# Building industry-ready tools: FAMA Framework & ADA[☆]

Pablo Trinidad, Carlos Müller, Jesús García-Galán, Antonio Ruiz-Cortés

*Dpto. Lenguajes y Sistemas Informáticos*
*ETS. Ingeniería Informática - Universidad de Sevilla*
*41012 Sevilla (Spain – España)*
{ptrinidad, cmuller, jegalan, aruiz}@us.es

**Abstract**

Developing good academic tools has become an art that forces researchers to achieve tasks they are not supposed to do. These include coding, web and logo designing, testing, etc. It compromises the quality of a product that differs from what the industry expects from it. In this paper we propose professionalising the tool development to build industry-ready products that are a middle term between academic and industrial products. We summarise many of the decisions we have made in the last three years to build two successful products of this kind, and that we think that may help other researchers their best in tool building.

*Keywords:* Automated Analysis, Software Product Line, SPL, Variability Models, Feature Models, Service Level Agreement, SLA

## 1. Context

### *1.1. Introduction*

One of the main objectives academic research must pursue is to turn ideas into industry-ready products. One of the most common ways software engineering researchers do this is by building software tools. However, many times academia only has resources to build just proof-of-concepts tools or prototypes that are not ready to be used by enterprises, but manage to demonstrate that the driving ideas are feasible and realisable into a product. When this is the case, prototypes are used to capture the interest of industry in academic research results. Alas, the time needed to transform a prototype into an industrial product is usually long enough to make businesses reconsider the real utility of this kind of prototypes. The industry expects mature products that can be incorporated into production as fast as possible. The academy's mind is on dissemination; the industry's mind is on *return on investment* (ROI).

In the spanish context, where most businesses are small or middle-sized, this distance is severe, since they are not usually prepared to assume the risk of incorporating the ideas behind a prototype into their products portfolio. It implies an opportunity loss that affects the dissemination of the research results and obviously the private incomes of our research structure.

[☆]This work has been partially supported by the European Commission (FEDER) and the Spanish Government under the CICYT project SETI (TIN2009–07366), and by the Andalusian Government under the project P07–TIC–2533.

We have observed that only academic tools are built when there is a lack of funding, and most of the times at the expense of researchers who invest too much time in tool building. In this scenario, researchers spend their time carrying out tasks they are not supposed to be assigned to; such as programming, testing, building a web page to disseminate their work or designing an interesting logo for the tool. Extra resources are not assigned to build tools because they increase expenditure, but letting researchers build tools is also an inefficient use of resources. In this scenario the inherited opportunity costs rocket since money is invested in good researchers but usually unexperienced tool-builders, rather than letting researchers investigate and allowing tool-builders develop tools.

The large distance between academic prototypes and industrial tools in quality terms is one of the main aspects that make companies refuse joint work with academia. We consider it necessary for academia to assume the costs of building better tools as efficient use of resources. As a projection of this reflection, for last three years we have been developing what we call industry-ready products that are a middle-term between academic tools and commercial products whose objectives are:

- Using tools to reduce the distance between academy and industry.
- Reducing the risk the companies assume when they are interested in research results.
- To allow a fast dissemination of research results in already existing industry-ready products.

In this paper we describe our experience from two points of view:

- Economic point of view: we have professionalised our tool-building activity so we have defined a business model that is briefly described in Section 2.
- Technological point of view: since we do not know our customers beforehand we have to build flexible products that could be adapted to the different companies who show some interest in our results. We show in Section 3 the main technologies that have helped to provide this flexibility and the methodologies that have been applied to lead our products to safe harbour.

### *1.2. Research Context*

Applied Software Engineering (ISA) Group is a research group composed of 22 researchers. Our research focuses on three main lines: *Software Product lines* (SPL) [1], software methodologies and *services oriented architectures* (SOA). Each line is independent of the others and there are only sporadic collaborations among researchers in different lines. For each line, several tool prototypes are built to demonstrate the realisation of the different advances and contributions. Within the wide set of prototypes we have built, there is a subset whose objective is analysing models of any kind: *Feature Models* (FMs)[2] in SPL, *Service-Level Agreements* (SLAs)[3, 4] in SOA, BPMN models in methodologies, etc.

