# DiverSE Coffee

Retrospective 10 years

# Triskell 2012


- Research Program (RA 2012)
  - Model Driven Engineering for Distributed Software
  - Software Product Lines
  - Object-Oriented Software Engineering
  - Design Pattern
  - Component
  - Contracts
  - Models and Aspects
  - Design and Aspect Weaving
  - Model Driven Engineering

**Software Language Engineering (SLE)**

Separation of concerns in the development of complex software-intensive systems leads to the use of various domain-specific modeling languages (DSMLs). SLE addresses the whole life cycle for designing, implementing and relating DSMLs to support heterogeneous modeling and analysis.

**Variability**

The systematic modeling of variability in software systems has emerged as an effective approach to document and reason about software evolutions and heterogeneity. Variability modeling characterizes an “envelope” of possible software variations.

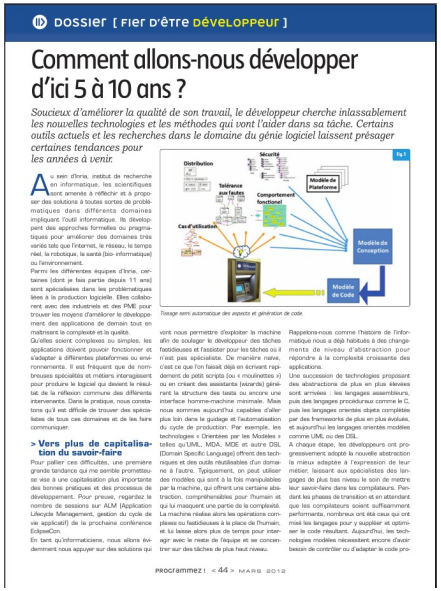
**Dynamic Adaptive System (DAS)**

Flexible yet dependable systems have to cope with heterogeneous hardware execution platforms ranging from smart sensors to huge computation infrastructures and data centers. We need to devise formalisms to reason about the impact of an evolution and about the transition from one configuration to another.

**Diversity**

Software diversity as the foundation for a novel software design principle and increased adaptive capacities in complex adaptive systems. Higher levels of diversity in the system provide a pool of software solutions that can eventually be used to adapt to unforeseen situations at design time.

- Programmez! Numéro #150 (01/03/2012)
  - Dossier “Fier d’être développeur !”

DOSSIER [ FIER D’ÊTRE DÉVELOPPEUR ]

## Comment allons-nous développer d’ici 5 à 10 ans ?

*Soucieux d’améliorer la qualité de son travail, le développeur cherche inlassablement les nouvelles technologies et les méthodes qui vont l’aider dans sa tâche. Certains outils actuels et les recherches dans le domaine du génie logiciel laissent présager certaines tendances pour les années à venir.*

Au sein d’Inria, institut de recherche en informatique, les scientifiques sont amenés à réfléchir et à proposer des solutions à toutes sortes de problématiques dans différents domaines impliquant l’outil informatique. Ils développent des approches formelles ou pragmatiques pour améliorer des domaines très variés tels que l’Internet, le réseau, le temps réel, la robotique, la santé (bio-informatique) ou l’environnement.

Parmi les différentes équipes d’Inria, certaines (dont je fais partie depuis 11 ans) sont spécialisées dans les problématiques liées à la production logicielle. Elles collaborent avec des industriels et des PME pour trouver les moyens d’améliorer le développement des applications de demain tout en maîtrisant la complexité et la qualité.

Qu’elles soient complexes ou simples, les applications doivent pouvoir fonctionner et s’adapter à différentes plateformes ou environnements. Il est fréquent que de nombreuses spécialités et métiers interagissent pour produire le logiciel qui devient le résultat de la réflexion commune des différents intervenants. Dans la pratique, nous constatons qu’il est difficile de trouver des spécialistes de tous ces domaines et de les faire communiquer.

### > Vers plus de capitalisation du savoir-faire

Pour pallier ces difficultés, une première grande tendance qui me semble prometteuse vise à une capitalisation plus importante des bonnes pratiques et des processus de développement. Pour preuve, regardez le nombre de sessions sur ALM (Application Lifecycle Management, gestion du cycle de vie applicatif) de la prochaine conférence EclipseCon.

En tant qu’informaticiens, nous allons évidemment nous appuyer sur des solutions qui vont nous permettre d’exploiter la machine afin de soulager le développeur des tâches fastidieuses et l’assister pour les tâches où il n’est pas spécialiste. De manière naïve, c’est ce que l’on faisait déjà en écrivant rapidement de petit scripts (ou « moulinettes ») ou en créant des assistants (wizards) générant la structure des tests ou encore une interface homme-machine minimale. Mais nous sommes aujourd’hui capables d’aller plus loin dans le guidage et l’automatisation du cycle de production. Par exemple, les technologies « Orientées par les Modèles » telles qu’UML, MDA, MDE et autre DSL (Domain Specific Language) offrent des techniques et des outils réutilisables d’un domaine à l’autre. Typiquement, on peut utiliser des modèles qui sont à la fois manipulables par la machine, qui offrent une certaine abstraction, compréhensibles pour l’humain et qui lui masquent une partie de la complexité. La machine réalise alors les opérations complexes ou fastidieuses à la place de l’humain, et lui laisse alors plus de temps pour interagir avec le reste de l’équipe et se concentrer sur des tâches de plus haut niveau.



