Rapid Prototyping Using a Configurable Platform

Antonio Misaka

> *"The only good idea is an implemented idea... that stays implemented!"*
>
> William C. Byham
> Entrepreneur, author, and organizational psychologist

This article describes an approach for speeding up the development of web applications using a configurable platform. The core idea of the approach is that developers can implement customer requirements by configuring platform components, instead of writing large amounts of "glue code" to wire the components together. This approach reduces the amount of glue code that still needs to be written and maintained, it shortens the time it takes developers to create a prototype, and it makes it easier for glue code to be reused in the future. It also allows developers to experiment with different configurations of platform components in order to find the configuration that best meets the customer's requirements. Developers are also able to manage a larger variation in customer requirements.

Introduction

Web applications are commonly assembled from a number of existing components that are combined together to support a custom business process. These are components such as Drupal (drupal.org) and SugarCRM (sugarcrm.com), which provide commonly used functionality for content management and user-profile management. The code that connects the components is known as "glue code" (tinyurl.com/q3vu3hz). Because this code is very specific to the assembled components, it can be difficult to maintain and reuse.

This development approach can best be described as "clone-and-own" reuse: a new application starts out by duplicating glue code from a previous application (tinyurl.com/pcruf2h). Code duplication causes significant maintenance problems. If any errors are subsequently found in the original code, they will need to be fixed in every copy. The match between the needs of the new and the old application is also often not perfect. The duplicated code often contains "orphaned" code that does not serve any purpose in the new application.

At the same time, the applications created often only differ in minor details, and thus much time is wasted by developers modifying and creating glue code and learning about new component APIs (tinyurl.com/6abeyab). A more systematic approach to selecting components and creating glue code is called for – one that reduces the amount of unnecessary glue code. Application developers could learn from the discipline of software product-line engineering (tinyurl.com/ps7wyob), which is concerned with the systematic creation of common assets and methods for enabling reuse across products in a product line. This approach is not yet used widely for developing web applications, but the benefits of using a software product-line engineering approach are threefold: i) the resulting applications are more maintainable, ii) time is saved when developing the application as a result of reuse, and iii) the details of using a specific component can be hidden from the developer behind common interfaces.

Box 1 provides examples of business processes that share many of their requirements, and could benefit from a software product-line approach.

Rapid Prototyping Using a Configurable Platform

Antonio Misaka

Box 1. Examples of business processes with similar requirements

Tony, Fred, and Bob are business owners with very similar needs:

- Tony wants to run a promotion for his restaurant. When diners pay their bill, they should also receive a printed ticket that enters them into a draw for a prize. At the end of the promotion period, the winning ticket numbers are announced on a board in the restaurant. Diners with a winning ticket can redeem it at the restaurant.

- Fred runs a construction company and wants to generate leads for his business. Potential customers can enter their email on the company's website, and they will be sent an email with a ticket that also enters them into a draw for a prize. At the end of the promotion, a winner will be selected and notified by email. The winner can print their ticket and redeem it by visiting the construction company's office.

- Bob is the owner of an independent bookstore and wants to increase the loyalty among his customers. Customers can receive a discount on future purchases if they register their email on the store's website. When customers make a purchase, they can enter the number of their sales receipt on the website, and they will receive a ticket worth 10% of the money they spent, which they can redeem at their next purchase.

Each of our three business owners approaches Tickets R Us to develop a custom application that implements their business processes. Traditionally, Tickets R Us might have built an application for Tony, chosen appropriate components – such as platforms for maintaining a database of tickets, printing a barcode on a ticket, and scanning the barcode – and wired them together using glue code. When creating Fred's application, Tickets R Us would have started with the code developed for Tony, added a new feature to send a ticket via email, and made tweaks to the existing code. Similarly, when creating Bob's application, reuse would be limited to a clone-and-own approach.

In order to apply the software product-line approach to web applications, two problems need to be overcome: i) how to reduce the amount of "glue code" required to wire the components together, and ii) how to hide the details of specific components from developers. The first problem can be addressed by creating a configurable platform that contains the reusable components (also known as common assets). A large part of the glue code that would otherwise have to be created can be replaced by specifying a configuration of platform components.

The second problem can be addressed by raising the level of abstraction at which developers write code that interacts with specific components. However, the second problem can really be considered a subproblem of the first one: a configurable platform would be of little use if developers had to have detailed knowledge of specific components.

This primary audience of this article are companies like our hypothetical company Tickets R Us who need to create more maintainable applications and achieve a higher degree of reuse.

The rest of this article first offers a closer look at the problem of raising the level of abstraction at which the glue code interfaces with components. It then describes the architecture of a configurable platform that increases the level of abstraction at which web applications can be built. Next, it outlines a process for creating a configurable platform that builds on the lessons from software product-line engineering and early requirements analysis. The article concludes with a discussion of managerial implications.

