می‌نماید که انجام پروسه‌ها چگونه تداخل دارند. بیان ویژگی‌ها در این مدل به کمک منطق زمانی خطی[6] امکان پذیر است. این ویژگی‌ها به صورت خودکار به زبان پرومﻻ ترجمه می‌شوند. همچنین امکان بیان جملاتی به صورت ادعا[7] نیز وجود دارد. در این تحقیق از ساختارهای ادعا برای بیان ویژگی‌های مدل که در بخش‌های بعدی آنها را معرفی می‌کنیم ، استفاده شده است.

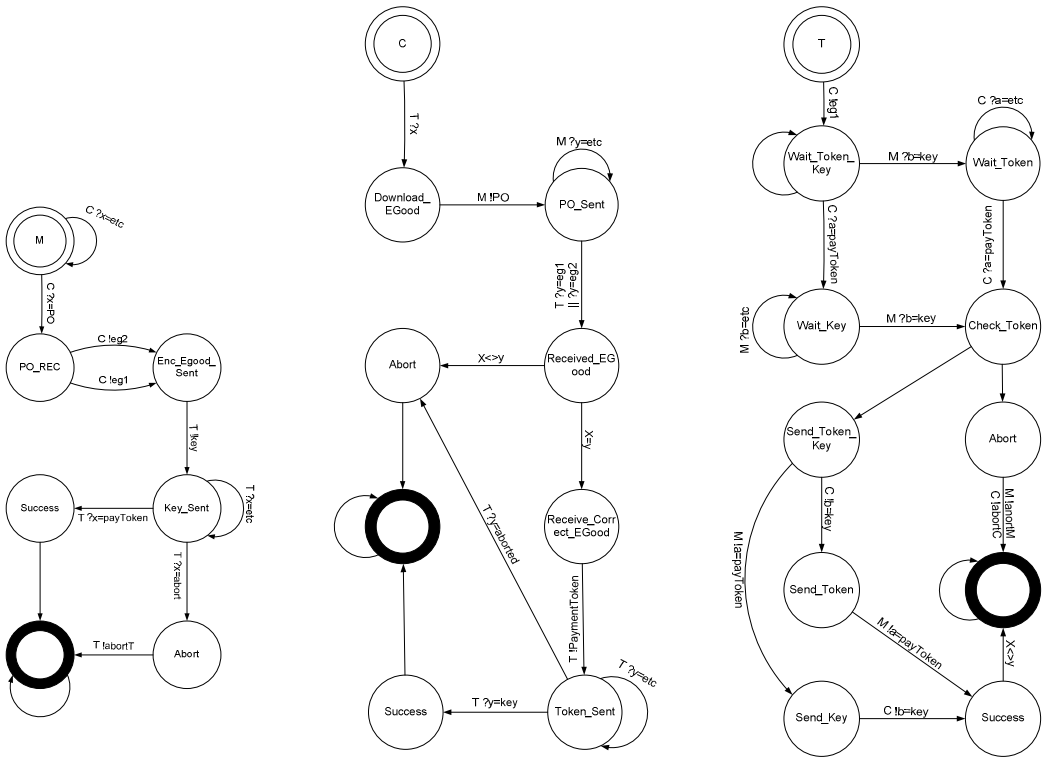
زبان پرومﻻ به کمک ابزار اسپین[8] پشتیبانی می‌شود که وظیفه وارسی مدل‌های پرومﻻ را بر عهده دارد و ما نیز از همین ابزار برای وارسی استفاده کرده‌ایم. در اینجا از بیان جزئیات این مدل صرف نظر می‌کنیم و پیشنهاد می‌کنیم که برای اطلاعات بیشتر به مراجع مراجعه شود.

۳- تعریف مساله:

