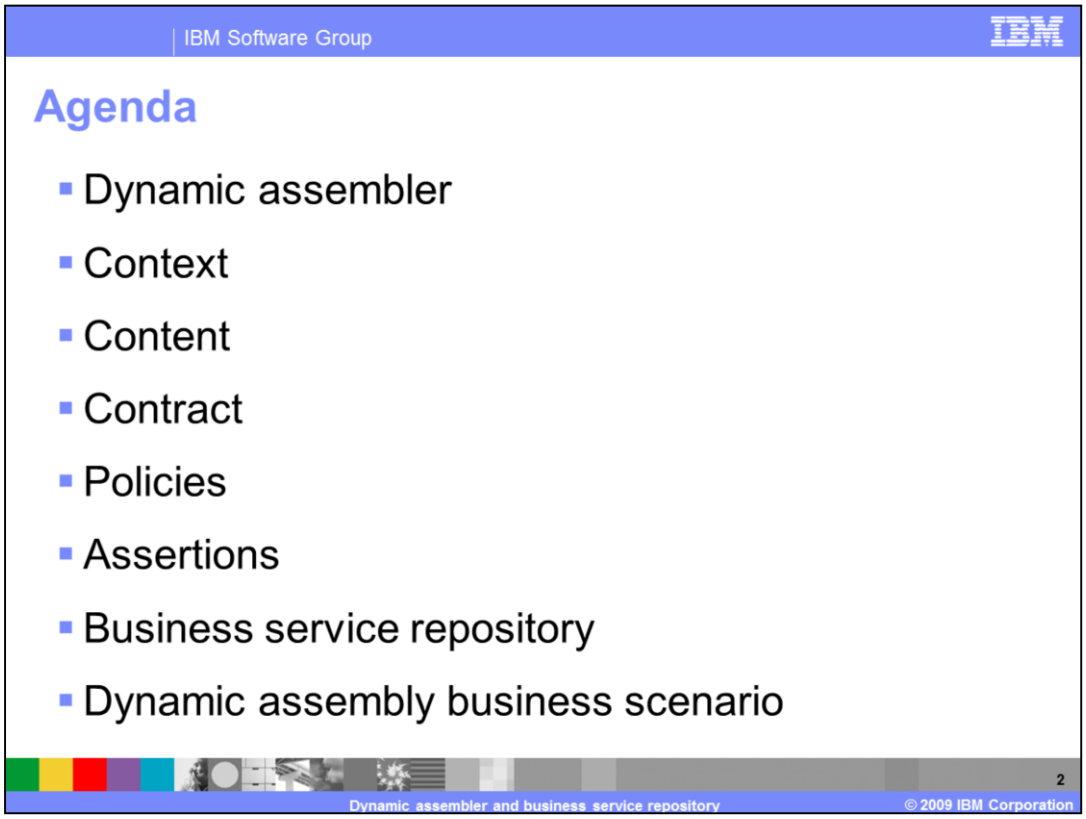
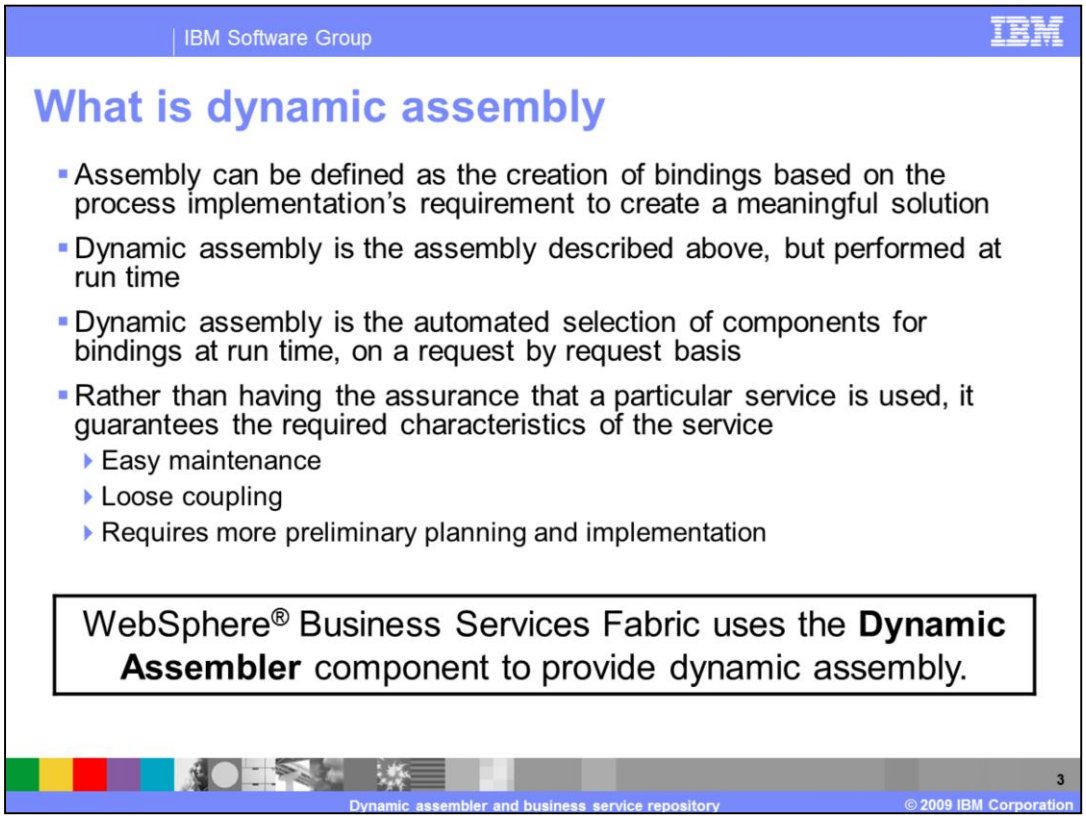

