

---

---

# *Easy Java/JavaScript Simulations*

(Simulations Simples en Java/JavaScript)

## Guide de l'utilisateur

---

Wolfgang Christian

et

Francisco Esquembre



---

## Chapitre Un

### Installation et exécution de Easy Java/JavaScript Simulations

Les machines doivent travailler. Les personnes penser. *Richard Hamming*

Ce chapitre d'introduction nous offre une vue d'ensemble sur *Easy Java/JavaScript Simulations* – Simulations Simples en Java/JavaScript – (*EjsS* sous forme abrégée), l'outil de modélisation et de création de haut niveau que nous utilisons pour enseigner la modélisation et la simulation. Nous allons premièrement décrire le processus d'installation et la structure de fichiers de notre programme et par la suite exécuter la console *EjsS* pour obtenir notre premier programme *EjsS* sur l'écran. Nous décrivons comment *Easy Java/JavaScript Simulations* supporte des différents langages de programmation pour le processus de modélisation. Les chapitres ultérieurs de ce document fournissent une introduction étape par étape des différentes parties de l'interface graphique utilisateur. Finalement nous vous donnons les instructions pour télécharger les modèles créés avec *EjsS* existants sur notre bibliothèque numériques en ligne.

#### 1.1 À PROPOS D' Easy Java/JavaScript Simulations

La modélisation informatique est intimement liée à la simulation informatique. Un modèle est une représentation conceptuelle d'un système physique et de ses propriétés, la modélisation est le processus par lequel nous élaborons cette représentation. La modélisation informatique nécessite de (1) la description et de l'analyse d'un problème, (2) de l'identification des variables et des algorithmes, (3) de la mise en oeuvre sur une plate-forme de matériel et logiciel, (4) de l'exécution de l'implémentation et de l'analyse des résultats, (5) de l'affinage et de la généralisation et (6) de la présentation des résultats. Une simulation informatique est une implémentation qui nous permet de tester le modèle sous de différentes conditions dans le but d'apprendre sur le comportement du modèle. L'applicabilité des résultats de

la simulation à ceux du système (physique) authentique dépend de jusqu'à quel point le modèle décrit la réalité. La science consiste à mettre au point des modèles plus généraux ou plus précis.

L'implémentation d'un modèle et la visualisation des données de sortie nous force à programmer un ordinateur. Programmer peut être amusant parce que cela nous donne un contrôle total de tous les détails visuels et numériques du monde simulé. Mais programmer c'est aussi une tâche technique qui peut intimider. Cependant cette barrière technique peut être réduite si vous utilisez un outil approprié. *Easy Java/JavaScript Simulations* est un outil de modélisation qui a été conçu pour permettre aux scientifiques, et non seulement aux scientifiques d'informatique, de créer des simulations dans de différents langages de programmation.<sup>1</sup> *EjsS* simplifie cette tâche autant d'un point de vue technique comme conceptuel.

*EjsS* nous fournit une simple mais puissante structure conceptuelle pour élaborer des simulations. Cet outil fournit une séquence de panneaux de travail pour implémenter le modèle et l'interface graphique utilisateur. *EjsS* automatise des tâches telles que la résolution numérique d'équations différentielles ordinaires et l'animation (en utilisant des traits séparés, en cas nécessaire). La communication de bas niveau entre le programme et l'utilisateur final qui a lieu lors de l'exécution, qui inclut le traitement des actions de la souris dans l'interface graphique de simulation, s'accomplit sans une programmation de bas niveau.

Évidemment une partie du travail dépend toujours de nous. Vous devez fournir un modèle pour le phénomène et concevoir et sélectionner une visualisation des données de sortie qui montre les principales caractéristiques du modèle. Ces tâches de haut niveau sont plus liées à la science qu'à la programmation. Nous vous encourageons de consacrer votre temps et énergie à étudier la science, tâche que l'ordinateur ne peut pas faire. L'objet de ce document est de prouver que cette modélisation par ordinateur n'est non seulement possible pour les non-programmeurs, mais aussi relativement facile, à l'aide de *Easy Java/JavaScript Simulations*.

## 1.2 INSTALLER ET EXECUTER LE LOGICIEL

Commençons par installer et exécuter *Easy Java/JavaScript Simulations*. *EjsS* est un programme Java qui peut s'exécuter sous tout système d'exploitation qui supporte une Machine Virtuelle (VM en anglais) Java. Comme Java est conçue comme une plate-forme indépendante, l'interface utilisateur *EjsS* sur Mac OS X, Windows et Linux présente une apparence très similaire bien qu'il puisse y avoir quelques petites différences. (Nous

---

1. Actuellement, *EjsS* supporte les langages de programmation Java et Javascript.

montrons l'interface Mac OS X dans les illustrations de ce document.)

Si *Easy Java/JavaScript Simulations* n'est pas installé sur votre ordinateur, vous devez l'installer. Pour **installer EjsS**, suivez ces indications :

1. **Installez Java Runtime Environment.** *EjsS* nécessite de Java Runtime Environment (JRE) — l'environnement d'exploitation de Java — **version 1.8 ou ultérieure**. C'est possible que le JRE soit déjà installé sur votre ordinateur mais s'il ne l'est pas, visitez le site Java sur `<http://java.com>` et suivez les instructions pour télécharger et installer la dernière version.
2. **Copiez le fichier d'installation de EjsS sur votre ordinateur.** *EjsS* est distribué dans un dossier compressé ZIP qui peut être téléchargé sur le site web de *EjsS* `<http://www.um.es/fem/EjsWiki>`. Le fichier d'installation sera intitulé d'une manière similaire à celui-ci : **EjsS\_X.x\_yymmdd.zip**. Les caractères **X.x** indiquent la version actuelle du logiciel et **yymmdd** indique la date à laquelle cette version a été créée. (Par exemple vous pouvez obtenir un titre similaire à **EjsS\_5.2\_161123.zip**.)
3. **Décompressez EjsS.** Décompressez le fichier d'installation *EjsS* sur le disque dur de votre ordinateur pour créer un répertoire intitulé **EjsS\_X.x** (**EjsS\_5.2** dans l'exemple). Le nouveau répertoire contient le programme *EjsS* complet.

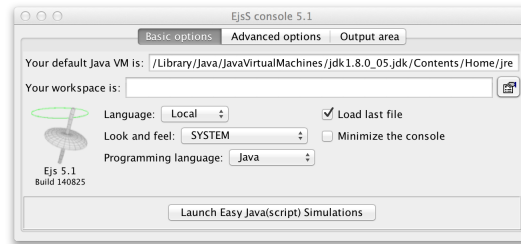
Dans les systèmes comme Unix, le répertoire **EjsS\_X.x** peut-être décompressé comme à lecture seule. Activez les accès d'écriture pour le répertoire **EjsS\_X.x** et tous ses sous-répertoires.

Et voilà! C'est tout ce que vous devez faire pour installer *EjsS*. Une fois que *Easy Java/JavaScript Simulations* est installé sur votre ordinateur, suivez ces indications pour **exécuter EjsS** :

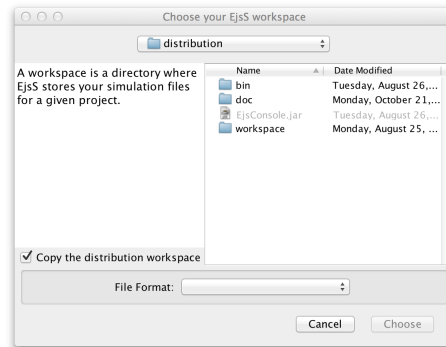
**Exécutez la console EjsS.** Dans le répertoire **EjsS\_X.x** récemment créé, vous trouverez un fichier intitulé **EjsConsole.jar**. Double-cliquez pour exécuter la console *EjsS* comme indiqué dans la figure 1.1.

Si le double-clic n'exécute pas la console, ouvrez une fenêtre de terminal, changez au répertoire **EjsS\_X.x** et tapez la commande : **java-jar EjsConsole.jar**. Vous devrez écrire le nom complet de la commande Java s'il n'est pas dans le *PATH* de votre système.

Vous devriez voir la console (Figure 1.1) et le dialogue de sélection de fichiers de la figure 1.2, décrite ci-dessous, sur votre écran.



**Figure 1.1:** La console *EjsS*.



**Figure 1.2:** Sélecteur de fichier pour choisir votre répertoire d'environnement de travail.

La console *EjsS* n'est pas une partie de *EjsS*, mais une utilité qui s'utilise pour lancer un ou plusieurs exemples (copies) de *EjsS* et pour effectuer d'autres tâches liées à *EjsS*. Vous pouvez utiliser la console pour personnaliser quelques aspects de l'apparence et du fonctionnement de *EjsS* au démarrage.<sup>2</sup> La console montre aussi l'information du programme *EjsS* et les messages d'erreur sur son onglet *Output area* (sortie), auquel nous ferons référence de temps en temps dans ce document. La console crée un exemple de *EjsS* au démarrage et le quitte automatiquement quand vous fermez le dernier exemple de *EjsS* en cours.

Cependant, avant que la console puisse exécuter *EjsS* juste après son installation, le sélecteur de fichier affiché sur la figure 1.2 apparaîtra et vous permettra de sélectionner dans le disque dur de l'ordinateur le répertoire que vous voulez utiliser comme environnement de travail (*workspace*). *EjsS* utilise le concept d'environnement de travail pour organiser votre travail. Un environnement de travail est un répertoire dans votre disque dur sur lequel


2. Par exemple, fixez le *look and feel* (la représentation) à *CROSS\_PLATFORM*, la représentation spécifique qui est identique sur toutes les plateformes, et lancez une nouvelle copie de *EjsS* pour appliquer le changement. Nous fixons la représentation par défaut à *SYSTEM* (système), la valeur par défaut pour votre plate-forme. Celle qui est signalée dans ce document correspond à la représentation par défaut (appelée *Aqua*) de Mac OS X.

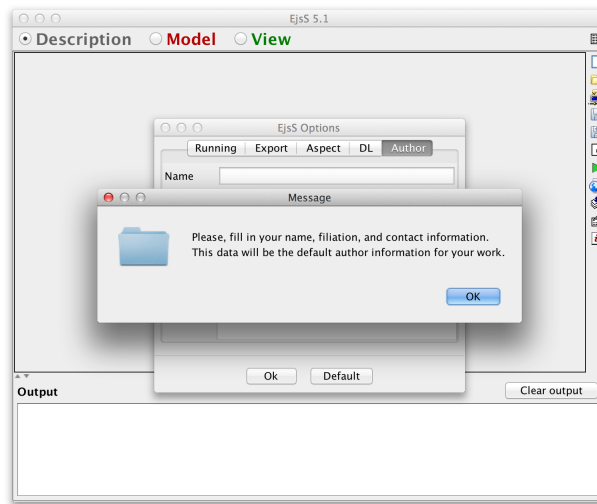
*EjsS* enregistre vos fichiers de simulation d'un projet déterminé. Un environnement de travail peut contenir un nombre illimité de simulations. Dans un répertoire d'environnement de travail, *EjsS* crée quatre sous-répertoires :

- **config** qui est le répertoire pour des fichiers de configuration et des options de l'utilisateur .
- **export** qui est le répertoire proposé par défaut quand *EjsS* génère des fichiers pour la distribution.
- **output** qui est le répertoire utilisé par *EjsS* pour placer les fichiers temporaires qui sont générés quand vous compilez une simulation.
- **source** qui est le répertoire dans lequel tous les fichiers de simulation (sources et auxiliaires) doivent être placés.