پروتکل مطرح شده برای انتقال کالاهای الکترونیکی برپایه‌ی تئوری اعتبارسنجی مزدوج می‌باشد. در این پروتکل سه موجودیت حضور دارند. خریدار که اقدام به دریافت کالای رمزدار می‌کند، اگر مطابق میلش بود در ازای دریافت کلید کالا اقدام به پرداخت پول می‌کند. فروشنده به کاربر کالای رمز شده را می‌دهد و کلید مربوط به آن را به واسطه می‌دهد. و منتظر آن می‌شود که واسطه پول پرداخت شده و تایید شده را به او منتقل کند. واسطه به در خواست کاربر برای دریافت کالای رمز شده اقدام کرده، از فروشنده کلید مربوطه را گرفته، و سپس از اعتبار پولی که از مشتری می‌گیرد مطمئن می‌شود و کلید را به مشتری و پرداخت پول را به فروشنده می‌فرستد. دو نکته در این‌جا باقی می‌ماند: اولا رابطه‌ی کالاهای رمز شده که واسطه دارد این است که توسط کلید قابل باز شدن نیست و فقط قابلیت مقایسه با کالای اصلی رمز شده را دارد. این خاصیت برای این است که در عین حال که خریدار از این‌که برای همان چیزی که می‌خواهد پول می‌پردازد مطمئن می‌شود، جلوی سوء استفاده توسط واسطه گرفته می‌شود. دوما در صورت بروز هرگونه اشکال در اعتبار سنجی این نقل و انتقال لغو شده و به اطلاع همه‌ی افراد می‌رسد. شایان ذکر است که درستی این پروتکل وابسته به عدم وجود خرابی است و در صورت وجود خرابی مشکلات مشخصی وجود دارد که از موضوع این مقاله خارج است. می‌توانید در شکل ۲ نمودار حالت هر کدام از سه مولفه‌ی ارتباطی را مشاهده کنید. در این اشکال روی کانال‌های ارتباطی در ابتدا مشخص شده طرف دیگر پیام کیست و T, C , M به ترتیب نماینده‌ی فروشنده، خریدار و واسطه‌اند. در ادامه ورود اطلاعات با علامت ؟ و خروج آن با علامت ! نمایش داده شده‌اند.

---

[5] Promela
[6] Linear Temporal Logic (LTL)
[7] Assert
[8] SPIN



شکل ۲ : از چپ به راست نمودار حالت فروشنده، مشتری و واسطه

خصوصیات مطرح شده که عبارت از صحت پرداخت پول و دریافت کالا بر این تاکید می‌کنند که اولا در سیستم پول حذف و یا تولید نشود (صحت پرداخت پول) و دوما ارتباط پرداخت پول و دریافت کالا به هم وابسته باشند و یا هردو با هم اتفاق بیافتند یا هیچ کدام اتفاق نیافتند (صحت دریافت کالا).

۴- مدل‌سازی مساله:

۴،۱- مدل‌سازی اجزای مساله:

پس از تعریف سیستم اقدام بعدی مدل‌سازی آن توسط زبان مدل‌سازی می‌باشد. همان طور که گفته شده زبان مدل‌سازی انتخاب شده زبان ربکا می‌باشد. در سیستم ما هر مولفه توسط یک ربک مدل می‌شود. پیغام‌های رد و بدل شونده بین مولفه‌های مختلف که در نمودار حالت آن‌ها هم قابل مشاهده هستند را به صورت توابعی در ربک‌های متناظر شده‌اند. حالات مولفه‌ها هم توسط متغیرهای حالت ربک‌های متناظر مشخص می‌شود و می‌توان از آن‌ها استفاده کرد. برای روشن شدن نحوه‌ی تبدیل، مدل فروشنده را با توضیحات بیشتری ارائه می‌دهیم (شکل ۳). حالت فروشنده با

متغیرهای‌موفقیت، خطا و صحت کالا مشخص می‌شود که دوتای اول برای نگهداری حالت ربک و آخری برای تصمیم‌گیری در مورد عمل ربک استفاده شده‌اند. در تابع دریافت سفارش (شکل ۳) هدف دریافت سفارش از مشتری و

```
statevars
{
   boolean abort;
   boolean success;

   boolean isCorrectGood;
}
```

```
msgsrv receivePurchaseOrder()
{
   isCorrectGood =?{true, false};

   if(isCorrectGood){
        c.receiveEncryptedGood(true);
   }else{
        c.receiveEncryptedGood(false);
   }

   tp.receiveKey();
}
```

شکل ۳ : تابع دریافت سفارش سمت راست و متغیرهای حالت سمت چپ

انجام اقدامات مقتضی می‌باشد. در این تابع ابتدا به صورت نا معین[9] تصمیم گرفته می‌شود که محصول درست فرستاده شود یا نه؛ این از آن جهت است که در درستی‌یابی حالتی که فروشنده جنس اشتباه را می‌دهد نیز مدل شود. در ادامه با توجه به تصمیم گرفته شده با صدا کردن تابعی از مشتری این کالا ارسال می‌شود و کلید آن نیز به واسطه فرستاده می‌شود. برای مشاهده‌ی کد کامل ربک‌ها به بخش ضمیمه مراجعه کنید.