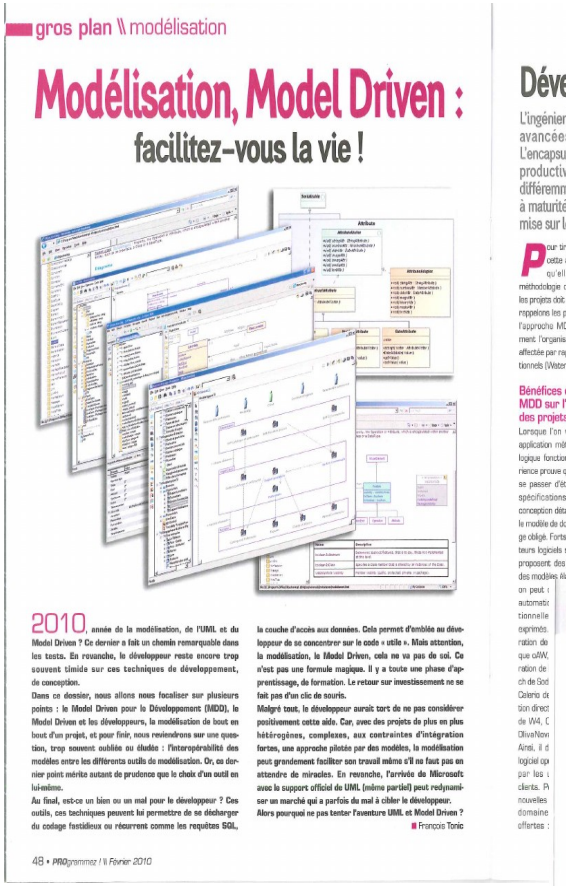

*Tissage semi automatique des aspects et génération de code.*

Rappelons-nous comme l’histoire de l’informatique nous a déjà habitués à des changements de niveau d’abstraction pour répondre à la complexité croissante des applications.

Une succession de technologies proposant des abstractions de plus en plus élevées sont arrivées : les langages assembleurs, puis des langages procéduraux comme le C, puis les langages orientés objets complétés par des frameworks de plus en plus évolués, et aujourd’hui les langages orientés modèles comme UML ou des DSL.

A chaque étape, les développeurs ont progressivement adopté la nouvelle abstraction la mieux adaptée à l’expression de leur métier, laissant aux spécialistes des langages de plus bas niveau le soin de mettre leur savoir-faire dans les compilateurs. Pendant les phases de transition et en attendant que les compilateurs soient suffisamment performants, nombreux ont été ceux qui ont mixé les langages pour y suppléer et optimiser le code résultant. Aujourd’hui, les technologies modèles nécessitent encore d’avoir besoin de contrôler ou d’adapter le code pro-

# Aute exemple de titres des années 2010...



gros plan \\ modélisation

## Modélisation, Model Driven : facilitez-vous la vie !

2010, année de la modélisation, de l'UML et du Model Driven ? Ce dernier a fait un chemin remarquable dans les tests. En revanche, le développeur reste encore trop souvent timide sur ces techniques de développement, de conception.

Dans ce dossier, nous allons nous focaliser sur plusieurs points : le Model Driven pour le Développement (MDD), le Model Driven et les développeurs, la modélisation de bout en bout d'un projet, et pour finir, nous reviendrons sur une question, trop souvent oubliée ou éludée : l'interopérabilité des modèles entre les différents outils de modélisation. Or, ce dernier point mérite autant de prudence que le choix d'un outil en lui-même.

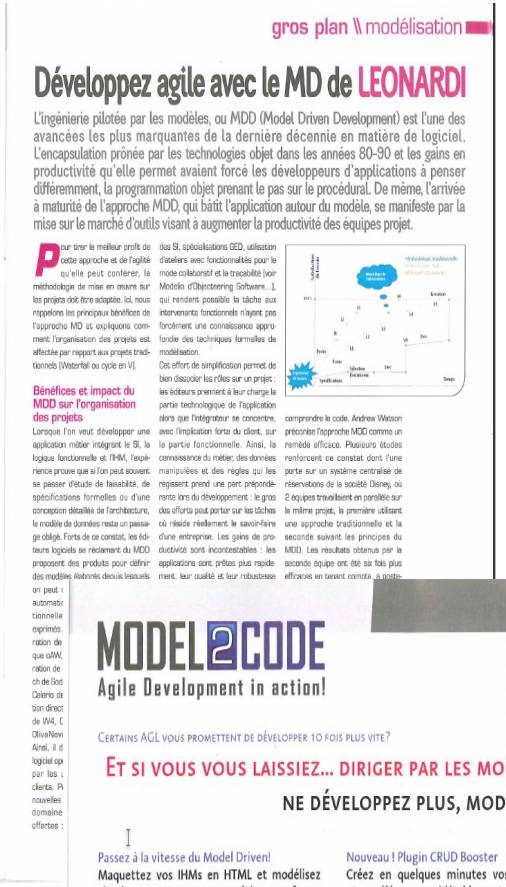
Au final, est-ce un bien ou un mal pour le développeur ? Ces outils, ces techniques peuvent lui permettre de se décharger du codage fastidieux ou récurrent comme les requêtes SQL, la couche d'accès aux données. Cela permet d'emblée au développeur de se concentrer sur le code « utile ». Mais attention, la modélisation, le Model Driven, cela ne va pas de soi. Ce n'est pas une formule magique. Il y a toute une phase d'apprentissage, de formation. Le retour sur investissement ne se fait pas d'un clic de souris.