This presentation looks deeper into the foundation pack, providing details about the dynamic assembler and business services repository.

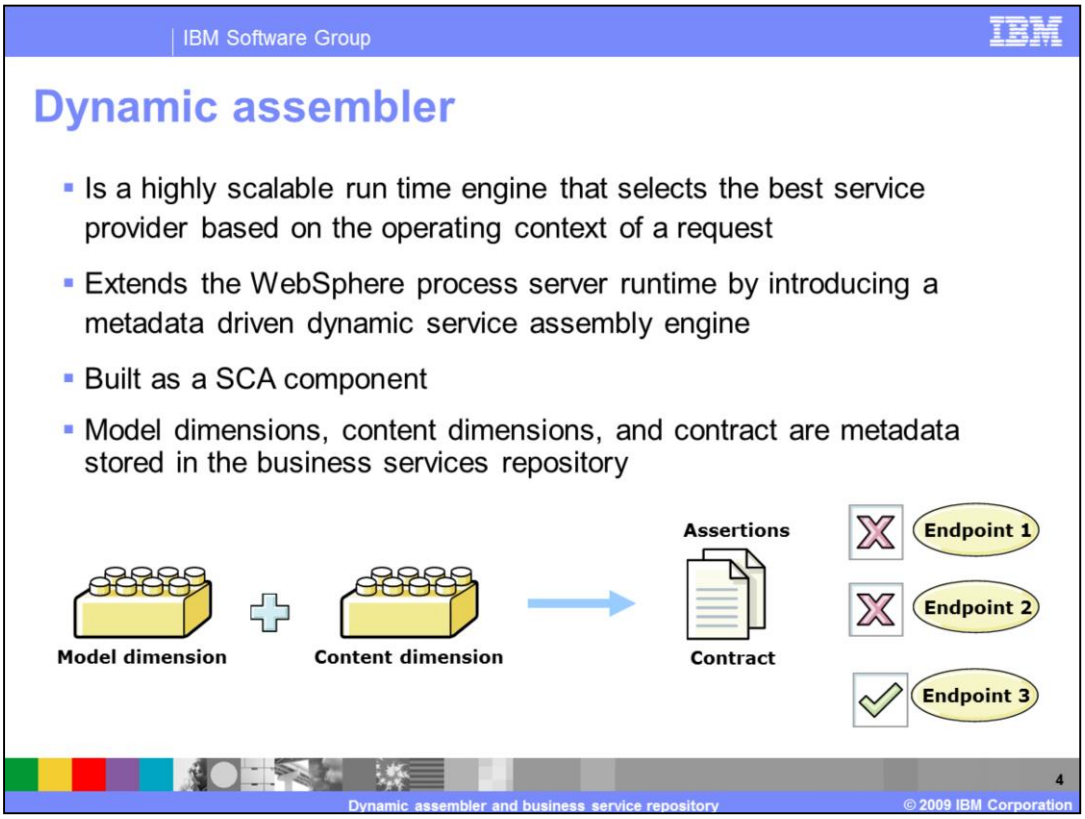



This presentation is focused on providing a basic understanding of the dynamic assembler, business polices and business services repository. First you will get an introduction to the dynamic assembler. Second you will look into context, content and contract and how they help define business policies and assertions. Next you will learn the basic concepts about the business services repository. In the end there is a walk through of dynamic assembly business scenario.




What does dynamic assembly mean? Lets look into the first the meaning of assembly. Assembly can be defined as the creation of bindings based on the process