**Domain Specific Languages in Software Architecture**

|  |  |
| --- | --- |
| Matthias Anlauff  GMD FIRST  Software Engineering Group  Rudower Chaussee 5  D-12489 Berlin, Germany  +49-30-6392-1827 ma@first.gmd.de | Asuman Sünbül  Technical University of Berlin  Computer-Aided Information Systems  Einsteinufer 17  D-10587 Berlin, Germany  +49-30-314-79463 asu@cs.tu-berlin.de |

# ABSTRACT

Building a software product requires knowledge in both software engineering and the domain the software is written for. However, common software engineering methods in fact exclude domain experts from the architectural design of a software system, because the generality of the notations defined by these methods require expert knowledge in the field of software engineering and especially software architecture. In order to avoid this effect, domain-specific languages (DSL) have been proposed as an alternative to the general purpose languages being normally used in software architecture. One of the advantages of DSLs is that they are less cryptic and easier to learn for domain experts. Of course the design of such a language requires deep understanding of the problem domain, on the other hand the existing tools for language design are only mastered by a few experts. In order to make DSL design applicable by a broader range of people - especially domain experts - we use Montages, a semi-visual meta language based on the Abstract State Machines approach. A DSL can be specified in Montages by means of visual descriptions of the language constructs. These descriptions can be fed to a rapid prototyping tool, called GemMex, which automatically generates a visual programming environment for the specified DSL.

ساختن یک محصول نرم افزاری نیاز به دانش در مهندسی نرم افزار و دامنه ای دارد که نرم افزار برای آن نوشته شده است. با این حال ، روشهای معمول مهندسی نرم افزار در حقیقت متخصصان دامنه را از طراحی معماری یک سیستم نرم افزاری محروم می کنند ، زیرا کلیات نمادهای تعریف شده توسط این روش ها نیاز به دانش تخصصی در زمینه مهندسی نرم افزار و به خصوص معماری نرم افزار دارد. به منظور جلوگیری از این تأثیر ، زبانهای اختصاصی دامنه (DSL) به عنوان جایگزینی برای زبانهای هدف کلی که معمولاً در معماری نرم افزار استفاده می شوند ، پیشنهاد شده اند. یکی از مزایای DSL ها این است که آنها رمزنگاری کمتری دارند و یادگیری آنها برای کارشناسان حوزه آسانتر است. البته طراحی چنین زبانی مستلزم درک عمیق از حوزه مسئله است ، از طرف دیگر ابزارهای موجود برای طراحی زبان فقط توسط تعدادی متخصص انجام می شود. برای اینکه طرح DSL توسط طیف وسیعی از مردم - بخصوص کارشناسان حوزه - قابل استفاده باشد ، از Montages ، یک زبان متا نیمه بصری مبتنی بر رویکرد دستگاه های حالت دولت انتزاعی استفاده می کنیم. DSL را می توان با استفاده از توضیحات بصری ساختارهای زبان در مونتاژها مشخص کرد. این توضیحات را می توان به یک ابزار نمونه سازی سریع با نام GemMex که به طور خودکار یک محیط برنامه نویسی بصری برای DSL مشخص ایجاد می کند ، تغذیه کرد.

# KEYWORDS

Domain-Specific languages, software architecture, visual specification, generation of visual programming environments.

# INTRODUCTION

Building a piece of software settled in a certain problem domain is usually done by software engineers. Although it would often make sense to integrate people with domain-specific knowledge into the software engineering process as a whole, these people normally just contribute at the very beginning and at the very end to that process: in the requirements specification and the software testing and evaluation phase.

ساختن یک قطعه از نرم افزار مستقر در یک دامنه مشکل خاص معمولاً توسط مهندسان نرم افزار انجام می شود. اگرچه اغلب ادغام افراد با دانش خاص دامنه در کل مهندسی نرم افزار کاملاً منطقی است ، اما معمولاً این افراد در همان ابتدا و در انتهای آن فرآیند مشارکت می کنند: در مشخصات الزامات و آزمایش نرم افزار و مرحله ارزیابی

Standard software engineering methods like UML [UML98] and Fusion[Fusion94] define general-purpose languages for the description of components occurring in a software development. Because of their generality, the use of these languages requires technical knowledge of software engineering and are not appropriate for being used by other people. This fact prevents experts of the domain the software is written for to actively contribute to the architecture of the software, because the threshold to get acquainted with the general-purpose design languages is too high. Domain-specific languages (DSL), if designed properly for the problem domain of the software, have the advantage, that this threshold can be substantially lowered for domain experts [Bentley86, DeuKli98, CM98].

روشهای استاندارد مهندسی نرم افزار مانند UML [UML98] و Fusion [Fusion94] برای توصیف مؤلفه های رخ داده در توسعه نرم افزار ، زبانهای عمومی را تعریف می کنند. به دلیل کلی بودن آنها ، استفاده از این زبانها نیاز به دانش فنی مهندسی نرم افزار دارد و برای استفاده افراد دیگر مناسب نیست. این واقعیت مانع از آن می شود که متخصصان دامنه نرم افزاری برای کمک به طور فعال در معماری نرم افزار مشارکت کنند ، زیرا آستانه آشنایی با زبان های طراحی کلی بسیار بالا است. زبانهای اختصاصی دامنه (DSL) ، اگر به درستی برای دامنه مشکل نرم افزار طراحی شده باشند ، از این مزیت برخوردار هستند که می توان این آستانه را برای کارشناسان دامنه بطور قابل ملاحظه ای کاهش داد [Bentley86، DeuKli98، CM98].

The approach presented in this paper addresses not only the use of DSL in software architecture but also the *design* of DSL. The advantages gained from the use of domain-specific languages in software architecture stand and fall with the appropriateness and acceptance of these languages. The design of a good domain-specific language requires even more knowledge than the standard software development using general purpose languages, because a DSL must form an abstraction over an entire domain while in the latter case a single concrete problem of that domain must be solved. Thus, domain experts should also participate in the design of a DSL and therefore it is necessary to open the field of DSL design to people not having experience in programming language design and implementation.

رویکرد ارائه شده در این مقاله نه تنها به استفاده از DSL در معماری نرم افزار بلکه به طراحی DSL می پردازد. مزایای به دست آمده از استفاده از زبانهای خاص دامنه در معماری نرم افزار با مناسب بودن و پذیرش این زبانها پدیدار می شود. طراحی یک زبان خاص برای دامنه خوب حتی نیاز به دانش بیشتری نسبت به توسعه نرم افزار استاندارد با استفاده از زبانهای هدف عمومی دارد ، زیرا یک DSL باید انتزاعی را در کل یک دامنه تشکیل دهد در حالی که در حالت دوم یک مشکل خاص آن دامنه باید حل شود. بنابراین ، کارشناسان حوزه نیز باید در طراحی DSL شرکت کنند و بنابراین لازم است زمینه طراحی DSL به افرادی که تجربه ای در طراحی و اجرای زبان برنامه نویسی ندارند ، باز شود.

In order to make domain-specific language design applicable especially for problem domain experts, we use *Montages* [KP97a,AKP97], a semi-visual meta language for the design of programming languages. Montages are based on the *Abstract State Machines* (ASM) [Gurevich94b] approach which has proven to be appropriate to formally specify the semantics of realistic programming languages. A DSL can be specified in Montages mainly by means of visual descriptions of the language constructs. These descriptions can be fed to a rapid prototyping tool, called Gem-Mex, which automatically generates a visual programming environment for the specified DSL.

