Technical note

IBS Consist two years later: The LEGO brick dream

by R. van der Salm

IBS Consist is a company that chose SanFrancisco™ as its strategical development environment, because this environment promised to deliver applications that could be assembled just like LEGO™ bricks. If the promise were true, the environment would enable actual component-based development. According to this principle, customers can buy open applications and assemble them to fit their individual needs. In a previously published paper, the author described the reasons why IBS Consist chose SanFrancisco and how this new technology was introduced in an AS/400® -oriented organization. As a result, many organizations around the world contacted IBS Consist to obtain more detailed information about their experiences. This technical note describes some further experiences of IBS Consist gained after the previous paper was written. From a management perspective, it can be seen whether SanFrancisco makes the LEGO brick dream come true and, if so, how that is done. Since it is important to know how to work in a LEGO brick environment, this note also describes how IBS Consist works in the SanFrancisco environment.

Cherish your visions and your dreams as they are the children of your soul; the blueprints of your ultimate achievements.

—Napoleon Hill

In 1998, an issue of the *IBM Systems Journal* featured the SanFrancisco* technology. One paper in the issue told why a typical AS/400* independent software vendor (ISV) such as the Dutch company IBS Consist B.V. invested in a technology like shareable frameworks.[1] Object orientation and the Java** programming language were at that time not often chosen by companies using the AS/400. Investing in such a technology meant investing a large amount of money in people, in training, and in building new applications.

The paper described how Consist B.V. began using SanFrancisco, including a training program and other facets. Throughout the world, other companies that were also accustomed to working with an AS/400 environment were very much interested in the SanFrancisco technology and how to become familiar with it. Many of them visited Consist B.V. to discuss the topic in more detail. This technical note stems from that interest and continues from where the previous paper stopped. It provides answers to the following questions:

- What has Consist B.V. done with the SanFrancisco technology so far?
- Where did SanFrancisco take Consist B.V.?
- Has the dream of component-based development come true, or is the story of "building software like using LEGO** bricks" a fairy tale after all?
- What is a good way of working in the SanFrancisco environment?

In the next section IBS Consist B.V. is introduced. Readers of the previously published paper will learn where SanFrancisco has brought Consist B.V. as a company, and new readers are given some back-

ground about the environment that is described in this note. The third section describes the business reasons for choosing SanFrancisco, based upon the component-based development paradigm. The subsequent section describes in what sense SanFrancisco meets the needs for "LEGO brick software engineering." The fifth section describes what Consist B.V. is now doing in this environment and how it is done. The sixth section talks about the benefits IBS Consist B.V. experiences from using this environment. The seventh section is a discussion of what IBS Consist thinks has to happen in the industry for components to be successful and grow in acceptance. The note ends with a summary and conclusions.

IBS Consist B.V.

Consist B.V. has sold financial and human resource management applications in the Dutch and Belgium market for over 20 years. Though it has delivered several generations of its applications, they have always been for an environment using IBM midrange computers. Their applications grew or were rebuilt on platforms such as the IBM System 3X and, since the late 1980s, on the AS/400. Although these platforms are, of course, quite different from one another, growing from one platform technology to the next was always relatively easy and natural. All environments had RPG (Report Program Generator) as the main programming language, so Consist developers could easily progress to the next generation of technology. For an application vendor, this capability is extremely important. Not only were the developers able to grow easily from one kind of technology to the next, customers could as well.

An application vendor like Consist has two important assets. First, the customer base is stable and large enough to make it possible to invest in product lines to keep up with the changing environments. Second, its employees have the necessary knowledge. Developers who have both the functional knowledge of what the basic needs and wants of the customer base will be in the future and who are able to translate those needs and wants into technical implications are of great value to the company. It must be clear that both the customer base and the developers are highly valued and, whenever possible, must be led to the next generation of technologies in incremental steps. So far, Consist has done a good job. It is the market leader for financial and human resource management applications aimed at middle-sized and large companies. Consist has 1400 of these companies as customers, mainly based in the Netherlands and Belgium, and about 150 implementations elsewhere in Europe.

In 1996 Consist B.V. decided to invest in SanFrancisco technology to develop future product lines. One reason was to be able to sell Consist B.V. products overseas. For a small company, selling overseas was then very difficult. The strategy was to find partners whose products Consist B.V. could sell in the "home" Benelux market and have the partners sell Consist B.V. products in their home markets. The only way to achieve this goal was, of course, by using a technology that enabled the integration of components that are built by the different partners. This meant choosing object technology. A thorough investigation pointed to SanFrancisco as the only promising environment to fulfill this strategy. It was the only architecture at that time to offer a platform-independent solution for open applications. Thus, since 1996 Consist B.V. has been heavily involved in everything related to SanFrancisco.

At the same time, the Swedish ERP (enterprise resource planning) vendor IBS had a similar strategy. IBS specialized in distribution and supply chain solutions, with worldwide offices serving about 3800 customers. Their major platform was the AS/400. IBS realized that its customers did not want the "confection" kind of ready-made applications that all large ERP vendors had been offering. IBS knew that "tailor-made standard solutions" based on components that can be bought from any vendor in the world are the next step in application development. In other words, the vision of IBS is that open applications have to be offered to the market. Although IBS had the intellectual resources and the financial power to develop its own closed technology like other large ERP vendors, it did want an open application standard. Therefore, the company has cooperated with IBM since the very beginning in the SanFrancisco development and together with IBM can be considered a founder of the SanFrancisco ideas.

IBS and Consist B.V. met each other because of their involvement with SanFrancisco. IBS was developing a next-generation open ERP solution, and Consist B.V. was developing next-generation open financial and human resource management components. IBS was looking for a strong partner in the Netherlands that could sell the current IBS ASW product in the Dutch market and that would be able to adapt the new technology in the future. Consist B.V. needed a product like ASW to complete their product offering. Thus, IBS and Consist B.V. were a perfect match.

IBS obtained a Dutch office with a strong position in the local market and, as a premium, a large group of skilled SanFrancisco developers. Consist B.V. fulfilled its goal of having a large worldwide network selling SanFrancisco components and obtained, as a premium, the ASW offering and access to a large amount of SanFrancisco knowledge from the IBS labs. IBS took over Consist B.V. from the former owners, IBM and Roccade, in 1998, forming IBS Consist B.V. as a first result of the mutual SanFrancisco investments of both companies.

Business reasons for choosing SanFrancisco: Component-based development

If you have built castles in the air, your work need not be lost; there is where they should be. Now put foundations under them.

—Henry David Thoreau

Component-based development (CBD) is currently a major issue in the information technology (IT) world. Everyone involved is at least talking about it. Every self-respecting application software vendor is telling its customers that it adapts to this paradigm. However, most vendors are just telling their customers a story because other vendors are saying the same thing in the market. To fully adapt to the CBD paradigm, a set of prerequisites has to be met, as described in the following subsections. In the next section these prerequisites will be matched with the SanFrancisco environment to see to what extent it is component-based.