implementation's requirement to create a meaningful solution. It is the creation of bindings for a service request to a service provider based on a predefined process implementation. Therefore, it can be said that the binding between the service request and service provider is static and inflexible to change depending upon the business scenario. If the assembly was done at runtime, it is then called dynamic assembly.

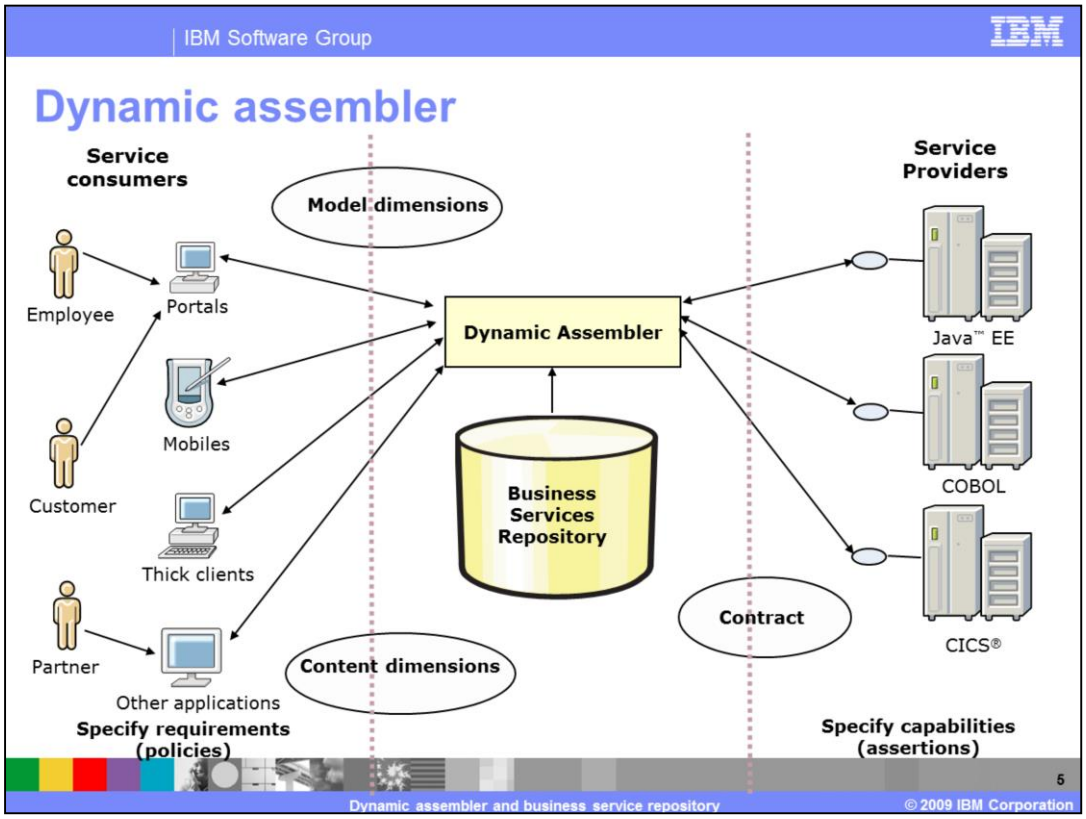
This ability to assemble dynamically at runtime makes the business process flexible to adapt to different business scenarios. It further enables easier maintenance, loose coupling and extracts the various decision points into business policies. The WebSphere Business Services Fabric uses the Dynamic Assembler component to provide this dynamic assembly.



