



فرم تعریف پروژه فارغ التحصیلی دوره کارشناسی

تاریخ:

شماره:

عنوان پروژه: طراحی و پیاده سازی ابزاری به منظور اعمال خط مشی امنیتی عدم تداخل مبتنی بر روش بازنویسی برنامه	
استاد راهنمای پروژه: دکتر مهران سلیمان فلاح	
امضاء:	
مشخصات دانشجو:	
نام و نام خانوادگی: سید محمدمهدی احمدپناه ^۱	گرایش: نرم افزار
شماره دانشجویی: ۹۰۳۱۸۰۶	ترم ثبت نام پروژه: دوم ۹۳-۹۴
داوران پروژه:	
۱- امضاء داور:	
۲- امضاء داور:	
شرح پروژه (در صورت مشترک بودن بخشی از کار که بعهدہ دانشجو می باشد مشخص شود):	
به پیوست آمده است.	
وسائل مورد نیاز:	
- امکان دسترسی به مقالات مرتبط	
- یک دستگاه کامپیوتر دارای دسترسی به اینترنت	
محل انجام پروژه: دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر	
تاریخ شروع: اردیبهشت ۱۳۹۴	

این قسمت توسط دانشکده تکمیل می گردد:

تاریخ تصویب در گروه:	اسم و امضاء:
تاریخ تصویب در دانشکده:	اسم و امضاء:
اصلاحات لازم در تعریف پروژه:	

توجه: پروژه حداکثر یک ماه و نیم پس از شروع ترمی که در آن در درس پروژه ثبت نام به عمل آمده است باید به تصویب برسد.

نسخه ۱- دانشکده	نسخه ۲- استاد راهنما	نسخه ۳- دانشجو
-----------------	----------------------	----------------

¹ Email: smahmadpanah@aut.ac.ir

تعریف مسئله:

امروزه و با گسترش سیستم‌ها و نرم‌افزارها، امنیت انتقال اطلاعات در برنامه‌های نوشته شده به زبان‌های برنامه‌نویسی گوناگون، بیش از پیش اهمیت پیدا کرده است. از این رو، می‌توان با افزودن ابزارهایی به یک زبان برنامه‌نویسی، باعث تولید نرم‌افزارهای مقاوم‌تری در برابر حملات و نفوذها شد. لذا باید مفهوم امن بودن یک سیستم یا برنامه به طور دقیق تعریف و مشخص شود که همین مسئله، چالشی برای متخصصان این حوزه است. در مرحله بعدی، نحوه و رویکرد اعمال آن مفهوم از امنیت مهم خواهد بود که وابستگی زیادی به تعریف ارائه شده دارد.

به طور کلی، خط مشی^۲ امنیتی، امن بودن یک سیستم یا برنامه را تعریف می‌کند. خط مشی امنیتی، قیود روی توابع و جریان‌های بین آن‌ها را مشخص می‌کند؛ مثل قیود دسترسی بر روی برنامه‌ها و سطوح دسترسی داده‌های بین کاربران که مانع از بروز مشکلات امنیتی از طریق سیستم‌های خارجی و نفوذگران شود.

یک خط مشی امنیتی را می‌توان به عنوان یک زیرمجموعه از مجموعه‌ای توانی همه اجراها، که هر اجرا یک دنباله دلخواه از حالت‌ها^۳ است، تعریف کرد. ضمناً می‌توان آن را به عنوان مجموعه‌ای برنامه‌هایی که آن خط مشی را برآورده می‌کنند، در نظر گرفت. بعضی از خط مشی‌های امنیتی، خاصیت^۴ هستند؛ به خاطر این که قابل دسته‌بندی و تشخیص توسط مجموعه اجراهای جداگانه می‌باشند. برخی از نیازمندی‌های مهم امنیتی، خاصیت نیستند. یک نمونه مهم از این گونه نیازمندی‌ها، خط مشی عدم تداخل^۵ است. نکته حائز اهمیت این است که روش اعمال خاصیت‌ها با نحوه اعمال خط مشی‌هایی که خاصیت نیستند، متفاوت است.