Malgré tout, le développeur aurait tort de ne pas considérer positivement cette aide. Car, avec des projets de plus en plus hétérogènes, complexes, aux contraintes d'intégration fortes, une approche pilotée par des modèles, la modélisation peut grandement faciliter son travail même s'il ne faut pas en attendre de miracles. En revanche, l'arrivée de Microsoft avec le support officiel du UML (même partiel) peut redynamiser un marché qui a parfois du mal à cibler le développeur.

Alors pourquoi ne pas tenter l'aventure UML et Model Driven ?

■ François Tonic

48 • PROgrammez ! \\ Février 2010



gros plan \\ modélisation

## Développez agile avec le MD de LEONARDI

L'ingénierie pilotée par les modèles, ou MDD (Model Driven Development) est l'une des avancées les plus marquantes de la dernière décennie en matière de logiciel. L'encapsulation prônée par les technologies objet dans les années 80-90 et les gains en productivité qu'elle permet avaient forcé les développeurs d'applications à penser différemment, la programmation objet prenant le pas sur le procédural. De même, l'arrivée à maturité de l'approche MDD, qui bâtit l'application autour du modèle, se manifeste par la mise sur le marché d'outils visant à augmenter la productivité des équipes projet.

Pour tirer le meilleur profit de cette approche et de l'agilité qu'elle peut conférer, la méthodologie de mise en œuvre sur les projets doit être adaptée. Ici, nous rappelons les principaux bénéfices de l'approche MD et expliquons comment l'organisation des projets est affectée par rapport aux projets traditionnels (Waterfall ou cycle en V).

### Bénéfices et impact du MDD sur l'organisation des projets

Lorsque l'on veut développer une application métier intégrant le SI, la logique fonctionnelle et, l'IHM, l'expérience prouve que si l'on peut souvent se passer d'étude de faisabilité, de spécifications formelles ou d'une conception détaillée de l'architecture, le modèle de données reste un passage obligé. Forts de ce constat, les éditeurs logiciels se réclamant du MDD proposent des produits pour définir des modèles. Abonnés depuis lesquels on peut

automatic

tionnelle

exprimés.

ration de

que oAW,

ration de

ch de Sod

Caleria de

tion direct

de WA, C

OliveNov

Ainsi, il d

logiciel op

par les

clients. Pa

nouvelles

domaine

offertes :

des SI, spécialisations GED, utilisation d'ateliers avec fonctionnalités pour le mode collaboratif et la traçabilité (voir Modello d'Objecteering Software...), qui rendent possible la tâche aux intervenants fonctionnels n'ayant pas forcément une connaissance approfondie des techniques formelles de modélisation.

Cet effort de simplification permet de bien dissocier les rôles sur un projet : les éditeurs prennent à leur charge la partie technologique de l'application alors que l'intégrateur se concentre, avec l'implication forte du client, sur la partie fonctionnelle. Ainsi, la connaissance du métier, des données manipulées et des règles qui les régissent prend une part prépondérante lors du développement : le gros des efforts peut porter sur les tâches où réside réellement le savoir-faire d'une entreprise. Les gains de productivité sont incontestables : les applications sont prêtes plus rapidement, leur qualité et leur robustesse



comprendre le code. Andrew Watson préconise l'approche MDD comme un remède efficace. Plusieurs études renforcent ce constat dont l'une porte sur un système centralisé de réservations de la société Disney, où 2 équipes travaillaient en parallèle sur le même projet, la première utilisant une approche traditionnelle et la seconde suivant les principes du MDD. Les résultats obtenus par la seconde équipe ont été six fois plus efficaces en tenant compte, a poste-

MODEL2CODE

Agile Development in action!

CERTAINS AGL VOUS PROMETTENT DE DÉVELOPPER 10 FOIS PLUS VITE?

ET SI VOUS VOUS LAISSIEZ... DIRIGER PAR LES MODÈLES ?

NE DÉVELOPPEZ PLUS, MODÉLISEZ !

Passez à la vitesse du Model Driven!

Maquettez vos IHMs en HTML et modélisez simplement vos processus métier sous forme de diagrammes UML, nos générateurs les transforment instantanément en application Spring, Java EE ou Flex, prête à être déployée !