In 2007, we decided to transform *SPLReasoner*, a promising prototype to analyse FMs, to *FAMA Framework* (FAMA FW), an industry-ready product. In 2009, we could say that this model worked for FAMA FW so we decided to replicate the process to transform SLAxplainer [5, 6] into ADA [7], both tools to analyse SLAs. In ADA, we have applied not only the economical aspects that led FAMA FW to be a success story, but also most of the technological issues since their context is close enough to reuse technologies and methodologies. Next we describe what we expect from an analysis tool in general and in FAMA FW and ADA in particular.

### *1.3. The products: Analysis Tools*

Generally speaking, an analysis tool is used to extract relevant information from a model. To achieve it, the tool transforms a model into different artificial intelligence paradigms such as constraint programming, satisfaction problems or genetic algorithms. Users can query for information by means of different analysis operations and the tool selects the most suitable technique to solve the analysis problem, having the best possible time-to-response.

In 2007, SPLReasoner had become an ongoing prototype in which the SPL community had shown much interest. We decided to take it to the next level building FAMA FW, an industry-ready tool that should fulfil two requirements:

- We will develop it as if it were an industrial product rather than an academic product. The main difference arises in how seriously we will take the development process and the support to users and third-party developers. There is an economic investment need, so business and dissemination models are defined to justify each cent that is invested in the project.
- The product must be customisable since there are many potential customers. This means that we will have a product per customer, thus transforming the product into an SPL. The methodologies and technologies must be adapted to support an SPL.

There were no new functional requirements added to FAMA FW, only non-functional or quality requirements that increase the flexibility of the product. These features did not depend on FAMA FW particularities so, ADA was built to fulfil the same requirements. The architecture of the product would be completely different so both products were built from scratch and only some isolated pieces of code could be reused from the original prototype. Human resources were assigned to the project and development methodologies were established to manage the development team.

### *1.4. Objectives*

At date, the practises that we describe in this paper are part of our internal procedures to build and maintain software products of near-industry quality; so the distance between research results and the industry is lessened. Our main objective is for the reader to see our story as a reflex in their context and to consider applying some of the ideas described in this paper to transform prototypes into industry-ready products.

We have divided our experience in two parts: a first part in Section 2 that explores the economic aspects that have been considered during FAMA FW and ADA's lifetime. These are the interaction with the industry, human resources hiring, how to improve product dissemination and which are the risks to be taken into account. A second part in Section 3 focuses in describing the development methodology and how the architecture and the chosen technologies have helped to manage the evolution of the product. To conclude we have built a *post-mortem* report in Section 4 that summarises the best and worst practises that we have detected during the development of the tools under study.

## 2. Business Model

In this section we describe some organisational aspects that caused a reorganisation of our structures and intense decision processes that took up a large amount of time and have to be taken into account whenever an industry-ready tool is built.

### *2.1. An iterative process to build industry-ready products*

***Why do we build tools?***. When we look to define a better way to build tools, it is mandatory to analyse the reasons why academia builds tools. We build tools to transform research ideas into a tangible product. If a company shows interest in the product, the product generates income. This income allows academia to keep on researching and the research continues to produce ideas that are incorporated to existing products, or in itself leads to the development of new products.

And why does a company pay for a product or finance a research topic? Because they foresee its transformation into a commercial product and they expect to obtain ROI from it.

There is also another scenario that must be considered; the creation of spin-off companies originating from these products. Mature products can become important assets in this kind of companies which are created to industrially exploit research results.

***Why does academia builds prototypes instead of building commercial products?***. A prototype is the less expensive and risky procedure to validate clipboard results. It reduces the time-to-validation by a proof of concept. Although they work for researchers, this is not the appropriate form to present the product to the industry. The industry expects a better quality product so researchers usually invest time in improving the quality of a prototype.

***How is a research prototype built?***. Usually a researcher is the only developer. The tool is generally created as part of the ideas maturing process. Since transmitting vague ideas to developers foreign to the problem may delay the project or even make it fail, the researcher is the one in charge of developing the prototype. Although this way of working is valid when first creating a prototype, extending the development in time may lead to a misuse of researchers time.

***Where are the limits with prototyping?***. This is one of the most important questions to be answered by each researcher that develops a prototype. In our case, we have found that the correct answer for us is *'When you can estimate the time and cost to build an industry-ready product, the core requirements and the target market'*. The point of this is that a researcher does not spend more time in development tasks than what is necessary to confirm ideas. What a research group needs is to collect all the available information to evaluate if it is worthy or not to assign resources to build an industrial-strength tool from a prototype.