به زبان ساده، خط مشی عدم تداخل بیان می‌کند که یک مشاهده‌گر^۶ سطح پایین که فقط به برنامه و مقادیر عمومی زمان اجرا دسترسی دارد، نتواند ورودی‌های سطح بالا یا خصوصی برنامه را بفهمد. به عبارت دیگر، این خط مشی بیان می‌کند که در هر جفت اجراهای برنامه که ورودی‌های عمومی یکسان دارند، مستقل از ورودی‌های خصوصی متفاوت، باید خروجی‌های عمومی یکی باشند.

نکته مهم این است که خط مشی عدم تداخل، یک خاصیت نیست؛ زیرا توسط اجراهای جداگانه که این خط مشی را برآورده می‌کند، قابل تعریف نیست. این نکته باعث ایجاد محدودیت‌هایی برای اعمال این خط مشی در برنامه‌ها می‌شود.

خط مشی عدم تداخل را می‌توان به دو دسته حساس به پیشرفت^۷ و غیر حساس به پیشرفت^۸ تقسیم کرد. در عدم تداخل غیر حساس به پیشرفت، مشاهده‌گر سطح پایین، تنها می‌تواند خروجی‌های میانی سطح پایین را ببیند؛ در حالی که یک مشاهده‌گر سطح پایین در عدم تداخل حساس به پیشرفت، علاوه بر دسترسی‌های قبلی، به وضعیت پیشرفت^۹ برنامه نیز دسترسی دارد. این باعث می‌شود تا بتواند تفاوت بین واگرایی^{۱۰} برنامه با موقعیتی که برنامه خاتمه می‌یابد یا در حال محاسبه مقادیر قابل مشاهده‌ی بعدی است، را تمیز دهد.

نکته مهم دیگر، بررسی جریان‌های غیرمجاز صریح و ضمنی است که می‌تواند مسئله‌ی داده شده را پیچیده کند.

² Policy

³ State

⁴ Property

⁵ Noninterference

⁶ Observer

⁷ Progress-Sensitive

⁸ Progress-Insensitive

⁹ Progress Status

¹⁰ Divergence

در این پروژه، هدف این است که با بهره‌گیری از روش بازنویسی برنامه^{۱۱}، برنامه‌های نوشته‌شده توسط یک زبان برنامه‌نویسی مدل به نام WL به نحوی تغییر داده شوند که خط مشی امنیتی عدم تداخل را برآورده سازند.

راه حل‌های فعلی و مشکلات آن‌ها:

وِنگتَکْرِیشَن و دیگران یک روش تبدیل برنامه‌ی ترکیبی برای اعمال عدم تداخل ارائه داده‌اند. برنامه تغییر داده شده، سطوح امنیتی انتساب^{۱۲} را دنبال می‌کند و زمانی که یک جریان غیرمجاز در حال وقوع باشد، خاتمه می‌یابد. این روش، تنها در فرمول‌بندی‌هایی از عدم تداخل که بدون توجه به رفتار خاتمه‌ی برنامه‌ها مطرح می‌شود، قابل استفاده است. مَگْزِینیوس و دیگران یک چارچوب برای ناظر^{۱۳}های امنیتی پویای درون‌برنامه‌ای- در حالی که برنامه در حال اجراست- ساخته‌اند. این روش، عدم تداخل غیر حساس به خاتمه را تضمین می‌کند و قابل به‌کارگیری در زبان‌های Perl و JavaScript است که از ارزیابی پویای کد پشتیبانی می‌کنند. ضمناً این روش نیاز دارد که تغییردهنده‌ی برنامه در زمان اجرا در دسترس باشد که یک ناظر مناسب بتواند در کدی که به صورت پویا تولید می‌شود، ورود کند.

چادَنُوف و نوَمَن یک ناظر ترکیبی برای عدم تداخل حساس به جریان در زبان‌های با ارزیابی پویای کد پیشنهاد داده‌اند. این روش ممکن است باعث وجود یک سربار غیرقابل قبول در زمان اجرا شود. همچنین این روش، اجازه وقوع مجراهای خاتمه^{۱۴} را نمی‌دهد. سانتوس و رزک نیز این روش را برای یک هسته JavaScript گسترش دادند.