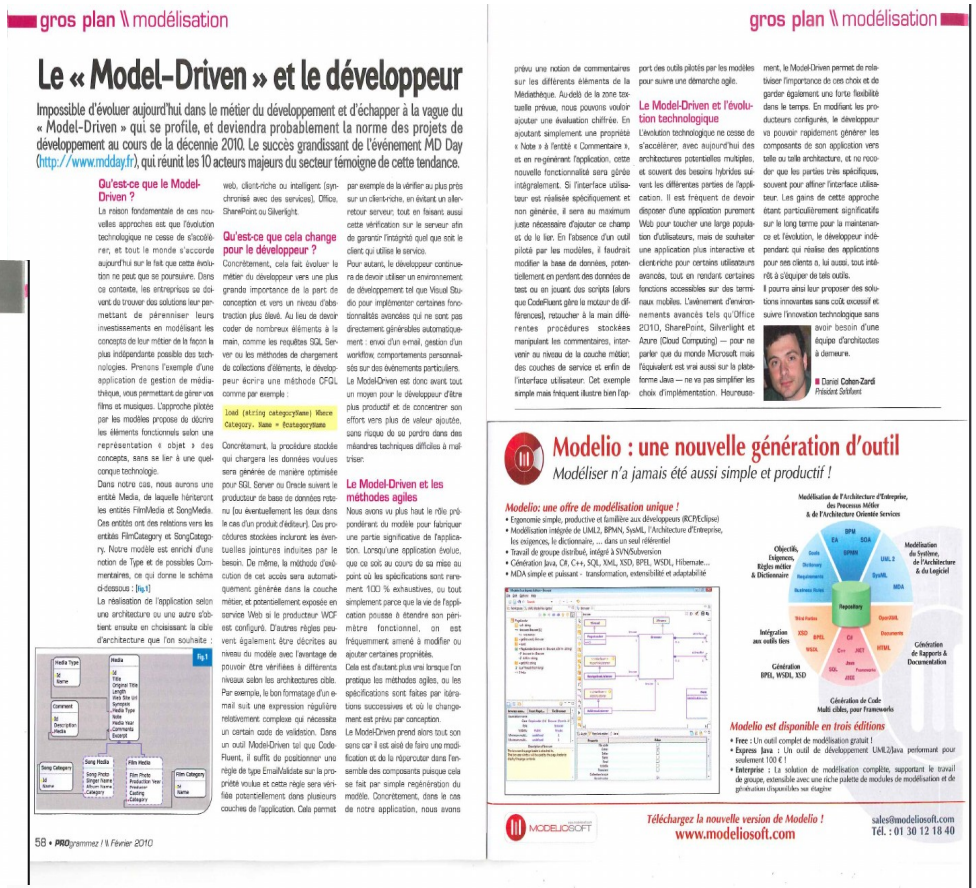
Nouveau ! Plugin CRUD Booster

Créez en quelques minutes vos prototypes et accélérez considérablement vos projets applicatifs. Générez une application CRUD fonctionnelle (y compris les IHMs) à partir d'un simple diagramme de classe !

VERSIONS D'ÉVALUATION GRATUITES & TUTORIAUX :

WWW.MODEL2CODE.COM

Model2Code propose des ateliers agiles et collaboratifs de génération d'applications web et riches, combinant les meilleures technologies Model Driven du marché : l'outil de modélisation UML Magicdraw® associé au moteur de génération par transformation de modèles BLU AGE®.



gros plan \\ modélisation

## Le « Model-Driven » et le développeur

Impossible d'évoluer aujourd'hui dans le métier du développement et d'échapper à la vague du « Model-Driven » qui se profile, et deviendra probablement la norme des projets de développement au cours de la décennie 2010. Le succès grandissant de l'événement MD Day (http://www.mdday.fr), qui réunit les 10 acteurs majeurs du secteur témoigne de cette tendance.

### Qu'est-ce que le Model-Driven ?

La raison fondamentale de ces nouvelles approches est que l'évolution technologique ne cesse de s'accélérer, et tout le monde s'accorde aujourd'hui sur le fait que cette évolution ne peut que se poursuivre. Dans ce contexte, les entreprises se doivent de trouver des solutions leur permettant de pérenniser leurs investissements en modélisant les concepts de leur métier de la façon la plus indépendante possible des technologies. Prenons l'exemple d'une application de gestion de médiathèque, vous permettant de gérer vos films et musiques. L'approche pilotée par les modèles propose de décrire les éléments fonctionnels selon une représentation « objet » des concepts, sans se lier à une quelconque technologie.

Dans notre cas, nous aurons une entité Media, de laquelle hériteront les entités FilmMedia et SongMedia. Ces entités ont des relations vers les entités FilmCategory et SongCategory. Notre modèle est enrichi d'une notion de Type et de possibles Commentaires, ce qui donne le schéma ci-dessous : [fig.1]

La réalisation de l'application selon une architecture ou une autre s'obtient ensuite en choisissant la cible d'architecture que l'on souhaite : web, client-riche ou intelligent (synchronisé avec des services), Office, SharePoint ou Silverlight.

### Qu'est-ce qui cela change pour le développeur ?

Concrètement, cela fait évoluer le métier du développeur vers une plus grande importance de la part de conception et vers un niveau d'abstraction plus élevé. Au lieu de devoir coder de nombreux éléments à la main, comme les requêtes SQL Server ou les méthodes de chargement de collections d'éléments, le développeur écrira une méthode CFQL comme par exemple :

```
Load (string categoryName) Where
Category. Name = $categoryName
```

Concrètement, la procédure stockée qui chargera les données voulues sera générée de manière optimisée pour SGL Server ou Oracle suivant le producteur de base de données retenu (ou éventuellement les deux dans le cas d'un produit d'éditeur). Ces procédures stockées incluront les éventuelles jointures induites par le besoin. De même, la méthode d'exécution de cet accès sera automatiquement générée dans la couche métier, et potentiellement exposée en service Web si le producteur WCF est configuré. D'autres règles peuvent également être décrites au niveau du modèle avec l'avantage de pouvoir être vérifiées à différents niveaux selon les architectures cible. Par exemple, le bon formatage d'un e-mail suit une expression régulière relativement complexe qui nécessite un certain code de validation. Dans un outil Model-Driven tel que Code-Fluent, il suffit de positionner une règle de type EmailValidate sur la propriété voulue et cette règle sera vérifiée potentiellement dans plusieurs couches de l'application. Cela permet par exemple de la vérifier au plus près sur un client-riche, en évitant un aller-retour serveur, tout en faisant aussi cette vérification sur le serveur afin de garantir l'intégrité quel que soit le client qui utilise le service.