***May a research prototype become a commercial product?***. The general answer is no. A prototype easily becomes a legacy system. Usability, extensibility and adaptability are quality attributes that are barely considered as initial requirements when a researcher decides to build a prototype. This transforms a prototype into a hard-to-maintain product. We need flexible architectures that can adapt a product to customer needs and incorporate new research results, and a prototype is not usually built following these two objectives.

***How do we build an industry-ready tool?***. We have defined a process to manage the development of tools that goes from the generation of research results to the dissemination of results to companies. The process is divided in four main phases:

1. **Prototype initiation**: a prototype is initiated under the researcher's consideration without constraints of any kind. It is important that the researcher does not try to build the definitive product; otherwise he will spend time on work that will be quickly discarded. Once a prototype is mature enough, it passes to the next phase.

2. **Prototype evaluation**: a research group committee evaluates how promising a prototype is based on the reports submitted by researchers, which have to analyse the development costs, product features and target market among other issues. If a prototype passes this phase, resources are assigned and the researcher is appointed leader of the product development.
3. **Prototype design**: a reference architecture is defined. It must be as flexible as possible so it supports any further extension or change in products requirements. Product leader and hired developers are involved in this phase.
4. **Product lifecycle**: whilst sufficient resources are assigned, products are developed following an iterative process. The lifecycle is carried out in three sub-phases:
    - Transfer: ideas are incorporated to the reference architecture as a new feature. The hired human resources are expected to get involved mainly in this phase. This phase ends when the feature is implemented, tested and correctly documented.
    - Tracking: the committee evaluates the product by means of the feedback received from companies that have shown interest in the product. Based on this feedback, future ideas to be investigated are chosen. The resources assignment is supervised and changed if needed.
    - Research: researchers produce new ideas. They are normally validated by the community in workshops, conferences or journals. When an idea is mature enough, *transfer* phase starts.

    Each of these sub-phases are executed by different human profiles: researchers, committee members and developers. This work distribution allows the sub-phases to overlap.

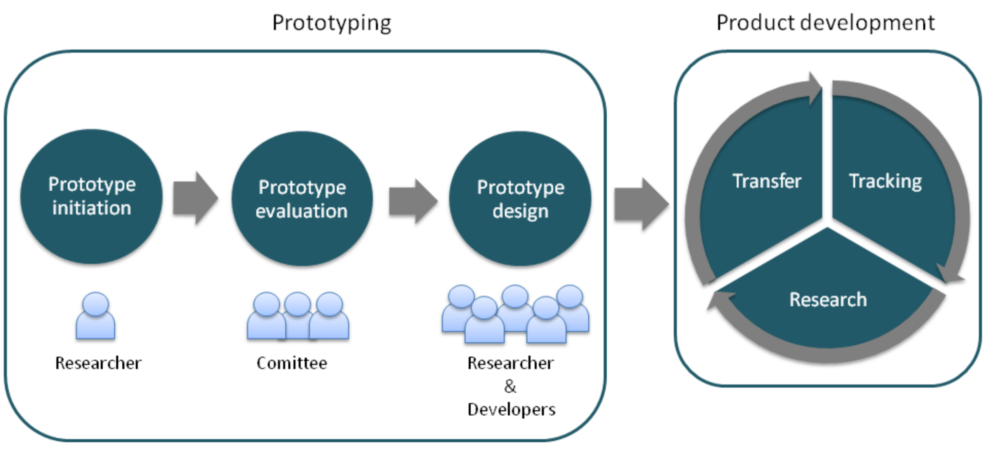

Figure 1: Our Business process model

***How many iterations are needed to produce an industry-ready product?***. We have to take into account that this form of building products is like creating a product whose market is unknown. When we have a customer, requirements are clearer. However, when there exist many potential customers but none of them acts as real customer, there is a tendency to be involved in a never-ending project. If a company shows interest in a product, the iterative process may be repeated indefinitely. If after given several iterations, no interest is shown, it is better to freeze the project until new opportunities arise so that resources are not wasted. In spite of this, the more iterations are fulfilled, the clearer the companies vision of the product. It is the commitee's responsibility to envision future interests and decide which risks are worth taking and which are not.

### *2.2. Human Resources*

With the above approach, we try to manage the product development as if we were a software development company. Thus, we have defined several roles that are played by our researchers and the hired human resources such as project leaders, developers, testers, etc. We have stated that researchers act as project leaders but, who plays the remaining roles?. It is not a secret that researchers commonly play roles they must not play such as developer, tester, designer, web-master, etc. We consider this is a waste of resources since researchers salary is usually (and hopefully) higher than developers. We attempt to hire different workers whose professional profile fits into the job they are assigned.