Prerequisite one: Components. To develop software based on the LEGO brick principle means, of course, that there are "LEGO bricks," or components, available. A component can be defined as "a piece of software that is only accessible via its interfaces." A component provides one or more business services. These services can be relied upon, or certified, and the component can then be used in conjunction with other components to rapidly assemble a complex business support infrastructure, which is inherently adaptable and of high quality.[2]

Prerequisite two: Legacy integration. CBD can only succeed if it offers the possibility of legacy integration. Companies such as IBS Consist have large customer bases. These customers are the main assets of the company. Not being able to help these customers to move up to the next generation of technology means losing your main asset. Legacy integration is thus extremely important to grow, step by step, into new technologies. It enables growth by evolution instead of replacement. Legacy integration is a two-way process from the newly built component-based environment to the legacy environment and vice versa.

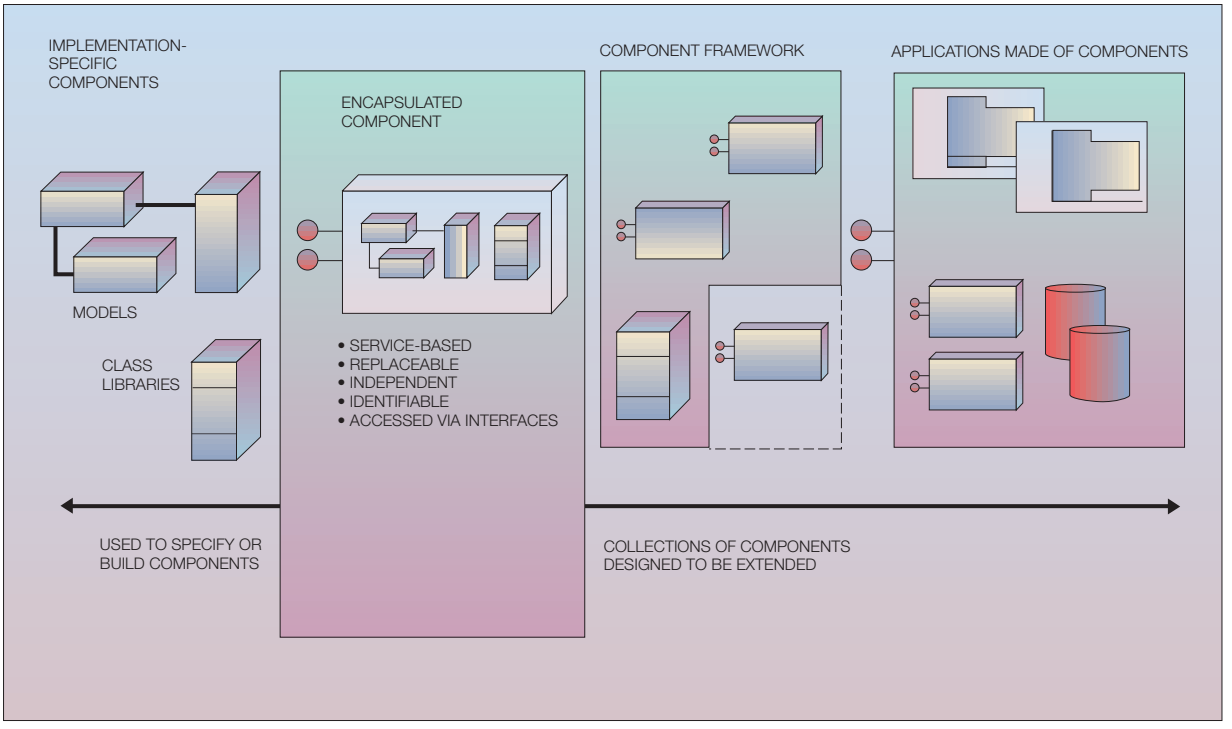
Prerequisite three: Technology independence. Customers will be demanding open applications, meaning that an organization does not want to depend on one application package vendor but wants to adopt a general architecture to be able to buy components or larger parts of an application from anywhere in the world. This general architecture needs to be based both on a generally available technology standard, such as Enterprise JavaBeans** (EJBs), Microsoft DCOM** (Distributed Component Object Model), and CORBA** (Common Object Request Broker Architecture), and on a general functional architecture such as a framework architecture. Although the IT world has been talking about open standards for a long time, it is still an illusion to expect one architecture to arise that supports all available standards. Therefore, it is important to choose a general architecture with broad support from the IT vendors rather than dreaming of an environment that supports everything.

Prerequisite four: Distributed object model. In the global Internet and e-business-based world we will soon live in, a distributed object model is extremely important as a base for a CBD environment. In the architecture for this environment, it is not important where objects are located or where processes are executed, as long as all components are related and synchronized throughout the network.

Prerequisite five: Easily adaptable business rules. Components will provide business rules. Sometimes these business rules will be just what an organization needs; often an organization has to fine-tune the business rules. It is important that this fine-tuning can be done easily and independently of the component itself, so the component can be looked at as a "black box," not needing to be maintained by software engineers who do not know all the details about it. Therefore, specialization of components must be possible through designated "plug points" rather than by copying a component and changing it.

Prerequisite six: Easily added attributes. Components offer a certain set of data elements that can be maintained by the component. Of course, it has to be possible to add data elements in a flexible way.

Figure 1 Component types



In an environment where components are bought from several vendors, it is very important that these vendor-specific components do not conflict with one another.

Of course, more prerequisites for a component-based development environment can be defined. The items mentioned above are the most important in the IBS Consist vision.

Do SanFrancisco frameworks enable LEGO brick software development?

The future belongs to those who believe in the beauty of their dreams.