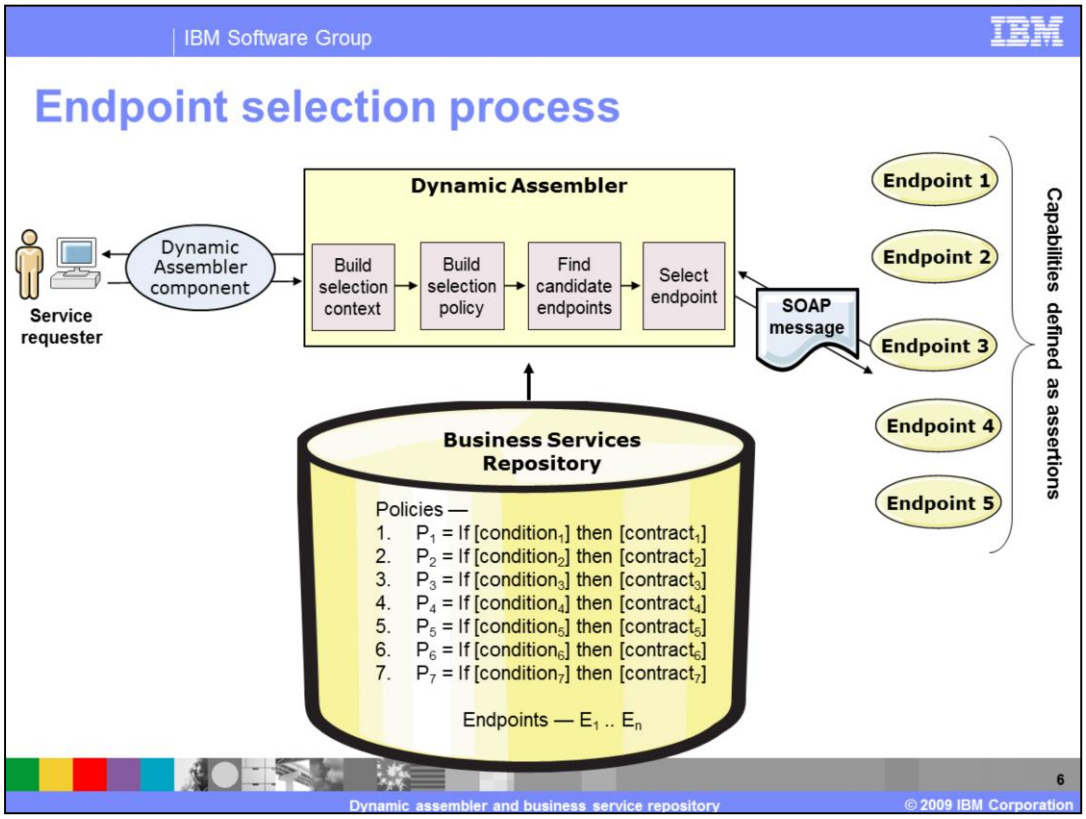

The WebSphere Business Services Dynamic Assembler determines the best service provider or endpoint that is to be used at runtime based on the incoming business request, business environment and context of the request. It analyzes the incoming business service request and the related relevant policies as per the operating context and from that assembles one composite policy or selection policy. Once the selection policy has been determined, candidate endpoints are chosen that match the criteria established by the selection policy. These candidates are then ranked and tiered. The most appropriate endpoint candidate is selected as the binding for the current request. The business service request is then sent to that particular endpoint or service provider to be processed.