When hiring people, a new problem arises: carrying out a selection process. We have delegated this task to the project leaders, however several factors make us wonder if this decision is the best choice:

- On the one hand, the University has plenty of motivated and well-prepared candidates. In our case, most of the researchers are lecturers in the University so they have probably taught some matters to the candidates, making it easier to evaluate their personal and technical abilities. This information is very valuable in the selection process.

- On the other hand, selecting people is not one of the responsibilities a researcher must assume. Furthermore, it is interesting to differentiate candidates who want to be part of a big software company from candidates expecting a professional career in academia. We want to avoid a developer leaving an ongoing project since it would imply a new selection process and probably the loss of knowledge. This makes the selection process harder and it usually takes longer than expected.

After 3 years following this procedure, we have had good and bad experiences. We have organised selection processes lead by the project leaders. Although the project leader is who best knows the technical profile a developer or a tester has to satisfy, there are other psychological and emotional factors involved in selecting the right candidate. Technical skills are as important as personal abilities. This is one of the reasons that makes us wonder if selection processes should be outsourced so there is more confidence in selecting the most decisive candidate.

### *2.3. Dissemination*

Tools dissemination is a task that should be carefully performed while the project takes place. We explain as follows the most important issues of our *marketing* and *communication plans* in order to define the different actions related to the dissemination of our analysis tools.

### 2.3.1. Marketing Plan

A marketing plan defines the products we aim to build, the policies to regulate its consumption, and its target market.

The basic policies needed to regulate the consumption of our developed analysis tools are the *price* and *usage policies*.

- Price policies: our goal is to capture investment; that means to obtain as much users as possible for our products. Users are not asked for a fee to use our tools. We decided to offer them as open-source tools, yet some kind of tool supporting such as tool customisation or integration would require an economic compensation.
- Usage policies: We set LGPLv3 licence [8] to our products. We decided to use this licence since it allows everyone to use our tools, even in commercial products, but a citation to the owners is mandatory.

We established as target market for our products companies and research institutions that have shown some interest in analysing models of any kind, specifically FMs and SLAs. We would offer our analysis tools and customising services to them. At this point we had to set a communication plan that considers how to reach the target market.

### 2.3.2. Communication Plan

Our communication plan's objective is two-fold; we want to be equally appealing to the two entities of our target market: private companies and research institutions.

To achieve such purpose we had to ask ourselves: What do private companies and research institutions expect from us?. While the former usually have the goal of getting a commercial product in order to obtain ROI; the latter commonly want to work together to obtain results in terms of research and tools integration. Therefore, we have to consider the kind of entity in the definition of our communication plan as follows:

- Dissemination releases for companies and research institutions:
  - *Advertising*: we have created web portals as direct communication channels[1]. Before releasing any versions of our products we designed logos and corporate images that identify the products in any publication relating them. Another kind of advertising is with posters and tool demonstrations sessions of conferences and commercial events.
  - *Technical documentation*: customers need to know how to use and extend our tools by means of: (1) *technical reports* published in the respective web portals. Such technical reports reduce the time to learn to develop an extension of our tools or to integrate our tool into third–party tools. Usually, the technical reports are published on–demand when an entity shows interest so our effort is not wasted. (2) *source documentation* by means of Javadoc [9] or Doxygen [10] tools. Doxygen allows introducing LaTeX pieces of text/images and producing a final documentation richer than the one produced by Javadoc. We also include source code documentation.

[1]FAMA portal is available at `http://www.isa.us.es/fama` and ADA portal is available at `http://www.isa.us.es/ada`

  - *Scheduling*: periodically, private companies go to commercial events and research institutions go to conferences. Once these forums are identified, we attend in order to show our products.
  - *Predefined forms*: some legal documents such as agreements and compliance forms and usage certificates are defined. These documents are necessary to validate the use of our tools by other entities and to guarantee the LGPLv3 compliance. Therefore, when an entity starts using a product of ours, they will be asked to request a signed copy of such documents.
  - *Periodical affiliation*: to keep the affiliation of research institutes and companies they are periodically sent quality assurance forms, satisfaction forms, etc.

- Dissemination releases only for research institutions:

  - *Research-oriented dissemination*: usually the communication of research tools is not mass marketing, so a much more selective marketing activity is needed. Neither leaflets, nor mailings are needed. Conferences and workshops are the most suitable events for tool demonstrations. In this case, scientific papers and journals are a first–term documentation, but they do not usually describe the internals of tools. Specially, when the research contribution is the structure and design of the tool instead of a new algorithm. Technical reports cover this gap so the insides of tools are described without constraints of any kind.