برای اینکه طراحی زبان خاص دامنه مخصوصاً برای متخصصان حوزه مشکل قابل اجرا باشد ، ما از Montages [KP97a، AKP97] ، یک متا نیمه بصری برای طراحی زبانهای برنامه نویسی استفاده می کنیم. مونتاژها براساس رویکرد دستگاههای دولت انتزاعی (ASM) [Gurevich94b] است که ثابت شده است که به طور رسمی برای مشخص کردن معانی زبانهای برنامه نویسی واقع گرایانه مناسب است. DSL را می توان در مونتاژها عمدتاً از طریق توصیفات بصری از سازه های زبان مشخص کرد. این توضیحات را می توان به یک ابزار نمونه سازی سریع با نام Gem-Mex منتقل کرد که به طور خودکار یک محیط برنامه نویسی بصری را برای DSL مشخص ایجاد می کند.

In the sequel, we further explain the use of DSLs in software architecture and described the formalism used -- Montages -- and its tool environment Gem-Mex. Section 4 shows how a DSL can be designed for a specific software architectural problem domain using Montages/Gem-Mex.

در ادامه ، ما بیشتر استفاده از DSL ها را در معماری نرم افزار توضیح می دهیم و فرمالیسم به کار رفته - Montages - و محیط ابزار آن Gem-Mex را شرح می دهیم. بخش 4 نشان می دهد که چگونه DSL می تواند برای یک دامنه مشکل معماری نرم افزار خاص با استفاده از Montages / Gem-Mex طراحی شود.

# DOMAIN-SPECIFIC LANGUAGES IN SOFTWARE ARCHITECTURE

A DSL can be regarded as a programming or specification language dedicated to a particular domain or problem. The advantage of a domain-specific language in contrast to a general purpose language is that the DSL provides appropriate built-in abstractions and notations. In a broad range of application domains, DSLs are used, although often not recognised as such. Popular examples are the UNIX-shell startup files (e.g. *.bashrc*, *.cshrc*) or the customisation files for X-window manager applications (e.g. *.fvwmrc*, *.twmrc*). However, although in each of the above cases the requirements of being a “domain-specific language” are fulfilled, this term is rarely used. More complex examples are languages providing interfaces to program libraries. A DSL makes it much easier to use the functionality of the library, because unnecessary technical details can be incorperated in the DSL. For example, Tcl/Tk provides a Tcl interface on top of the Tk graphic toolkit. More generally speaking, each DSL is specialised for a set of problems that share enough characteristics that it is worthwhile to study them as a whole. Examples for application domains where DSL have been used are graphics [Ell97,KH97], telephone switching systems [GJKW97], financial software [ADR95], robot languages [Bja96], and data-warehouse application [AKP98d].

DSL را می توان به عنوان یک زبان برنامه نویسی یا خصوصی سازی اختصاص داده شده به یک دامنه یا مشکل خاص در نظر گرفت. مزیت یک زبان خاص دامنه در مقایسه با یک زبان عمومی این است که DSL دارای انتزاعات داخلی و علائم مناسب می باشد. در طیف گسترده ای از دامنه های کاربردی ، از DSL ها استفاده می شود ، اگرچه اغلب به عنوان اینگونه شناخته نمی شوند. نمونه های رایج عبارتند از: پرونده های راه اندازی پوسته UNIX (مانند .bashrc ، .cshrc) یا پرونده های سفارشی سازی برای برنامه های مدیر X-windows (به عنوان مثال .fvwmrc ، .twmrc). با این حال ، اگرچه در هر یک از موارد فوق الزامات "زبان خاص دامنه" برآورده می شود ، این اصطلاح بندرت استفاده می شود. نمونه های پیچیده تر زبانهایی هستند که رابط هایی برای کتابخانه های برنامه ارائه می دهند. DSL استفاده از قابلیت های کتابخانه را بسیار آسان تر می کند ، زیرا جزئیات فنی غیر ضروری را می توان در DSL اشتباه گرفت. به عنوان مثال ، Tcl / Tk رابط Tcl را در بالای ابزار گرافیکی Tk فراهم می کند. به طور کلی تر ، هر DSL برای مجموعه ای از مشکلات تخصصی است که ویژگی های کافی را به اشتراک می گذارد که مطالعه آنها به عنوان یک کل ارزش دارد. نمونه هایی برای دامنه های کاربردی که در آن از DSL استفاده شده است عبارتند از: گرافیک [Ell97، KH97] ، سیستم های سوئیچینگ تلفنی [GJKW97] ، نرم افزار مالی [ADR95] ، زبان های روبات [Bja96] و برنامه انبار داده [AKP98d].

**Why using DSLs in Software Architecture?**

In software architecture, several aspects make the use of DSLs interesting:

* *Involving domain experts in the software development process.* Due to the fact that software engineering methods contain general purpose languages for describing the development of a software product, people other than software engineers usually are excluded from the actual production process. On the other hand, experts of the problem domain are often needed not only during the requirements specification but also in the architectural phase. For example, if a developer is uncertain about the desired functionality of a feature that he or she has to specify, this feature must be traced back to the requirement specification, in order to make a communication with the domain expert about this problem possible. The general purpose languages are inappropriate for this purpose, because they are software architecture languages and not domain languages. If properly designed, DSLs provide a chance to also involve these domain experts in the architecture of the software product. We don`t claim, that software architects are no longer needed, if DSLs are used, but we claim, that DSLs extend the range of people being able to contribute to the architecture of the software product.
* - درگیر شدن کارشناسان دامنه در فرایند توسعه نرم افزار. با توجه به اینکه روشهای مهندسی نرم افزار حاوی زبانهای هدف کلی برای توصیف توسعه محصول نرم افزاری است ، افراد دیگری به غیر از مهندسین نرم افزار معمولاً از فرآیند تولید واقعی خارج می شوند. از طرف دیگر ، متخصصان حوزه مشکل اغلب نه تنها در طی مشخصات الزامات بلکه در مرحله معماری نیز مورد نیاز هستند. به عنوان مثال ، اگر یک توسعه دهنده درباره عملکرد مطلوب یک ویژگی که باید آن را مشخص کند ، اطمینان ندارد ، باید این ویژگی را در قسمت مشخصات مورد نیاز ردیابی کرد تا ارتباط با کارشناس دامنه در مورد این مشکل امکان پذیر باشد. زبانهای هدف عمومی برای این منظور نامناسب هستند ، زیرا آنها زبانهای معماری نرم افزار هستند و نه زبانهای دامنه. اگر به درستی طراحی شود ، DSL ها فرصتی را برای شرکت درگیر این متخصصان دامنه در معماری محصول نرم افزار فراهم می کنند. ما ادعا نمی کنیم ، در صورت استفاده از DSL ها ، دیگر نیازی به معماران نرم افزار نداریم ، اما ما ادعا می کنیم که DSL ها محدوده افرادی را که قادر به کمک به معماری محصول نرم افزار هستند ، گسترش می دهد.
* *Software Architecture using DSLs.* DSLs provide enough abstraction that they can serve as architectural specification language during the design phase of a software development. Nevertheless, because many technical details are already built in the semantics of the DSL, the specification written in a DSL can often be used to automatically generate code that forms the implementation. Thus, DSLs often bridge the gap that exists between the phases of the software engineering process, especially between the design and the implementation phase.
* - معماری نرم افزار با استفاده از DSL. DSL انتزاع کافی را فراهم می کند که می تواند به عنوان زبان خصوصیات معماری در مرحله طراحی یک نرم افزار خدمت کند. با این وجود ، از آنجا که بسیاری از جزئیات فنی در معناشناسی DSL در حال حاضر ساخته شده اند ، مشخصات نوشته شده در DSL اغلب می تواند برای تولید خودکار کدی که اجرای را تشکیل می دهد ، مورد استفاده قرار گیرد. بنابراین ، DSL ها اغلب شکافی را که بین مراحل فرآیند مهندسی نرم افزار وجود دارد ، خصوصاً بین طراحی و مرحله اجرا ، ایجاد می کنند.
* *Domain-specific software architecture.* The idea of having domain-specific support for a class of similar problems can also be mapped to software architecture. Domain-specific software architecture approaches have been defined as specialisation of general purpose software architecuture description languages in order to provide optimised support for a specific domain [MG92, HRPL+95, Tra94, Ves94]. The definition of a domain specific software architecture methods reflects a commonly occuring situation in practise: “off-the-shelve” software engineering and especially software architecture methods are modified so that they meet the requirements of the domain they are appointed to. In this respect, DSLs play a central role while defining a specialised software architecture approach, because they can be used at all places, where abstractions from the general purpose approach should be constituted.
* - معماری نرم افزار خاص دامنه. ایده داشتن پشتیبانی اختصاصی از دامنه برای طبقه ای از مشکلات مشابه نیز می تواند در معماری نرم افزار ترسیم شود. رویکردهای معماری نرم افزار خاص دامنه به عنوان ارائه تخصص پشتیبانی از زبانهای توصیف معماری نرم افزارهای عمومی تعریف شده است تا بتواند پشتیبانی بهینه از یک دامنه خاص ارائه دهد [MG92، HRPL + 95، Tra94، Ves94]. تعریف روشهای معماری نرم افزاری خاص دامنه ، یک وضعیت معمول در حال وقوع را منعکس می کند: مهندسی نرم افزار "خارج از قفسه" و به ویژه روشهای معماری نرم افزار به گونه ای اصلاح شده اند که نیازهای دامنه مورد نظر خود را برآورده سازد. از این نظر ، DSL ها ضمن تعریف یک رویکرد تخصصی در معماری نرم افزار ، نقش اصلی را ایفا می کنند ، زیرا می توانند در همه مکان ها ، در جایی که انتزاع از رویکرد هدف کلی باید ایجاد شود ، استفاده شوند.

