# Archimedes – o CAD Aberto
# Uma aplicação para desenho técnico baseada na plataforma do Eclipse

**Hugo Corbucci**[1], **Mariana V. Bravo**[1]

[1]Instituto de Matemática e Estatística – Universidade de São Paulo (USP)
R. do Matão, 1010 – Cidade Universitária – CEP 05508-090 – São Paulo – SP – Brasil

{corbucci,marivb}@ime.usp.br

***Abstract.*** *Archimedes is a free software for computer aided design (CAD) focused on architecture. The project has recently moved to Eclipse's Rich Client Platform. In this article, the result of such migration will be presented, as well as the extension points that allow new funcionality to be added to the software.*

***Resumo.*** *Archimedes é um programa livre para desenho técnico desenvolvido para arquitetos. O projeto mudou recentemente para a plataforma de aplicações (Rich Client Platform) do Eclipse. Neste artigo serão apresentados os resultados da migração, bem como os pontos de extensão que permitem que novas funcionalidades sejam agregadas ao programa.*

## 1. Introdução

O projeto Archimedes ([Archimedes Team 2005]) nasceu a partir de um desafio feito por Jon "Maddog" Hall no FISL 6.0 (6º Fórum Internacional de Software Livre) para prover uma solução aberta para profissionais usuários do AutoCAD. O objetivo inicial do projeto é um software livre multi-plataforma para ser usado principalmente por arquitetos. Para atingi-lo, além de desenvolvedores, participam da equipe estudantes e profissionais de arquitetura. Essas pessoas colaboram com o projeto definindo o que deve ser feito a cada etapa do desenvolvimento, segundo a metodologia de Programação Extrema ([Beck 1999], [Beck and Andres 2004]).

A meta para a versão 1.0 é de um *software* de cunho educacional cujo comportamento seja similar ao de uma prancheta de desenho. Desde março de 2006, a equipe de desenvolvedores está seguindo este rumo assim como definindo uma segunda versão do programa que, desta vez, terá como foco o ambiente profissional.

Após quase um ano de trabalho, o projeto conseguiu atingir um estado razoável de desenvolvimento, porém o programa claramente apresentava algumas carências. Novos usuários tinham dificuldade em utilizá-lo, tanto pela pobre estrutura dos menus quanto pela ausência de ajuda embutida no programa. Além disso, como era esperado que o software permitisse formas de desenho e trabalho diferentes das existentes, tanto desenvolvedores quanto usuários desejavam que o programa fosse mais flexível. Com isso, a equipe decidiu migrar o Archimedes para o RCP (*Rich Client Platform*) do projeto Eclipse ([Eclipse Foundation 2001]), por ser amplamente adotado pela comunidade de Software Livre e por disponibilizar diversas das funcionalidades desejadas.

Na próxima seção, será apresentado o sistema inicial do Archimedes, que deveria ser mantido; e na seguinte a arquitetura resultante da migração.

## 2. Uma visão geral do sistema

O programa funciona de forma semelhante a um *prompt* de comandos com uma área de desenho como pode ser visto na figura 1. Comandos são entrados pelo teclado, e os parâmetros podem ser passados tanto em texto quanto pelo *mouse*. Dessa forma, os profissionais podem mudar o contexto de seu trabalho rapidamente sem a necessidade de procurar atalhos em menus grandes e complexos. A seguir será descrita a camada de interface, responsável por cuidar da interação com o usuário.

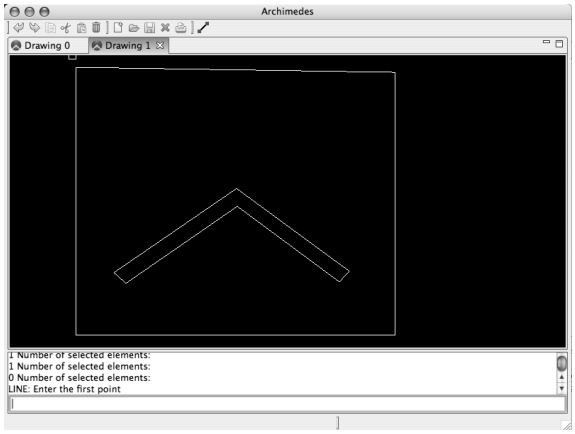


**Figure 1. A janela principal do Archimedes no MacOS X**

### 2.1. Camada de interface

Para lidar com os dois tipos de entrada de maneira uniforme, as informações recebidas pelo *mouse* são transformadas em texto. Esses dados, assim como os obtidos pelo teclado, são enviados para um controlador de entradas (`InputController`).