Raising the Level of Abstraction

Glue code that developers write to wire together components is hard to maintain for a number of reasons. One reason is that there is a lot of it: the more code there is, the harder it is to maintain. The other reason is that glue code tends to be very specific to the components that are being assembled. On top, glue code is likely to be "reused" in an improper manner from one application to the next; this is the problem that we referred to earlier as clone-and-own.

Rapid Prototyping Using a Configurable Platform

Antonio Misaka

The first part of the solution to these issues is to create a configurable platform. When using a configurable platform, developers do not need to write as much glue code. In the next two sections, we outline an architecture and process of constructing such a configurable platform.

The second part of the solution involves raising the level of abstraction at which developers interface with components. If developers do not apply proper constraint, the glue code can become very dependent on specific details of the components used. Not only does this lead to more complicated glue code, but it also limits the opportunities to replace the components with other functionally equivalent components, should this become necessary later. For example, the glue code to send emails to customer should ideally be the same irrespective of which protocol is being used to access emails.

This dependency is a well-known problem when programming user interfaces, where the application code and user-interface code can become tightly intertwined. As in that case, decoupling the glue code from the components can help create code that is significantly easier to understand and maintain. In general, decoupling can be achieved by defining interfaces that abstract the functionality of components with similar functionality into a common set of operations, and requiring developers to invoke the components only through those operations. It is not incidental that creating such common interfaces creates a "language" that is much closer to a business owner's model of the domain.

For example, in the Tickets R Us example, business owners will be used to specifying the requirements for what a ticket should show in terms of concepts such as ticket numbers, barcodes, and expiration date. Those concepts are a natural part of the language used by anyone who intends to use tickets for a promotion. These users are less likely to be familiar with expressing this information in the format required by a particular barcode component. Creating these common interfaces thus closes the "gap" that exists between how business owners express their requirements and the way developers think about writing glue code.

Architecture of the Configurable Platform

Figure 1 shows a proposed architecture of the configurable platform. Users of the platform (the business owners) are shown as subscribers on the top left. The configuration of platform components for each application can be specified in a configuration table. A configuration is a list of services that can be invoked by each application and specifies the values of configuration parameters for each service. Examples of services are Email, Login, or Ticket Generation.

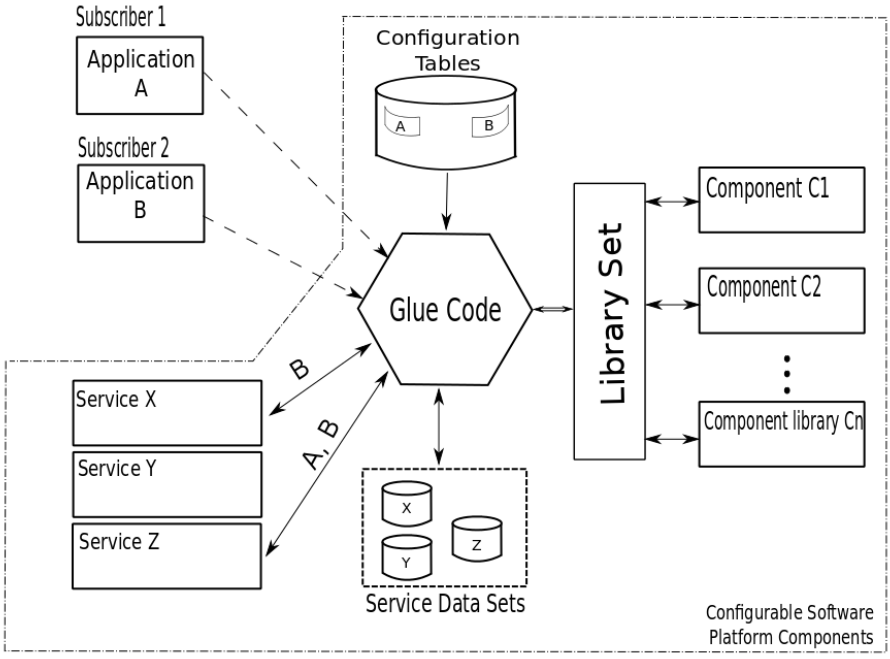

Figure 1. Architecture of the configurable platform

Rapid Prototyping Using a Configurable Platform

Antonio Misaka

Each service provides an abstraction for platform components with similar functionality and can be configured through parameters. For example, whether or not a ticket should be sent by email is a configurable parameter of the Ticket service. If the parameter is set to sending email, the corresponding glue code that invokes the Email service will be executed. The type of barcode to use on the ticket is another parameter that can be specified in a configuration.

Process

This section describes a process for creating a configurable platform and building applications based on this platform. The benefits of this approach are:

1. **It raises the level of abstraction:** Software platform configurations are defined in the language of the business owner (also known as the domain level), not at the implementation level.