Quand vous exécutez pour la première fois *EjsS*, la console vous demande de choisir un répertoire de votre disque dur comme votre environnement de travail (*workspace*). Celui-ci doit être un répertoire enregistrable n'importe où dans votre disque dur. Vous pouvez décider d'utiliser l'environnement de travail qui est inclus dans la distribution par exemple, le répertoire d'environnement de travail dans le répertoire **EjsS\_X.x** qui est créé quand vous décompressez le paquet *EjsS*. Mais il est **fortement recommandé** de créer un nouveau répertoire dans votre répertoire personnel habituel (ou mieux encore, un répertoire sur le cloud — comme *Dropbox* <<http://dropbox.com>> ou similaire). Le dialogue de sélection de fichier qui vous permet de choisir l'environnement de travail dispose d'une case qui, une fois cochée, copie tous les exemples de fichiers de distribution dans le nouvel environnement de travail. Maintenez cette case cochée et vous trouverez quelques sous-répertoires dans le répertoire **source** de votre environnement de travail qui contient des exemples de simulations. En particulier vous trouverez les répertoires **JavaExamples** (Exemples de Java) et **JavascriptExamples** (Exemples de JavaScript) décrits dans les chapitres ultérieures de ce document.

Bien que cela ne soit pas nécessaire en général, vous pouvez créer et utiliser plus d'un environnement de travail pour de différents projets ou tâches. La console prévoit un sélecteur qui vous permet de changer l'environnement de travail en cours et *EjsS* retiendra l'environnement de travail actuel entre les séances, même si vous réinstallez *EjsS*.

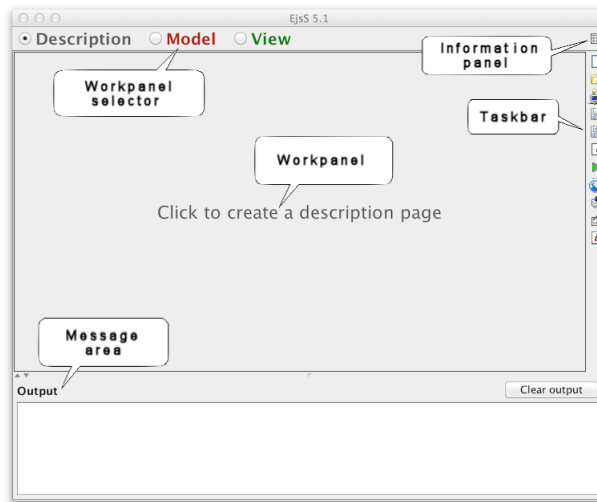
Finalement la première fois que vous exécutez *EjsS*, le programme vous demandera d'indiquer votre nom et affiliation (Figure 1.3). Cette démarche est optionnelle mais elle est recommandée car elle vous aidera à documenter vos simulations futures. Vous pouvez décider de saisir ou de modifier cette information plus tard en utilisant l'icône EjsS *options*, , de la barre des tâches de *EjsS*.



**Figure 1.3:** Optionnellement indiquez votre nom et affiliation.

### 1.3 INTERFACE GRAPHIQUE UTILISATEUR

Vous êtes prêt maintenant pour diriger votre attention sur l'outil de modélisation *EjsS*, affiché avec des annotations à la figure 1.4. En dépit de son interface simple, *EjsS* présente tous les outils nécessaires pour un cycle de modélisation complet.



**Figure 1.4:** L'interface utilisateur de *EjsS* avec des annotations.

La barre de tâche de la droite fournit une série d'icônes pour effacer, ouvrir, rechercher et enregistrer un fichier, pour configurer *EjsS*, et pour afficher l'information du programme et la bulle d'aide. Elle fournit aussi



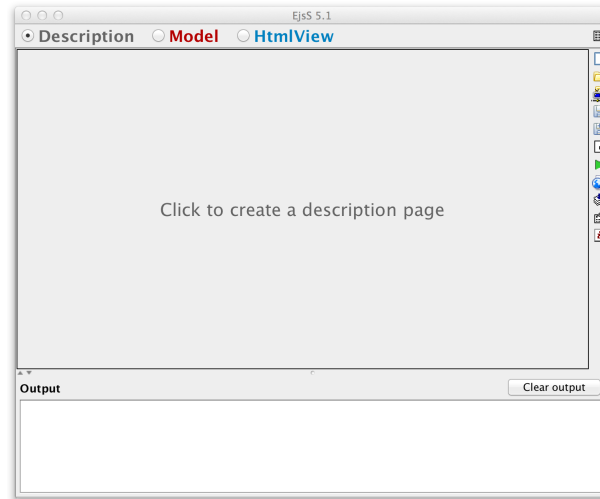
des icônes pour exécuter une simulation et pour stocker une ou plusieurs simulations sur un fichier autonome. Si vous faites un clic droit sur les icônes de la barre de tâche, des actions alternatives (mais en relation) s'affichent, celles-ci seront décrites en fonction de la nécessité. La partie inférieure de l'interface contient une zone de données de sortie sur laquelle *EjsS* affiche des messages. La partie centrale de l'interface contient des panneaux (espaces) de travail sur lesquels la modélisation est élaborée.

*Easy Java/JavaScript Simulations* fournit trois panneaux de travail pour la modélisation. Le premier panneau *Description* nous permet de créer et d'éditer un texte multimédia basé sur HTML qui décrit le modèle. Chaque page qui contient un texte apparaît dans un panneau étiqueté dans le panneau de travail et un clic droit sur la barre étiqueté permet à l'utilisateur d'éditer le texte ou d'importer un texte additionnel. Le deuxième panneau de travail, *Model* (Modèle) est dédié au processus de modélisation. Nous utilisons ce panneau pour créer des variables qui décrivent le modèle, pour initialiser ces variables et pour écrire des algorithmes qui décrivent comment ce modèle change dans le temps. Le troisième panneau de travail *View* (Vue) ou *HtmlView* (Vue Html, pour les interfaces basées sur HTML5) est dédié à la tâche d'élaborer l'interface graphique utilisateur qui permet à l'utilisateur de contrôler la simulation et d'afficher les données de sortie. Nous élaborons l'interface en sélectionnant les éléments de la palette et en les ajoutant au *Tree of elements* (arbre des éléments) de la vue. Par exemple, la palette *Interface* contient des boutons, des sliders (curseurs) et des champs de saisie, et la palette *2D Drawables* contient des éléments pour représenter les données en 2D.

*Easy Java/JavaScript Simulations* supporte plus d'un langage de programmation pour implémenter les algorithmes nécessaires pour le processus de modélisation. L'interface pour tous ces langages est basée sur les mêmes principes, donc si vous apprenez à l'utiliser pour un de ces langages, vous arriverez à tous les comprendre. Quelquefois nous faisons référence à de différentes interfaces de *Easy Java/JavaScript Simulations* pour de différents langages de programmation comme les arômes de *EjsS*. Donc, nous pouvons, par exemple, faire référence à l'arôme *Java* ou à l'arôme *JavaScript* de *EjsS*.

La console *EjsS* lance, par défaut, un exemple-copie de *EjsS* qui supporte le langage de programmation Java (c'est-à-dire un exemple de l'arôme *Java* de *EjsS*). Vous pouvez changer le langage de programmation supporté – ou l'arôme – en utilisant le sélecteur *Programming language* (langage de programmation) dans la barre *Basic options* de la console *EjsS*, et lancer un autre exemple de *EjsS* pour ce langage de programmation en cliquant le bouton *Launch EjsS*. La figure 1.5 affiche l'interface graphique utilisateur

pour la création de modèles JavaScript sur *Easy Java/JavaScript Simulations*.



**Figure 1.5:** L'interface utilisateur de *Easy Java/JavaScript Simulations* pour le support Javascript.


Les deux chapitres suivants de ce document vous guideront pour voir en détail et exécuter une simulation existante dans chaque langage de programmation supporté. (La simulation montre le même phénomène physique dans tous les cas.) Cela vous aidera à comprendre comment les panneaux *Description*, *Model* et *View* (ou *HtmlView*) fonctionnent collaborativement pour vous aider à modéliser une simulation. Si vous êtes débutant sur *Easy Java/JavaScript Simulations*, veuillez lire le chapitre de l'arôme *EjsS* qui vous intéresse le plus avant d'examiner un des modèles existants dans notre bibliothèque numérique.

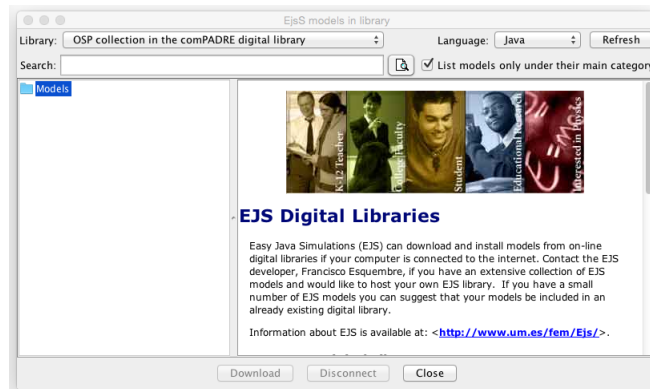
## 1.4 TROUVER DES MODÈLES

Une fois que vous avez abordé les éléments basiques de *EjsS* et que vous savez comment télécharger, examiner, exécuter et même modifier un exemple, vous serez sûrement intéressé à trouver des simulations existantes pour voir ce que d'autres utilisateurs ont fait avec *EjsS*. Vous trouverez peut-être un modèle qui s'adapte à vos besoins ou que vous pouvez facilement modifier pour son utilisation en cours.

Il y a deux lieux que vous pouvez consulter pour trouver plus de modèles. Le premier lieu à consulter c'est le répertoire exemple **source** qui est associé à la distribution de *EjsS*. Dans le répertoire **source** de l'environnement de travail de distribution, vous trouverez quelques répertoires avec

des exemples de simulations. Ces répertoires exemples ont été aussi copiés sur votre propre environnement de travail (à moins que vous ayez désactivé cette option) quand vous avez exécuté pour la première fois *EjsS*.

Le deuxième lieu (en fait, lieux), et probablement le plus intéressant, pour chercher des nouveaux modèles est disponible sur Internet. L'icône de la bibliothèque numérique de *EjsS* sur la barre d'outils, , ouvre une fenêtre qui nous permet de nous connecter à des référentiels de modèles *EjsS* disponibles sur Internet. Cette fenêtre, affichée dans la figure 1.6, contient une boîte combinée dans sa partie supérieure qui fournit une liste des bibliothèques numériques disponibles. Une deuxième boîte combinée vous permet de sélectionner le langage de programmation souhaité pour les modèles. Sélectionnez une de ces bibliothèques, le langage de programmation ou cliquez sur le bouton *Refresh* (actualiser) pour obtenir une liste des modèles disponibles. Toutes ces bibliothèques fonctionnent d'une manière similaire et nous utilisons le référentiel de bibliothèque numérique **comPADRE** pour illustrer comment nous y accédons sur *EjsS*.