Pour autant, le développeur continuera de devoir utiliser un environnement de développement tel que Visual Studio pour implémenter certaines fonctionnalités avancées qui ne sont pas directement générables automatiquement : envoi d'un e-mail, gestion d'un workflow, comportements personnalisés sur des événements particuliers. Le Model-Driven est donc avant tout un moyen pour le développeur d'être plus productif et de concentrer son effort vers plus de valeur ajoutée, sans risque de se perdre dans des mêlandres techniques difficiles à maîtriser.

### Le Model-Driven et les méthodes agiles

Nous avons vu plus haut le rôle prépondérant du modèle pour fabriquer une partie significative de l'application. Lorsqu'une application évolue, que ce soit au cours du sa mise au point où les spécifications sont rarement 100 % exhaustives, ou tout simplement parce que la vie de l'application pousse à étendre son périmètre fonctionnel, on est fréquemment amené à modifier ou ajouter certaines propriétés.

Cela est d'autant plus vrai lorsque l'on pratique les méthodes agiles, ou les spécifications sont faites par itérations successives et où le changement est prévu par conception.

Le Model-Driven prend alors tout son sens car il est aisé de faire une modification et de la répercuter dans l'ensemble des composants puisque cela se fait par simple regénération du modèle. Concrètement, dans le cas de notre application, nous avons prévu une notion de commentaires sur les différents éléments de la Médiathèque. Au-delà de la zone textuelle prévue, nous pouvons vouloir ajouter une évaluation chiffrée. En ajoutant simplement une propriété « Note » à l'entité « Commentaire », et en regénérant l'application, cette nouvelle fonctionnalité sera gérée intégralement. Si l'interface utilisateur est réalisée spécifiquement et non générée, il sera au maximum juste nécessaire d'ajouter ce champ et de le lier. En l'absence d'un outil piloté par les modèles, il faudrait modifier la base de données, potentiellement en perdant des données de test ou en jouant des scripts (alors que CodeFluent gère le moteur de différences), retoucher à la main différentes procédures stockées manipulant les commentaires, intervenir au niveau de la couche métier, des couches de service et enfin de l'interface utilisateur. Cet exemple simple mais fréquent illustre bien l'apport des outils pilotés par les modèles pour suivre une démarche agile.

### Le Model-Driven et l'évolution technologique

L'évolution technologique ne cesse de s'accélérer, avec aujourd'hui des architectures potentielles multiples, et souvent des besoins hybrides suivant les différentes parties de l'application. Il est fréquent de devoir déployer d'une application purement Web pour toucher une large population d'utilisateurs, mais de souhaiter une application plus interactive et client-riche pour certains utilisateurs avancés, tout en rendant certaines fonctions accessibles sur des terminaux mobiles. L'avènement d'environnements avancés tels qu'Office 2010, SharePoint, Silverlight et Azure (Cloud Computing) — pour ne parler que du monde Microsoft mais l'équivalent est vrai aussi sur la plate-forme Java — ne va pas simplifier les choix d'implémentation. Heureusement, le Model-Driven permet de relativiser l'importance de ces choix et de garder également une forte flexibilité dans le temps. En modifiant les producteurs configurés, le développeur va pouvoir rapidement générer les composants de son application vers telle ou telle architecture, et ne recoder que les parties très spécifiques, souvent pour affiner l'interface utilisateur. Les gains de cette approche étant particulièrement significatifs sur le long terme pour la maintenance et, l'évolution, le développeur redépendant qui réalise des applications pour ses clients a, lui aussi, tout intérêt à s'équiper de tels outils.

Il pourra ainsi leur proposer des solutions innovantes sans coût excessif et suivre l'innovation technologique sans avoir besoin d'une équipe d'architectes à demeure.



■ Daniel Cohen-Zardi
Président Sofeliant

58 • PROgrammez ! \\ Février 2010







Modelio : une nouvelle génération d'outil

*Modéliser n'a jamais été aussi simple et productif !*

**Modelio: une offre de modélisation unique !**

- Ergonomie simple, productive et familière aux développeurs (RCP/Eclipse)
- Modélisation intégrée de UML2, BPMN, SysML, l'Architecture d'Entreprise, les exigences, le dictionnaire, ... dans un seul référentiel
- Travail de groupe distribué, intégré à SVN/Subversion
- Génération Java, C#, C++, SQL, XML, XSD, BPEL, WSDL, Hibernate,...
- MDA simple et puissant - transformation, extensibilité et adaptabilité

**Modelio est disponible en trois éditions**

- **Free :** Un outil complet de modélisation gratuit !
- **Express Java :** Un outil de développement UML2/Java performant pour seulement 100 € !
- **Entreprise :** La solution de modélisation complète, supportant le travail de groupe, extensible avec une riche palette de modules de modélisation et de génération disponibles sur étagère