2. **It simplifies configuration:** Glue code that specifies a selection of components and sets configurable parameters is easier to reuse than component-specific code.

3. **It makes reuse more systematic and efficient:** Glue code can be reused across multiple applications through shared services, not in the form of "clone-and-own" reuse.

A domain is an area of knowledge or expertise. It typically reflects the business owner's mental model of a domain. In software product-line engineering, a distinction is made between domain engineering and application engineering. Developing a platform that contains the core assets is referred to as domain engineering, and developing products from the platform is referred to as application engineering (tinyurl.com/p6xn7zh). Assets created during domain engineering are reusable, whereas the assets created during application engineering tend to be specific to a particular application, unless they recur across applications, in which case they should be turned into reusable assets to avoid future duplication of work.

The requirements are captured in the form of form of goals and expectations (goal models) and business process descriptions (scenarios). In the research we conducted, those models are represented in user requirements notation (URN). However, for sake of the exposition, we will not go into details of this notation here, but refer the interested reader to the project website (usecasemaps.org). For readers familiar with use cases and the unified modeling language (UML; tinyurl.com/anyno), we might add that URN bridges between use cases and object models in the UML.

The process comprises five steps:

1. Modelling domain requirements

- Gather user requirements in the form of goals and expectations (goal models) and business process descriptions (scenarios) by interviewing the business owners.

- A goal model is created for each business owner or a group of business owners that share the same functionality. A specific key identification is created for the configuration table.

- Links between goal models and scenarios are captured.

2. Identifying commonalities and variabilities in the requirements model

- Identify common and variable elements in goals models and scenarios. These represent the configurable features of the system.

- Commonalities are all those elements repeated in each model (goal and scenario models), and variabilities are elements that are unique to a model. Variabilities are candidates for configurable variations in the features provided by the platform. For a variation to be supported by the platform, it must generally occur more than once in the models.

- Identify candidate components that can provide those features. Those components can be selected by a developer when implementing the requirements. Identify parameters through which the components can be configured.

3. Modelling application requirements

- Create a model application using all the necessary elements to create the configurable platform. Existing software components, both third-party components and internally developed components, are possible candidates for reuse in the configurable platform. The model should incorporate the requirements to be satisfied and all functionalities expected by the configurable software platform.

Rapid Prototyping Using a Configurable Platform

Antonio Misaka

4. Identifying existing components

- Match components in the scenario models against the available software components.
- Identify configuration parameters to be included in the configuration tables.

5. Binding variabilities to components

- Develop and implement the necessary glue code to run an application. The developer now has all the necessary information to build a prototype using the selected set of components.
- Test the prototype and verify it with potential customers.

Box 2 provides an example of the first two steps of the process.

Figure 2 shows how the architecture from Figure 1 was instantiated for the Tickets R Us example (steps 3 to 5). Note that, for purposes of illustration, some details have been removed from the diagram.

Conclusion

If a company plans to create a series of web applications in the same application domain, it should consider building a configurable platform first. A configurable platform offers two advantages over the traditional "clone-and-own" approach: i) developers save time when building applications with similar functionality and can take on more projects, and ii) it raises the level of abstraction at which web applications can be built. The approach also reduces the translation errors developers can make when mapping high-level user requirements to low-level application details. Creating a configurable platform does not come without initial expense, however, but will pay off after a few applications.