—Eleanor Roosevelt

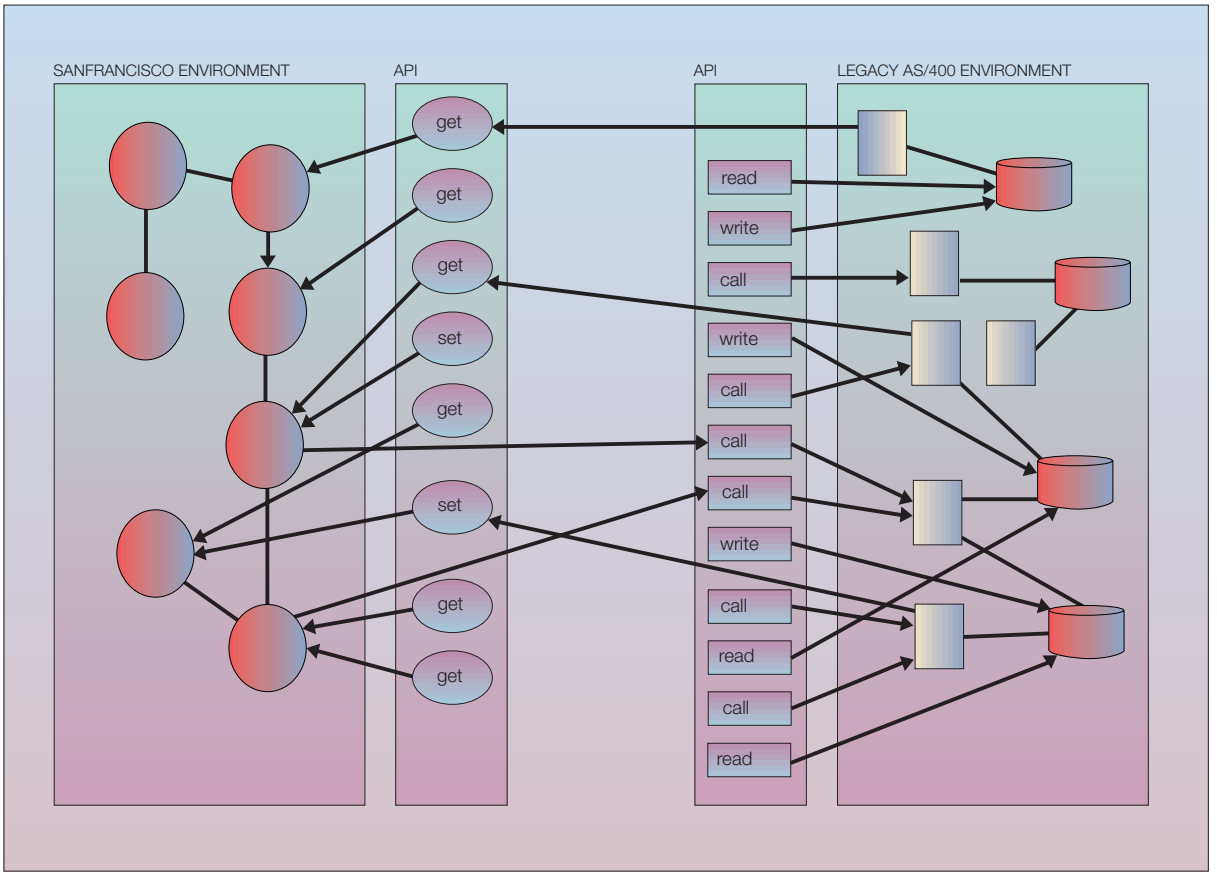
To check whether the SanFrancisco environment is truly a CBD environment, the experiences of IBS Consist B.V. will now be matched with the prerequisites listed in the previous section. This will be done both at an object-oriented class diagram level and in a way that is accessible for readers who are not trained in the use of the art of object orientation, but who are nevertheless interested in the promises of component-based development.

Prerequisite 1: Components. Components can be available in different ways. The Butler group has defined four main types of components.[3] They are displayed in Figure 1.

Components can be (1) small implementation-specific building bricks (individual classes), (2) encapsulated with several other components to form a larger building brick, (3) a framework consisting of a prebuilt assembly of components specifically designed to be extended, or (4) a prebuilt application.

It is clear that SanFrancisco components can be seen as component frameworks. An especially strong feature of the SanFrancisco frameworks is their ability to be extended through very well-designed add-on mechanisms. Thus, SanFrancisco easily meets this first prerequisite. Experience gained by IBS Consist B.V. indicates that the component contents are very rich. Compared to what has to be done to develop

Figure 2 Legacy integration through APIs



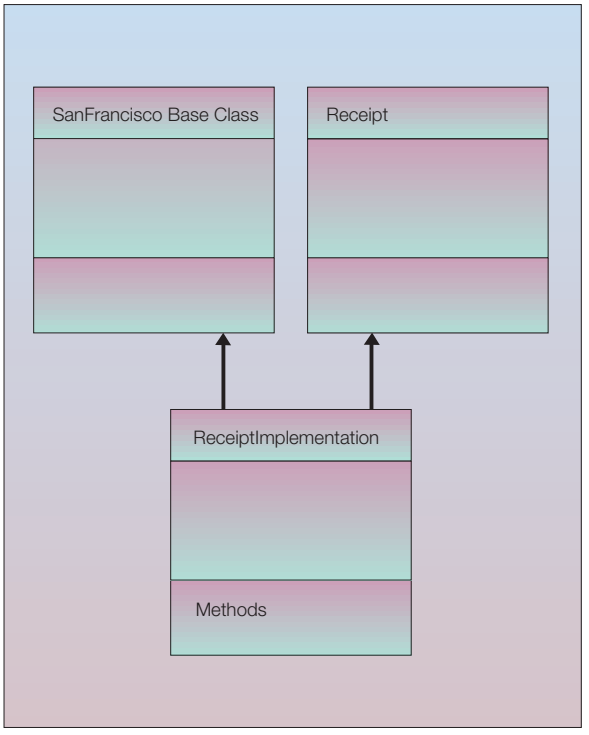
applications from scratch in a pure Java environment, the amount of system behavior that is offered exceeds the 40 percent IBM is claiming.

Prerequisite two: Legacy integration. Legacy integration is extremely important. Looking at the IBS Consist situation where sophisticated and broad financial and human resource management applications are currently offered in the traditional RPG environment, we can see that it would be impossible to replace all these applications at once by a new SanFrancisco-based application suite. Therefore, a careful integration and migration path has to be developed. Of course, integration is possible both at a database level and on a program call level as standard SanFrancisco functionality.

The solution IBS Consist offers is slightly different. The current applications that the company delivers have large and extended sets of application programming interfaces (APIs). On the SanFrancisco application development side APIs are also developed. By using these API layers, both the legacy applications and the new-generation SanFrancisco applications can be developed independently of one another and still use the services of one another. This independent development might look like extra work but will in the long run give the better return on investment. The traditional relational databases in the legacy environment and the databases that are storing the objects of the SanFrancisco environment are independent of one another. Integration is possible in a two-way direction. This integration can be pictured as shown in Figure 2.

From this prerequisite it can be concluded that legacy integration can be done in several ways. What is the best way has to be determined for each sit-

Figure 3 Proxy pattern



uation. The example above implies that there may be duplicate data in both the legacy and object store. Although this duplication may provide an option to choose, it is not necessary. SanFrancisco offers components the ability to share common persistent data or objects. These components can be SanFrancisco-based components as well as legacy components.

The legacy integration options give IBS Consist B.V. the possibility of offering small SanFrancisco-based applications to their customers with extra functionality as compared to the current offerings. Customers can grow into the SanFrancisco architecture step-by-step without having to replace their current applications all at once.