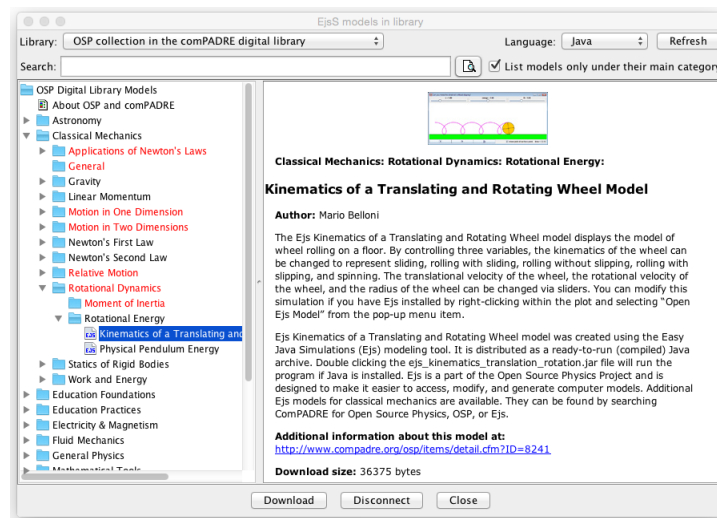


**Figure 1.6:** La fenêtre de bibliothèque numérique de *EjsS*. Sélectionnez un des référentiels disponibles en utilisant la boîte combinée dans la partie supérieure de la fenêtre, le langage de programmation souhaité ou cliquez sur le bouton *Refresh* pour obtenir la liste des modèles disponibles.

Le **comPADRE Pathway** (sentier *comPADRE*), qui est une partie de la Bibliothèque Nationale Numérique de Sciences des États-Unis, est un réseau en expansion de collections de ressources éducatives conçu pour venir en aide aux enseignants et aux étudiants de physique et en astronomie. De spéciale relevance pour nos intérêts, c'est la collection Open Source Physics (OSP) — physique de source ouverte — de comPADRE, disponible sur <http://www.compadre.org/OSP>. Cette collection contient des ressources informatiques pour l'enseignement sous forme de simulations exécutables et des ressources pédagogiques qui attire l'attention des étudiants vers la physique, le calcul numérique et la modélisation informatique. En particulier,

elle contient des modèles auxquels nous pouvons accéder à travers de leurs codes source (XML) directement à partir de *EjsS* en utilisant l'icône de bibliothèques numériques.

Si vous êtes connecté à Internet, sélectionnez la saisie *OSP collection on the comPADRE digital library* (la collection OSP de la bibliothèque numérique comPADRE) de la boîte combinée supérieure et *EjsS* vous connectera à la bibliothèque pour obtenir le tout dernier catalogue des modèles *EjsS* de la bibliothèque. Au moment de rédiger ce document, il y avait plus de 500 modèles Java et de 200 modèles JavaScript organisés dans de différentes catégories et sous-catégories et on estime que la collection devrait augmenter. Comme nous indique le cadre de gauche de la figure 1.7 la collection est organisée en catégories et sous-catégories. Quand le nom d'une sous-catégorie apparaît en rouge, faites un double clic pour ouvrir le nœud avec la liste des modèles de la sous-catégorie. Comme beaucoup de modèles ont une classification primaire et secondaire, une case à cocher dans le volet supérieur à côté du champ de recherche (*Search*) vous permet de décider si vous voulez que les modèles soient énumérés uniquement sous leur classification primaire ou qu'ils apparaissent dans toutes les catégories correspondantes (dans ce cas ils apparaîtront plus d'une fois)



**Figure 1.7:** La collection OSP de la bibliothèque numérique comPADRE. Cette collection est organisée en catégories et sous-catégories. La saisie pour un modèle vous fournit de l'information sur le modèle.

Quand vous cliquez sur un nœud modèle, le cadre de droite vous affiche l'information sur le modèle obtenu instantanément de la bibliothèque. L'information décrit le modèle et inclut un lien direct à la bibliothèque comPADRE pour des informations supplémentaires. Faites double clic sur l'entrée du modèle ou cliquez le bouton *Download* pour extraire le modèle

et les fichiers auxiliaires de la bibliothèque, on vous demandera un emplacement dans votre répertoire **source** de votre environnement de travail pour les télécharger, et découvrira le modèle sur *EjsS* quand le téléchargement est complété. Comme les fichiers sources sont normalement petits, le téléchargement se fera d'une manière presque immédiate. Maintenant vous pouvez examiner, exécuter, modifier le modèle tel que signalé dans les chapitres d'exploration ci-dessous pour le modèle de masse-ressort.

La collection OSP de la bibliothèque numérique comPADRE est fortement recommandée pour chercher des modèles *EjsS* et du matériel pédagogique d'accompagnement.

## 1.5 RÉSUMÉ

*Easy Java/JavaScript Simulations* est un outil de modélisation et d'édition-crédation expressément dédié à la création de simulations informatiques pour étudier et afficher une grande variété de phénomènes qui vont du plus simples au plus complexes. Ces simulations informatiques sont utilisées pour obtenir des données numériques à partir de nos modèles en fonction de la progression dans le temps et pour montrer ces données d'une manière que l'être humain puisse les comprendre.

*EjsS* a été conçu pour nous permettre de travailler à un haut niveau conceptuel, pour concentrer la plupart de notre temps aux aspects scientifiques de notre simulation et demander à l'ordinateur de réaliser automatiquement toutes les tâches nécessaires mais qui sont facilement automatisées. Chaque outil, y compris *EjsS*, présente une courbe d'apprentissage. Le reste de ce document contient une série d'exemples détaillés qui vous permettront de vous familiariser avec les possibilités de modélisation, visualisation et interaction de *EjsS*.

La modélisation est une science et un art. *Easy Java/JavaScript Simulations* est un outil qui vous permet d'exprimer vos connaissances en sciences en vous facilitant les techniques nécessaires de l'art, et en vous fournissant un accès simple à beaucoup d'exemples de modélisation créés par d'autres auteurs.



---

## Chapitre Trois

### Examiner l'arôme Javascript de Easy Java/JavaScript Simulations

Le changement est la loi de la vie. Et ceux dont le regard est tourné vers le passé ou le présent sont certains de rater l'avenir.

*John F. Kennedy*


Pour offrir une perspective du processus de modélisation nous allons, dans ce chapitre, premièrement ouvrir, examiner et exécuter une simulation d'un oscillateur harmonique simple existant. Par la suite, nous allons modifier la simulation pour montrer comment *EjsS* engage l'utilisateur dans le processus de modélisation et réduit en grande partie la quantité de programmation qui est nécessaire. Le chapitre utilise JavaScript comme langage de programmation pour la modélisation et c'est un chapitre similaire au Chapitre 2 (dans lequel Java est utilisé).

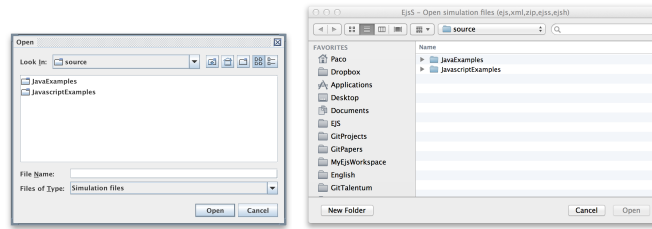
#### 3.1 EXAMINER LA SIMULATION

Comme nous l'avons mentionné dans le Chapitre 1, *Easy Java/JavaScript Simulations* fournit trois panneaux de travail pour construire des simulations. Le premier panneau, *Description*, nous permet de créer et d'éditer un texte multimédia basé sur HTML qui décrit la simulation. Le deuxième panneau, *Model*, est dédié au processus de modélisation. Nous utilisons cet espace pour créer des variables qui décrivent le modèle, pour initialiser ces variables et pour écrire des algorithmes qui décrivent comment ce modèle change dans le temps. Le troisième panneau, *HtmlView*, est dédié à la tâche d'élaborer l'interface graphique utilisateur qui permet à l'utilisateur de contrôler la simulation et d'afficher les résultats ou données de sortie.

Pour comprendre comment les panneaux *Description*, *Model* et *HtmlView* fonctionnent collaborativement, nous allons examiner et exécuter une simulation existante. Une capture d'écran ne remplace pas une démonstration en direct et nous vous encourageons à continuer à suivre sur votre


ordinateur au fur et mesure que vous lisez.

Cliquez sur l'icône *Open* icon, , de la barre de tâche de *EjsS*. Une boîte de dialogue de fichier similaire à celle qui est sur la figure 3.1 s'affiche pour montrer le contenu du répertoire **source** de votre environnement de travail. Dirigez-vous sur le répertoire **JavascriptExamples** où vous trouverez un fichier intitulé **MassAndSpring.ejss**. Sélectionnez ce fichier et cliquez sur le bouton *Open* de la boîte de dialogue.



**Figure 3.1:** La boîte de dialogue de fichier ouverte vous permet de naviguer sur votre disque dur et de ouvrir une simulation existante. L'apparence de la boîte de dialogue (deux types différents de *look and feels* – présentations – sont affichés ici) peut varier en fonction de votre système d'exploitation et de la présentation sélectionnée.

Maintenant les choses deviennent réelles ! *EjsS* lit le document **MassAndSpring.ejss** qui comble le panneau et un navigateur intitulé “EjsS-PREVIEW” apparaît sur votre écran comme indiqué dans la figure 3.2.<sup>1</sup> Un petit avertissement. Vous pouvez déplacer les objets et cliquer sur des boutons de la maquette de la fenêtre mais le modèle ne représentera pas le comportement complet. Pour cela vous devez exécuter la simulation.

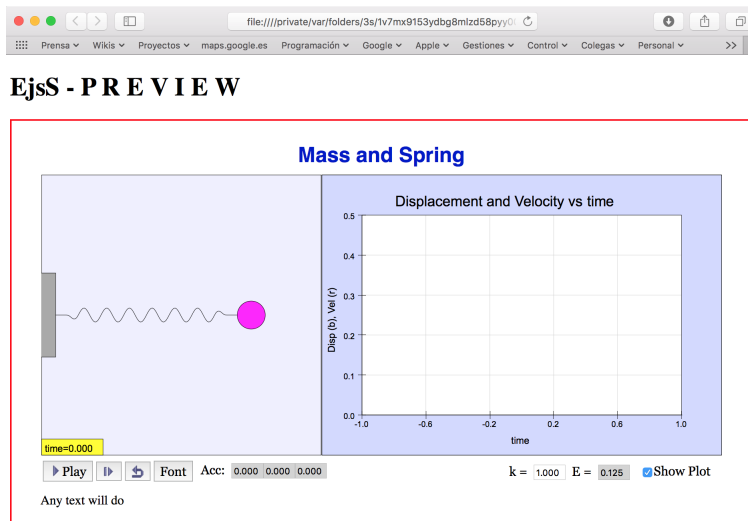
Les lecteurs impatientes ou précoces peuvent être tentés de cliquer sur l'icône vert d'activation  de la barre de tâches pour exécuter notre exemple avant de procéder avec le tutoriel. Les lecteurs qui feront cela n'interagiront plus directement avec *EjsS*, mais avec un programme complet d'exécution de JavaScript sur une nouvelle page HTML. Quittez le programme d'exécution en fermant ce nouveau page web *Mass and Spring* avant de procéder.