این موضوع اثبات شده است که هیچ روش کاملاً پویایی برای اعمال عدم تداخل حساس به جریان وجود ندارد. این موضوع باعث می‌شود که پروژه‌هایی که محدودیت‌های نحوی^{۱۵} بر روی کد دارند، از اطلاعات ایستا در ناظری بر اجراهای چندگانه‌ی برنامه‌ها استفاده کنند.

بلو و بونلی یک ناظر اجرایی^{۱۶} پیشنهاد دادند که از یک تحلیل وابستگی زمان اجرا بهره می‌برد. برای یافتن یک جریان غیرمجاز، همان‌طور که در طرح پیشنهادی آن‌ها و کارهای مشابه دیگر آمده است، ممکن است نیاز به چندین اجرا از برنامه مورد نظر داشته باشد که در بسیاری از کاربردها این امکان وجود ندارد.

لِگوئِرَنیک و دیگران یک ماشین طراحی کرده‌اند که رخدادهای انتزاعی^{۱۷} در زمان اجرا را دریافت می‌کند و اجرا را توسط بعضی از اطلاعات ایستا، ویرایش می‌کند. این روش نیز جالب است اما اجازه وقوع مجراهای خاتمه را می‌دهد.

به طور کلی، مسئله‌ی تشخیص برنامه‌هایی که عدم تداخل را برآورده می‌کنند، تصمیم‌ناپذیر^{۱۸} است. پس در حالت کلی، عدم تداخل توسط روش‌های ایستا قابل اعمال نیست؛ به همین دلیل است که نوع‌سامانه^{۱۹}های ارائه‌شده برای این مسئله، محافظه‌کارانه^{۲۰} هستند و ممکن است بعضی برنامه‌های امن را نیز رد کنند. از طرفی، این مسئله هم‌بازگشتی شمارش‌پذیر^{۲۱} نیز نیست. بنابراین، قابل اعمال توسط ناظرهای اجرایی که نقض خط مشی عدم تداخل در یک برنامه‌ی در حال اجرا را بررسی می‌کنند، نیست.

¹¹ Program Rewriting

¹² Assignment

¹³ Monitor

¹⁴ Termination Channels

¹⁵ Syntactic

¹⁶ Execution Monitor

¹⁷ Abstract Events

¹⁸ Undecidable

¹⁹ Type System

²⁰ Conservative

²¹ Co-recursively Enumerable

با توجه به مشکلات اخیر مطرح شده در روش‌های قبلی، روش بازنویسی برنامه تغییراتی را به ذاتِ عدم تداخل وارد نمی‌کند؛ بلکه به جای آن، یک برنامه جدید با حداقل تغییرات ممکن نسبت به برنامه اصلی، که عدم تداخل را برآورده می‌کند، تولید می‌کند. در واقع، می‌توان روش بازنویسی برنامه را روشی بین روش‌های ایستا و روش‌های پویا دانست.

راه حل پیشنهادی:

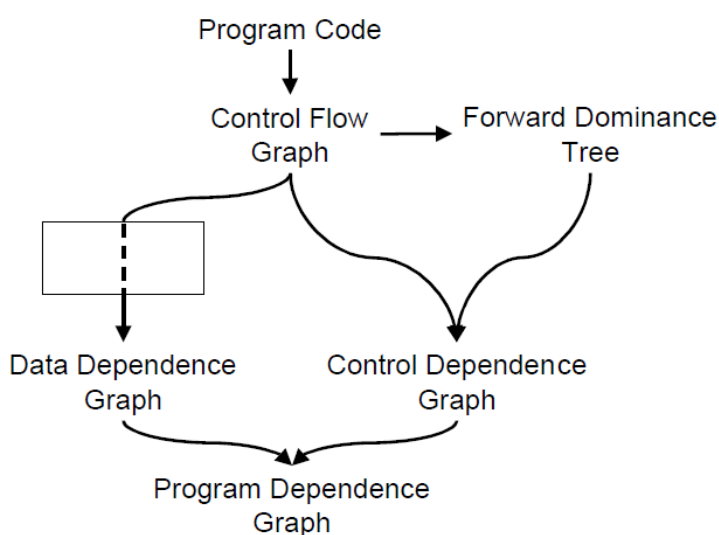
راه حل مورد توجه در این پروژه، از گراف‌های وابستگی برنامه^{۲۲} استفاده می‌کند. گراف وابستگی برنامه، ابزاری است که توسط آن، یک برنامه با یک گراف جهت‌دار که گره‌های آن گزاره‌ها یا عبارت‌های برنامه است و یال‌های آن، وابستگی‌های کنترلی یا داده‌ای بین گره‌ها را نشان می‌دهد، بیان می‌شود. گراف وابستگی یک برنامه، همه وابستگی‌های بین گزاره‌های آن برنامه را منعکس می‌کند. البته ممکن است مسیری بین گره‌ها در این گراف وجود داشته باشد ولی جریانی بین آن دو برقرار نباشد.