۴،۲- مدل‌سازی خصوصیات:

همان طور که اشاره شد برای فهمیدن این که آیا یک خصوصیت در سیستم برقرار است یا نه آن خصوصیت را نیز مدل کرده و به یک ابزار داده تا بر مدل اعمال کند. زبان ربکا هنوز امکانی برای بیان مستقیم خصوصیات در کد خود ندارد، بنابراین باید در کد تولیدی (در اینجا پرومِلا) به صورت دستی، شرایط این خصوصیات را وارد کرد. ما دو خصوصیت صحت پرداخت پول و دریافت کالا را مدل می‌کنیم.

زمان آزموده شدن وقتی است که یا سیستم به کار خود با موفقیت پایان داده باشد و یا روند کار به دلیل عدم توافق طرفین قطع شده باشد (Abort)؛ بنابراین ابتدا چک می‌کنیم که این شرایط برقرارند یا نه (آیا سیستم در حالت مورد نظر هست) و سپس اقدام به بررسی خصوصیات مورد نظر می‌کنیم. برای خاصیت صحت پرداخت پول چندین حالت داریم؛ یا پول پرداخت شده و کالا گرفته شده، یا پول پرداخت شده ولی حساب مشتری معتبر بوده و یا اصلا پولی پرداخت نشده. برای صحت دریافت کالا هم چندین حالت داریم؛ یا هردو به موفقیت رسیده اند و یا هردو صرف نظر کرده‌اند(شکل ۴). البته در سیستم حاضر به دلیل عدم خرابی و مطمئن بودن در کانال‌ها مشکلی وجود ندارد و خاصیت‌های گفته شده تامین می‌شوند.

[9] Non deterministic

```
if
::
   ((s_success_Customer[1] || s_abort_Customer[1]) &&
   (s_success_Merchant[1] || s_abort_Merchant[1]) &&
   (s_success_ThirdParty[1] || s_abort_ThirdParty[1]))
->
   assert(((s_payment_Customer[1] && s_success_Merchant[1]) ||
   (s_payment_Customer[1] && s_abort_Customer[1])) ||
   (!s_payment_Customer[1]))
fi
```

```
if
::
   ((s_success_Customer[1] || s_abort_Customer[1]) &&
   (s_success_Merchant[1] || s_abort_Merchant[1]) &&
   (s_success_ThirdParty[1] || s_abort_ThirdParty[1]))
->
   assert( (s_success_Customer[1] && s_success_Merchant[1]) ||
   (s_abort_Customer[1] && s_abort_Merchant[1]));
fi
```

شکل ۴ : بالا شرط صحت پرداخت پول، پایین شرط صحت دریافت کالا

۵- معرفی و مدل‌سازی خطا:

همان طور که گفته شد پروتکل ارائه شده [2] در غیاب خطا درست عمل می‌نماید. لذا در این قسمت با معرفی خطا در قسمت‌های مختلف پروتکل سعی می‌کنیم که خطاهای محتمل را در حضور خطا شناسایی نماییم. برای این منظور شکست‌هایی را که ممکن است در مراحل اجرای پروسه‌های مشتری ، فروشنده و واسطه رخ دهد ، یک به یک مدل نموده و پروتکل را در حضور آنها وارسی نمودیم.

نکته مهم در مورد معرفی شکست ، نحوه مدل‌سازی آن در پروسه‌ها است. برای این منظور به این صورت عمل شد که اولا یک متغیر حالت شکست به هر یک از ربک‌ها اضافه شد. سپس در نقاطی که لازم بود شکست اضافه شود به صورت نامعین بین اینکه روند اصلی طی شود و یا اینکه شکست روی دهد تصمیم گیری شده است. برای این منظور نیز مقدار دهی نامعین به متغیر حالت شکست استفاده شده است. شکل ۵ نمونه‌ای از این شکست‌ها را که در یکی از توابع مشتری اضافه شده است نشان می دهد.

```
.
.
.
statevars
{
        boolean crash;
        .
        .
msgsrv receiveKey ()
{
        crash = ?{true,false};
        if(!crash)
        {
                success = true;
                self.halt();
        }
        else
        {
                Tp.doAbort();
        }
}
```

شکل ۵ : نمونه‌ای از معرفی خطا

با توجه به نکاتی که در بالا گفته شد حال نتایج حاصل را بررسی می‌نماییم.