- Dissemination releases only for companies:

  - *Company-oriented dissemination*: Any research group grows surrounded by local, national and worldwide companies who are potential consumers of our products. Research projects usually include some technology partners that are interested in knowing at first hand the results of the projects. This contact usually produce constant meetings between researchers and companies that may lead to projects where our products play a key role.

### *2.4. Risk Analysis*

A *risk* is any event that implies a delay in time or a deviation in expenditure. We have identified the following risks during industry-ready product development:

1. *a developer leaves*. It is a well-known risk in any development process. Avoiding it is complex, but we attempt to reduce its probability trying to offer our technical staff obtainable goals to prepare professional careers accordingly. If it cannot be avoided, it is important to minimise the impact of a developer leaving. It implies a constant focus on source code documentation and reporting.
2. *a product is not interesting for the community*. When this happens, it affects the cost assumed to date. Therefore, we establish the iterations of process depicted in Section 2.1 to determine as early as possible if the target market is interested in a product or not.
3. *A researcher has to play other roles*. It usually happens when the project has been assigned insufficient human resources. A thorough project scheduling helps to avoid these situations, anticipating the needed human resources.

## 3. Development model

In this section we describe the software development methodology followed in the development of our analysis tools. We also detail the technology incorporated in our products and the reference architectures.

### *3.1. Methodologies*

Due to the particular characteristics of our products, a methodology to develop our industry-ready products should support: (1) changing requirements, because requirements become clearer when the results can be observed, and (2) a straightforward and continuous communication between researchers and developers to ease the incorporation of research results into the existing products.

To cover these needs, we chose agile methodologies [11] which are a set of methodologies based on teamwork and adaptation to changes to provide a rapid high quality software product. These methodologies requires small teams, frequent and straightforward communication as periodic meetings to monitor the progress of the tool, a short time planning and the know-how developers reuse from similar projects to take advantage of their experience. The opposite of heavyweight methodologies, agile methodologies are more incremental, iterative, present short iterations, and requires less documentation; only the essentials, so third-party developers can understand how to extend the tool, users know how to use the product and the know-how remains inside the organisation.

We have specifically used the following agile methodologies to develop our products:

- *Feature-Driven Development* (FDD). Firstly, groups of features, which represent user visible characteristics, are identified. Later, such features are classified in core and satellite features for a priority assignment. A core feature is mandatory for any potential customer; a satellite feature is only relevant for a subset of customers. In an iteration, a subset of features is selected, designed, developed and tested starting with the core features and following the priority order. FDD allows the separation of different satellite projects, each considered with satellite features.

- *Test-Driven Development* (TDD). Before developing or upgrading a feature, one or more test-cases are designed, implemented and executed. Whenever new source code is produced or existing one changes, code is tested. Finally we refactor the source code to grant its quality. In addition, before building a distribution for a product, test-cases are executed to detect possible errors introduced by changes. It increases the quality, reduces the time-to-failure, and accelerate the development since time to repair errors is reduced. TDD is used in our projects in conjunction with FDD to increase the confidence in the feature development correctness.

The experience on using these methodologies has been very satisfactory. On FAMA FW, several versions of the tool have been released with significant functionality upgradings and the amount of detected errors has been drastically reduced. On ADA, the experience on developing and managing FAMA FW helped us to improve the development process. Thus, we built the first version of ADA which covered a complete iteration in all the development phases in three months.

## *3.2. Software Architecture*

Before defining an architecture we need to discuss the common aspects of analysis tools in general. Basically, an analysis tool works with a *model* to be analysed, several *analysis operations*, and one or more *reasoners* that are able to solve those operations. The main goal is extracting information from the models using the analysis operations. Using this approach we may characterise a flexible analysis tool as the one that is able to support:

- Different kinds of models (a.k.a. metamodels) and file formats to store and retrieve them.
- Different analysis operations.
- Several alternatives to solve an analysis operation.

FAMA FW and ADA have a SPL-based design. This means that instead of building a product, we pretend to build a flexible production platform that is able to produce different kinds of products depending on the customer needs. It is very valuable for our purposes since we are unable to determine the companies that are going to use our products.