O objetivo dessas informações é ativar um comando, passar parâmetros para ele até que ele termine e então realizar a ação que o comando gera. Do ponto de vista do sistema, o que é executado é a ação, que implementa o padrão `Command` ([Gamma et al. 1995]), de onde vem seu nome. Uma ação é gerada por uma fábrica de `Command`, que é chamada de `CommandFactory`. Por exemplo, para criar uma linha, primeiro é preciso ativar a fábrica de linhas, depois são definidos dois pontos como parâmetros e finalmente é feita a ação de criar a linha.

O processamento dessas informações por parte do `InputController` depende do seu estado. Existem duas principais possibilidades:

- Não existe nenhuma fábrica ativa.
  Nesse caso, o `InputController` pede ao registro de comandos (`CommandParser`) que lhe envie a fábrica associada entrada do usuário, para que esta seja ativada.
- Existe uma fábrica ativa.
  Se for assim, essa fábrica é quem deve lidar com as entradas do usuário. Como podem existir muitos parâmetros para criação de um comando e muitas fábricas recebem parâmetros semelhantes, a tradução das entradas em texto para objetos do sistema é realizada por um objeto chamado `Parser`. A cada passo, a fábrica informa qual `Parser` deverá cuidar da entrada e espera que ele devolva um objeto

específico. Dessa forma, as fábricas podem trabalhar com objetos de mais alto nível e reaproveitar o trabalho de processamento realizado pelos *parsers*.

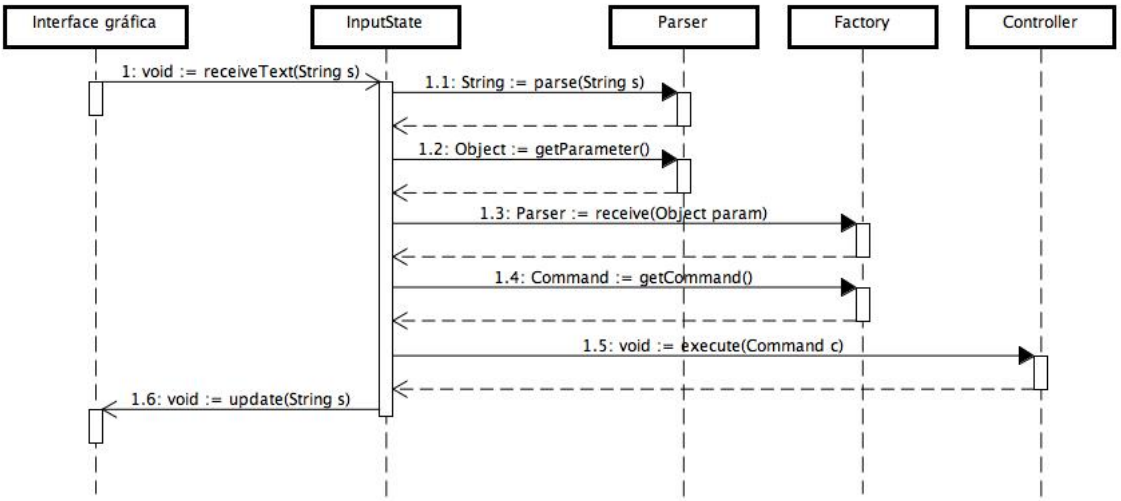


**Figure 2. Diagrama de seqência da entrada de dados**

Uma vez que o usuário passou todos os parâmetros necessários a uma fábrica, esta cria um `Command` que é recebido pelo `InputController`. Este o envia para o controlador geral do sistema (`Controller`). O fluxo descrito até aqui pode ser visto na figura 2. Nesse ponto, deixa-se a camada de interface para entrar na camada do modelo, descrita a seguir. Vale a pena notar que essa passagem só pode ser realizada passando pela camada de controle de acordo com o padrão `MVC` ([Schmidt et al. 1996]).

A evolução que levou às soluções anteriormente apresentadas e às seguintes está detalhada no trabalho de formatura dos autores ([Corbucci and Bravo 2006]).

### 2.2. Camada de modelo

Antes de executar o comando recebido, o controlador se encarrega de armazenar informações sobre o estado atual do modelo para permitir que a ação seja desfeita. Depois disso, o comando é executado no desenho ativo. Esse desenho (`Drawing`) é a principal unidade de trabalho no modelo. Ele contém camadas de trabalho (`Layer`) que, por sua vez, contêm os elementos (`Element`), como linhas, círculos, etc.

O sistema de modelo utiliza o desenho como um `Composite` ([Gamma et al. 1995]) repassando todos os pedidos de renderização para os elementos abaixo na árvore. Assim o desenho é renderizado a partir de seus componentes.