# DESIGN OF DOMAIN-SPECIFIC LANGUAGES

In general, using DSLs means that a huge amount of domain knowledge is moved from the actual program to the compiler and runtime system of the implementation language. Thus, the design of a DSL requires even more understanding of the problem domain than writing a single program for that domain in a general purpose language. DSLs only make sense, if they contain useful abstractions for the whole class of problems occuring in the corresponding domain. Thus, domain experts must also be involved in the \emph{design} of an DSL, so that the resulting language is a useful vehicle for realisticly specifying problem solutions in the corresponding domain.

به طور کلی ، استفاده از DSL به این معنی است که حجم عظیمی از دانش دامنه از برنامه واقعی به کامپایلر و سیستم زمان اجرای زبان پیاده سازی منتقل می شود. بنابراین ، طراحی DSL حتی نیاز به درک بیشتر از حوزه مشکل دارد تا نوشتن یک برنامه واحد برای آن دامنه به یک زبان کلی. DSL فقط در صورتی معنی دارد که دارای انتزاعی مفید برای کل کلاس از مشکلات در دامنه مربوطه باشد. بنابراین ، کارشناسان دامنه همچنین باید در \ emph {طراحی} DSL شرکت داشته باشند ، به طوری که زبان نتیجه وسیله نقلیه مفیدی برای تعیین واقع بینانه راه حل های مسئله در دامنه مربوطه باشد.

The way how the semantics of a DSL is given determines for which purposes it can be used during software development. If the semantics is given informally, the DSL is good for documentation purposes in order to ease the communication between developers and customers, for example. In this case, the DSL is not appropriate for being used as specification language, because the lack of an unambiguous semantics disqualifies the DSL for that purpose. In this case using a formally defined general purpose specification language would be the better choice. If the semantics of the DSL is given formally using some mathematical notation, then the DSL can be used as specification language, because an unambiguous description of the semantics exists. In addition -- if tool support is provided -- the DSL program can be used as part of the implementation of the resulting product, at least in a prototypical version.

نحوه ارائه معناشناسی DSL تعیین می کند که در طی توسعه نرم افزار می تواند از چه اهدافی استفاده شود. اگر معنایی به طور غیررسمی داده شود ، DSL به منظور سهولت در برقراری ارتباط بین توسعه دهندگان و مشتریان ، برای اهداف مستند سازی مناسب است. در این حالت ، DSL برای استفاده به عنوان زبان خصوصی مناسب نیست ، زیرا عدم وجود معنایی بدون ابهام DSL را برای آن منظور رد می کند. در این حالت استفاده از یک زبان خصوصی برای اهداف عمومی به طور رسمی تعریف شده انتخاب بهتری خواهد بود. اگر معنایی DSL به طور رسمی با استفاده از برخی از نمادهای ریاضی داده شود ، می توان از DSL به عنوان زبان خصوصیات استفاده کرد ، زیرا توضیحی مبهم از معناشناسی وجود دارد. علاوه بر این - در صورت ارائه پشتیبانی ابزار - برنامه DSL می تواند به عنوان بخشی از اجرای محصول حاصل ، حداقل در یک نسخه نمونه اولیه استفاده شود.

In our approach, we try to meet these two seemingly contradicting requirements concerning the design of DSLs: We describe how the Montages method allows to specify the semantics of a DSL formally in an intelligible way using the mathematical notation of Abstract State Machines. Furthermore, DSL specifications are processed by a support tool (Gem-Mex) which automatically generates executable code purely based on the formal semantics descriptions.

در رویکرد ما ، ما سعی داریم این دو الزام به ظاهر متناقض در مورد طراحی DSL ها را برآورده کنیم: ما توصیف می کنیم که چگونه روش Montages اجازه می دهد تا معانی یک DSL را بطور رسمی و با روشی قابل فهم با استفاده از نماد ریاضی ماشینهای دولتی انتزاعی مشخص کند. علاوه بر این ، مشخصات DSL توسط یک ابزار پشتیبانی (Gem-Mex) پردازش می شود که بطور خودکار کد اجرایی را صرفاً بر اساس توصیف های معنایی رسمی تولید می کند.

# VISUAL FORMAL SEMANTICS DESCRIPTIONS: MONTAGES

Montages[KP97a,AKP97] constitute a specification formalism for describing all aspects of programming languages. Syntax, static analysis and semantics, and dynamic semantics are given in an unambiguous and coherent way by means of semi--visual descriptions. The static aspects of Montages resemble control and data flow graphs, and the overall specifications are similar in structure, length, and complexity to those found in common language manuals. Montages are designed to provide a mathematical basis for a number of activities from initial language design to prototyping.

مونتاژها [KP97a، AKP97] یک تشریفات ویژه برای توصیف کلیه جنبه های زبان های برنامه نویسی را تشکیل می دهند. نحو ، تجزیه و تحلیل استاتیک و معناشناسی و معنایی پویا با توصیفات نیمه بصری به صورت واضح و منسجم ارائه می شوند. جنبه های استاتیک Montages شبیه به نمودارهای کنترل و جریان داده ها است ، و مشخصات کلی از نظر ساختار ، طول و پیچیدگی با مواردی که در کتابچه های زبان مشترک یافت می شود مشابه هستند. مونتاژها به گونه ای طراحی شده اند که پایه ای ریاضی برای تعدادی از فعالیت ها از طراحی زبان اولیه گرفته تا نمونه سازی اولیه باشد.