Prerequisite three: Technology independence. SanFrancisco previously has been defined as a complete Java architecture. Java in its first, rather primitive form mainly supported the client side of development. There was a lack of support for the technical infrastructure every application needs to be stable and robust. This technical infrastructure should support services such as authorization, transaction management, and persistency. A typical Java environment did not have these kinds of services. SanFrancisco was the first environment to provide them by means of the foundation layers. Two years ago these foundation layers were probably the most important part of SanFrancisco because of the enormous added value it gave to Java developers. However, Java-only adepts were not happy because they saw SanFrancisco as strictly an IBM product and not as a part of the new open Java world.

With Enterprise JavaBeans (EJBs) as the definition of the component model for building robust business-critical Java applications, an alternative became available for the SanFrancisco foundation layers. IBM realized what had occurred and announced that the foundation layer would be replaced by EJBs. With that move, the SanFrancisco environment truly opened up for everyone who sees EJBs as the architecture for future application development and deployment.

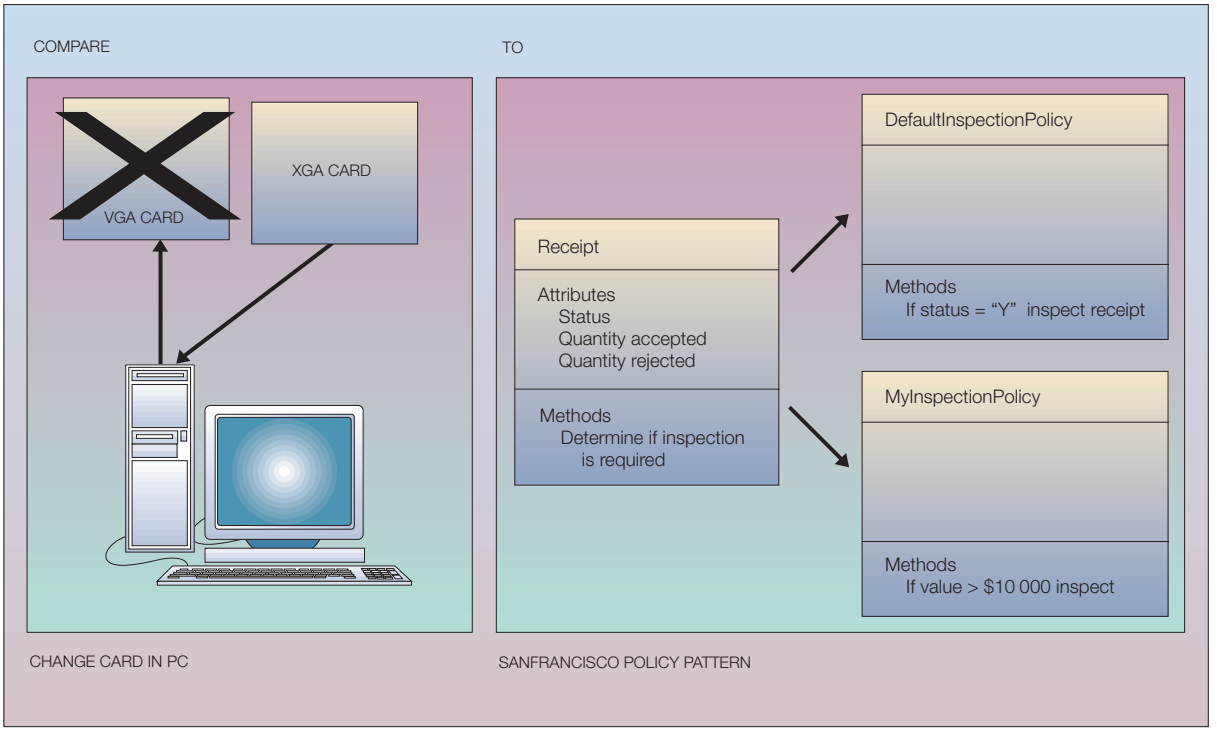
Thus, SanFrancisco will meet this prerequisite as soon as the migration to EJBs is completed.

Prerequisite four: Distributed object model. The growing Internet world will demand applications that can be distributed in a large network. Running an application in New York on a UNIX** machine, in Amsterdam on an AS/400, or in Stockholm on a PalmPilot** must be transparent to the user. Therefore, a component-based architecture must have the ability to know how to behave in such a heterogeneous network environment. The way this works is shown in Figure 3.

SanFrancisco classes all inherit from a SanFrancisco base class that knows how to behave in a distributed environment. Therefore, all SanFrancisco objects know this behavior and can act in any network environment. Thus this prerequisite is also matched.

Prerequisite five: Easily adaptable business rules. An environment that enables components from several vendors to be assembled together has as a very important requirement: the possibility of changing the business rules that are delivered in the components.[4] Examples of business rules that have to be changed are policies for sending chase, or follow-up, letters or country-specific legislation. For the long-term continuation and stability of components and for the possibility of replacing components, it is important that the components themselves not change. Suppose that in the SanFrancisco environ-