### 3.1.1 The panneau *Description*

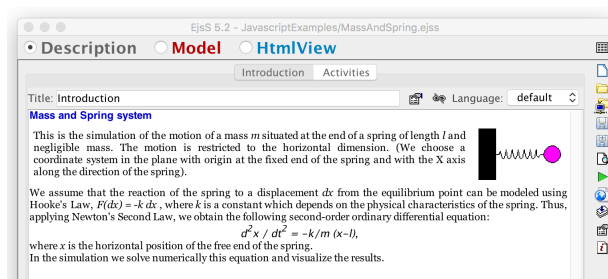
Sélectionnez le panneau *Description* en cliquant sur le bouton radio correspondant dans la partie supérieure de *EjsS* et vous verrez deux pages de texte pour cette simulation. La première page affichée dans la figure 3.3 contient un petit débat sur le modèle de masse-ressort. Cliquer sur la barre *Activities* (activités) pour voir le deuxième texte.

1. Si le navigateur n'apparaît pas, de-sélectionnez la case à cocher *Keep preview hidden* (Garder la prévisualisation cachée) du panneau *HtmlView*.





**Figure 3.2:** La maquette de fenêtre de *EjsS* de la simulation **MassAndSpring**. La barre de titre et la bordure rouge indique que c’est une page de prévisualisation HTML et que le programme ne s’exécute pas.



**Figure 3.3:** Les pages de description pour la simulation de la masse-ressort. Cliquez sur la barre pour afficher la page. Faites un clic-droit sur la barre pour éditer la page.

Une *Description* est un texte multimédia HTML ou XHTML qui fournit de l’information et des instructions sur la simulation. HTML représente Hypertext Markup Language (langage de balisage d’hypertexte) et c’est le protocole le plus habituellement utilisé pour le formatage et l’affichage de documents sur le Web. Le X de XHTML signifie eXtensible. XHTML est essentiellement un HTML exprimé comme un XML valide ou, d’un mode plus simple, un HTML parfaitement formaté.

*EjsS* fournit un éditeur HTML simple qui vous permet de créer et de modifier des pages sur *EjsS*. Vous pouvez aussi importer des pages HTML ou (préférentiellement) XHTML à *EjsS* en faisant un clic droit sur la barre de l’espace de travail *Description*. (Voir la Section 3.4.3.) Les pages de description sont une partie essentielle du processus de modélisation et ces pages

sont inclus dans le modèle compilé quand le modèle est exporté pour sa distribution.

### 3.1.2 Le panneau *Model*

Le panneau *Model* est l'espace où le modèle est défini pour qu'il puisse être converti en un programme par *EjsS*. Dans cette simulation, nous étudions le mouvement de la particule de masse  $m$  attaché à une extrémité du ressort sans masse de la longueur d'équilibre  $L$ .

Le ressort est fixé au mur par son autre extrémité et son mouvement est restreint en direction horizontale. Bien que la masse oscillante présente une solution analytique bien connue, il est utile de commencer par le modèle oscillateur harmonique simple pour que nos données de sortie puissent être comparées avec le résultat analytique exact.

Notre modèle assume des petites oscillations pour que le ressort réponde à un déplacement donné horizontal  $\delta x$  de la longueur de l'équilibre  $L$  sous une force déterminée par la Loi de Hooke,  $F_x = -k \delta x$ , dans laquelle  $k$  est la constante élastique du ressort, qui dépend de ses caractéristiques physiques. Nous utilisons la deuxième Loi de Newton pour obtenir une équation différentielle du second ordre pour la position de la particule :

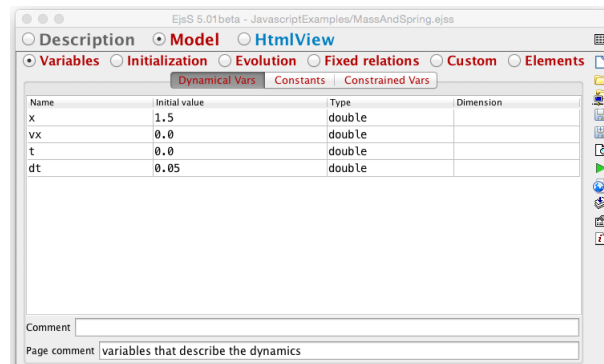
$$\frac{d^2 x}{dt^2} = -\frac{k}{m} (x - L). \quad (3.1.1)$$

Remarquez que nous utilisons un système de coordonnées avec l'axe- $x$  au long du ressort et son origine est à l'extrémité du ressort qui est fixée. La particule est placée à  $x$  et le déplacement de l'équilibre  $\delta x = x - L$  équivaut à zéro quand  $x = L$ . Nous résolvons ce système numériquement pour étudier l'évolution de son état dans le temps.

Voyons comment nous allons implémenter le modèle de masse-ressort en sélectionnant le bouton radio de *Model* et en examinant chacun des six panneaux.

#### 3.1.2.1 Déclaration des variables

Quand vous implémentez un modèle, une des bonnes choses à faire premièrement c'est identifier, définir et initialiser les variables qui décrivent le système. Le terme *variable* est très général et fait référence à tout ce que l'on peut attribuer un nom, y compris une constante physique et un



**Figure 3.4:** Le panneau *Model* contient six sous-panneaux de travail. Le sous-panneau pour la définition des variables dynamiques masse-ressort est affiché. D'autres onglets dans ce sous-panneau de travail définissent les variables additionnelles telles que la longueur naturelle du ressort  $L$  et l'énergie  $E$ .

graphique. La figure 3.4 affiche un panneau de variable *EjsS*. Chaque ligne définit une variable du modèle en précisant le nom de la variable, le type, la dimension et la valeur initiale.

Les variables dans les programmes informatiques peuvent être de différents types en fonction de l'information qu'elles détiennent. Les types plus habituellement utilisés sont **boolean** pour les valeurs vrai/faux, **int** pour les nombres entiers, **double** pour les valeurs numériques à haute précision ( $\approx 16$  chiffres significatifs) et **String** pour les textes. Nous utiliserons tous ces types de variables dans ce document mais le modèle masse-ressort utilise seulement les variables de type **double** et **boolean**.

Si vous avez déjà appris un peu de JavaScript, vous saurez probablement que JavaScript n'a aucun type pour les variables. Toutes les variables sont déclarées avec un mot-clé **var**. Ce qui veut dire que, en principe, JavaScript ne fait aucune différence entre integers (les nombres entiers), double (les nombres à haute précision) ou String (les textes), etc. . . Mais il le fait ! Par exemple vous ne devez pas utiliser une variable double comme index pour un tableau (*array*, en anglais). C'est pour cela et parce qu'il vous aide à clarifier l'utilisation des variables dans votre modèle (quelles valeurs elles peuvent détenir et où vous pouvez ou vous ne pouvez pas les utiliser) nous vous recommandons d'attribuer un type à chaque variable de votre modèle.

Les variables peuvent s'utiliser comme paramètres, comme variables d'état ou comme données d'entrée et de sortie du modèle. Les panneaux de la figure 3.4 définissent les variables utilisées dans notre modèle. Nous avons déclaré une variable pour la position- $x$  de la particule,  $x$ , pour sa vitesse dans la direction- $x$ ,  $vx$ , pour le temps,  $t$ , et pour l'incrément de temps à chaque étape de la simulation,  $dt$ . Nous définissons quelques variables dans

cet onglet et dans d'autres qui n'apparaissent pas dans l'équation(3.1.1). Les raisons pour l'utilisation de variables auxiliaires telles que  $\mathbf{vx}$  ou les énergies cinétiques, potentielles ou totales seront indiquées clairement par la suite. La partie inférieure du panneau de variables contient un champ de commentaires qui fournit une description du rôle de chaque variable dans le modèle. Cliquez sur une variable pour afficher le commentaire correspondant.

### 3.1.2.2 Initialisation du modèle

Fixer les conditions initiales correctement est très important quand vous implémentez un modèle car le modèle doit commencer dans un état qui puisse se réaliser physiquement. Notre modèle est relativement simple et nous l'initialisons en introduisant des valeurs (ou des expressions simples de JavaScript telles que  $0.5*m*vx*vx$ ) dans la colonne *Initial value* (valeur initiale) du panneau de variables. *EjsS* utilise ces valeurs quand il initialise la simulation.

Les modèles avancés peuvent nécessiter un algorithme d'initialisation. Par exemple un modèle dynamique moléculaire peut fixer la vitesse de chaque particule pour un ensemble de particules. Le panneau *Initialization* nous permet de définir une ou plusieurs pages de code JavaScript qui réalisent les calculs nécessaires. *EjsS* transforme ce code en une fonction JavaScript et reprend cette fonction au démarrage et chaque fois que la simulation est réinitialisée. Le panneau *Initialization* de masse-ressort n'est pas affiché parce qu'il est vide. Voir le Paragraphe 3.1.2.4 pour un exemple de comment le code JavaScript apparaît sur *EjsS*.

### 3.1.2.3 L'évolution du modèle

Le panneau *Evolution* nous permet d'écrire le code JavaScript qui détermine comment le système masse-ressort évolue dans le temps et il utilisera cette option fréquemment pour les modèles qui ne sont pas basés sur des équations différentielles ordinaires (EDO, ODE en anglais). Cependant, il y a une deuxième option qui nous permet d'introduire des équations différentielles ordinaires telles que (3.1.1) sans programmer. *EjsS* fournit un éditeur dédié qui nous permet de préciser des équations différentielles dans un format qui ressemble à des notations mathématiques et qui génère automatiquement le code JavaScript correct.

Voyons comment l'éditeur d'équations différentielles fonctionne pour le modèle de masse-ressort. Étant donné que les algorithmes EDO résolvent des systèmes d'équations différentielles ordinaires de premier ordre, une équation

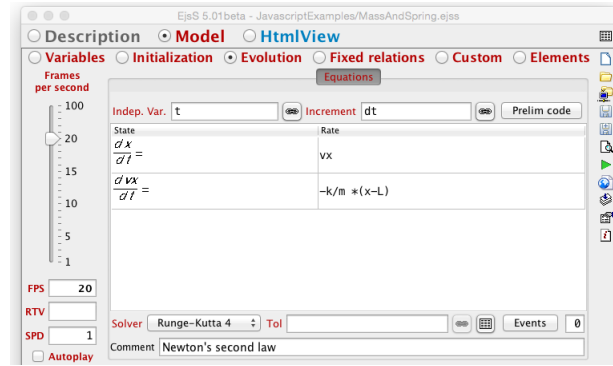
tion d'ordre supérieur tel que (3.1.1) doit être refondée dans un système de premier ordre. Nous pouvons faire cela en considérant la vitesse comme une variable indépendante qui obéit à sa propre équation :

$$\frac{d x}{d t} = v_x \quad (3.1.2)$$

$$\frac{d v_x}{d t} = -\frac{k}{m} (x - L). \quad (3.1.3)$$

La nécessité d'une équation différentielle additionnelle explique pourquoi nous déclarons la variable `vx` dans notre panneau de variables.