MODELIOSOFT

*Téléchargez la nouvelle version de Modelio !*
www.modeliosoft.com

sales@modeliosoft.com
Tél. : 01 30 12 18 40

# Vision en 2012

# Capitalisation du savoir faire inter disciplines

- Inter actions entre experts de différents domaines
- Vers une capitalisation du savoir faire
  - Bonnes pratique et process de développment
    - ALM (Application Lifecycle Management)
  - IDM/MDE permet d’aller plus loin en permettant
    - une manipulation à la fois par les machines et par les humains
    - Réapplication de techniques inventées pour un domaine à de nouveaux domaines

# Utilisation d’abstractions

- Les développeurs sont habitués au changements de niveau d’abstraction pour gérer une complexité croissante
  - Assembleur > language procéduraux > language orientés objets + framework > langages orientés modèles / DSL
  - Utilisation de l’abstraction adaptée au besoin + utilisation  de compilateurs
    - Périodes de transitions pour pallier aux limites des compilateurs

# Separation of concerns

- Convergence modèle/code
  - Syntaxes textuelles / graphiques
- Découplage des préoccupations
  - Techniques orientés aspects
    - Au niveau code  (via des fonctionnalités de langage telles que scala, Ruby C#, ou via injection par des annotations)
    - Et généralisées à toutes les étapes de conception

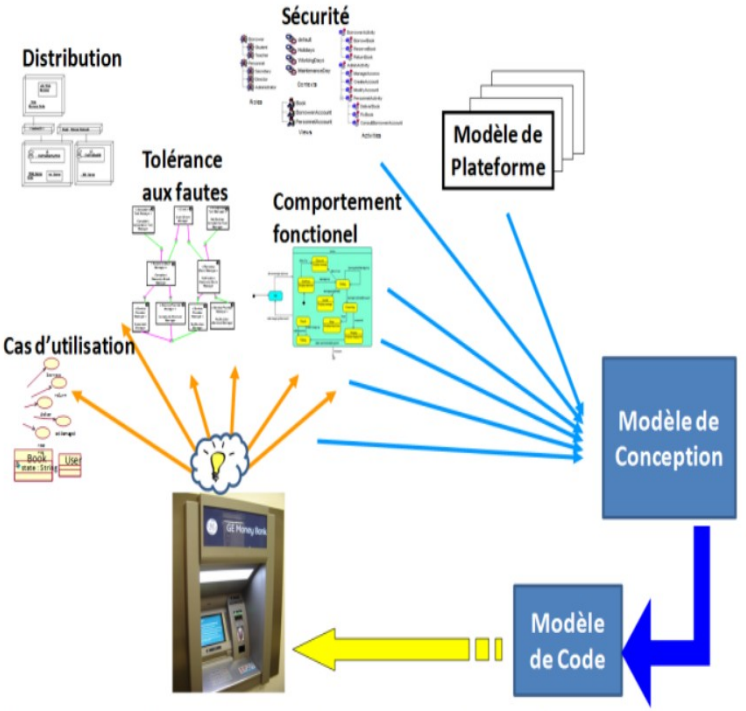



Figure 1 : Tissage semi automatique des aspects et génération de code

# Importance des processus

- Humain dans la boucle
  - Paramétrage, choix des outils, itérations successives sur les modèles
  - Gestion du cycle de développement (BPMN/SPEM/…)
  - Adaptabilité permettant la mise en oeuvre de ligne de produit logicielle

# Vers plus de fiabilité

- support aux systèmes dynamiques
- Adaptation @DesignTime (grace au MDE dans les phases de conceptions)
- Adaptation @Runtime
  - Reconfiguration à chaud

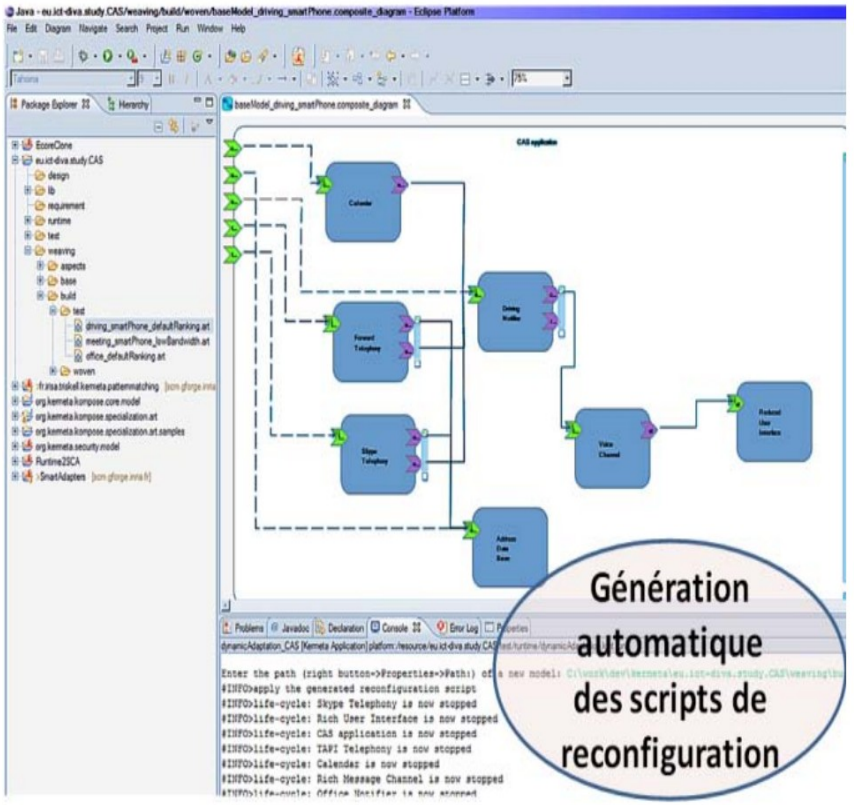



Figure 2 : Reconfiguration dynamique des fonctions d'un téléphone pour s'adapter à une nouvelle situation.