The Business Dynamic Assembler extends the WebSphere Process Server runtime by introducing a metadata driven dynamic service assembly engine. The Dynamic Assembler is packaged as a Service Component Architecture (SCA) component and is managed by the SCA runtime within the WebSphere Process Server.

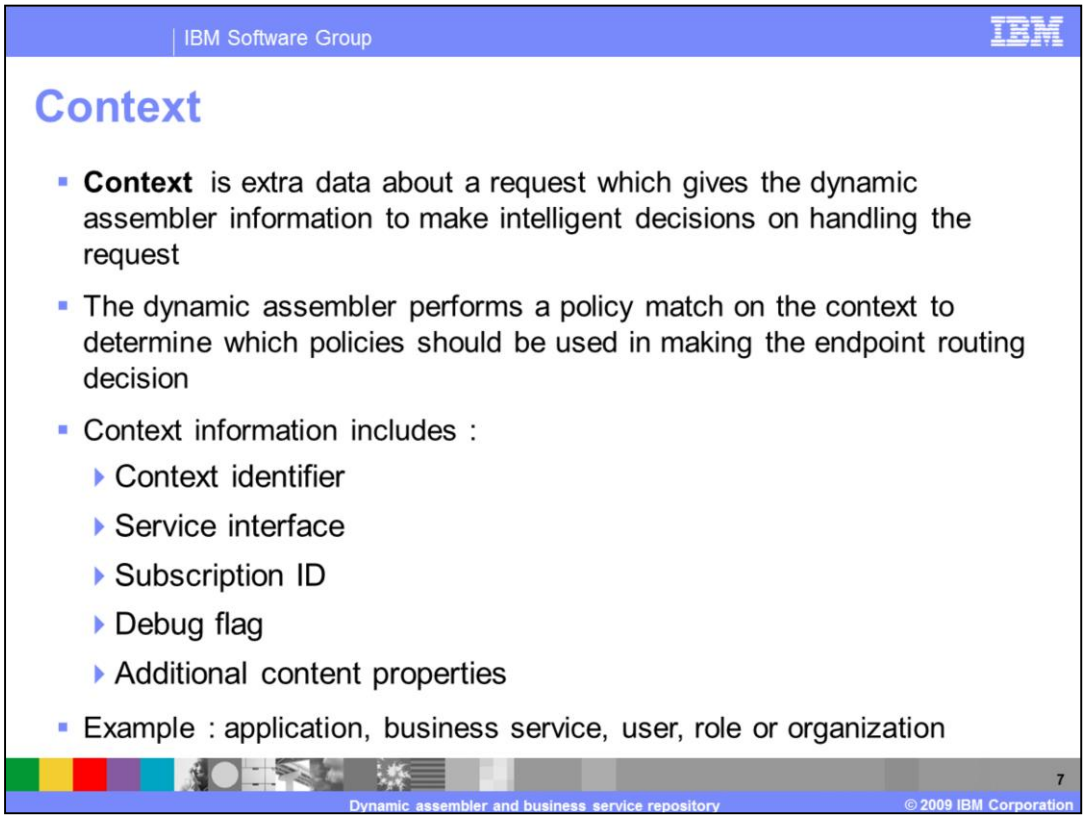
There are extensions available for the dynamic assembler component giving users more visibility and control in the dynamic selection process run by this engine.