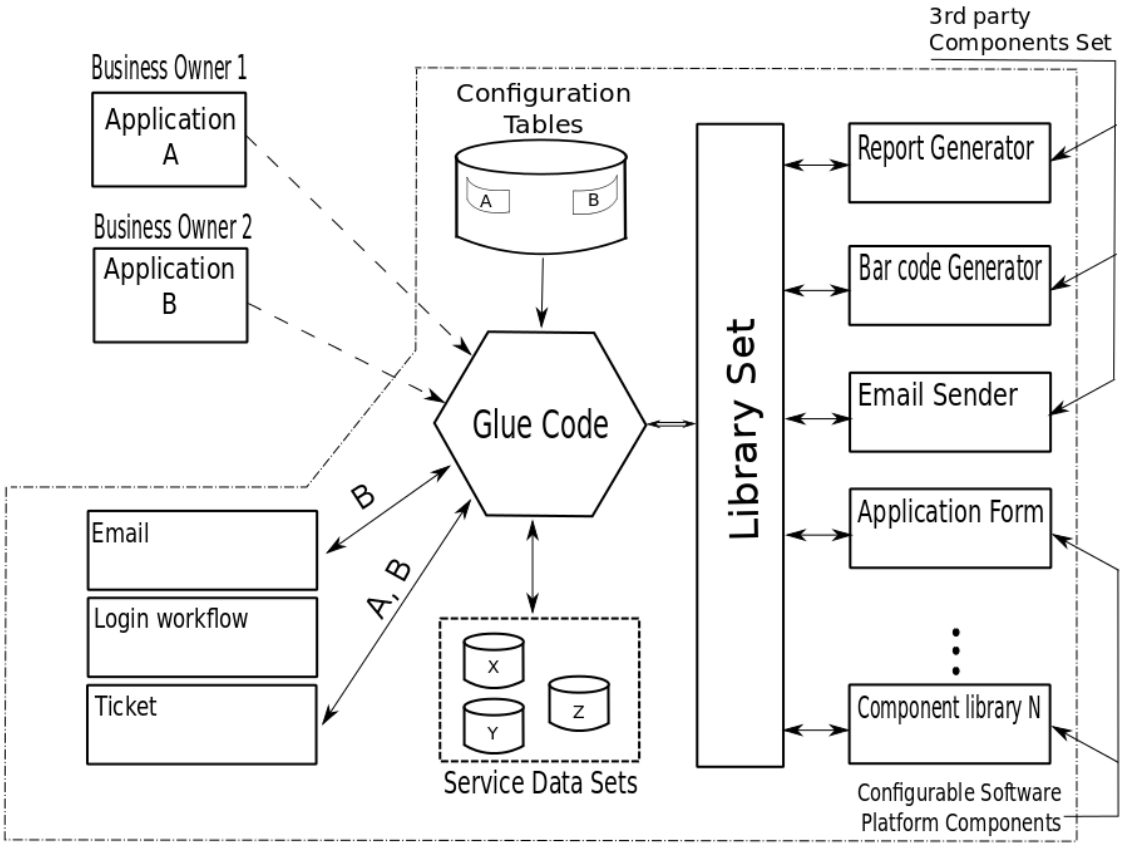

Figure 2. Instantiation of the architecture for the example

Rapid Prototyping Using a Configurable Platform

Antonio Misaka

Box 2. Applying the process to the Tickets R Us example (steps 1 and 2)

In the first step (modelling domain requirements), we capture the business owner's domain requirements in terms of their goals and business processes. Here are samples of the requirements in plain language:

- Tony, the restaurant owner, wants to use promotions to get diners to return. His needs include the ability to generate tickets, print them, and allow winners to redeem tickets for a prize.
- Fred wants to use promotions to generate leads for his construction company. In addition to being able to generate tickets, he needs to be able to collect email addresses from potential customers.
- Bob wants to increase his customers' loyalty by giving them discounts on future purchases. He also needs his customers to be able to enter their sales receipts on the bookstore's website.

Note that "wants" indicate goals and "abilities" indicate steps in a business process.

In the second step (identifying commonalities and variabilities in the requirements model), we look for what is common among the models and in which ways they differ. For example:

- All business owners want to increase their sales through promotions.
- They want to collect information about their customers, but plan to do so in slightly different ways (sales receipts for Tony and Bob, and email addresses in Fred's case).
- They all need to generate tickets, but in some cases (Tony) the tickets are generated at the point of purchase, and in the other cases (Fred and Bob), they are generated via a website.
- All tickets have barcodes, but there can be different types of barcodes.
- All business owners need to allow winners to redeem their prizes, but they use different ways of informing winners (through a board for Tony, or via email for the others).

From this information, we can identify common and variable features, choose candidate components that provide those features, and identify configuration parameters for the components.

Examples of common features that all business owners require include:
- prompting users to enter data
- generating tickets
- selecting the winning tickets
- redeeming winning tickets

Examples of variable features that require different implementations for different business owners, or that only some business owners have asked for include:
- supporting multiple types of barcodes on tickets
- sending emails to winners
- registering and logging in customers

Examples of candidate components include:
- PHP Barcode to create and read barcodes
- PHP Mailer and SMTP in PHP to send emails
- MyDB database framework for PHP
- Tickets R Us' own components to generate random ticket numbers
- Tickets R Us' own components to check submitted tickets

Examples of configuration parameters include:
- text to display on the tickets
- barcode type
- flag whether to send emails to customers
- expiry date of the promotion

Rapid Prototyping Using a Configurable Platform

Antonio Misaka

About the Author

Antonio Misaka is a recent graduate of the Technology Innovation Management program at Carleton University in Ottawa, Canada. He is a former consultant for IBM and R&D researcher for NEC-Brazil. His research interests include software engineering and technology management. He also holds an MSc degree in Computer Science and Mathematics from the University of São Paulo, Brazil.

Citation: Misaka, A. 2013. Rapid Prototyping Using a Configurable Platform. *Technology Innovation Management Review*. May 2013: 18–24.

Keywords: web applications, rapid prototyping, configurable platform, requirements analysis, software product-line engineering