En cliquant sur le panneau *Evolution*, l'éditeur EDO représenté dans la figure 3.5 s'affiche.



**Figure 3.5:** Le panneau évolution ODE (EDO en français) représente l'équation différentielle et l'algorithme numérique de la masse-ressort.

Remarquez que l'éditeur EDO affiche (3.1.2) et (3.1.3) (en utilisant le caractère `*` pour désigner la multiplication). Les champs qui se trouvent dans la partie supérieure de l'éditeur précisent la variable indépendante `t` et l'incrément de variable `dt`. Les algorithmes numériques estiment la solution EDO exacte en indiquant l'état dans les étapes distinctes et l'incrément détermine cette intervalle. Le bouton *Prelim code* dans la partie supérieure droite de l'éditeur nous permet d'introduire un code préliminaire pour réaliser les calculs avant d'évaluer les équations (une circonstance nécessaire dans des situations plus complexes que celle dont nous nous occupons dans cet exemple). Un menu déroulant dans la partie inférieure de l'éditeur nous permet de sélectionner le *solutionneur* EDO, un algorithme numérique qui anticipe la solution à partir de la valeur actuelle de temps, `t` à la suivante valeur `t+dt`. Le champ de tolérance (*Tol*) est grisé et vide car Runge-Kutta 4 est une méthode à étape fixe qui ne nécessite pas de paramètres de tolérance. Le bouton *Fine-tune...* affiche un dialogue qui nous permet de régler l'exécution de ce solutionneur bien que les valeurs par défaut soient normalement appropriées. Finalement le champ d'événements (*Events*) dans la partie in-

férieure du panneau nous indique que nous n'avons défini aucun évènement pour cette équation différentielle. Vous pourrez trouver des exemples avec des codes préliminaires et des évènements dans les pages suivantes de ce document. Les différents algorithmes de résolution et leurs paramètres sont commentés dans la bulle d'aide de *EjsS*.

La partie gauche de l'espace de travail *evolution* comprend des champs qui déterminent à quel niveau de *douceur* et à quelle vitesse la simulation s'exécute. L'option de *frames per second (FPS)* — images par seconde — qui peut être sélectionnée soit en utilisant un curseur, soit un champ de saisie, précise combien de fois par seconde nous voulons que notre simulation soit redessinée sur l'écran. Le champ de saisie *steps per display (SPD)* — étape par affichage — précise combien de fois nous voulons faire progresser le modèle avant de le redessiner. La valeur actuelle de 20 images par seconde produit une bonne animation qui, conjointement avec la valeur prescrite d'une étape par affichage et 0.05 pour  $\Delta t$ , donne lieu à une simulation qui s'exécute (à peu près) en temps réel. Nous utiliserons pratiquement toujours le paramètre par défaut d'une étape par affichage. Néanmoins il y a des situations dans lesquelles les données de sortie graphique du modèle utilisent une quantité considérable de la puissance de traitement et au cours desquelles nous voulons accélérer les calculs numériques. Dans ce cas nous pouvons augmenter la valeur du paramètre d'étape par affichage pour que le modèle progresse plusieurs fois avant que la visualisation soit redessinée. La case à cocher *Autoplay* indique si la simulation doit démarrer quand le programme démarre. Cette case est désactivée pour que vous puissiez changer les conditions initiales avant de démarrer l'évolution.

Le panneau *Evolution* gère les aspects techniques du modèle EDO de masse-ressort sans programmer. La simulation fait progresser l'état du système en résolvant numériquement les équations différentielles du modèle et en utilisant l'algorithme intermédiaire. L'algorithme passe de l'état actuel à un temps  $t$  à un nouvel état à un nouveau temps  $t + \Delta t$  avant que la visualisation soit redessinée. La simulation répète cette étape évolution 20 fois par seconde sur les ordinateurs ou dispositifs avec une puissance de traitement faible. La simulation peut s'exécuter plus lentement ou incorrectement sur des ordinateurs ou des dispositifs avec une puissance de traitement insuffisante ou si l'ordinateur est occupé mais il devrait fonctionner.

Bien que le modèle masse-ressort puisse se résoudre avec un simple algorithme EDO, notre bibliothèque de méthodes numériques contient des algorithmes très sophistiqués et *EjsS* peut appliquer ces algorithmes à de grands systèmes d'équations différentielles de vecteur avec ou sans évènements discontinus.

### 3.1.2.4 Relation entre les variables

Toutes les variables d'un modèle ne sont pas calculées en utilisant un algorithme du panneau *Evolution*. Les variables peuvent se calculer après que l'évolution soit appliquée. Nous faisons référence aux variables qui sont calculées en utilisant l'algorithme *evolution* comme des variables d'état ou des variables dynamiques et nous faisons référence aux variables qui dépendent de ces variables comme des variables auxiliaires ou de sortie. Dans le modèle masse-ressort les énergies cinétiques, potentielles et totales du système sont des variables de sortie parce qu'elles sont calculées à partir de variables d'état :

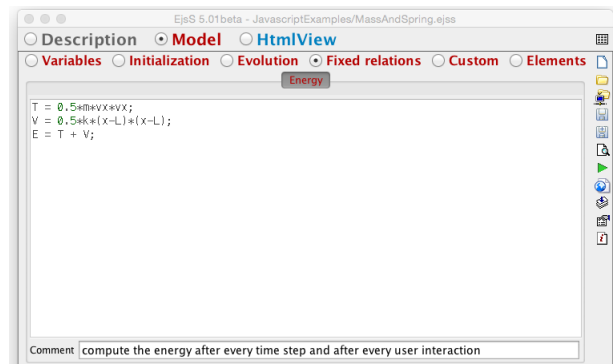
$$T = \frac{1}{2}mv_x^2, \quad (3.1.4)$$

$$V = \frac{1}{2}k(x - L)^2, \quad (3.1.5)$$

$$E = T + V. \quad (3.1.6)$$

On dit qu'il existe des relations fixes entre les variables du modèle.

Le panneau *Fixed relations* représenté dans la figure 3.6 s'utilise pour écrire des relations fixes entre les variables. Remarquez la facilité pour convertir (3.1.4) jusqu'à (3.1.6) à travers de la syntaxe JavaScript. Assurez-vous d'utiliser le caractère de multiplication `*` et d'écrire un point-virgule à la fin de chaque énoncé JavaScript.



**Figure 3.6:** Les relations fixes pour le modèle masse-ressort.

**Voilà une importante remarque !** Vous vous demandez peut-être pourquoi nous n'écrivons pas des expressions de relation fixe en incorporant une deuxième page de code après la page ODE (EDO) dans le panneau *Evolution*. Après tout, les pages évolution exécutent séquentiellement et une deuxième page d'évolution actualiserait correctement les variables de sortie après chaque étape. . . La raison pour laquelle le panneau *Evolution* ne doit pas être utilisé c'est parce que les relations fixes doivent *toujours* se main-

tenir car il y a d'autres manières, telles que les actions de la souris, pour modifier les variables d'état. Par exemple, déplacer la masse change la variable  $x$  et ce changement affecte l'énergie. *EjsS* évalue automatiquement les relations après l'initialisation, après chaque étape d'évolution et chaque fois qu'il y ait une interaction de l'utilisateur avec l'interface de la simulation. C'est pour cela que c'est important que les relations fixes entre variables soient annotées sur le panneau *Fixed relations*.

#### 3.1.2.5 Les pages personnalisées (*Custom*)

Il y a un cinquième panneau dans le panneau *Model* intitulé *Custom* (Personnalisation). Ce panneau peut s'utiliser pour définir des fonctions JavaScript qu'on veut utiliser dans le modèle. Ce panneau est vide parce que notre modèle actuel ne nécessite pas de méthodes additionnelles mais nous utiliserons ce panneau quand nous modifierons l'exemple masse-ressort dans la Section 3.4. Une fonction personnalisée ne s'utilise pas à moins qu'elle soit explicitement indiquée par un autre panneau.

#### 3.1.2.6 Les éléments du modèle

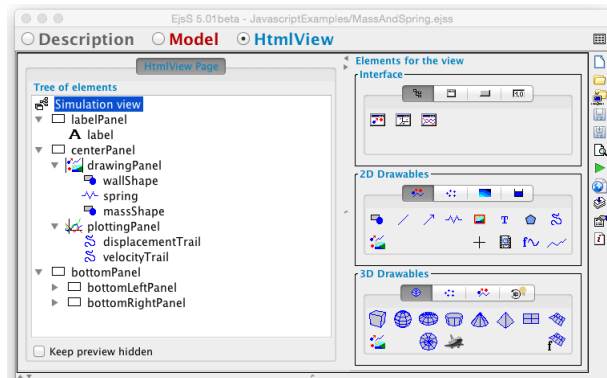
Le sixième et dernier sous-panneau du panneau *Model* est intitulé *Elements* et il vous fournit un accès aux bibliothèques JavaScript externes en forme de icônes glisser-déposer. Vous incorporez ces bibliothèques à votre programme en glissant l'icône correspondant à la liste des éléments *Model* à utiliser pour ce modèle. Cela crée des objets JavaScript que vous pouvez utiliser dans votre code modèle. Ce panneau est aussi vide pour ce modèle parce que notre modèle masse-ressort ne nécessite pas de bibliothèques JavaScript additionnelles.

### 3.1.3 Le panneau *HtlnView*

*HtlnView* est le troisième panneau de *Easy Java/JavaScript Simulations*. Ce panneau nous permet de créer une interface graphique basée sur HTML qui comprend la visualisation, l'interaction de l'utilisateur et le contrôle du programme avec un minimum de programmation. La figure 3.2 représente un affichage basée sur HTML pour le modèle de masse-ressort. Sélectionnez le bouton radio *HtlnView* pour examiner comment un affichage basé sur HTML est créé.



Le cadre de droite du panneau *HttmlView* de *EjsS* représenté dans la figure 3.7 contient une collection d'éléments d'affichage (*view elements*) basé sur HTML, regroupés en base à leur fonctionnalité. Les éléments d'affichage sont des blocs de construction qui peuvent se combiner pour former une complète interface utilisateur basé sur HTML, et chaque élément d'affichage est un objet spécialisé avec une représentation sur l'écran. Pour afficher l'information d'un élément déterminé, cliquez sur son icône et tapez la touche *F1* ou faites un clic droit pour sélectionner le menu *Help* (aide) de l'objet. Pour créer une interface utilisateur, nous créons un affichage HTML vide (imaginez-le comme une page blanche HTML) et incorporez les éléments tels que le panneau, les boutons, les graphiques... en utilisant glisser-déposer tel que décrit dans la Section 3.4.



**Figure 3.7:** Le panneau *HttmlView* représentant le *Tree of elements* (l'arbre des éléments) pour l'interface utilisateur masse-ressort.