## 3. Arquitetura RCP

A arquitetura do RCP ([Eclipse Foundation 2003]) é organizada em *plugins*. Existe um ncleo comum a todos os aplicativos baseados nessa plataforma que é responsável por carregar um *plugin* definido como inicial. No caso do Archimedes, esse *plugin*, identificado como `br.org.archimedes`, possui apenas a lógica do fluxo de controle explicada na seção anterior. Sozinho, esse *plugin* não permite nada além de criar um desenho vazio. Porém, ele serve de base para que outros *plugins* adicionem as funcionalidades desejadas, como elementos e fábricas de comandos.

Essa estrutura do sistema garante bastante flexibilidade, pois com ela é possível criar, por exemplo, um conjunto de *plugins* para trabalhar em duas dimensões e outro para trabalhar em três. Além disso, a plataforma RCP permite organizar facilmente os menus e provê uma boa base para construir um sistema de ajuda.

Para poder ser carregado, todo *plugin* deve possuir um arquivo chamado "plugin.xml" em sua raiz. Esse arquivo XML ([World Wide Web Consortium 1998]) descreve as contribuições que o *plugin* faz aplicação. Essas contribuições recebem o nome de extensões, e a cada extensão corresponde um ponto de extensão de algum outro *plugin*. O ncleo do Archimedes define alguns pontos de extensão para que novas funcionalidades possam ser adicionadas a ele.

## 3.1. Extensões disponíveis

Além dos pontos definidos pelo projeto, existem pontos já disponíveis pela plataforma do Eclipse, como `org.eclipse.ui.actionSet`, que permite criar elementos de menus, ou `org.eclipse.ui.view`, que permite acrescentar elementos gráficos à interface, como novas janelas.

### 3.1.1. Elemento (`br.org.archimedes.element`)

Esse ponto permite criar novos elementos do desenho, sejam eles simples, como linhas e círculos, ou complexos, como paredes e janelas, ou até elementos que não devem aparecer em um desenho final, como anotações. Uma extensão de elemento deve possuir um identificador que permite ao núcleo manter um registro dos elementos existentes. Além disso, o *plugin* deve indicar uma classe que estende `br.org.archimedes.interfaces.Element`, para que o sistema possa carregar o elemento quando necessário. Opcionalmente, pode-se definir uma fábrica responsável por criar aquele elemento e um atalho para ela.

Pelo que foi apresentado do sistema até agora, essa extensão não é realmente necessária já que o programa consegue lidar com a interface de `Element` sem precisar saber exatamente a implementação que está instanciada. Esse ponto, no entanto, é necessário para o sistema de exportação que será explicado a seguir.

### 3.1.2. Exportador (`br.org.archimedes.exporter`)

Para permitir *plugins* que exportem o desenho para novos formatos, existe um ponto de extensão para exportadores. O *plugin* que estende esse ponto trata apenas de exportar o desenho e suas camadas, e não deve possuir código para exportar elementos específicos. O ncleo do Archimedes é encarregado de procurar exportadores daquele formato para cada elemento (ver a seção 3.1.3). Além disso, usa-se o registro de *plugins* para listar os exportadores disponíveis no sistema para o usuário.

A vantagem dessa arquitetura é que se um elemento ou um exportador específico não estiver disponível, o exportador ainda funciona. Isto é, não existe dependência entre o exportador e os elementos que ele sabe exportar.

### 3.1.3. Exportador de elemento ( `br.org.archimedes.elementExporter`)

Um *plugin* que estende esse ponto deve definir a extensão e o ID do elemento que ele exporta. Além disso, ele precisa de um identificador próprio e uma classe que implemente `br.org.archimedes.interfaces.ElementExporter`. Essa interface possui apenas um método que será chamado pelo mecanismo de exportação quando for necessário exportar o elemento ao qual ele está associado. O fluxo de exportação pode ser visto no diagrama de seqência na figura 3.