This chart gives a more pictorial high level view of the dynamic assembler workflow. On the left side are listed the service providers that connect or invoke the business process through various channels like portals and mobiles. The dynamic assembler upon receiving the business request takes into account the model dimensions and the content dimensions along with the request to identify all applicable business policies in the business service repository. It consolidates these policies into selection policy and establishes a contract based on this policy. The dynamic assembler then looks at the capabilities of the set of available endpoints and chooses the endpoint or service provider that best satisfies that particular business request. The business request is then transmitted to the selected service provider to complete the transaction; thereby completing an end to end execution of the workflow.

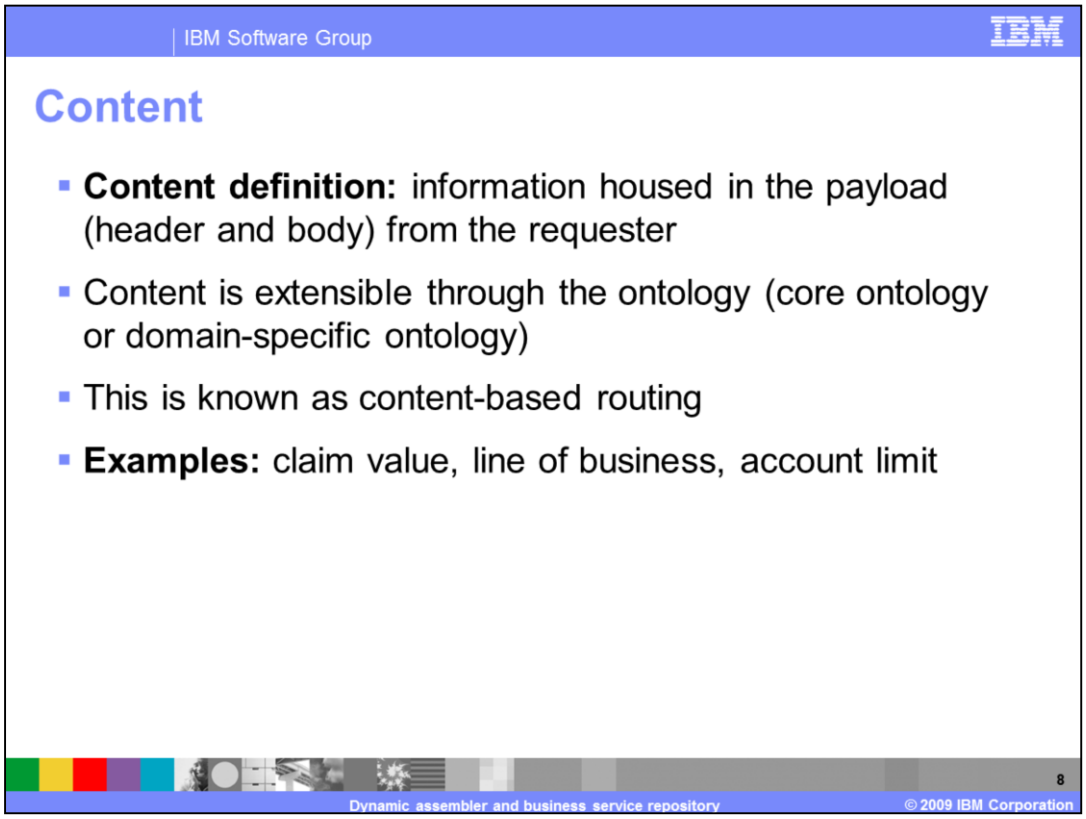



This chart describes how the dynamic assembler applies business policies and selects an endpoint. The first step in the process is to build a selection context for the incoming request. This means it assembles personalized information about the service request such as the channel, user, group, role or organization. Once the dynamic assembler defines a selection context it looks into the business service repository for the all the business policies that apply to the selection context. The policies are then processed on the context and the content of the incoming request which results in selection contract. Next the dynamic assembler looks into the business service repository to find all the candidate endpoints that can satisfy this contract. The candidate endpoints are then tiered and ranked and the most suitable endpoint is selected that satisfies that business scenario. The SOAP message is then sent to that selected endpoint to complete the transaction.