# Conclusion de 2012

- Technique permettant de répondre aux enjeux de:
  - Accélération des technologies
    - Multiplication des architectures
  - Besoins hybrides
  - Complexité croissante / spécialisation croissante
  - Maitrise du processus dans son ensemble
    - Modularité des outils

# Et aujourd'hui ?

## Sujet "modelling" dans cette revue

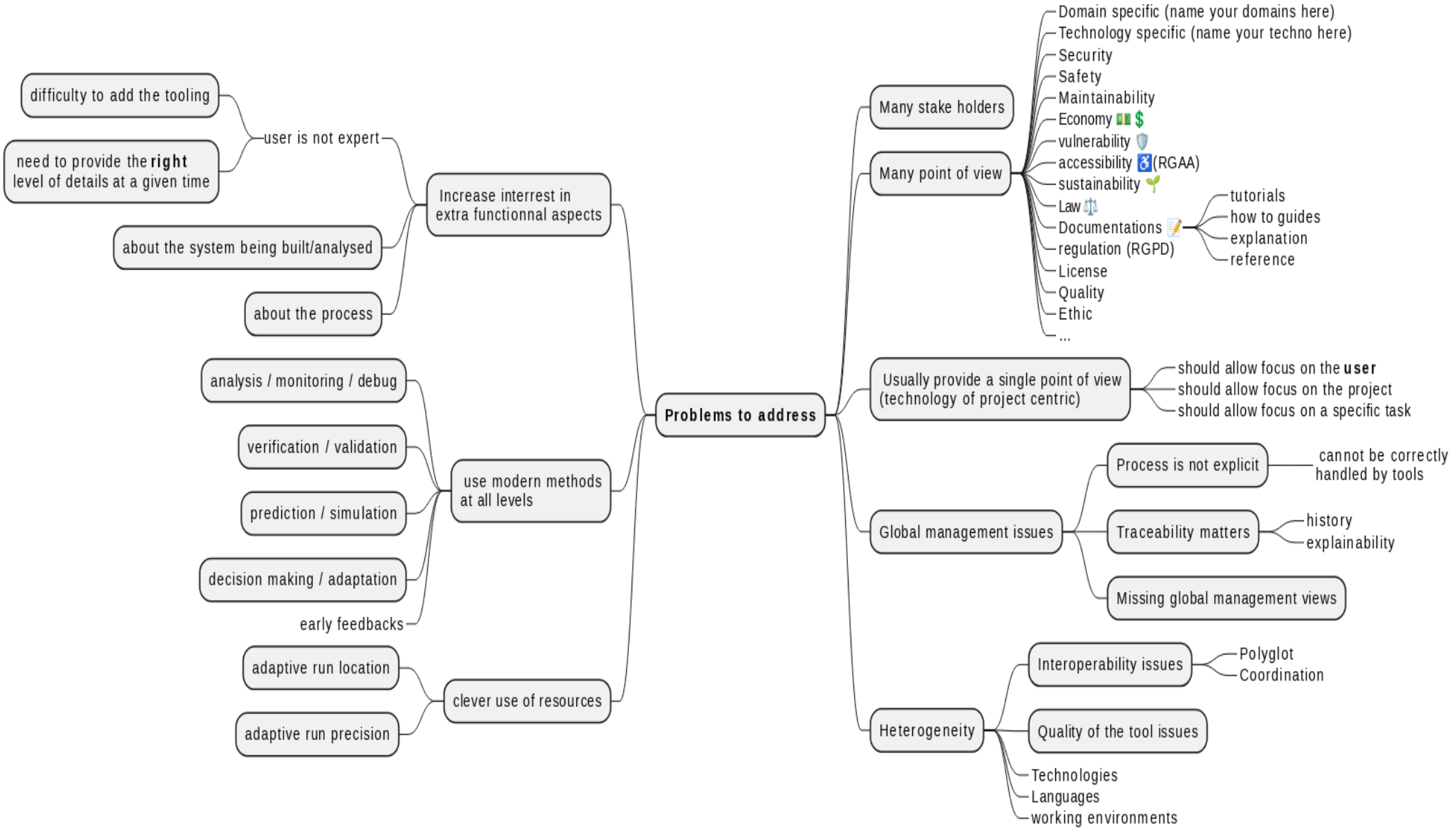
- Les termes IDM/MDE n'apparaissent plus récemment dans cette revue généraliste
- Mais on trouve
  - LowCode/NoCode
    - #260 (03/11/2023)
    - #255 (02/12/2022)
    - Hors serie #7 (05/2022)
    - #244 (30/10/2020)
  - Domain specific
    - Thread Modeling #255