SPL is a building concept rather than a bunch of methodologies, so specific procedures have to be defined for each kind of project. One of the ways to implement a SPL is by a component-based architecture [12, 13]. A component-based architecture emphasises on developing functionality in independent components, integrating them later to provide a complete functionality. It allows dividing the tool in many small-sized projects and reducing the coupling among them. It permits parallelising the development of these components so development time can be reduced. Among the advantages of using a component-based architecture we remark the following:

1. Modularity: coupling among components is reduced to minimum since each of them is specific enough to perform a feature without needing a direct coupling with other components. It can only be achieved when the nature of the problem allows this kind of division, as is the case for our products.
2. Extensibility: new functionalities are easily added registering a new component in the reference architecture.
3. Integrability: due to the reduced coupling, communication lines between components is minimum which eases the integration process and the substitution of components.

In our component-based architecture, we have identified three component layers as depicted in Figures 2 and 3:

1. Core: a core is a subset of components that must be in any product. They are the first pieces to be defined in any tool. Core components define basic functionalities, public interface for the final users, and load and use extensions.
2. Extensions: an extension is an implementation of one or more features that are pre-defined by core. Several extensions can be available offering the same feature so the core is in charge of selecting the most suitable extension to carry out the job. The extensions we have identified for FAMA and ADA are:
   - Metamodels to be analysed.
   - Analysis operations.
   - Reasoners to solve the analysis operations.
   - Selection criteria to choose the better reasoner to solve an analysis operation.

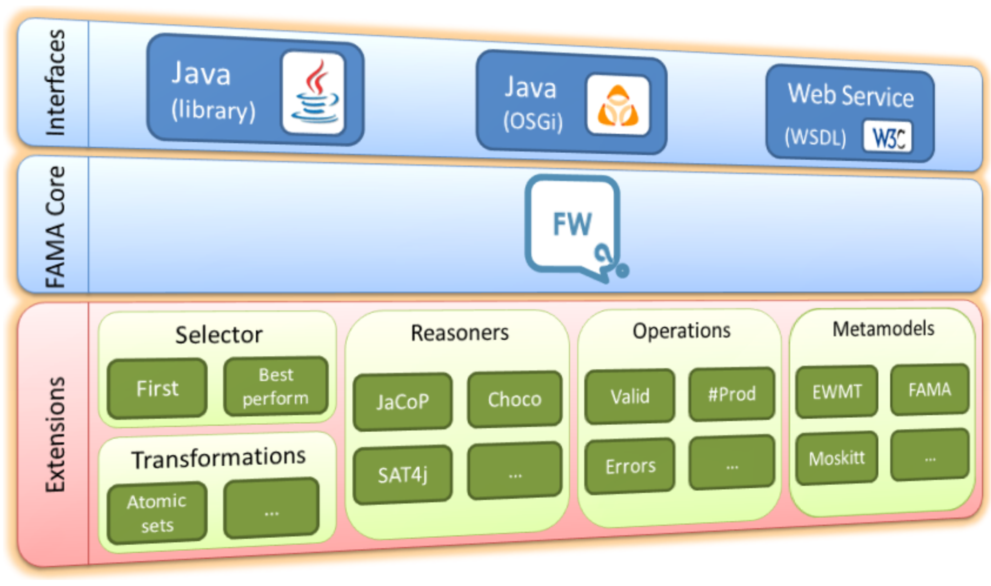

Figure 2: Architecture of FaMa framework

- Transformations between models.

3. Interfaces: they are the facade that allows third-party entities to use our products. They are defined as independent satellite projects that use the API of the analysis tool, considering tool API the core and extensions, and give us new functionality. A typical example can be a GUI, that consumes the API of the tool and makes the human use of the tool easier than the programmatic use. An example of this kind of interface is available for ADA, specifically a rich internet application called *ADA-FrontEnd* that is accessible at *try it online!* section of ADA portal (`http://www.isa.us.es/ada`).

In both tools, FAMA FW and ADA, a three-layer architecture (see Figures 2 and 3) is designed to support several *variation points*. In our context, a variation point is any extension component.

From a developer's point of view, the tool becomes a framework, since any new extension that is developed by third-parties can be integrated in the architecture and the core is in charge of consuming the extension. From an end-user's point of view, those variation points are transparent to the user since it is the core that is in charge of choosing the extensions needed without any kind of user interaction.

The architecture permits dividing the product development into several sub-projects each of them involving an extension. It eases the community collaboration and the hiring of new developers for our development team who can start developing satellite projects without affecting core functionality.

However, the adoption of this architecture involves a double challenge: (1) the integration between independent components requires a technological solution, such as a tool or framework to support it; and (2) the management of dependencies between core and extensions. The following Section 3.3 tackles these challenges with some technology decisions.