Figure 4 Policy pattern

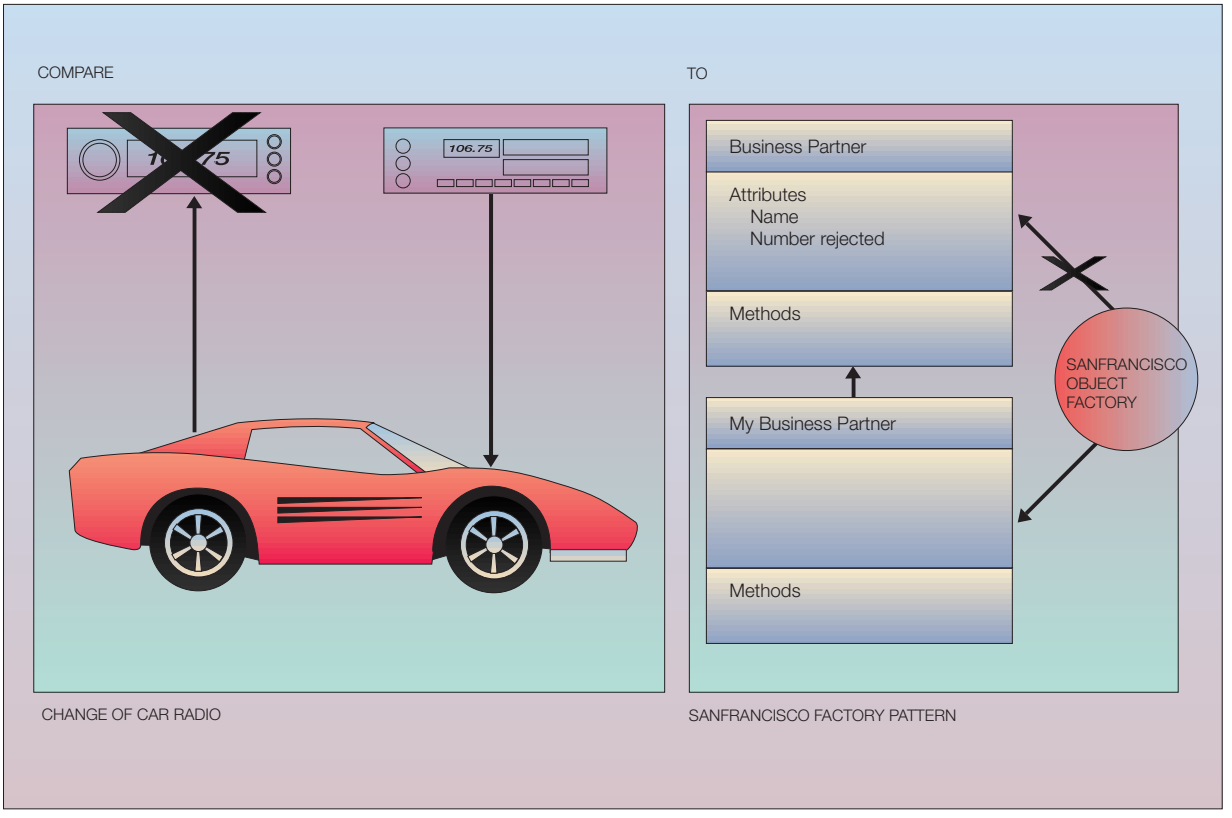


ment an individual user changed the framework containing the chase letter generation and the standard chase policy. After that it would not be possible to replace a current version of SanFrancisco with a new version without again doing all the change work to whatever changed the policies.

Suppose there is a standard business rule in a framework for determining whether, upon receipt of goods, a quality check of the goods is needed or not. Suppose the standard business rule is to check a status code on article level, and if this code is yes, then a check has to be done. This rule may not be sufficient for a certain company, and instead the business rule states that a check must always be done if the value of the goods is larger than $10 000. This business rule must be replaced in a component-based manner. This replacement can be compared, for instance, to the way in which PCs are built out of components. A PC with a VGA card (a business rule requiring VGA) can be easily changed to a PC with an XGA card. All that must be done is to plug the XGA card into the proper place. SanFrancisco uses the same principle where important business rules are part of the frameworks. This solution is known as the "policy pattern." The policy pattern makes it possible to easily change business rules in a release-independent and vendor-independent way. It can be visualized as the class diagram shown on the bottom right of Figure 4.

Another way of changing business rules is by class replacement. Suppose a framework architecture delivers a class "Business Partner" with some business rules in it. If this class is not suitable for use in a distribution application and is replaced with another class, problems could arise. Suppose the Business Partner class is used in the financial component but not in the distribution component. If these components come from different ISVs, it is not clear whether the Business Partner class has been replaced in the financial application. To solve this problem, the factory pattern is implemented on every class within SanFrancisco. In short, this pattern implements a factory for each class. The class-specific factory handles the creation or instantiation of that class. It is the BaseFactory class (retrieved by a Global.factory( ) method call) that handles the deletion of objects. Changes to objects are accomplished by the

Figure 5 Factory pattern



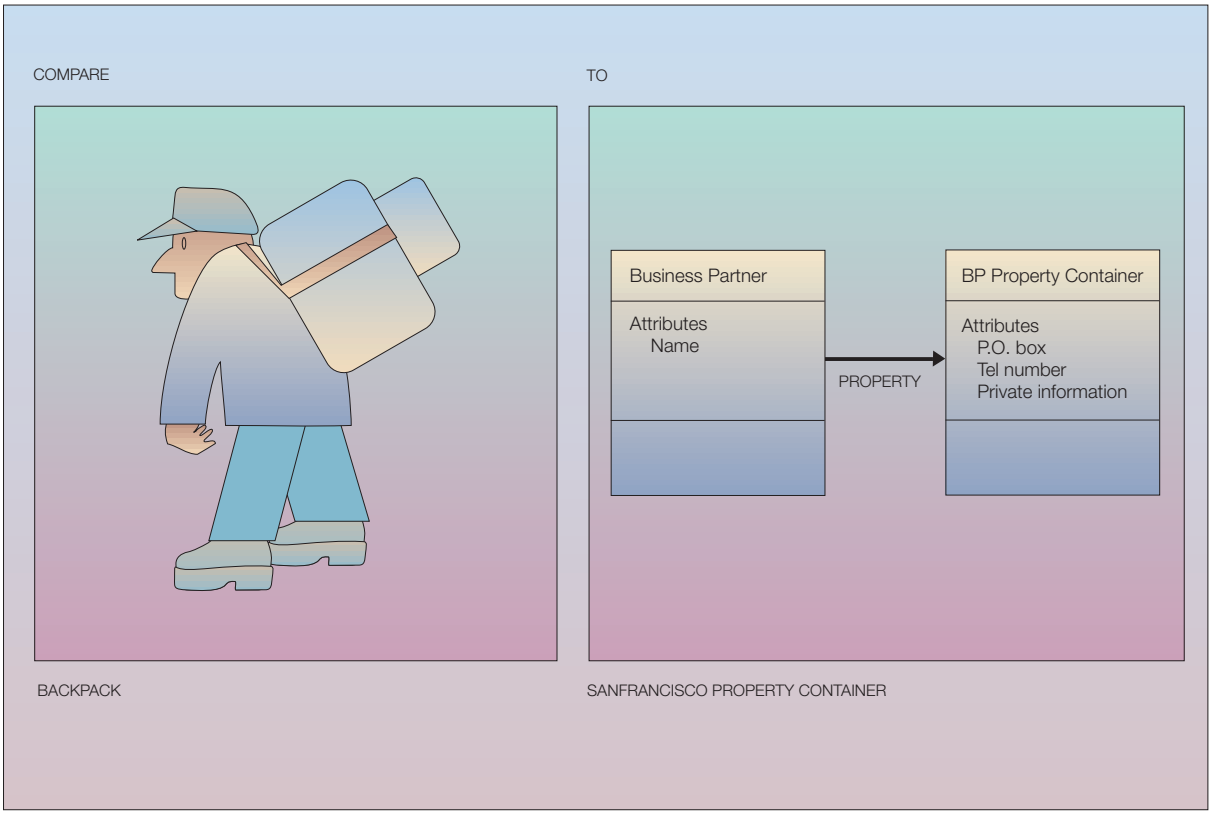
objects themselves. This implementation can be pictured as in Figure 5.

The SanFrancisco architecture has more standards to change the business rules than the two described above. The two described above are the best examples to explain the possibilities of the "LEGO brick."