برخلاف سایر روش‌های امنیتی مطرح شده، روش این پروژه، یک برنامه مغایر با خط مشی موردنظر را، چه قبل یا چه در طول زمان اجرا، رد نمی‌کند؛ بلکه آن‌ها بازنویسی می‌شوند و از برنامه‌های ناامن به برنامه‌های امن تبدیل می‌شوند. در این روش، هر دو نوع عدم تداخل - حساس به پیشرفت و غیر حساس به پیشرفت - در برنامه‌ها به همراه مقادیر قابل مشاهده میانی، مورد توجه هستند. ضمناً قابل اثبات است که بازنویس^{۲۳}‌های مورد نظر، در اعمال عدم تداخل، سالم^{۲۴} و شفاف^{۲۵} هستند. شفاف بودن یک روش بدین معناست که تا حد ممکن، مجموعه اجراهای ممکن برنامه‌ی تبدیل شده مشابه برنامه‌ی ورودی باشد، خواه امن باشد یا نباشد.

توصیف نرم‌افزار:

ورودی: کد برنامه (که ممکن است خط مشی عدم تداخل را برآورده نکند)

خروجی: کد برنامه تغییر داده شده (که خط مشی مورد نظر را برآورده می‌کند)



شکل ۱- نمودار کلی نحوه به دست آوردن گراف وابستگی برنامه

پردازش: با استفاده از کد برنامه ورودی،

گراف وابستگی برنامه از روی آن ساخته می‌شود. سپس با توجه به این که خط مشی مورد نظر حساس به پیشرفت یا غیر حساس به پیشرفت است، پردازش مربوط به هر کدام انجام می‌شود.

برای ساخت گراف وابستگی برنامه، باید گراف‌های وابستگی کنترل و وابستگی داده را از روی کد برنامه تشکیل داد. گراف وابستگی برنامه از گراف کنترل جریان به دست می‌آید. با تشخیص وابستگی‌های کنترل و داده، گراف کنترل جریان به گراف وابستگی برنامه تبدیل می‌شود.

²² Program Dependence Graph (PDG)

²³ Rewriter

²⁴ Sound

²⁵ Transparent

گراف کنترل جریان گرافی است که گره‌های آن، دستورهای برنامه و جریان‌های کنترلی بین گره‌ها نمایش داده می‌شود. دو گره شروع و پایان نیز نقاط ورود و خروج برنامه را مشخص می‌کند.

وابستگی داده بین گره X به Y یعنی گره Y شامل متغیری است که در X منتسب شده است. وابستگی کنترل بین گره X به Y یعنی اجرای Y توسط مقداری که در X محاسبه شده است، کنترل می‌شود. بنابراین، هر مسیر در گراف وابستگی برنامه ممکن است جریانی بین دو گره باشد. در گراف وابستگی برنامه، می‌گوییم جریانی از X به Y وجود دارد، اگر مقدار محاسبه شده در Y یا صرف اجرای Y ، وابستگی به مقدار محاسبه شده در X داشته باشد.