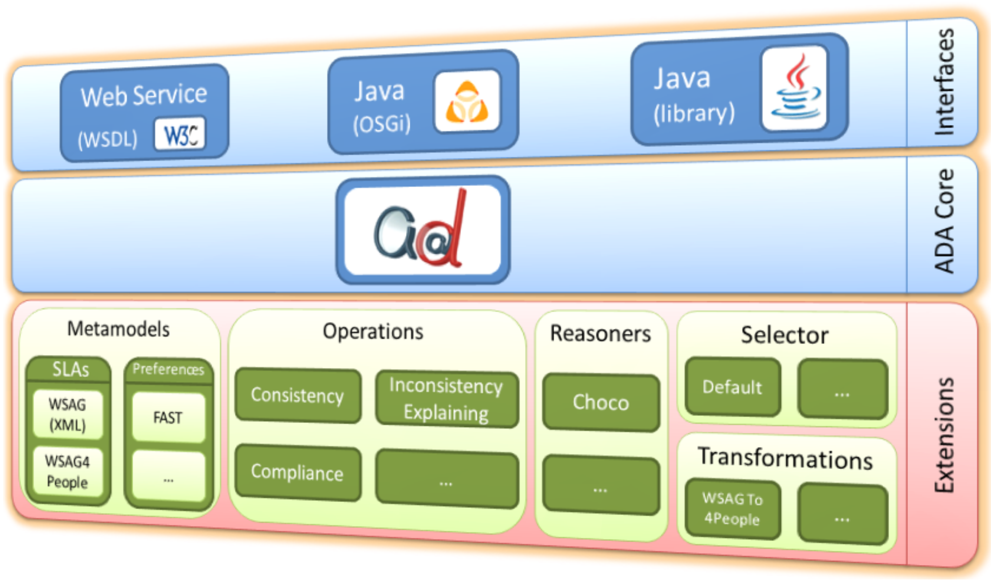

Figure 3: Architecture of ADA framework

### *3.3. Technologies*

Making the right decisions regarding technology is key to support product evolution. Any wrong decision may lead to an important economic loss.

One of the most important choices to develop software is the programming language to use. A stable and mature language, with the right paradigm is needed to minimise risks. If we want to involve a community behind a product, we should choose a general purpose programming language with many open source projects. These projects implies to have several free tools available that can be incorporated into our products. We have chosen Java [14] because it is widely used and has a large open source community behind, and many high-quality tools.

In previous Section 3.2, we mentioned two challenges in architecture designing: integration between components and dependencies management. OSGi [15], Maven [16] and Subversion [17] have been the chosen tools that help us with these problems as detailed in the following:

1. *OSGi* is a framework specification that provides a container to deploy, run and integrate components (a.k.a. *bundles*) for this framework. It has several implementations, such as Equinox [18], Felix [19] or Knoplerfish [20]. OSGi allows the integration between OSGi compliant tools, and it offers some interesting plug-in tools. One of them is Distributed OSGi (DOSGi) [21] which publishes OSGi bundles as web services providing a WSDL specification [22][2].
   ADA and FAMA FW are OSGi-compliant. FAMA FW is currently being used by Moskitt Feature Modeller [23] thanks to OSGi framework, and ADA public interface can be used as a web service through its WSDL specification.

[2]*DOSGi* was used to generate the *WSDL* facade of commented ADA interface *ADA-FrontEnd*

2. *Maven* is a software tool for the project management and for automating tasks on java projects. For FAMA FW and ADA, Maven manages the dependencies between core, extensions and satellite projects, OSGi metadata, binaries, and tests execution before packaging.
3. *Subversion* is a popular version control system to manage changes on source code, resources and documentation. Our subversion repositories comprise three folders: (1) *trunk* for the current development, (2) *branches* for separated lines of development, and (3) *tags* to keep snapshots of the repository at a concrete date or milestone.

## 4. Conclusions and Postmortem Report

With this experience we have learned that we, the researchers in academia have to put ourselves in industry's shoes so our prototypes are closer to what the industry expects. As of today, more than 20 companies and research institutions are using, adapting or are interested in using FAMA FW and we expect ADA achieves the same goal in the next months. During this process there have been right and wrong choices. To analyse them we have realised a post-mortem report which summarises those practices that we will keep in the future and that we recommend others to do. Besides, we have committed errors that we will try not to repeat in the future and are also part of this report.