The mathematical semantics of Montages is given with Abstract State Machines (formally called Evolving Algebras[Gurevich94b,ASMHomepage]. In short, ASMs are a state--based formalism in which a state is updated in discrete time steps. Unlike most state based systems, the state is given by an algebra, that is, a collection of functions and universes. The state transitions are given by rules that update functions point-wise and extend universes with new elements.

معانی ریاضیات مونتاژها با Machines Abstract State (رسماً با نام Evolving Algebras [Gurevich94b، ASMHomepage]) ارائه شده است. خلاصه اینکه ، ASM ها یک فرمالیسم مبتنی بر ایالت هستند که در آن حالت در مراحل زمانی گسسته به روز می شود برخلاف اکثر سیستم های مستقر در ایالت ، دولت توسط یک جبر داده می شود ، یعنی مجموعه ای از توابع و جهان ها. انتقال دولت توسط قوانینی انجام می گیرد که توابع را به صورت دقیق و به روز رسانی می کنند و جهان ها را با عناصر جدید گسترش می دهند.

ASMs have already been used to model the dynamic semantics of a number of programming languages, such as C [GurHug93], Occam [BoDuRo94], C++ [Wallace94], Oberon [Kutter97], and Java [BoeSch98a] to mention a few. At the risk of oversimplifying somewhat, one defines the *initial state* of the functions and specifies how they evolve by means of *transition rules*. The *initial state* is assumed to include the results of a static analysis. After this analysis the program's control and data flow is represented in the form of functions between parts of the program text. As usual the control flow functions specify the order in which statements are executed, and the data flow functions specify how values flow via variables through operations. The corresponding *transition rules* update the system state and let the control evolve through the control flow.

قبلاً از ASM برای مدل سازی معنای پویا تعدادی از زبان های برنامه نویسی ، مانند C [GurHug93] ، اوپام [BoDuRo94] ، C ++ [Wallace94] ، اوبرون [Kutter97] و جاوا [BoeSch98a] استفاده شده است تا به ذکر چند مورد بپردازیم. در معرض خطر بزرگنمایی برخی از موارد ، وضعیت اولیه توابع را تعریف می کند و نحوه تکامل آنها را با استفاده از قوانین انتقال مشخص می کند. فرض بر این است که حالت اولیه شامل نتایج آنالیز استاتیک است. پس از این تحلیل ، کنترل برنامه و جریان داده ها در قالب توابع بین بخش هایی از متن برنامه نمایش داده می شود. طبق معمول ، توابع جریان کنترلی ترتیب اجرای جملات را مشخص می کنند ، و توابع جریان داده چگونگی گردش مقادیر از طریق متغیرها را از طریق عملیات مشخص می کنند. قوانین انتقال مربوطه حالت سیستم را به روز می کنند و اجازه می دهند کنترل از طریق جریان کنترل تکامل یابد.

The existing case studies showed that it is possible to model with ASMs the dynamic semantics of realistic programming languages, but they have the disadvantage that they do not formalize the static aspects. Montages engineered the ASM's approach to programming language semantics showing how to model consistently not only the dynamic semantics, but the static analysis and semantics as well. In particular, Montages describe how to define intensionally the abstract syntax[[1]](#footnote-1), i.e. the control and data flow, starting from the concrete syntax. This mapping is provided by means of graphs which confer to the specification a great intelligibility.

مطالعات موردی موجود نشان داد که می توان با ASM ها معنایی پویای زبان های برنامه نویسی واقع گرایانه را مدل سازی کرد ، اما این ضرر را دارند که جنبه های استاتیک را به طور رسمی نمی رسند. مونتاژها رویکرد ASM را برای معناشناسی زبان برنامه نویسی طراحی کردند که نشان می دهد چگونه نه تنها معانی پویا ، بلکه از تحلیل استاتیک و معناشناسی نیز به طور مداوم مدل می شوند. به طور خاص ، مونتاژها چگونگی تعریف نحوی انتزاعی ، یعنی کنترل و جریان داده ها را با شروع از نحو بتن شرح می دهند. این نقشه برداری با استفاده از نمودارهایی انجام می شود که به مشخصات یک درک قابل فهم اعطا می کنند.

A language specification is given by a collection of Montages, which is hierarchically structured according to the corresponding context-free grammar. Each Montage is a “BNF-extension-to-semantics”, that is a self contained capsule in which all the properties of a given construct are formally defined.

مشخصات زبان توسط مجموعه ای از Montages ارائه شده است ، که بصورت سلسله مراتبی مطابق دستور زبان بدون متن مربوطه ساخته می شود. هر مونتاژ یک "BNF- پسوند به معناشناسی" است ، یعنی یک کپسول خود حاوی آن است که در آن تمام خصوصیات یک ساختار خاص به طور رسمی تعریف می شود.

In Fig. 1 the Montage specification of a “While” construct is presented. The topmost part in the working area is the production rule defining the context--free syntax. Below is the local flow, i.e. the graphical definition of the mapping between the concrete and the abstract syntax of the while statement. It describes the portion of the parse tree associated with the while statement and enriches it with some additional information, that is the control and data flow edges. In Montages, the inner nodes of the parse tree are represented with boxes and leaves with ovals, whereas the node corresponding to the while statement is denoted by the *self* oval node. Moreover, nested boxes can be used to represent nodes on lover levels of the parse trees. The solid and dotted arrows denote the data and control flow, respectively. Control flow arrows may be labeled by means of firing conditions, i.e. predicates which determine through which edges the control must flow. For instance, the predicate *guard.value* indicates that the control is passed from the while node (denoted by self) to the sequence of statements *StatementSequence* whenever the value of the expression *Expression*, which is retrieved by means of the data flow arrow *guard*, is evaluated to *true*. The control flow arrows I (initial) and T (terminal) are special arrows which serve to plug together the local flow-information to the global one. In other word, the abstract syntax (graph) or global flow of a program is obtained from its parse tree by inductively decorating it with the information defined in the local flows of the statements occourring in the program itself.

در شکل 1 مشخصات Montage از ساختار "در حالی که" ارائه شده است. مهمترین بخش در منطقه کار ، قانون تولید است که نحو عاری از متن را تعریف می کند. در زیر جریان محلی ، یعنی تعریف گرافیکی نقشه برداری بین بتن و نحو انتزاعی جمله در حالی که وجود دارد ، آمده است. این بخش از درخت پارگی مرتبط با دستور موقع را توصیف می کند و آن را با برخی از اطلاعات اضافی ، یعنی لبه های کنترل و جریان داده غنی می کند. در مونتاژها ، گره های درونی درخت جعفری با جعبه ها و برگ هایی با تخمک نشان داده شده است ، در حالی که گره مربوط به عبارت در حالی که گره خود بیضی شکل دارد مشخص می شود. علاوه بر این ، از جعبه های تو در تو می توان برای نشان دادن گره های سطح عاشق درختان پارسی استفاده کرد. فلش های جامد و نقطه به ترتیب جریان داده و کنترل را نشان می دهند. فلش های جریان کنترل با استفاده از شرایط شلیک می توانند برچسب گذاری شوند ، یعنی گزاره هایی که تعیین می کنند کنترل از چه لبه هایی باید جریان داشته باشند. به عنوان مثال ، predate protection.value نشان می دهد که این کنترل از گره در حالی که (توسط خودش مشخص می شود) به توالی بیانیه ها اعطا می شود StatementSequence هر زمان که مقدار عبارت Expression ، که با استفاده از گارد فلش جریان داده بازیابی می شود ، است. ارزیابی به درست است. فلش های جریان کنترل I (اولیه) و T (ترمینال) فلش های ویژه ای هستند که برای وصل کردن اطلاعات جریان محلی به اطلاعات جهانی ، به کار می روند. به عبارت دیگر ، نحو انتزاعی (نمودار) یا جریان جهانی یک برنامه از درخت تجزیه شده آن با تزریق استقراء آن با اطلاعات تعریف شده در جریان های محلی عبارات در حال اجرا در برنامه ، به دست می آید.