Le *Tree of elements* (l'arbre des éléments) représenté sur le côté gauche de la figure 3.7 affiche la structure de l'interface utilisateur masse-ressort. Remarquez que la simulation présente trois panneaux principaux : *labelPanel*, *centerPanel* et *bottomPanel* (panneau étiquette, panneau central et panneau inférieur) qui sont alignés verticalement sur la page HTML du navigateur. Ces trois panneaux appartiennent à la classe d'éléments *Containers* (de conteneurs), dont le but principal est de regrouper (organiser) visuellement d'autres éléments sur l'interface utilisateur. L'arbre des éléments affiche des noms descriptifs et des icônes pour ces éléments. Faites un clic-droit sur un élément de l'arbre pour obtenir un menu qui aidera l'utilisateur à changer cette structure. Alternativement, vous pouvez glisser et déposer les éléments d'un conteneur dans un autre pour changer la relation parent-enfant ou dans un conteneur pour changer l'ordre des enfants.

Il y a des conditions pour qu'un conteneur accepte un élément donné comme enfant. Par exemple un panneau d'affichage en deux dimensions peut seulement accepter des éléments d'affichage en 2D.

Chaque élément d'affichage présente une série de paramètres internes, dénommés *Properties* (propriétés) qui configurent l'apparence et le comportement de l'élément. Nous pouvons éditer ces propriétés en faisant double-clic sur l'élément de l'arbre pour afficher un panneau connu comme *properties inspector* (le contrôleur des propriétés). Les propriétés d'apparence telles que la couleur sont souvent établies par une valeur constante tel que "Red" (rouge). Nous pouvons aussi utiliser une variable du modèle pour fixer la propriété d'un élément. Cette capacité pour connecter (relier) une propriété à une variable sans programmer est l'aspect essentiel pour transformer notre affichage en une visualisation dynamique et interactive.

Voyons comment ce processus fonctionne dans la pratique. Faites double clic sur l'élément **massShape** (le suffixe 'Shape' – forme – que nous ajoutons au nom de l'élément vous aide à savoir de quel type d'élément il s'agit) dans l'arbre pour afficher le contrôleur de propriétés des éléments. Cet élément est la masse qui est associée à l'extrémité libre du ressort. Le panneau de propriétés de *massShape* apparaît tel qu'il est représenté dans la figure 3.8.

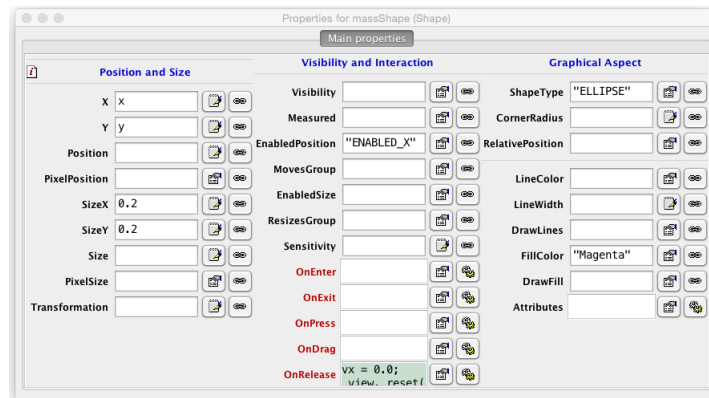


Figure 3.8: Le panneau de propriétés de l'élément **massShape**.

Observez les propriétés qui ont une valeur constante déterminée. Les propriétés **ShapeType**, **SizeX**, **SizeY**, et **FillColor** produisent une ellipse d'unité de grandeur (0.2,0.2) qui produit un cercle (colorié en couleur Magenta). Plus important encore, les propriétés **X** et **Y** de la forme sont reliés aux variables **x** et **y** du modèle. Cette simple attribution établit une connexion bidirectionnelle entre le modèle et l'affichage. Ces variables changent conformément le modèle évolue et les formes suivent les valeurs **x** et **y**. Si l'utilisateur glisse la forme à un nouvel emplacement, les variables **x** et **y** du modèle changent en conséquence. Remarquez cependant, que la propriété **Draggable** (glissable) est fixée pour vous permettre uniquement de déplacer horizontalement la masse.

Les éléments peuvent aussi avoir des propriétés d'action qui peuvent

s'associer à un code. (Les étiquettes des propriétés d'actions sont affichées en rouge.) Les actions de l'utilisateur telles que glisser ou cliquer, comporte leur correspondante propriété d'action qui fournit donc, une manière simple de contrôler la simulation. Quand l'utilisateur libère la masse après l'avoir glissée, le code suivant (précisé sur la propriété d'action `OnRelease`) est exécuté :

```
vx = 0.0;          // sets the velocity to zero
_view.reset();    // clears all plots
```

Cliquez sur l'icône près du champ pour afficher un petit éditeur qui affichera ce code.

Comme le code d'action `OnRelease` occupe plus d'une ligne, le champ de propriété dans le contrôleur affiche un fond (vert) plus sombre. D'autres types de données telles que les propriétés *boolean* ont des éditeurs différents. Cliquez sur le deuxième icône pour afficher une boîte de dialogue avec une liste des variables et des fonctions qui peuvent être utilisées pour fixer la valeur de la propriété.

### Exercice 3.1. Les contrôleurs d'éléments

Le contrôleur de la masse affiche de différents types de propriétés et leurs possibles valeurs. Examinez les propriétés des autres éléments de l'affichage. Par exemple les éléments `displacementTrail` et `velocityTrail` correspondent au déplacement et à la vitesse tracés sur le panneau de représentation de tracés de l'extrême droite de l'affichage respectivement. Quel est le nombre maximum de points qui peuvent être attribués à chaque tracé? □

#### 3.1.4 La simulation complétée

Nous avons vu que *Easy Java/JavaScript Simulations* est un puissant outil qui nous permet d'exprimer les connaissances d'un modèle à un haut niveau d'abstraction. Pour la modélisation masse-ressort, nous avons premièrement créé un panneau de variables qui décrivent le modèle et nous avons initialisé ces variables en utilisant une colonne du panneau. Nous avons ensuite utilisé un panneau *evolution* avec un éditeur à haut niveau pour les systèmes d'équations différentielles ordinaires de premier ordre pour préciser comment l'état progresse dans le temps. Par la suite nous avons écrit les relations pour calculer les variables auxiliaires ou de sortie qui peuvent être exprimées en utilisant des expressions comportant des variables d'état. Finalement l'interface graphique utilisateur du programme et des visualisations de haut niveau sont créés en glissant des objets de la palette *Elements* (éléments) jusqu'au arbre des éléments (*Tree of elements*). Les propriétés

des éléments sont fixées en utilisant un éditeur de propriété et quelques propriétés sont associées aux variables procédant du modèle.

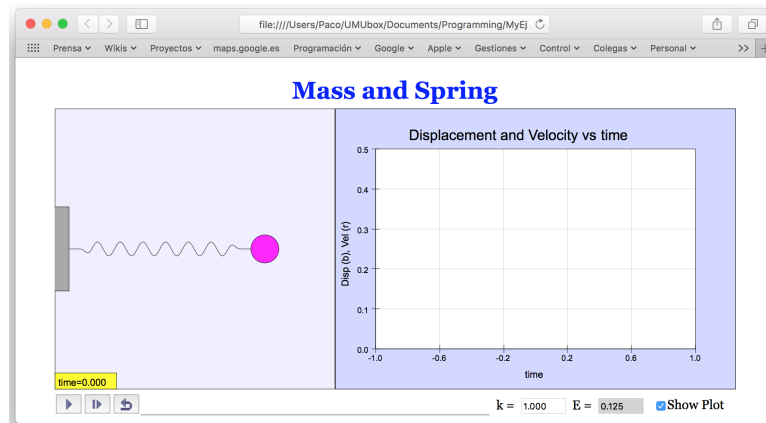
Il est important de remarquer que les trois lignes du code du panneau de relations fixes — *Fixed relations*, figure 3.6 — et les deux lignes du code de la méthode d'action de la particule sont les uniques codes JavaScript explicites nécessaires pour implémenter le modèle. *Easy Java/JavaScript Simulations* crée un programme JavaScript complet en processant l'information des panneaux quand l'icône exécuter est activé comme décrit dans la Section 3.2.

### 3.2 EXECUTER LA SIMULATION

Le moment est venu d'exécuter la simulation en cliquant sur l'icône *Run* (exécuter) de la barre des outils, ►. *EjsS* génère le code JavaScript, recueille les fichiers auxiliaires et de bibliothèque et génère et ouvre une page HTML qui contient le programme complet. Tout cela en un seul clic.


L'exécution d'une simulation initialise ses variables et exécute les relations fixes pour s'assurer que le modèle est dans un état consistant. L'évolution de temps du modèle démarre quand vous appuyez sur le bouton *Play/Pause* de l'interface utilisateur (le bouton *Play/Pause* affiche l'icône ► quand la simulation est en pause et || quand elle est en marche.) Sur notre exemple actuel, le programme exécute une méthode numérique pour calculer l'équation différentielle d'oscillateur harmonique par unité de temps de 0.05 et il exécute ensuite le code de relations fixes. Par la suite, les données sont passées au graphique et le graphique est redessiné. Ce processus est répété 20 fois par seconde.

Quand nous exécutons une simulation, *EjsS* imprime des messages d'information qui indiquent que la simulation a été générée correctement et qu'elle est en fonctionnement. Remarquez que la maquette de la page du navigateur (si elle a été ouverte) est remplacée par une nouvelle mais très similaire page sans la bordure rouge autour du titre. (Alternativement, la simulation peut apparaître à une nouveau page sur votre navigateur HTML.) Ces affichages répondent maintenant aux actions de l'utilisateur. Glissez et déposez la particule à la position horizontale initiale souhaitée et cliquez sur le bouton *Play/Pause*. La particule oscille sur son point d'équilibre et le graphe affiche les données de déplacement et de vitesse tel que représenté dans la figure 3.9. Pour quitter le programme, fermez la page du navigateur.



**Figure 3.9:** La simulation de masse-ressort affiche un dessin interactif du modèle et un graphique avec les données de déplacement et de vitesse.

### 3.3 DISTRIBUTION DE LA SIMULATION

Les simulations créées avec *EjsS* sont des programmes JavaScript autonomes qui peuvent être distribués sans *EjsS* pour être utilisé par d'autres personnes. La manière plus facile de les distribuer c'est de compresser la simulation dans un unique fichier compressé exécutable en cliquant sur l'icône *Package*, . Un explorateur de fichier s'affiche et il vous permettra de choisir un nom pour le paquet de contenus autonomes. Le répertoire par défaut pour sauvegarder ce fichier c'est le répertoire **export** de votre environnement de travail mais vous pouvez choisir le nom de répertoire ou de paquets de votre choix. Le fichier compressé autonome est prêt pour être distribué sur un CD ou sur Internet. D'autres mécanismes de distribution sont disponibles en faisant un clic droit sur l'icône.