### *4.1. How to continue doing what was done Right*

1. LGPLv3 license is the best option.
2. The components architecture has reduced the bugs in the products, moreover in those that have been affected by many iterations.
3. The business model used in FAMA FW has been validated in the ADA project. The human resources reuse between the two projects has been especially interesting.
4. The dissemination model is effective enough as shown by the high number of companies and research institutions interested in FAMA and ADA.

### *4.2. How to correct what Needs Improvement*

1. We have not focused our efforts in providing much documentation to end-users and third-party developers. We have had to invest much time in documenting after several iterations since we have been asked for more and better documentation. We have solved it by adding a new procedure to FDD that requires documenting before ending the development of a feature or extension.
2. Some times our selection processes were massive and they took longer than expected. We will find a solution for this in the future.
3. At the beginning we prepared a detailed planning for the whole project and we used a tasks tracker to monitor if tasks were on schedule. However, we had to redo the planning due to the changing requirements. Therefore, the detailed planning should define short-term goals within a month. The long-terms goals should be defined in a more relaxed planning.

[1] P. Clements, L. Northrop, Software Product Lines: Practices and Patterns, SEI Series in Software Engineering, Addison–Wesley, 2001.

[2] K. Kang, S. Cohen, J. Hess, W. Novak, S. Peterson, Feature–Oriented Domain Analysis (FODA) Feasibility Study, Tech. Rep. CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University (Nov. 1990).

[3] L. Lewis, Managing Business and Service Networks, Kluwer Academic Publishers, Norwell, MA, USA, 2001.

[4] D. Verma, Supporting Service Level Agreements on IP Networks, Macmillan Technical Publishing, 1999.
[5] C. Müller, A. Ruiz-Cortés, M. Resinas, An Initial Approach to Explaining SLA Inconsistencies, in: Proc. of the $6^{th}$ Int. Conf. on Service-Oriented Computing (ICSOC), Vol. 5364 of LNCS, Springer Verlag, Sydney, Australia, 2008, pp. 394–406.
[6] C. Müller, M. Resinas, A. Ruiz-Cortés, Explaining the Non-Compliance between Templates and Agreement Offers in WS-Agreement*, in: Proc. of the $7^{th}$ International Conference on Service Oriented Computing (ICSOC), Vol. 5900 of LNCS, Springer Verlag, Sweden, Stockholm, 2009, pp. 237–252.
[7] C. Müller, A. Durán, M. Resinas, A. Ruiz-Cortés, O. Martín-Díaz, Experiences from building a ws–agreement document analyzer tool (including use cases in ws–agreement and wsag4people), Tech. Rep. ISA-10-TR-03, ISA Research Group, `http://www.isa.us.es/modules/publications/getPdf.php?idPublication=322` (Jul 2010).
[8] Free Software Foundation (FSF) Inc., Gnu lesser general public licence version 3, http://www.gnu.org/copyleft/lesser.html.
[9] Oracle–Sun Developer Network (SDN)., Javadoc tool, http://java.sun.com/j2se/javadoc/.
[10] Dimitri van Heesch et al., Doxygen -Dox(Document) Gen(Generator)-, http://www.doxygen.org.
[11] M. Fowler, J. Highsmith, The agile manifesto, In Software Development, Issue on Agile Methodologies (August 2001).
[12] D. Coppit, K. Sullivan, Multiple mass-market applications as components, in: Proc. of the 2000 International Conference on Software Engineering, 2000, pp. 273–282.
[13] H. Kienle, Component-based tool development, in: Frontiers of Software Maintenance, 2008. FoSM 2008., 2008, pp. 87–98.
[14] Sun Microsystems, Java, http://java.sun.com.
[15] OSGi Alliance, Osgi, www.osgi.org.
[16] Apache Software Foundation, Apache maven, http://maven.apache.org.
[17] Apache Software Foundation, Subversion, http://subversion.apache.org.
[18] Eclipse, Equinox, www.eclipse.org/equinox.
[19] Apache Software Foundation, Apache felix, http://felix.apache.org.
[20] Makewave, Knopflerfish, www.knopflerfish.org.
[21] Apache Software Foundation, Distributed osgi, http://cxf.apache.org/distributed-osgi.html.
[22] World Wide Web Consortium, Web Services Description Language (WSDL) 1.1, `http://www.w3.org/TR/wsdl` (2001).
[23] Conselleria de Infraestructuras y transportes, Moskitt, http://www.moskitt.org/cas/moskitt0.