|  |
| --- |
| While ::= “while” Expression “do”  StatementSequence  “end” |
|  |
| **condition** *S-Expression.staticType = Boolean* |

Figure 1: The While Montage

The third part of the while Montage contains the static semantics, i.e. the type-checking condition which requires that the expressions in the while statement must be booleans. In general, the designer may make use of full firstorder logic to express context sensitive constraints. In this example, the part containing explicit dynamic semantics rules is missing. This is usual for most of the control statements, but there are also cases in which an additional transition rule is needed to define the complete dynamic semantics as illustrate, for instance, in Fig 7 and Fig 8.

The semantics of a Montages specification of a programming language is therefore the following, given a program

بخش سوم این در حالی است که Montage شامل معانی ایستایی است ، یعنی شرط بررسی نوع که مستلزم این است که عبارات موجود در عبارت هنگام باید بول باشند. به طور کلی ، طراح ممکن است از منطق کامل درجه اول برای بیان محدودیت های حساس متن استفاده کند. در این مثال ، بخشی که حاوی قوانین معنایی پویای صریح است ، از دست نمی رود. این برای اکثر گفته های کنترل معمول است ، اما مواردی نیز وجود دارد که یک قانون انتقال اضافی برای تعریف معنایی کامل پویا به عنوان مثال در شکل 7 و شکل 8 لازم است.  
معنای معیار مشخصات Montages از یک زبان برنامه نویسی به شرح زیر است ، با توجه به برنامه

* the context-free grammar obtained by collecting all the EBNF-rules in each Montage defines the concrete syntax;
* the local flow, i.e. the visual part of each Montage, defines how to provide the parse tree with all the information which are needed to define in turn the dynamic semantics, in particular the control and data flow, respectively, of that given construct;
* the condition part is a first-order logic predicate which is evaluated while traversing the parse tree, i.e. for each internal node the corresponding predicate is checked. In general, different traversal strategies can be specified, predicates can be checked before or after the analysis of subtrees, or maybe the designer may prefer to define several passes, e.g. one additional pass to check declarations before the other fragments. A more leisured and detailed discussion can be found in [KP97a].
* the dynamic semantics part is a transition rules which is fired whenever the control reach that given construct, i.e. the *self* node. For instance, in Fig. 8 whenever the control reaches the node denoted by “self” (which denotes to the node of the construct *EventConj*) the the following rule is enabled

*fire\_condition := checkSensor(sensor.Name,value.Name)*

which updates locally the value of the function *fire\_condition*.

- گرامر عاری از متن که با جمع آوری کلیه قوانین EBNF در هر مونتاژ به دست می آید ، نحو بتن را تعریف می کند.  
- جریان محلی ، یعنی بخش بصری هر مونتاژ ، چگونگی تهیه درخت تجزیه را با تمام اطلاعات موردنیاز برای تعریف به معنای معناشناختی پویا ، به ویژه کنترل و جریان داده ، به ترتیب ، به ترتیب ، از ساختار داده شده تعریف می کند.  
- قسمت شرط یک گزاره منطق مرتبه اول است که هنگام پیمایش درخت پارس ارزیابی می شود ، یعنی برای هر گره داخلی محمول مربوطه بررسی می شود. به طور کلی ، استراتژی های مختلف مسافرتی را می توان مشخص کرد ، پیش بینی ها را می توان قبل یا بعد از تجزیه و تحلیل زیر درختان بررسی کرد ، یا شاید طراح ممکن است ترجیح دهد چندین پاس را تعیین کند ، به عنوان مثال. یک پاس اضافی برای بررسی اظهارات قبل از قطعات دیگر. بحث بیشتر و تفصیلی بیشتری را می توان در [KP97a] یافت.  
- بخش معناشناختی پویا یک قانون گذار است که هر زمان کنترل به آن ساختار داده شده ، یعنی گره خود برسد ، اخراج می شود. به عنوان مثال ، در شکل 8 هر زمان که کنترل به گره مشخص شده توسط "خود" برسد (که به گره EventConj اشاره دارد) قاعده زیر فعال شده است

The logical and formal aspects of Montages are not supposed to be illustrated deeply here, a more detailed discussion can be found in [KP97a]. Nevertheless, it should be stressed how Montages represents a rigorous and formal instrument providing a mathematical framework which can be used by the designer to record unambiguously decisions about a particular language and obtaining new insight into the nature of the language developing description of it. However, our interests focussed more on the pragmatic qualities of the formalism and its effectiveness, rather than its mathematical properties.

جنبه های منطقی و رسمی مونتاژها قرار نیست در اینجا به تصویر کشیده شوند ، بحث مفصل تری در [KP97a] یافت می شود. با این وجود ، باید تأکید کرد که Montages چگونه یک ابزار دقیق و رسمی را ارائه می دهد که یک چارچوب ریاضی را ارائه می دهد و می تواند توسط طراح مورد استفاده قرار گیرد تا تصمیمات نامشخصی راجع به یک زبان خاص ثبت کند و بینش جدیدی در مورد ماهیت زبان در حال توصیف آن ارائه دهد. با این حال ، منافع ما بیشتر بر ویژگیهای عملی واقع گرایی و اثربخشی آن متمرکز شده اند تا خصوصیات ریاضی آن.

# RUNNING EXAMPLE: A DSL FOR THE SOFTWARE ARCHITECTURE OF VENDING MACHINES

In this section, we will illustrate the design of a DSL using Montages and Gem-Mex. As an example, we want to design a language for specifying the software architecture of simple, state-based vending machines, we will call it „VML“ for „vending machine language“ in the sequel. With this specific DSL one should be able to describe the constituents and the behavior of the control software of these machines. Examples are beverage and food dispensers, automated teller machines etc. . In order to describe these kind of machines, VML must contain constructs to describe

در این بخش ، طراحی یک DSL را با استفاده از Montages و Gem-Mex نشان خواهیم داد. به عنوان نمونه ، ما می خواهیم یک زبان را برای مشخص کردن معماری نرم افزارهای ماشین های فروش ساده و مستقر در کشور طراحی کنیم ، در ادامه آن را "VML" برای "زبان ماشین فروشی" می نامیم. با استفاده از این DSL خاص باید بتوان ترکیبات و رفتار نرم افزار کنترل این دستگاه ها را توصیف کرد. نمونه هایی از مواد پخش کننده نوشیدنی و مواد غذایی ، دستگاه های اتوماتیک خودکار و غیره. به منظور توصیف این نوع ماشین ها ، VML باید شامل سازه هایی برای توصیف باشد

* input and output devices, like sensors or displays together with their possible values;
* actions that can be triggered by the control software; - states together with permissible events for each state;
* state transitions.

In order to get an idea of the example domain-specific language VML, Figure 2 contains an example description of a vending machine dispensing beverages.