#### Exercice 3.2. Distribution d'un modèle

Cliquez sur l'icône *Package* de la barre de tâches pour créer une archive compressée autonome de la simulation masse-ressort. Copiez ce fichier compressé dans un répertoire de travail séparé de votre installation *EjsS*. Fermez *EjsS* et vérifiez que la simulation s'exécute comme une application autonome en décompressant le fichier et en faisant double-clic sur le fichier **MassAnsSpring.xhtml** qui est extrait du fichier compressé. □

Une caractéristique pédagogique importante est que c'est très facile de distribuer votre code source de simulation pour que d'autres personnes puissent l'utiliser avec *EjsS* à tout moment pour examiner, modifier ou adapter le modèle. (*EjsS* doit, évidemment, être installé.) *EjsS* écrit toutes les informations de ses panneaux de travail dans un petit fichier de description Extensible Markup Language (XML) – Langage de Balisage Extensible – et

il peut aussi créer un unique fichier compressé ZIP avec toute l'information XML et tous les fichiers de ressources (tels que les images) que vous avez utilisés dans votre simulation. Ce fichier ZIP peut alors être distribué et, si vous essayez de l'ouvrir avec *EjsS*, cela extraira les fichiers nécessaires du ZIP et copiera les fichiers dans un dossier de votre environnement de travail et téléchargera *EjsS* avec la simulation. Si un modèle du même nom existe déjà, il peut être remplacé. L'utilisateur peut alors examiner, exécuter ou modifier le modèle exactement comme nous avons fait dans ce chapitre. Un étudiant peut, par exemple, obtenir un exemple ou un modèle procédant d'un enseignant et il peut plus tard compresser à nouveau les sources du modèle modifié dans un nouveau fichier ZIP pour le rendre comme un exercice complété.

### Exercice 3.3. Compresser et extraire les fichiers source d'un modèle

Faites un clic droit sur l'icône *Package* et sélectionnez l'option *ZIP the simulation source files* (les fichiers sources de la simulation) sur le menu qui s'affiche. Ouvrez le fichier ZIP créé avec *EjsS* et sélectionnez un dossier de destination (différent de **JavascriptExamples**) dans votre environnement de travail pour que *EjsS* décompresse les sources. Remarquez que *EjsS* copie pour vous tous les fichiers de l'exemple original et ouvre le nouveau fichier XML **MassAndSpring.ejss**. □

*EjsS* est conçu comme outil de modélisation et d'édition-crédation, et nous vous suggérons que vous expérimentiez avec celui-ci pour apprendre à créer et distribuer vos propres modèles. Pour commencer, nous vous recommandons que vous exécutiez la simulation masse-ressort et que vous examiniez les activités de la deuxième page du panneau *Description*. Nous modifierons cette simulation dans la section suivante.

## 3.4 MODIFIER LA SIMULATION

Comme nous avons vu, un aspect important et caractéristique de *Easy Java/JavaScript Simulations* c'est qu'il nous permet de créer et étudier une simulation à un haut niveau d'abstraction. Nous avons examiné un modèle existant de masse-ressort et son interface utilisateur dans la section antérieure. Nous allons maintenant illustrer les possibilités additionnelles de *Easy Java/JavaScript Simulations* en incorporant une friction et une force motrice et en incorporant une visualisation de l'espace des phases du système.

### 3.4.1 Étendre le modèle

Nous pouvons incorporer un amortissement dans notre modèle en introduisant une force visqueuse (Loi de Stoke) qui est proportionnelle au négatif de la vitesse  $F_f = -b v_x$  dans laquelle  $b$  est le coefficient amortisseur. Nous pouvons aussi incorporer une force motrice externe et temporellement dépendante qui prend la forme d'une fonction sinusoïdale  $F_e(t) = A \sin(\omega t)$ . L'introduction de ces deux forces change l'équation différentielle de second ordre (3.1.1) à

$$\frac{d^2 x}{dt^2} = -\frac{k}{m} (x - L) - \frac{b}{m} \frac{dx}{dt} + \frac{1}{m} F_e(t), \quad (3.4.1)$$

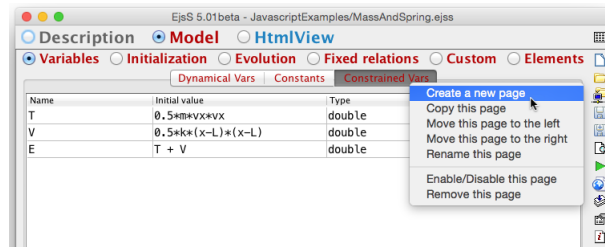
ou, comme dans les équations (3.1.2) et (3.1.3) :

$$\frac{dx}{dt} = v_x, \quad (3.4.2)$$

$$\frac{dv_x}{dt} = -\frac{k}{m} (x - L) - \frac{b}{m} v_x + \frac{1}{m} F_e(t). \quad (3.4.3)$$

#### 3.4.1.1 Incorporer des variables

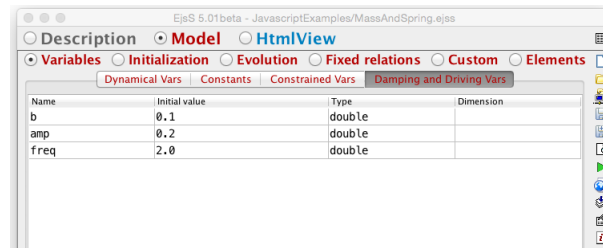
L'introduction de nouveaux termes de force nécessite de l'incorporation de variables pour le coefficient de friction dynamique et pour l'amplitude et la fréquence de la force motrice sinusoïdale. Retournez sur le panneau *Model* de *EjsS* et sélectionnez le panneau *Variables*. Faites un clic-droit sur la barre de la page existante de variables pour voir le menu déroulant, comme dans la figure 3.10. Sélectionnez l'entrée *Add a new page* (ajoutez une nouvelle page) comme indiqué dans la figure 3.10. Tapez **Damping and Driving Vars** pour dénommer le nouveau panneau et un panneau vide s'affichera.



**Figure 3.10:** Le menu déroulant pour une page de variables.

Nous allons maintenant utiliser le nouveau panneau pour déclarer les variables nécessaires. Nous pourrions avoir utilisé des panneaux déjà existants mais déclarer des pages multiples nous aidera à organiser les variables par catégories. Faites double clic sur une cellule du panneau pour l'éditer

et naviguer sur le panneau en utilisant les flèches et la touche tabulation. Tapez **b** dans la cellule *Name* (nom) de la première ligne et introduisez la valeur 0.1 dans la cellule *Initial value* (valeur initiale) à droite. Vous ne devez faire rien de plus car le type **double** sélectionné est correct. *EjsS* vérifie la syntaxe de la valeur introduite et l'évalue. Si nous introduisons une valeur fausse, le fond de la cellule de valeur s'affichera en rose. Remarquez que quand vous complétez le nom d'une variable, une nouvelle ligne s'affiche automatiquement. Procédez de manière identique pour déclarer une nouvelle variable pour l'amplitude **amp** de la force motrice avec une valeur 0.2 et pour sa fréquence **freq** avec une valeur 2.0. Expliquez la définition de ces variables en tapant un petit commentaire pour chacune d'entre elles dans la partie inférieure du panneau. Notre panneau final de variables est représenté dans la figure 3.11. Vous pouvez ignorer les lignes vides à la fin du panneau ou les éliminer en faisant un clic droit sur chaque ligne et en sélectionnant *Delete* du menu déroulant qui s'affiche.



**Figure 3.11:** Le nouveau panneau avec les termes d'amortissement et de force.

### 3.4.1.2 Modifier l'évolution

Maintenant nous allons modifier l'équation différentielle de la page d'évolution en incorporant des expressions pour les nouveaux termes de l'équation (3.4.3). Dirigez-vous au panneau evolution, faites double clic sur la cellule *Rate* de la deuxième équation et éditez-la pour lire :

$$-k/m * (x-L) - b*vx/m + force(t)/m$$

Remarquez que nous utilisons une fonction intitulée **force** qui n'a pas encore été définie. Nous pourrions avoir utilisé une expression explicite pour la fonction sinusoïdale. Cependant définir la méthode **force** favorise un code plus claire et plus lisible et qui nous permet d'introduire des fonctions personnalisées.



### 3.4.1.3 Incorporer un code personnalisé

La fonction **force** est définie en utilisant le panneau *Custom* de *Model*. Dirigez-vous sur ce panneau et cliquez sur la zone centrale vide pour créer une nouvelle page de code personnalisé. Intitulez cette page *force*. Vous remarquerez que cette page est créée avec un modèle de code qui définit la fonction. Éditez ce code pour lire :

```
function force (time) {  
    return amp*Math.sin(freq*time); // sinusoidal driving force  
}
```

Tapez exactement ce code tel qu'il est indiqué y compris les majuscules car les compilateurs réagissent négativement aux erreurs de syntaxe.

Remarquez que nous indiquons le temps auquel nous voulons calculer la force motrice à la fonction **force** comme un paramètre d'entrée. Indiquer la valeur de temps est très important. Ce serait incorrect de demander à la méthode d'utiliser la valeur de la variable **t** comme dans :





```
function force () { // incorrect implementation of the force method  
    return amp*Math.sin(freq*t);  
}
```

La raison pour laquelle le temps doit être indiqué dans méthode est que le temps change au cours de l'étape evolution. Pour que le solutionneur EDO calcule correctement la force temporellement dépendante pendant l'étape evolution, le temps doit être indiqué dans la méthode qui calcule la valeur.

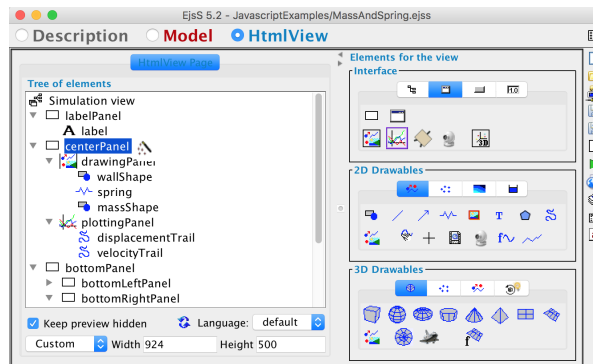
Les variables qui changent (évoluent) doivent être indiquées dans les méthodes qui sont utilisées pour calculer la valeur car les solutionneurs numériques évaluent la colonne *Rate* (valeur) dans le panneau EDO avec des valeurs intermédiaires entre  $t$  et  $t + dt$ . Autrement dit, la variable indépendante et tout autre variable dynamique qui est différenciée dans la colonne *State* de l'éditeur EDO doivent être indiquées pour toute méthode qui est signalée dans la colonne *Rate*. Les variables qui demeurent constantes pendant l'étape evolution peuvent s'utiliser sans être indiqué comme paramètre d'entrée car la valeur de la variable au début de l'étape evolution peut être utilisée.

### 3.4.2 Améliorer l'affichage

Nous allons maintenant introduire une visualisation de l'espace des phases (déplacement versus vitesse) de l'évolution du système dans *HtmlView*. Nous allons aussi incorporer de nouveaux champs de saisie pour afficher et modifier la valeur des paramètres d'amortissement, amplitude et fréquence.