- Quels points vous semblent'ils toujours d'actualité ?
  - Éventuellement sous une autre terminologie ?
- Quels points semblent en perte de vitesse ?
- Avenir d'un écosystème MDE dans l'open source ?
  - Sensibilité des écosystèmes de dev

# “My” Vision 2024

- better integration/collaboration of the all activities (incl. DevOps)
- Tomorrow developers tools
  - Concern aware collaborative workspaces
    - Or “Concern aware social workspaces” ?
  - Human is the center, the tool(s) should help keeping the focus while providing info about events in its environment/dependencies
- Content of what is dreamed next already exists in a myriad of tools, let’s imagine all this assembled and rationalized

# Some problems to address



# Concern aware collaborative workspaces requirements

- Concern/profile aware
  - Focus on some activities
  - Other information shown using more abstract/compact representations (zoom
    - ( all?) other concerns should still be present without requiring to switch to the concern dedicated workspace (kind of “Gemini cricket” showing impact on/of other concerns)
      - User/project should be able to customize level of details / threshold for warning / frequency of reminders
    - Easily switch to dedicated workspace if focus changes
  - Non exhaustive list of concerns (as main or insight):
    - Domain specific (name your domains here), Technology specific (name your techno here), security, safety, Economy , vulnerability , accessibility (RGAA), sustainability , regulation (RGPD), law , License, Documentations (tutorials, how to guides, explanation, reference), …
  - Some concerns should appear both about the *build process* and the *system under construction/analysis*
    - Ex: sustainability, vulnerability, economy,

# Concern aware collaborative workspaces requirements

- Verification support
  - “Are we developing the software application/model correctly?”
  - Should include all static checks (and proof)
    - Automated locally, remotely
    - Not automated (done by the user or any collaborator)
  - Should apply to both the application and the build system
    - Ex: detection of supply chain vulnerability

# Concern aware collaborative workspaces requirements

- Validation support
  - “Are we developing the **right** software application/model?”
  - Should include support for all executions
    - Test suite executions
    - Local executions
    - Production execution (monitoring and debug of the prod

# Concern aware collaborative workspaces

# requirements

- Natively polyglot
  - Works on both code AND models
    - ie. know that a given concept can be represented using different formalisms
  - With debug support
    - Omniscient (ie. forward and backward)
  - With for forecast
    - Prediction
    - Live programming

# Concern aware collaborative workspaces requirements

- Adaptable
  - Adapt fidelity of related models/codes
    - from mock to high fidelity to real/physical system
  - Adapt tool execution location
    - Move any tool or execution between User computer and remote resources (such as CI/CD runners, LLM queries, verification...)

# Concern aware collaborative workspaces

# requirements

- Traceable
  - Every “significant” action should have a mean to be related its origin
    - Ex: see a debug break point state in its different facets (models, codes, assembler)
    - Ex: human activities are saved and/archived (kind of version control) (on demand / automatically)

# Concern aware collaborative workspaces

# requirements

- Explainable/reproducable
  - List of tool used (dependencies, configuration, version, ...)
    - (also required to be able move to remote computing resource
  - Store evolutions of sources AND ( store meaningful executions results OR a way to reproduce results on demand a given version)
    - Results include functional data, but also non functional data such as execution time
    - For regression detection

# Concern aware collaborative workspaces requirements

- Natively Social
  - Should allow human in the loop for any task of the workflow
  - Should allow pair programming

# Concern aware collaborative workspaces requirements

- Explicitable overall process
  - if not customized by the user, the tool must be able to present the selected process

# Concern aware collaborative workspaces requirements

- LLM integration
  - Workflow should define the validation process of LLM inputs
  - Integration using Human interface
    - Ex: create pull request (ex: dependabot)
    - similar to human, activity and result should be traced
    - Ex: chat
    - Ex: assign task or issues similarly to  a human but to a bot
  - Integration as tool
    - Ex: wizard, refactor, translation, transformation, ...
    - As any tool of the process, usage and result should be traced
  - traceability in case of auto validation without human control

# Concern aware collaborative workspaces requirements

- Open
  - Plugin...

# How to achieve that ?

- We have several work in the team that can contribute
- Opportunity to build something larger than prototypes ?
- Which base technology ?
  - Risk if too specific: reproduce Eclipse twilight ?
- Many tools to integrate how to smoothly integrate them ?
- How to avoid a big bloatware ?
  - Create collaborating subsets ? (eg. forge de l'ingénieur + ? )
- Suggestion:
  - Massively invest on protocols ?
    - BSP, DAP, LSP, ...
    - Improve protocol specification for better interoperabilty (ie. REST vs SocketRPC vs GraphQL vs  gRPC vs tRPC vs Protocol Buffers…) or find bridges to avoid the protocol hell about protocol type, transport protocol, request/response format, schema language, support of subscriptions, ...

# Autres sujets du dossier de 2012

- Mixité
  - 2012
    - environnement très masculin
  - 2024
    - Et aujourd’hui pensez vous que l’on ai progressé ?
- Développeur
  - 2012
    - mal vu d’être développeur toute sa vie. (peu valorisant)
    - Il faut être chef de projet, architecte.
    - La France reste toujours en retard sur d’autres pays comme l’Angleterre ou les Etats-Unis où être développeur n’est pas une tare, et même le contraire
  - 2024
    - Est ce toujours le cas ?