- دستگاه های ورودی و خروجی ، مانند سنسورها یا نمایشگرها با مقادیر احتمالی آنها.  
- اقداماتی که توسط نرم افزار کنترل انجام می شود. - همراه با رویدادهای مجاز برای هر ایالت؛  
- انتقال دولت.  
به منظور به دست آوردن ایده ای از مثال مربوط به زبان اختصاصی دامنه VML ، شکل 2 شامل نمونه ای از نوشیدنی های پخش کننده دستگاه توزیع کننده است.

|  |
| --- |
| BEGIN SimpleBeverageDispenser  DISPLAY BevLed: {on,off};  SENSOR MoneySlot:{active},  BevButton:{pressed},  CancelButton:{pressed},  BevEmptySensor:{empty};  ACTION DispenseBeverage();  ACTION ReturnMoney(),GetMoney();  FORWARD WaitForMoney; FORWARD Initial;  STATE Reset IS  BevEmptySensor SIGNALS empty ->  BevLed <- off;  GOTOSTATE Initial;  BevEmtpySensor SIGNALS nothing ->  BevLed <- on;  GOTOSTATE Initial;  ENDSTATE Reset;  STATE Initial IS  BevButton SIGNALS pressed AND  BevEmptySensor SIGNALS nothing ->  GOTOSTATE WaitForMoney;  ENDSTATE Initial;  STATE WaitForMoney IS  MoneySlot SIGNALS active ->  GetMoney();  DispenseBeverage();  GOTOSTATE Reset;  CancelButton SIGNALS pressed ->  GOTOSTATE Initial;  ENDSTATE WaitForMoney; END |

Figure 2: A “VML”-specification of the architecture of a beverage vending machine

The vending machine specified is able to dispense one kind of beverage. The customer presses the “BevButton” when the system is in its initial state. After that, the machine wait for an “active” signal from the “MoneySlot” sensor or the user may press the “CancelButton”. In the “Reset” state the systen checks whether the “BevEmptySensor” signals that there are no more beverages in the vending machine and sets the value of the display components accordingly. For a more realistic modeling other components may be introduced.

دستگاه فروش مشخص شده قادر به نوشیدن یک نوع نوشیدنی است. هنگامی که سیستم در وضعیت اولیه خود است ، مشتری "BevButton" را فشار می دهد. پس از آن ، دستگاه منتظر یک سیگنال "فعال" از سنسور "MoneySlot" یا کاربر ممکن است "CancelButton" را فشار دهد. در حالت "تنظیم مجدد" سیستن بررسی می کند که آیا "BevEmptySensor" سیگنال می دهد که دیگر هیچ نوشیدنی در دستگاه فروش وجود ندارد و مقدار اجزای صفحه نمایش را بر این اساس تنظیم می کند. برای یک مدل سازی واقع بینانه سایر مؤلفه ها ممکن است معرفی شوند

## Specifying VML using Montages

The Montages specification of VML will be sketched in the following. Note, that the specification parts shown here are an excerpt from a running version generated within the Gem-Mex support environment.

We will first have a look at the definition of the general structure of a VML program. This is given by the Montage for the nonterminal symbol “VML” as shown below.

مشخصات Montages از VML در شکل زیر ترسیم می شود. توجه داشته باشید که بخش مشخصات نشان داده شده در اینجا گزیده ای از نسخه در حال اجرا است که در محیط پشتیبانی Gem-Mex تولید شده است.  
در ابتدا نگاهی خواهیم داشت به تعریف ساختار کلی یک برنامه VML. این توسط Montage برای نماد غیر انتهایی "VML" آورده شده است همانطور که در زیر آمده است.

|  |
| --- |
| VML ::= “BEGIN” Ident  { Declaration “;” }  { ForwardState “;” } { StateDescr “;” }  “END”  Declaration = Display | Sensor | Action |
|  |
| **condition** *(forall s in StateTable*  *StateTable(s) ≠ FORWARD)* |

The grammar rule define the syntax of an VML program: after the declaration of display, sensor, and action items, the states can be specified. A “FORWARD“ construct is introduced, in order to simplify the static analysis part. The static condition requires, that each entry in the „StateTable“ points to a actual node in the abstract syntax tree and doesn`t contain the special constant “FORWARD“. In the static analysis phase, all descendants of the VML node are visited, e.g. the lists of Declaration, ForwardState, and StateDescr, then the condition is checked. After the static analysis phase, the dynamic semantics can be executed by passing control to the initial node of the start symbol, e.g. the first StatDescr, as graphically specified by the I-arrow in the VML Montage.

قانون دستور زبان نحو یک برنامه VML را تعریف می کند: پس از اعلام نمایش ، سنسور و موارد عملی ، می توان حالت ها را مشخص کرد. یک ساختار "FORWARD" به منظور ساده سازی قسمت آنالیز استاتیک معرفی شده است. شرط استاتیک ایجاب می کند که هر ورودی در "StateTable" به گره واقعی در درخت نحو انتزاعی اشاره کند و حاوی ثابت مخصوص "FORWARD" نباشد. در مرحله آنالیز استاتیک ، از همه فرزندان گره VML بازدید می شود ، به عنوان مثال لیست های اعلامیه ها ، ForwardState و StateDescr ، سپس شرط بررسی می شود. پس از مرحله تحلیل استاتیک ، معناشناسی پویا را می توان با انتقال کنترل به گره اولیه نماد شروع ، به عنوان مثال اجرا کرد. اولین StatDescr ، همانطور که از طریق فلش I در مونتاژ VML مشخص شده است.

The following figure contains the Montage for the nonterminal symbol “SensorDecl“. In the static part, the name of the sensor item together with the list of possible values for the sensor are stored into global tables named “SensorTable“ and “SensorValue“. These tables are declared at another place of the specification as one- resp. twoary dynamic ASM functions.

شکل زیر حاوی Montage برای نماد غیر انتهایی "SensorDecl" است. در قسمت استاتیک ، نام مورد حسگر به همراه لیست مقادیر احتمالی سنسور در جداول جهانی به نام های "SensorTable" و "SensorValue" ذخیره می شود. این جداول در یک مکان دیگر از مشخصات به عنوان یک اعلام شده است. توابع ASM پویا دوتایی.

|  |
| --- |
| SensorDecl ::= SensorIdent “:”  “{“ ValueIdent  { “,” ValueIdent } “}”  SensorIdent = Ident  ValueIdent = Ident |
| *let dname = S-SensorIdent.Name in SensorTable(dname) := self vary v over list S-ValueIdent SensorValue(dname,v.Name) := true endvary*  *SensorValue(dname,“nothing”) := true endlet* |
| **condition** *UniqueNames(S-ValueIdent)* |

The condition states that the values specified must be pairwise disjoint, where “UniqueNames“ is a macro defined as

macro UniqueNames(IDENTLIST) is (forall v1 in list IDENTLIST: (forall v2 in list IDENTLIST:

(v1\_ind=v2\_ind) or (v1.Name!=v2.Name)

)) endmacro

where “*x*\_ind“ refers to the corresponding list index of the iteration variable *x*. The declaration of sensor values is specified accordingly.

Montage “StateDescr” shows the specification of the semantics of the “STATE“ construct. In the static analysis, the “StateTable“ is updated to contain the root node of the state in the syntax tree (“StateTable(S1-Ident.Name) := self“). This node serves as entry point when a “GOTOSTATE“ construct is executed. Reaching the end of the inner constructs of a state means, that none of the specified events have occured. In this case, the state is not changed, the control flow returns to the initial task of the same state, which is specified by the dotted arrow in the figure below.