۵،۱- خطا در مشتری:

نتایج نشان داد که بروز خطا در مراحل مختلف کار مشتری به غیر از هنگامی که پرداخت صورت گرفته و هنوز کلید دریافت نشده است تاثیری در شرایط امن بودن پروتکل ندارد و همیشه شرایط امن بودن برقرار خواهند بود. اما در صورتی که بعد از پرداخت و قبل از دریافت کلید شکست روی دهد ، وارسی نشان می‌دهد که هر دو شرط صحت پرداخت پول و صحت دریافت کالا نقض می‌شوند. دنباله خطا دار به این صورت است که مشتری کالای کد شده را دریافت می‌کند و در خواست را به فروشنده می‌دهد. در عین حال پول را در اختیار واسطه قرار می‌دهد. فروشنده نیز کلید را در اختیار واسطه قرار می‌دهد. سپس واسطه کلید را به مشتری و پول را به فروشنده تحویل می‌دهد. در این فاصله اگر که مشتری قبل از دریافت کلید دچار شکست شود با آنکه پرداخت را انجام داده ولی کلیدی دریافت نکرده است. این موضوع باعث نقض هر دو شرط امنیت می‌شود. این نشان می‌دهد که در این پروتکل به منظور حفظ شرایط ، مشتری بعد از پرداخت نباید دچار شکست شود.

۵،۲- خطا در فروشنده:

بررسی بروز خطا در مراحل مختلف اجرای کار فروشنده نشان داد که ، همانند مشتری ، بروز خطا فقط زمانی شرایط امنیت را نقض می‌کند که فروشنده کلید را برای واسطه فرستاده باشد ولی هنوز پرداخت را دریافت نکرده باشد. در سایر موارد با وجود خطا همچنان تمام شرایط برقرار خواهند بود. در اینجا از بیان مثال خطا دار صرف نظر می‌کنیم و به همین نکته بسنده می‌کنیم که مثال آن بسیار شبیه مثال خطا در مشتری است. همانند مشتری ، فروشنده نیز در مراحل پایانی نباید دچار شکست شود.

۵،۳- خطا در واسطه:

واسطه فقط در مرحله اول می‌تواند دچار شکست شود در حالی که شرایط امنیت پروتکل نقض نشود. این امر نیز کاملا بدیهی است زیرا در این پروتکل واسطه نقش کلیدی ارائه دهنده سرویس را بر عهده دارد که به هر دوی مشتری و فروشنده سرویس می‌دهد و در نتیجه شکست آن باعث شکست کل پروتکل خواهد شد.

۶- نتیجه‌گیری و کار آینده:

در ضمن مدل‌سازی پروتکل کاملا مشخص شد که سادگی این کار با استفاده از زبان ربکا به مراتب بیش‌تر از کار با زبان پرومِلا می‌باشد. البته زبان پرومِلا یک زبان قدیمی، پایدار و امتحان پس داده می‌باشد؛ استراتژی تبدیل کد

ربکا به کد پرومِلا استراتژی مقعولی به نظر می‌رسد. اما اگر بتوان امکان نوشتن خصوصیات را مستقیما به زبان ربکا اضافه کرد بسیار به‌تر است تا دیگر استفاده کننده درگیر استفاده از پرومِلا نشود. در ضمن پیاده‌سازی بعضی از قسمت‌های زبان ربکا هنوز به اتمام نرسیده و بنابراین با توجه به پیش‌بینی خصوصیاتی در آن هنوز نمی‌توان از آن‌ها استفاده کرد. به نظر می‌رسد زبان ربکا در صورت اضافه شدن این خصوصیات تا اندازه‌ی زیادی قدرت لازم را بدست آورد و در کنار سادگی در استفاده ابزار مناسبی برای درستی‌یابی شود.

پروتکل مطرح شده می‌تواند پیچیده‌تر از این حرف‌ها باشد. به خصوص در قسمت واسطه ممکن است کارها و اتفاقات بیشتری بیافتد. در ضمن در مدل ما از هر موجود فقط یک نمونه وجود داشت و مدل دنیای بسته بود، می‌توان این مدل را به مدلی حقیقی‌تر و مدل دنیای باز تبدیل کرد. در ضمن به دلیل کوچک بودن مدل درستی‌یابی ترکیبی خیلی معنا ندارد ولی در صورت پیچیده شدن مدل می‌توان این انتخاب را هم در نظر گرفت. در آینده به این کارها خواهیم پرداخت.

۷- منابع:

[1] Ray I., Ray I. "Failure Analysis of an E-Commerce Protocol Using Model Checking," wecwis, p. 176, Second International Workshop on Advance Issues of E-Commerce and Web-Based Information Systems (WECWIS 2000), 2000, pp 176-183.