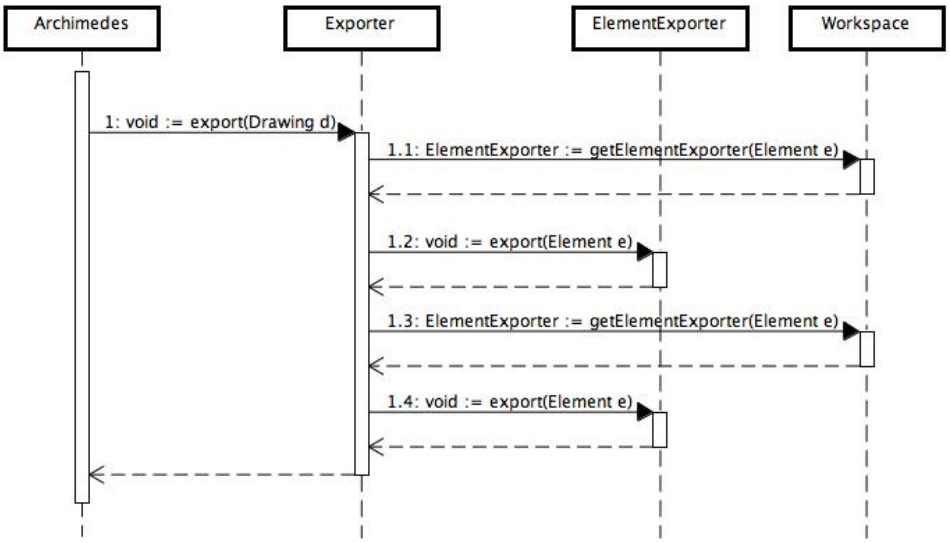


**Figure 3. Diagrama de seqüência da exportação**

### 3.1.4. Importador (`br.org.archimedes.importer`)

A extensão de importação tem a mesma intenção da de exportação: permitir que sejam adicionados importadores independentemente dos elementos disponíveis. Porém, a leitura de dados não segue obrigatoriamente uma estrutura bem definida que permita delegar o trabalho de criação dos elementos a importadores específicos. Por isso, é impossível adotar a mesma solução dos exportadores sob pena de limitar as possibilidades de importação.

Por essas dificuldades, o sistema de importação é constituido apenas de uma classe que deve se encarregar de fazer a total leitura do arquivo, criando a estrutura necessária de objetos. Idealmente, ele não irá depender dos *plugins* de elementos para criá-los, mas isso ainda não está definido. O importador também deverá assumir que podem existir trechos desconhecidos nos arquivos lidos.

### 3.1.5. Formato nativo (`br.org.archimedes.nativeFormat`)

Um formato nativo é apenas um formato para o qual existe um exportador e um importador. Informalmente, exige-se que um formato seja nativo apenas se não houver perda de informação ao salvar e abrir um desenho com seu importador e exportador. Porém, essa

verificação não é feita pelo programa, ficando a critério dos desenvolvedores seguir essa regra.

### 3.1.6. Fábrica (`br.org.archimedes.factory`)

Por fim, existe um ponto de extensão para permitir a criação de novas fábricas de comandos. Essas fábricas permitem criar comandos do sistema, sejam eles novos ou composições de comandos existentes, para realizar novas operações nos desenhos. Com isso, é possível aumentar facilmente o nmero de comandos disponíveis para o usuário.

Esse ponto de extensão exige apenas um identificador e uma classe que implemente `br.org.archimedes.interfaces.CommandFactory`. São opcionais um nome e um conjunto de atalhos sugeridos para essa fábrica. Tanto o identificador quanto o nome e os atalhos são textos que o usuário pode usar para ativar essa fábrica.

## 4. Conclusão

Graças migração do projeto, espera-se que a quantidade de contribuições de código aumente consideravelmente já que os desenvolvedores não precisarão participar diretamente da equipe central do projeto para poder acoplar suas próprias modificações. Essa flexibilidade só pôde ser atingida com o modelo de *plugins* que é a base do RCP do Eclipse.

Com o tempo, espera-se que os desenvolvedores descubram necessidades por outras extensões, que a equipe central poderá disponibilizar e agrupar em conjuntos específicos para certas atividades relacionadas ao programa.

## References


Archimedes Team (2005). Archimedes - The Open CAD. `http://www.archimedes.org.br/`.

Beck, K. (1999). *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional, us ed edition.

Beck, K. and Andres, C. (2004). *Extreme Programming Explained: Embrace Change, 2nd Edition*. The XP Series. Addison-Wesley Professional, 2 edition.

Corbucci, H. and Bravo, M. (2006). *Archimedes: Um CAD Livre desenvolvido com programação extrema e orientação a objetos*. IME-USP. Trabalho de conclusão de curso.

Eclipse Foundation (2001). Eclipse Project. `http://www.eclipse.org/`.

Eclipse Foundation (2003). Eclipse RCP. `http://www.eclipse.org/rcp`.

Gamma, E., Helm, R., Vlissides, J., and Johnson, R. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, first edition.

Schmidt, D., Stal, M., Rohnert, H., and Buschmann, F. (1996). *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns*, volume 1. John Wiley & Sons, 1 edition.

World Wide Web Consortium (1998). Xml definition. `http://www.w3.org/XML/`.