Montage "StateDescr" مشخصات معنایی ساختار "STATE" را نشان می دهد. در تجزیه و تحلیل استاتیک ، "StateTable" به روز شده است که شامل گره ریشه ای از حالت در درخت نحو ("StateTable (S1-Ident.Name): = خود") باشد. این گره به عنوان نقطه ورود هنگام اجرای یک طرح "GOTOSTATE" عمل می کند. رسیدن به پایان سازه های داخلی یک حالت بدان معنی است که هیچ یک از وقایع مشخص شده رخ نداده است. در این حالت ، حالت تغییر نمی یابد ، جریان کنترل به کار اصلی همان حالت باز می گردد ، که توسط پیکان نقطه ای در شکل زیر مشخص شده است.

|  |
| --- |
| StateDescr ::= “STATE” Ident “IS”  { Reaction “;” }  “ENDSTATE” Ident |
| StateTable(S1-Ident.Name) := self |
| **condition** *S1-Ident.Name = S2.Ident.Name* |

The following figure shows the “Reaction“-Montage. This construct is used in VML to specify the reaction of the system to an event described by the “EventDescr“. If this is the case, the actions specfied in the “Exec“ list are executed and thereafter the system changes its state to the one specified after the “GOTOSTATE“ keyword.

شکل زیر "عکس العمل" -Montage را نشان می دهد. این سازه در VML برای مشخص کردن واکنش سیستم به رویدادی که توسط "EventDescr" شرح داده شده است ، استفاده می شود. در این صورت ، اقدامات مشخص شده در لیست "Exec" اجرا می شوند و پس از آن سیستم حالت خود را به حالت مشخص شده پس از کلید واژه "GOTOSTATE" تغییر می دهد.

|  |
| --- |
|  |
|  |
|  |

The control flow of this construct is completely specified grapohically: if the fire condition of the event is fulfilled, then the “Exec” list is executed, otherwise this branch is skipped and the construct is left. After the actions of the “Exec” list have been processed, the control flow is directed to the next state the entry point of which is stored in the global “StateTable”. Note, that this can be done directly without checking whether a corresponding entry exists, because this has already been done by the static condition.

The following Montage contains the semantics of the “EventDescr” construct. The static condition checks whether the sensor value is defined for the given sensor item. In this Montage an additional dynamic semantics rule is specified, which updates the attribute “fire\_condition” according to the actual values of the sensors.[[2]](#footnote-2)

جریان کنترل این سازه کاملاً بصورت واضح مشخص شده است: اگر شرایط حادثه برآورده شود ، لیست "Exec" اجرا می شود ، در غیر این صورت این شاخه پرش می شود و سازه باقی می ماند. پس از پردازش اقدامات لیست "Exec" ، جریان کنترل به حالت بعدی هدایت می شود که نقطه ورود آن در "StateTable" جهانی ذخیره می شود. توجه داشته باشید که این کار می تواند بطور مستقیم و بدون بررسی وجود ورودی مربوطه انجام شود ، زیرا این کار قبلاً توسط شرایط استاتیک انجام شده است.  
مونتاژ زیر شامل معناشناسی ساختار "EventDescr" است. وضعیت استاتیک بررسی می کند که آیا مقدار سنسور برای مورد سنسور معین تعریف شده است یا خیر. در این مونتاژ یک قانون معناشناختی پویا اضافی مشخص شده است ، که ویژگی "fire\_condition" را با توجه به مقادیر واقعی سنسورها به روز می کند.

Finally, in the Montage “EventConj” an example is given how lazy evaluation of a conjunction can be expressed. The second part of the conjunction is only reached, if the fire condition of the first part is fulfilled, otherwise the construct is left via the self node. The actual value of the fire condition of the combined construct is given textually.

سرانجام ، در Montage "EventConj" مثالی آورده شده است که چگونه می توان ارزیابی تنبل یک رابطه را بیان کرد. قسمت دوم اتصال فقط در صورت رسیدن شرایط آتش قسمت اول ، به دست می آید ، در غیر این صورت سازه از طریق گره خود باقی می ماند. مقدار واقعی وضعیت آتش سازه ترکیبی به صورت متنی ارائه می شود.

|  |
| --- |
| EventConj ::= EventDescr “AND” EventExpr |
|  |
| *fire\_condition := (if left.fire\_condition then right.fire\_condition else false)* |

The Montages presented in this section together with a handful more are sufficient to formally specify the semantics of the DSL. The specification can be processed by the Gem-Mex system, which also automatically generates a visual debugging and animation interface for VML.

مونتاژهای ارائه شده در این بخش به همراه تعداد معدودی دیگر برای مشخص کردن رسمی معانی DSL کافی است. مشخصات را می توان با سیستم Gem-Mex پردازش کرد ، که به طور خودکار یک رابط اشکال زدایی بصری و انیمیشن را برای VML تولید می کند.

# GEMMEX: TOOL SUPPORT FOR MONTAGES

The development environment for Montages is given by the Gem-Mex tool. It is a complex system which assists the designer in a number of activities related with the language life cycle, from early design to routine programmer usage.

محیط توسعه برای Montages توسط ابزار Gem-Mex ارائه شده است. این سیستم پیچیده ای است که از طراحان در تعدادی از فعالیت های مرتبط با چرخه عمر زبان ، از طراحی اولیه گرفته تا استفاده روزمره برنامه نویس ، به طراح کمک می کند.

It consists of a number of interconnected components:

* the Graphical Editor for Montages (Gem) is a sophisticated graphical editor in which Montages can be entered; furthermore high-quality documentation can be generated, see the Montages of Section 5 which have been generated this way;
* the Montages executable generator (Mex) which automatically generates correct and efficient implementations of the language;
* the generic animation and debugger tool visualizes the static and dynamical behavior of the specified language at a symbolic level; source programs written in the specified language can be animated and inspected in a visual environment.
* - ویرایشگر گرافیکی برای Montages (Gem) یک ویرایشگر گرافیکی پیشرفته است که در آن می توانید Montages وارد کنید. علاوه بر این می توان اسناد با کیفیت بالا تولید کرد ، به مونتاژهای بخش 5 که از این طریق تولید شده اند ، مراجعه کنید.  
  - ژنراتور اجرایی Montages (Mex) که بطور خودکار پیاده سازی صحیح و کارآمد زبان را تولید می کند.  
  - ابزار انیمیشن و اشکال زدایی عمومی رفتار استاتیک و دینامیکی زبان مشخص شده را در سطح نمادین تجسم می کند. برنامه های منبع که به زبان مشخص نوشته شده اند در یک محیط تصویری قابل انیمیشن و بازرسی هستند.

A broad range of professionals may find interesting and convenient to use Gem-Mex. The whole development of a DSL can be supported with an effective impact on the productivity and robustness of the design. The designer can enter the specification, browse it and especially maintain it. Specifications may evolve in time even in a nonmonotonic way since modifications can be localized within very neat boundaries. By doing so, different experimentation can take place with different versions of the syntax and semantics of the specified language in a very short time.

طیف گسترده ای از حرفه ای ها ممکن است جالب و مناسب برای استفاده از Gem-Mex را پیدا کنند. توسعه کل DSL می تواند با تأثیر مؤثر بر بهره وری و استحکام طراحی پشتیبانی شود. طراح می تواند مشخصات را وارد کند ، آنرا مرور کرده و به خصوص آن را حفظ کند. مشخصات ممکن است در زمان حتی به صورت غیر حرکتی تکامل یابد زیرا تغییرات را می توان در مرزهای بسیار مرتبا بومی سازی کرد. با این کار می توانید آزمایش های مختلفی را با نسخه های مختلف نحو و معانی زبان مشخص شده در مدت زمان بسیار کمی انجام دهید.