Prerequisite six: Easily added attributes. The first question asked in a discussion of "LEGO brick" applications is: "Can I add my own attribute types to whatever information is already available?" The answer, "Of course, that can be done in any environment," is a little too simple. The problem in adding extra data elements, including the correct business rules for the data integrity in a LEGO brick environment, can be explained as follows:

For this example, the Business Partner framework is used again. Suppose one ISV decides to add the data element "P.O. Box" to the Business Partner class. Because this ISV delivers a financial application, the business rule for this data element states that it is mandatory to fill it in on a form. The reason is that an invoice should always be sent to a post office box. Suppose another ISV implements the same Business Partner class. This second ISV delivers a warehousing application. Because goods always have to be delivered to a physical warehousing address, the business rule for this partner is that a P.O. box should never be filled in on the form. As can be seen in this oversimplified example, a conflict of business rules between several applications can occur. In SanFrancisco this problem is solved by the use of so-called property containers. Classes that extend DynamicEntity or implement the PropertyContainer interface have a property container. A property container can be seen as a backpack where all kinds of extra data for the class can be added with their own business rules (Figure 6).

Figure 6 Property container



In this way each class can have all the attributes that each ISV needs, along with all the specific business rules an ISV needs, but without changing the frameworks themselves and without affecting other applications with the attributes.

How to work in a SanFrancisco environment

> *An expert is a man who has made all the mistakes which can be made, in a narrow field.*
>
> —Niels Bohr

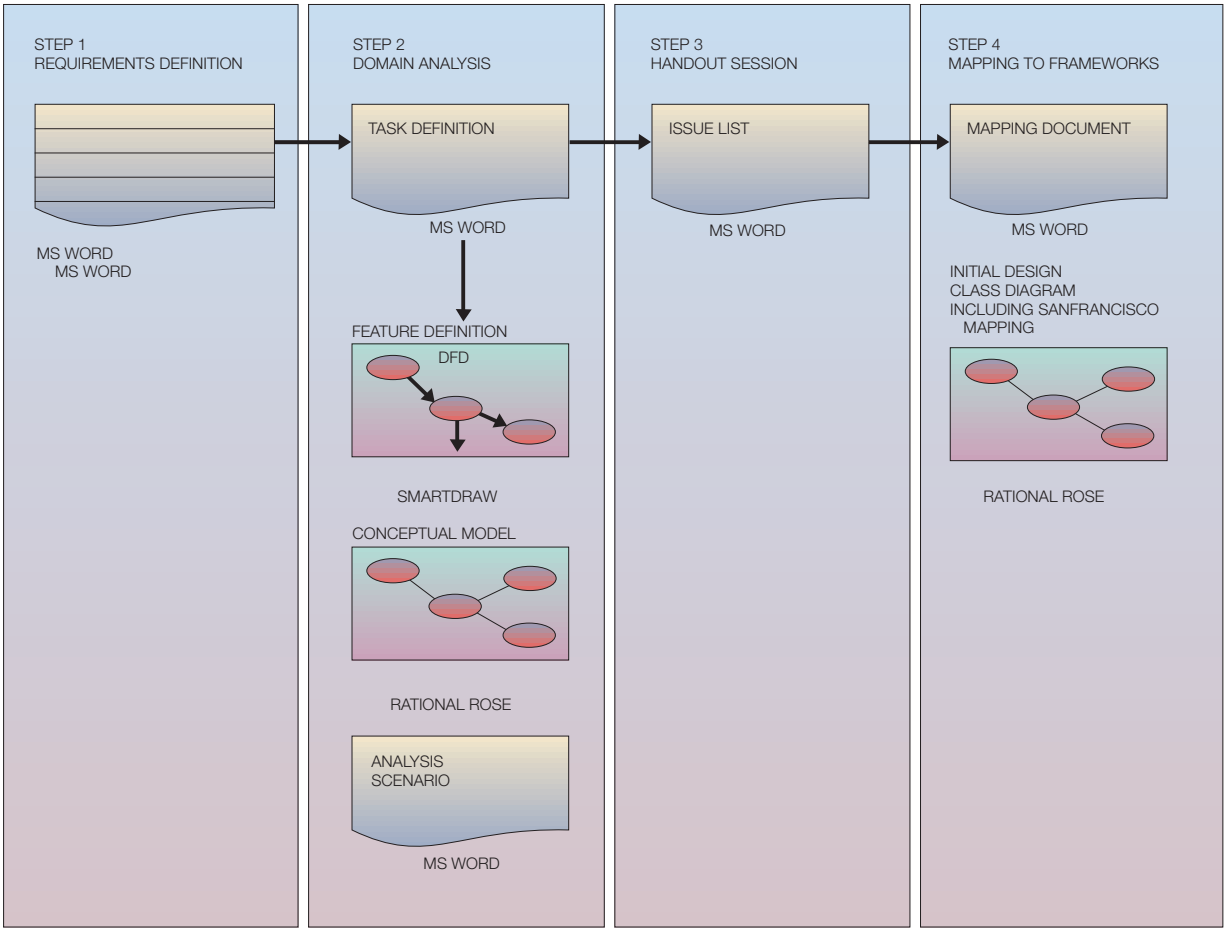
SanFrancisco enables the prerequisites if good application design principles are followed. There is no magic involved. Good methodology and design work are still necessary. In this section the working methodology used by IBS Consist is described.[5]

Working in a SanFrancisco environment requires a different method compared to working in an RPG-based AS/400 environment. Apart from the work being iterative and based on time periods, the biggest difference, of course, is to define what one wants to build and to map that to what the frameworks are offering. For each process and every part of the application that is to be developed, it is necessary to find the balance between the following questions:

1. What do I need in my application?
2. What is available in the frameworks?
3. Can I obtain exactly what I want using the frameworks?
4. If not, how much does it cost me to fulfill my requirements by not using the frameworks?
5. How close can I come to my original goal by using the frameworks?
6. What are the differences in results, costs, and revenues when I use the frameworks compared to when I do not use them?

To obtain the correct answers to these questions and to make the correct choices, the working method (or

Figure 7 IBS roadmap domain steps



the roadmap) to be followed has to be laid out carefully. Although the documentation for SanFrancisco lays out a recommended roadmap for developing applications with SanFrancisco, IBS Consist B.V. has developed its own roadmap. The development process is differentiated in two main stages. In the first stage the domain analysts are in the lead, and in the second stage the technical people are in the lead. During the development process many different roles are fulfilled in the project team. The domain analysts know everything about the domain, the market, the users, etc., and almost nothing about the technology. The technical people know how to develop software in a SanFrancisco environment and know a great deal about Java, object orientation, and similar things. Some of them have domain knowledge, but not as much as the domain analysts.

The first part of the process focuses on the domain of the system to be developed. The steps to be followed can be pictured as in Figure 7.