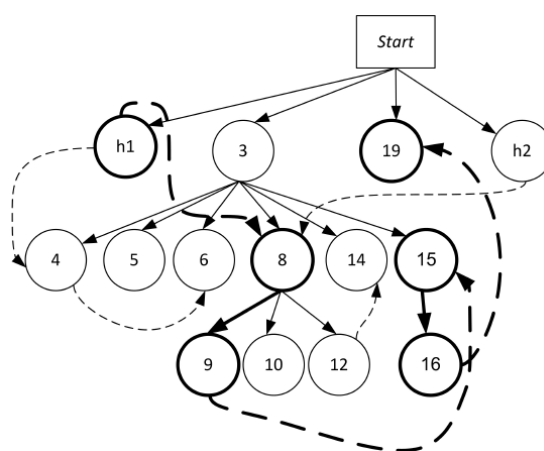
با استفاده از درخت غلبه رو به جلو^{۲۶}، وابستگی کنترلی Y به X این گونه تعریف می‌شود که یک مسیر در گراف کنترل جریان از X به Y وجود دارد که حاوی غلبه‌کننده رو به جلوی بلافاصله^{۲۷} نیست.

تعاریف و الگوریتم‌های مبتنی بر گراف کنترل جریان نیازمند وجود یک گراف هم‌بند است که در این زبان مدل، به دلیل عدم وجود تابع یا رویه، مشکلی ایجاد نخواهد کرد.

```

1.  $in_H h_1, h_2;$ 
2.  $in_L l_1, l_2, l_3;$ 
3.  $if (l_1 \leq 0) then$ 
4.    $l_2 = h_1;$ 
5.    $out_L l_1;$ 
6.    $out_L l_2;$ 
7.  $else$ 
8.    $if(h_2 == h_1) then$ 
9.      $l_3 = 0;$ 
10.     $out_L l_1;$ 
11.    $else$ 
12.      $l_1 = 1;$ 
13.    $endif;$ 
14.   $out_L l_1;$ 
15.   $if(l_3 == 0) then$ 
16.     $l_3 = 1;$ 
17.   $endif;$ 
18.  $endif;$ 
19.  $out_L l_3;$ 

```



شکل ۲- نمونه کدی به زبان WL و گراف وابستگی برنامه آن

در شکل شماره ۲، نمونه کدی به زبان مدل مورد نظر و همچنین، گراف وابستگی برنامه متناظر با آن وجود دارد. در این گراف، یال‌های با خطوط ممتد، بیانگر وابستگی‌های کنترلی و یال‌های خط‌چین، بیانگر وابستگی‌های داده در برنامه هستند. البته به دلیل بررسی خط مشی عدم تداخل، وابستگی‌های داده‌ای که از ورودی‌های سطح پایین ناشی می‌شوند، در شکل نیامده است. به عنوان مثال، یال‌های پررنگ شده نشان دهنده مسیر از ورودی سطح بالای h_1 تا دستور خروجی متغیر l_3 در خط پایانی برنامه است.

در عدم تداخل غیر حساس به پیشرفت، از روی گراف وابستگی برنامه، مسیرهای حاوی ورودی سطح بالا که در ادامه آن‌ها خروجی سطح پایینی نمایش داده خواهد شد، مورد بررسی قرار می‌گیرند. با استفاده از شرط‌های اجرا^{۲۸} و شرط‌های مسیر^{۲۹}، بسته به وجود جریان صریح یا ضمنی، دستور نمایش یک مقدار سطح پایین، به عبارتی شرطی تبدیل می‌شود که در صورت برقراری آن شرط، مقدار خروجی نشان داده می‌شود و در غیر این صورت، آن مقدار قابل مشاهده برای مشاهده‌گر سطح پایین نخواهد بود.