Dirigez-vous au panneau *HtmlView* et remarquez que la palette *Interface* contient plusieurs sous-panneaux. Cliquez sur l'onglet avec l'icône  pour afficher la palette *Windows, containers and drawing panels* des éléments d'affichage. Cliquez sur l'icône  pour obtenir un panneau de représentation de tracés dans cette palette, . Vous pouvez poser (maintenir) le curseur de la souris sur un icône pour afficher une indication qui décrit l'élément si vous avez des problèmes pour reconnaître cet icône. Quand vous sélectionnez un élément, une bordure en couleur autour de l'icône de la palette s'active et transforme le curseur en une baguette magique, . Ces changements indiquent que *EjsS* est prêt pour créer un élément du type sélectionné. (Retourner au mode design — faites disparaître la baguette magique — en cliquant sur n'importe quelle zone blanche sur the *Tree of Elements* ou appuyez sur la touche *Esc*.)

Cliquez sur l'élément *Simulation view* intitulé `centerPanel` comme indiqué dans la figure 3.12 pour incorporer le panneau de traçage à l'affichage (comme enfant de ce panneau).

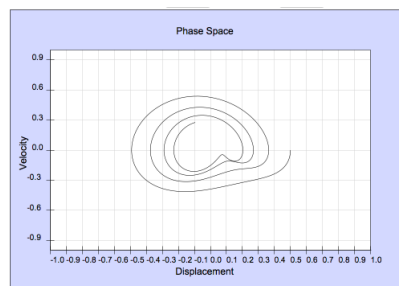


**Figure 3.12:** Création d'un panneau de traçage comme un nouveau (et dernier) enfant du panneau central.

*EjsS* vous demandera le nom du nouveaux élément et il crée alors l'élément comme un nouvel enfant du panneau central. Un nouveau *plot* (traçage) s'affiche sur la droite de l'antérieur mais la fenêtre du navigateur est peut-être trop petite et c'est possible qu'il l'affiche sous les autres deux


panneaux. (Il y a d'autres manières d'améliorer cette configuration mais nous ne nous occuperons pas de cela maintenant.) Redimensionner la fenêtre de votre navigateur et changer la dimension de la prévisualisation en éditant les champs *Width* et *Height* (largeur et hauteur) de la partie inférieure de l'arbre des éléments *HtmlView*. La dimension de la fenêtre est utile pour vous orienter sur l'aspect de la simulation finale sur des dispositifs qui ont une résolution de l'écran fixe (tels que les tablettes). Finalement éditez le panneau des propriétés de l'élément récemment créé du panneau de traçage pour fixer la propriété *Title* à **Phase Space**, la propriété *TitleX* à **Displacement** et la propriété *TitleY* à **Velocity**. (*EjsS* incorpore les citations principales et de traçage à ces ressorts pour conformer la syntaxe JavaScript correcte). Fixez le minimum et le maximum des échelles  $x$  et  $y$  à  $-1$  et  $1$ , respectivement, et ne variez pas les autres propriétés.

Le panneau de représentation de tracés est, comme son nom l'indique, le conteneur pour la tracé de l'espace des phases. Les données de l'espace des phases sont représentées sur ce panneau en utilisant un élément 2D du type *Trail*, [S](#). Retrouver l'élément *Trail* sur la palette *2D Drawables* et suivez le même processus qu'antérieurement. Sélectionnez l'élément *Trail* et créez un élément de ce type en cliquant avec la baguette magique sur le nouveau panneau de traçage d'espace des phases. Finalement éditez les propriétés du nouveau élément *Trail* pour fixer la propriété *InputX* à **x - L** et la propriété *InputY* à **vx**. Cette attribution fera que la simulation incorporera un nouveau point ( $x - L$ ,  $vx$ ) au traçage après chaque étape evolution et par la suite réalisera le traçage de l'espace des phases représenté dans la figure 3.13.




**Figure 3.13:** Le nouveau traçage de l'espace des phases incorporé à l'affichage de la simulation.

Pour finir les modifications nous incorporons un nouveau panneau qui affiche les paramètres de la force motrice sinusoïdale.

- Sélectionnez l'icône élément **Panel**, , sur le sous-groupe *Windows containers and drawing panels* de la palette *Interface*. Cliquez avec la baguette magique sur le nœud racine **Simulation view** dans le *Tree*

of *Elements* pour créer un nouveau panneau intitulé `forceParamPanel` comme étant son dernier enfant. Faites un clic-droit et utilisez l'option *Move up* pour le placer avant le panneau de traçage de l'espace des phases. (Vous pouvez aussi le glisser et le déposer dans sa nouvelle position sur l'arbre des éléments.)

- Sélectionnez l'icône élément **Label**, **A**, sur le sous-groupe *Buttons and decorations* de la palette *Interface* et créez un nouveau élément de ce type dans le panneau `forceParamPanel` et fixez la propriété du texte de l'étiquette à `"frequency ="`.
- Sélectionnez l'icône élément **Field**, **20**, et créez un nouveau élément intitulé `freqField` dans le panneau de paramètre de la force. Éditez le panneau des propriétés de `freqField` tel que indiqué dans la figure 3.14. La connexion à la variable `freq` est établie en utilisant la propriété *Value*. Cliquez sur le deuxième icône de la droite du champ propriété, , et choisissez la variable appropriée. La liste des variables affiche toutes les variables de modèles qui peuvent être utilisés pour fixer le champ propriété. La propriété *Format* indique le nombre de chiffres décimaux avec lesquels nous pouvons afficher la valeur de la variable.
- Répéter ce processus pour incorporer la variable `amp` à l'interface utilisateur

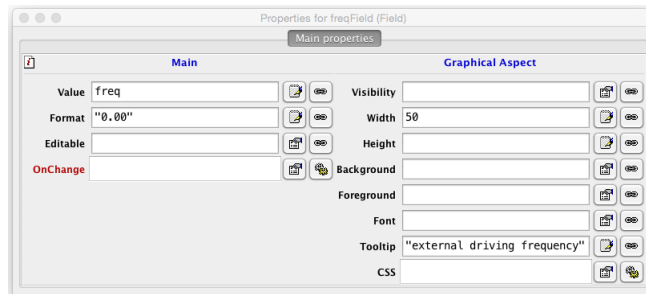





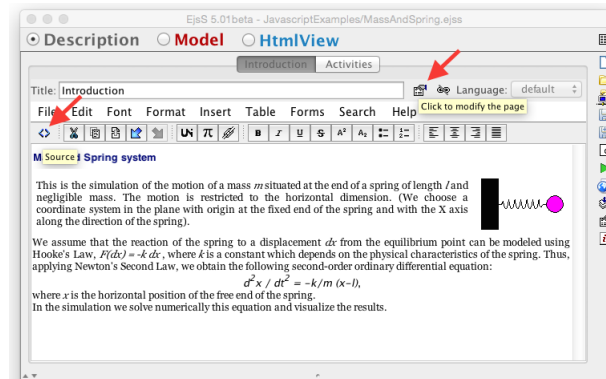
Figure 3.14: Le panneau des propriétés de l'élément `freqField`.

### 3.4.3 Changer la description


Une fois que nous avons changé le modèle et l'affichage, nous devons modifier les pages de description de notre simulation. Dirigez-vous au panneau **Description** et cliquez sur l'onglet de la première page, celui intitulé **Introduction**. Une fois que vous visualisez cette page, cliquez sur l'icône *Click to modify the page*, . La page description changera pour éditer le mode comme indiqué dans la figure 3.15 et un éditeur simple s'affichera pour vous fournir un accès direct aux caractéristiques HTML plus habituelles.

Si vous préférez utiliser votre propre éditeur vous pouvez copier-coller

des fragments HTML de votre éditeur sur l'éditeur de *EjsS*. Si vous connaissez la syntaxe HTML, vous pouvez introduire directement un texte étiqueté (balisé) en cliquant sur l'icône *Source*, , de la barre des outils. Vous pouvez même importer des pages HTML complètes à *EjsS* en cliquant sur l'icône *Link/Unlink page to external file*, .



**Figure 3.15:** L'éditeur HTML de *EjsS*. Les flèches rouges supplémentaires signalent les icônes de mode d'édition et le code d'édition et source.

Éditez les pages de description selon votre convenance. Changez au moins le texte du modèle pour incorporer les forces motrices et d'amortissements. Une fois que cela soit fait, sauvegardez la nouvelle simulation sous un nom différent en cliquant sur l'icône *Save as* de la barre d'outils de *EjsS*, . Lorsqu'on vous le demande, introduisez un nouveau nom pour le fichier XML de votre simulation. La simulation modifiée est enregistrée dans le fichier **MassAndSpringComplete.ejss** dans le répertoire **source** pour ce chapitre.

### 3.5 PROBLEMS ET PROJETS

**Problem 3.1 (Énergie).** Incorporez un troisième panneau de traçage à la fenêtre de dialogue de la simulation **MassAndSpringComplete.ejss** qui affichera l'évolution des énergies cinétiques, potentielles et totales.

**Problem 3.2 (Traceur de fonction).** La solution analytique pour un oscillateur harmonique simple déroulé est

$$x(t) = A \sin(w_0 t + \phi) \quad (3.5.1)$$

dans laquelle  $A$  est l'amplitude (déplacement maximum),  $w_0 = \sqrt{k/m}$  est la fréquence naturelle d'oscillation et  $\phi$  est l'angle de phase. Consultez un manuel de mécanique pour déterminer la relation entre l'amplitude et la phase d'angle et le déplacement initial et la vitesse. Utilisez la simulation **FunctionPlotter.ejss** dans les répertoires exemples pour comparer la so-

lution analytique à la solution numérique générée par le modèle **MassAndSpringComplete.ejss**.

**Project 3.1** (Oscillateur bidimensionnel). Modifiez le modèle de la simulation masse-ressort pour considérer que le mouvement n'est pas restreint à la direction horizontale. Assumez que un deuxième ressort avec une constante de ressort  $k'$  produit une force de restauration verticale  $F_y(\delta y) = -k' \delta y$ . Modifiez la simulation pour permettre à l'utilisateur de préciser les constantes de la Loi de Hooke ainsi que les conditions initiales dans les deux directions. Décrivez le mouvement produit sans une force motrice mais sous de différentes conditions initiales et avec des constantes de ressort différentes. (Essayez  $k = 1$  et  $k' = 9$ .) Démontrez qu'il est possible d'obtenir un mouvement circulaire si  $k = k'$ .

**Project 3.2** (Une pendule simple). Créez une simulation similaire à celle qui est décrite dans ce chapitre pour une pendule dont l'équation différentielle de second ordre de mouvement est

$$\frac{d^2\theta}{dt^2} = -\frac{g}{L} \sin(\theta), \quad (3.5.2)$$

dans laquelle  $\theta$  est l'angle de la pendule avec la verticale,  $g$  l'accélération causée par la gravité et  $L$  est la longueur des bras de leviers. Utilisez des relations fixes pour calculer la position de  $x$  et de  $y$  du balancier de la pendule en utilisant les équations :

$$\begin{aligned} x &= L \sin(\theta) \\ y &= -L \cos(\theta). \end{aligned}$$