[2] Ray I., Ray I., and Narasimhamurthy N., “A Fair-exchange E-commerce Protocol with Automated Dispute Resolution”. Proceedings of the IFIP Tc11/ Wg11.3 Fourteenth Annual Working Conference on Database Security: Data and Application Security, Development and Directions (August 21 - 23, 2000).

[3] Sirjani M., Shali A., Jaghouri M. M., Iravanchi H., and Movaghar A., “A Front-End Tool for Automated Abstraction and Modular Verification of Actor-Based Models”, in Proceedings of ACSD 2004, Hamilton, Canada, pp. 145-148, IEEE Computer Society, June 2004.

[4] Sirjani M., Movaghar A., Shali A., and de Boer F.S.,”Modeling and Verification of Reactive Systems using Rebeca” , Fundamenta Informaticae, Vol. 63, Nr. 4, December 2004.

[5] Sirjani M., Movaghar A., and Mousavi M.R., “Compositional Verification of an Actor-Based Model for Reactive Systems”, in Proceedings of Workshop on Automated Verification of Critical Systems (AVoCS'01), Oxford University, April 2001.

[6] Rebeca Homepage, URL: http://khorshid.ece.ut.ac.ir/~rebeca

[7] Spin Model Checker, URL: http://spinroot.com

ضمیمه‌ی ۱ – کد ربکا برای مدل‌سازی سیستم:

```
reactiveclass Customer (4)
{
	knownobjects
	{
		Merchant m;
		ThirdParty tp;
	}

	statevars
	{
		boolean abort;
		boolean success;
		boolean payment;
		boolean egood;
	}

	msgsrv initial()
	{
		abort = false;
		success = false;
		payment = false;
		egood = false;

		self.downloadEGood();
	}


	msgsrv downloadEGood()
	{
		egood = true;
		m.receivePurchaseOrder();
	}

	msgsrv receiveEncryptedGood(boolean isCorrectGood)
	{
		if(isCorrectGood){
			tp.receivePaymentToken();
		}else{
			tp.doAbort();
		}
	}

	msgsrv correctPayment()
	{
		payment = true;
	}

	msgsrv receiveKey()
	{
		success = true;
		self.halt();
	}

	msgsrv halt()
```

```
	{
		self.halt();
	}

	msgsrv doAbort()
	{
		abort = true;
		self.halt();
	}
}

reactiveclass Merchant (4)
{
	knownobjects
	{
		Customer c;
		ThirdParty tp;
	}

	statevars
	{
		boolean abort;
		boolean success;

		boolean isCorrectGood;
	}

	msgsrv initial()
	{
		abort = false;
		success = false;

		isCorrectGood = false;
	}


	msgsrv receivePurchaseOrder()
	{
		isCorrectGood =?{true, false};

		if(isCorrectGood){
			c.receiveEncryptedGood(true);
		}else{
			c.receiveEncryptedGood(false);
		}

		tp.receiveKey();
	}

	msgsrv receivePaymentToken()
	{
		success = true;
	}

	msgsrv doAbort()
	{
		abort = true;
```

```
	}
}

reactiveclass ThirdParty (4)
{
	knownobjects
	{
		Customer c;
		Merchant m;
	}

	statevars
	{
		boolean waitForToken;
		boolean waitForKey;
		boolean abort;
		boolean success;

		boolean isValidToken;
		boolean firstSendToken;
	}

	msgsrv initial()
	{
		waitForToken = true;
		waitForKey = true;
		abort = false;
		success = false;

		isValidToken = false;
		firstSendToken = false;
	}

	msgsrv receiveKey()
	{
		waitForKey = false;

		if(!waitForToken){
			self.checkToken();
		}
	}

	msgsrv receivePaymentToken()
	{
		waitForToken = false;

		if(!waitForKey){
			self.checkToken();
		}
	}

	msgsrv checkToken()
	{
		isValidToken =?{true, false};

		if(isValidToken){
			c.correctPayment();
```

```
                self.sendTokenKey();
            }else{
                self.doAbort();
            }
        }

        msgsrv doAbort()
        {
            c.doAbort();
            m.doAbort();
            abort = true;
        }

        msgsrv sendTokenKey()
        {
            firstSendToken =?{true, false};

            if(firstSendToken){
                m.receivePaymentToken();
                c.receiveKey();
            }else{
                c.receiveKey();
                m.receivePaymentToken();
            }

            success = true;
            waitForToken = true;
            waitForKey = true;
        }
}

main
{
        Customer c(m, tp):();
        Merchant m(c, tp):();
        ThirdParty tp(c, m):();
}
```