²⁶ Forward Dominance Tree

²⁷ Immediate Forward Dominator

²⁸ Execution Conditions

²⁹ Path Conditions

در عدم تداخل حساس به پیشرفت، به دلیل این که مشاهده گر سطح پایین دسترسی های بیشتری دارد، لذا رفتار برنامه نیز اهمیت بیشتری پیدا می کند؛ یعنی باید بازنویس به گونه ای کد برنامه را تغییر دهد که وضعیت پیشرفت در برنامه ی تغییر پیدا کرده، به مقادیر سطح بالا وابستگی نداشته باشد. به همین دلیل، واگرایی یا خاتمه برنامه از دید مشاهده گر سطح پایین نیز حاوی اطلاعاتی است که باعث نقض خط مشی عدم تداخل در این حالت می شود. در این حالت، روش پردازش و بازنویسی برنامه، به یک تحلیل گر حلقه³⁰ وابسته خواهد شد. در ابتدا، مراحل حالت غیر حساس به پیشرفت بر روی کد برنامه ورودی انجام می شود و در صورتی که برنامه دارای مسیرهایی باشد که در آن قبل از رسیدن به یک حلقه، مقدار ورودی سطح بالایی وجود داشته باشد، مطابق با نتیجه ی تحلیل گر حلقه در هر کدام از حلقه ها، کدهای مربوط به آن بازنویسی و اصلاح می شوند. این گونه کد برنامه خروجی نیز خط مشی عدم تداخل حساس به پیشرفت را برآورده می سازد.

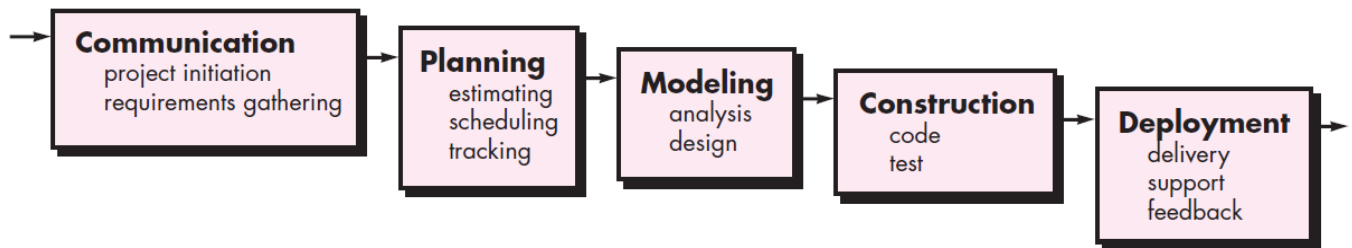
ضمناً می توان با استفاده از ابزارهای موجود مانند lex و yacc، کدهای به زبان WL را به کد میانی به زبان C تبدیل کرد.

همچنین برای آشنایی بیشتر کاربر با نرم افزار و الگوریتم ها، بخش راهنمای استفاده از برنامه تهیه خواهد شد.

³⁰ Loop Analyzer

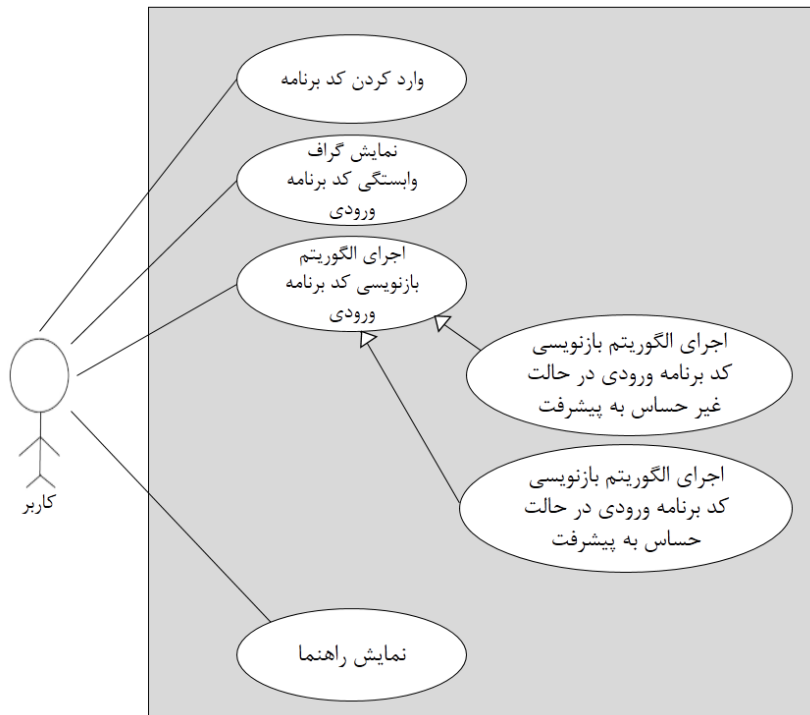
تحلیل نرم افزار:

با توجه به مشخص و ثابت بودن نیازهای این نرم افزار در ابتدای تعریف پروژه، می توان از مدل فرآیندی آبشاری^{۳۱} یا چرخه حیات کلاسیک^{۳۲} استفاده کرد. این مدل فرآیندی شامل پنج مرحله ارتباط^{۳۳}، برنامه ریزی^{۳۴}، مدل سازی^{۳۵}، ساخت^{۳۶} و استقرار^{۳۷} است.



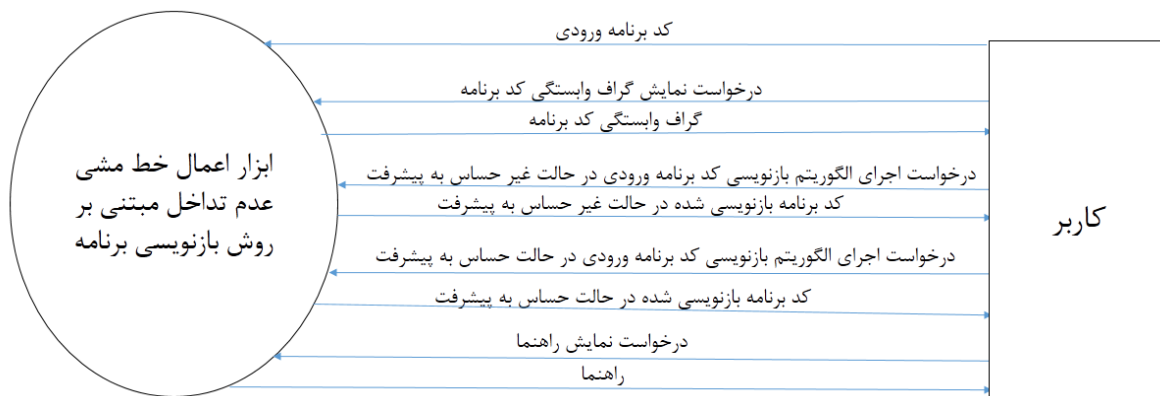
شکل ۳ - مدل فرآیندی آبشاری

نمودار Use Case:



شکل ۴ - نمودار Use Case

³¹ Waterfall
³² Classic Life Cycle
³³ Communication
³⁴ Planning
³⁵ Modeling
³⁶ Construction
³⁷ Deployment



شکل ۵ - نمودار Context

برای پیاده‌سازی این ابزار می‌توان از زبان‌های سطح بالای برنامه‌نویسی نظیر جاوا و محیط‌های توسعه مانند نت‌بینز^{۳۸} بهره برد.

³⁸ NetBeans IDE

- [1] A. Lamei and M.S. Fallah, "Rewriting-based Enforcement of Noninterference in Programs with Observable Intermediate Values", To appear in the Journal of Universal Computer Science, Vol. XX, No. XX, Month 20XX, 1-24.
- [2] J. Ferrante, K.J. Ottenstein, and J.D. Warren, "The program dependence graph and its use in optimization, ACM Transactions on Programming Languages and Systems 9, 1987, pp. 319-349.
- [3] D. Wasserrab, D.Lohner, and G. Snelting, "On PDG-based noninterference and its modular proof", in Proceedings of the ACM SIGPLAN Fourth Workshop on Programming Languages and Analysis for Security, PLAS '09, ACM, 2009, pp. 31-44.
- [4] C. Hammer and G. Snelting, "Flow-Sensitive, Context-Sensitive, and Object-sensitive Information Flow Control Based on Program Dependence Graphs", the International Journal of Information Security, Volume 8, Issue 6, 2009, pp 399-422