The first step is to define the requirements of the application to be developed. The domain analyst, managed by the domain lead, performs this step. By looking at trends in the market, talking with customers and marketing people, etc., the analyst produces a document that describes the requirements of the new system. SanFrancisco is not an issue here. This step is about describing *what* is needed, not about *how* the frameworks provide support. The document is created in Microsoft Word** and disseminated throughout the whole company to be discussed with everyone involved.

The second step is the domain analysis. During this step the requirements are structured according to the processes to be supported, tasks that are part of the processes, features that are part of the tasks, and descriptions in plain English about what these processes, tasks, and features should do. To create a clear view of the processes and to define the borders for each process, each process is structured in a data flow diagram (DFD) with a drawing tool. Within IBS Consist, SmartDraw** is used. Of course, DFDs are not part of any object-oriented design methodology. However, using DFDs helps to lay out the process on a domain level very well. In communications between domain people and technical people and among domain people it is particularly helpful. In the long term, this documentation will not be maintained. It is just used to define the processes. A domain analyst is able to create a DFD, but creating a correct class diagram is much more difficult. High-level class diagrams are created during this step. Design leads make the diagrams when talking with the domain people. At this stage SanFrancisco is still not involved, so there is nothing that has anything to do with SanFrancisco in the class diagrams.

The handout session is an important element of the first part of the process. During the domain analysis, features are defined. A feature is a comprehensive part of work that can be done in a relatively small amount of time (±1 month). During the handout session, the domain analyst tells the developers all the details about the feature that they will work on. It is in these discussions that SanFrancisco becomes important for the first time. Items not clear to the developers are written down on an issue list.

From then on the developers are in the lead. After the handout session the requirements are mapped to the frameworks. Standard forms are filled in, describing in detail how and where the mapping will be done and what "LEGO brick" mechanism will be used. The design class diagrams are then created (using Rational Rose**). The mapping to the frameworks is part of these designs.

The steps from the domain analysis to the mapping are performed iteratively on a feature level. Much communication takes place between the developers and the domain analysts about the trade-off between using the frameworks "plain" or enhancing or replacing them. On the one hand, most of the time it is not clear to the domain people how much effort will be needed to change or enhance the frameworks to meet their exact requirements. On the other hand, it is not always clear to the developers how important some requirements are for the application to be accepted or to have a unique appearance in the marketplace. If developers and domain analysts cannot decide what is best, the domain lead will make the decision. If the choice means that the amount of time needed will be too great, the project manager decides whether or not to implement the specific feature in the first release.

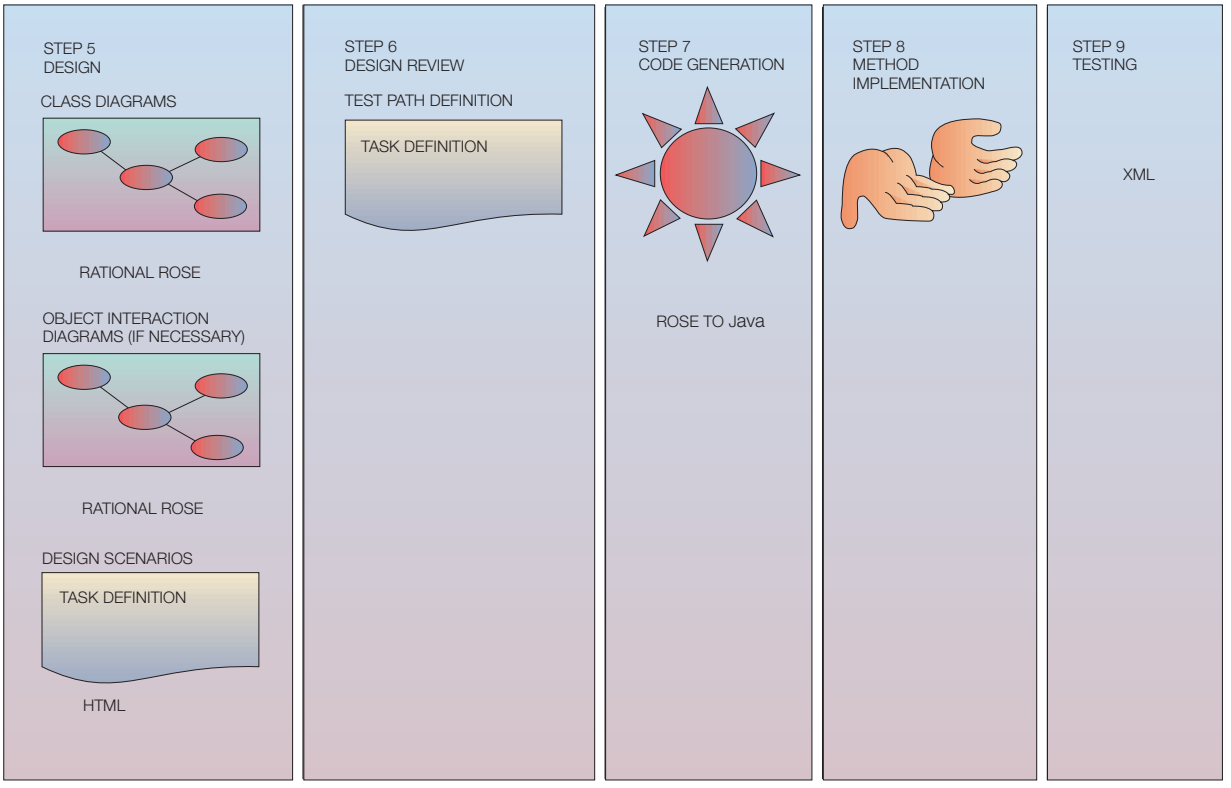
A large part of the development process consists of all the technical work.

A second, and large, part of the development process consists of all the technical work. This stage can be pictured as in Figure 8.

After the mapping to the frameworks has been done and all the choices have been made about adding functionality to the frameworks, the developers complete the design. This work includes producing the class diagrams, writing design scenarios in HTML (HyperText Markup Language) format, and creating object interaction diagrams, if necessary. Part of the design work is coding the requirements in such a way within Rational Rose that much of the Java code can be generated automatically. After the design is finished, the design lead reviews it to be sure all the standards have been followed and a sound and correct design has been developed. After this review, the test paths are defined to make clear what is to be tested after the coding. This operation is done using XML (Extensible Markup Language) so that the test scripts will be generated and processed automatically later.

The next step is generation of the Java code with use of a Rose-to-Java generator (named the SanFrancisco Code Generator, or SFCG) that is part of the SanFrancisco tool set. After that step the method implementation has to be coded manually. Of course, subsequent steps include technical testing, integration testing, and usability testing, the same as in any other software development environment. A large

Figure 8 IBS roadmap development stage



part of the testing is done automatically. The test script generator uses XML and is being developed by IBS Consist.

From experience with this method, IBS Consist has found that it is a structured way to obtain high-quality applications. Especially from a financial perspective, the SanFrancisco frameworks provide very rich functionality that can be easily enhanced.