Besides the pure editing functionality, Gem can be used to generate documents suitable for specification presentation. Experience suggests how lack in documentation is a dangerous bottleneck for the consistency and coherence of a project. Both, paper and online presentation of the language specification are automatically generated by Gem:

علاوه بر قابلیت ویرایش خالص ، از Gem می توان برای تولید اسناد مناسب برای ارائه مشخصات استفاده کرد. تجربه نشان می دهد که عدم وجود مستندات ، تنگنای خطرناکی برای انسجام و انسجام یک پروژه است. هر دو ، ارائه مقاله و مشخصات آنلاین زبان به طور خودکار توسط Gem تولید می شود:

LaTeX documents illustrate the Montages and the grammar; such documents are easily customizable for the non-specialist user;

* HTML versions of the language specification allows to browse the specification and retrieve pieces of specification.

Moreover, intelligibility is enhanced by means of “literate specification” techniques directly supported by Gem. Formal parts of the specification can be substituted with textual elements by means of a “literate programming” tool integrated in the system. “Literate specification” means that the Montages text fields may contain references to other parts of the formalization specified outside of the Montages modules. Thus, the readability and comprehension of a Montages specification results very much similar to those of language manuals.

اسناد LaTeX مونتاژها و دستور زبان را نشان می دهد. چنین اسنادی برای کاربر غیر متخصص به راحتی قابل تنظیم است.  
- نسخه های HTML از مشخصات زبان اجازه می دهد تا فهرست مشخصات و بازیابی قطعات از مشخصات.  
علاوه بر این ، قابل فهم با استفاده از تکنیک های "مشخصات سواد آمیز" به طور مستقیم توسط Gem پشتیبانی می شود. با استفاده از ابزار «برنامه نویسی با سواد» که در سیستم یکپارچه شده است ، قطعات اصلی مشخصات را می توان با عناصر متنی جایگزین کرد. "مشخصات با سواد" به این معنی است که قسمتهای متنی Montages ممکن است شامل بخشهایی از رسمی شدن مشخص شده در خارج از ماژولهای Montages باشد. بنابراین ، خوانایی و درک مشخصات Montages بسیار شبیه به کتابچه های راهنمای زبان است.

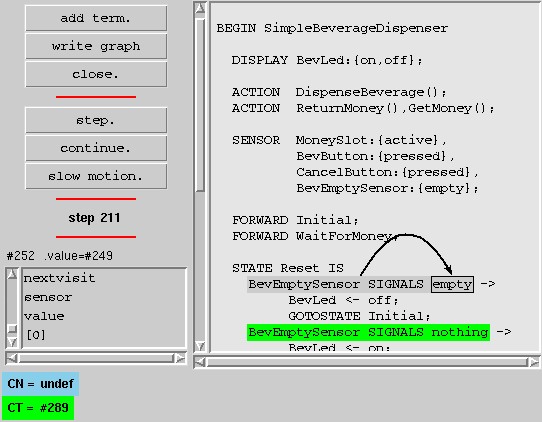
## Generation of Visual Animation Tools

Gem-Mex generates an executable version of the Montages specification by simply fedig the formal ASM-semantics of Montages to Aslan [Anl98]. Aslan is an efficient and flexible implementation of the ASM approach. Besides a plain textual version of the DSL interpreter, Gem-Mex also generates a visual debugging and animation tool for the specified DSL. The generated graphical user interface allows the language designer to browse the data structures of the formalisation and to trace the program run graphically.

Gem-Mex یک نسخه قابل اجرا از مشخصات Montages را با استفاده از اصول رسمی ASM- معانی رسمی Montages به Aslan تولید می کند [Anl98]. Aslan یک اجرای کارآمد و انعطاف پذیر از روش ASM است. علاوه بر یک نسخه متنی ساده از مفسر DSL ، Gem-Mex همچنین یک ابزار اشکال زدایی بصری و انیمیشن را برای DSL مشخص تولید می کند. رابط کاربری گرافیکی ایجاد شده به طراح زبان اجازه می دهد ساختار داده های رسمی را مرور کند و برنامه را بصورت گرافیکی ردیابی کند.

The following figure shows the “view source” subwindow of the generated graphical user interface for VML. The movement from one task to the next one is animated by an arrow that starts from the previously active node and ends in the currently active node. In addition, there exist “textual” subwindows, where the values of globally defined functions can be inspected.

شکل زیر زیر نمایش "منبع نمایش" رابط کاربری گرافیکی ایجاد شده برای VML را نشان می دهد. حرکت از یک کار به کار دیگر توسط یک فلش متحرک می شود که از گره فعال قبلی شروع می شود و به گره فعال فعلی پایان می یابد. علاوه بر این ، زیرنویس های "متنی" وجود دارد ، که در آن می توان مقادیر توابع تعریف شده در سطح جهان را بررسی کرد.



# CONCLUSION

We have introduced a tool support environment that can be used for designing DSL to be used in software architecture. With this framework, one is able to intelligibly specify the syntax and semantics of the DSL based on a

formal mathematical approach, Abstract State Machines. We pointed out, that domain experts should also be involved into the software architecture phase which is much easier, if DSLs are used in these phases rather than general purpose software architecture notations. On the other hand, these DSL must also be well-designed, so that domain knowledge is also needed during the design of an DSL. Our approach offers a possibility for a broader range of people to also participate in the design of an DSL, especially domain experts. Using a simple DSL named VML (vending machine language) we illustrated the design of DSLs. With Montages/Gem-Mex it is possible to specify the semantics of a DSL formally and in an intelligible way; a task that is very important, if DSLs should be used during software architecture.

ما یک محیط پشتیبانی ابزار معرفی کرده ایم که می تواند برای طراحی DSL در معماری نرم افزار مورد استفاده قرار گیرد. با استفاده از این چارچوب ، فرد می تواند نحو و معانی معانی DSL را براساس a مشخص کند  
رویکرد رسمی ریاضی ، دستگاه های دولتی چکیده. ما خاطرنشان كرديم كه در صورت استفاده از DSL ها در اين مراحل به جاي اينكه از كلمات معماري نرم افزارهاي هدفي استفاده شود ، متخصصان حوزه نيز بايد وارد فاز معماري نرم افزار شوند. از طرف دیگر ، این DSL نیز باید به خوبی طراحی شده باشد ، به طوری که دانش دامنه نیز در حین طراحی DSL مورد نیاز است. رویکرد ما این امکان را برای طیف گسترده تری از مردم فراهم می کند که می توانند در طراحی DSL ، به ویژه کارشناسان حوزه نیز شرکت کنند. با استفاده از DSL ساده به نام VML (زبان ماشین فروش) ما طراحی DSL ها را نشان دادیم. با استفاده از Montages / Gem-Mex می توان معانی یک DSL را بطور رسمی و به روشی قابل فهم مشخص کرد. اگر از DSL ها در معماری نرم افزار استفاده شود ، کاری که بسیار مهم است.

1. Since the Montages framework is state-based and not functional, by *abstract syntax* it denotes an abstract syntax graph rather than a tree. [↑](#footnote-ref-1)
2. In the implementation “checkSensor” is an external function returning a random boolean value.

   [↑](#footnote-ref-2)