Benefits for IBS Consist from using SanFrancisco

> *But Charlie, remember what happened to the man who suddenly got everything he always wanted. . . . He lived happily ever after.*
>
> —Roald Dahl (from *Charley and the Chocolate Factory*)

IBS Consist was one of the first companies to start using the SanFrancisco frameworks worldwide. After making the effort to step into the object-oriented and Java world, the benefits of working in the SanFrancisco environment and being such an early adopter are tremendous. The original goal of developing SanFrancisco-based applications that could be sold worldwide by other SanFrancisco partners has been reached earlier than expected with much better results than expected. Because IBS Consist is now a part of the global IBS enterprise, it is able to develop products in a worldwide organization with a worldwide distribution channel. Being able to build a large network of SanFrancisco partners worldwide that use and sell one another's components is still a goal, both for IBS and for IBS Consist.

By informing customers about SanFrancisco, its "LEGO brick" possibilities, and the open architecture, IBS Consist has developed a much closer relationship with many customers. Customers believe in the SanFrancisco goals and stay with IBS Consist to grow together into this new world. The decision by IBM to put an EJB foundation underneath SanFrancisco based on WebSphere* has done much to help, be-

cause it makes the SanFrancisco architecture a truly open one.

A new opportunity to give service to IBS Consist customers has been developed. Because customers realize that it takes time to learn about the framework technology, they like to be helped in obtaining this knowledge. IBS Consist has a great deal of experience in gaining such knowledge because it trained its own large group of developers in object orientation, Java, and SanFrancisco. The company maintains a structured training program, available through its intranet, that helps interested employees obtain the proper knowledge. Apart from theoretical aspects, this environment offers many practical cases, based on real-life experience, that help developers grow into the SanFrancisco world. Customers are very interested in this experience, and IBS and IBS Consist help them to obtain this kind of knowledge.

But mainly, IBS will deliver, with the help of its Center of Excellence within IBS Consist, a state-of-the-art, next-generation, Java- and SanFrancisco-based application. This application is built and delivered according to the LEGO brick principle so customers can buy open applications.

The component-based software industry

An important question to answer is: "Will the LEGO brick dream really come true?" In other words: "What has to happen in the industry for components to be successful and grow in acceptance?" Is CBD just another kid on the block like Computer Assisted Software Engineering tools, AD Cycle,* etc.—technical promises that had their use in the software industry but never fulfilled everything they promised? Every new trend always promises better time to market, more productivity, better applications, more standardization, etc. Why is CBD different?

For IBS Consist B.V. the question is not: "Is CBD actually going to happen?" The real question is: "When are all prerequisites filled in to make it happen?" There is no doubt about the need for the IT industry to become truly industrialized. All major package vendors are criticized about poor implementation and enormous implementation times. As a response, all major players are investing in CBD so that software implementation will become more and more a "plug and play" world.

Another important trend that reinforces investments in CBD technology is the demand from customers to become independent of their software suppliers. Open applications are wanted. In the last decade, the trend for large organizations has been to get rid of the tailor-made software they had and implement standard applications. To a certain extent this change has been successful. Although implementation times are long, the results are not always exactly what is needed, and the costs are higher than expected, the process of implementing "ERP-like solutions" is much more manageable than the process of developing and maintaining all needed information systems by themselves. However, the real downside is that if everyone implements the same kind of ERP system from the same supplier, it is difficult to obtain competitive advantage from the information systems. Therefore, the demand of organizations will be to obtain the best of both worlds—the maintainability and manageability of standard application environments and the perfect fit of tailor-made solutions. CBD is necessary to fulfill this demand.

Before the IT industry can make this happen, many things have to change. The most important ones follow.

1. The first step is to standardize the "plumbing" underneath all information systems. This standardization is needed to make applications truly open. This process is going very fast at this moment. The main types of plumbing that will remain are the EJB model versus the COM/DCOM model. For SanFrancisco developers, IBM's decision to move SanFrancisco to the EJB plumbing environment means that SanFrancisco frameworks are becoming a fully open standard.
2. The tools available for development in a CBD environment must be improved. The Java language is easier to use than C++ but is still considered low-level programming. To improve development speed, the productivity of Java tools has to improve.
3. CBD has long been a technical issue. However, what is important about components is their content. Technology vendors have to become content providers. It will take some time for these vendors to realize this and change their way of working.
4. The major application vendors have to enable CBD. However, they also have to protect their current markets and, therefore, are looking for an evolutionary way to grow into CBD. This will take some time.
5. CBD needs to set standards on methods, technol-

ogies, tools, etc. It will take some time for these standards to be accepted worldwide.

The gurus predict that in the future 60 to 80 percent of all software development will be based on CBD. IBS Consist agrees with this assessment but thinks it will take at least five years before it happens. The market is being set now. Because of the necessary learning curve and the component development time, application vendors should invest now in this paradigm.

Summary and conclusions

IBS Consist has long been involved with the SanFrancisco technology. The promise of SanFrancisco was that applications could be delivered like LEGO bricks, that is, by assembling all kinds of components on a general architecture. It meant that customers had a choice of truly open applications.

Since the technology has been used for some time and the first applications developed with the technology have been delivered, this LEGO brick promise could be reviewed. Experience shows that SanFrancisco contains several mechanisms that fulfill the promise. Of course, it is important to use these mechanisms so that LEGO brick environments will flourish.

Working in a component-based development environment requires new ways of working compared to working in a more traditional environment. The most critical part is mapping the requirements to the functionality that is already in the frameworks. It is difficult to choose between the quick solution of using the rich framework functionality or using more time to change the frameworks so that they support the requirements more precisely.

This technical note has provided an overview of how the LEGO brick dream is implemented in the SanFrancisco environment. In addition, the way in which IBS Consist works in this environment is described. It is intended to provide the reader with a high-level understanding of these topics.

Cited references

1. R. L. van der Salm, "Introducing Shareable Frameworks into a Procedural Development Environment," *IBM Systems Journal* **37**, No. 2, 200–214 (1998).
2. *Component Based Development*, Management Guide, Butler Consulting Group Limited, Netherlands (March 1998).
3. *Component Based Development*, 1998 Report Series, Butler Consulting Group Limited, Netherlands (April 1998).
4. E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Publishing Co., Reading, MA (1995).
5. Based on the *California Roadmap* by E. Callebaut, IBS (1998).

Accepted for publication December 10, 1999.

Rob van der Salm *IBS Consist B.V., Nevelgaarde 20, P.O. Box 500, 3430 AM Nieuwegein, Netherlands (electronic mail: Rob.van.der.Salm@consist.nl).* Mr. van der Salm is a deputy director for research and development within IBS Consist B.V. He has been involved in the SanFrancisco project since 1996. Before that he worked as a project manager for several application vendors.