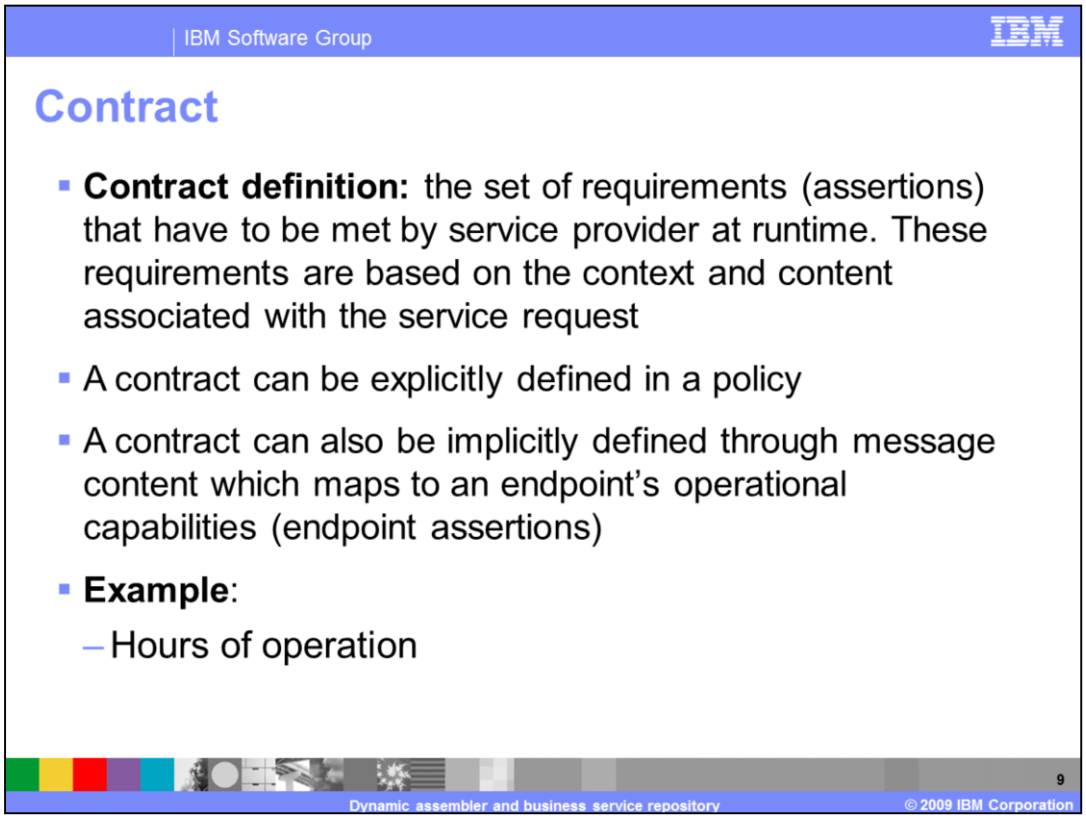



The last slide walked through the end point selection process. In that process you came across concepts like context, content, contract and policies. In the next few slides, you will look deeper into these concepts in terms of what they mean in WebSphere Business Services Fabric. First let's look at context.

Context is extra data about a request which gives the dynamic assembler information to make intelligent decisions on handling the request. It uses this information to look into the Business Service Repository to identify all the policies that are applicable to a particular request. The context includes information such as the context identifier, which is the WS-Context identifier. The service interface which is the interface name universal resource identifier or URI of the interface which needs to be invoked. The subscription identifier is the subscription identifier URI of the user's subscription of the Business Service. This identifier can be seen in the Business Services Subscriber Manager during subscription time. The debug flag which is a boolean flag that clients can use to request the endpoint computation. This is only for SOAP requests. Lastly, additional content properties which are additional context properties can be specified by using properties of content based assertions extensions. Examples of context information are the application, business service interface, user, role or organization associated with the request.

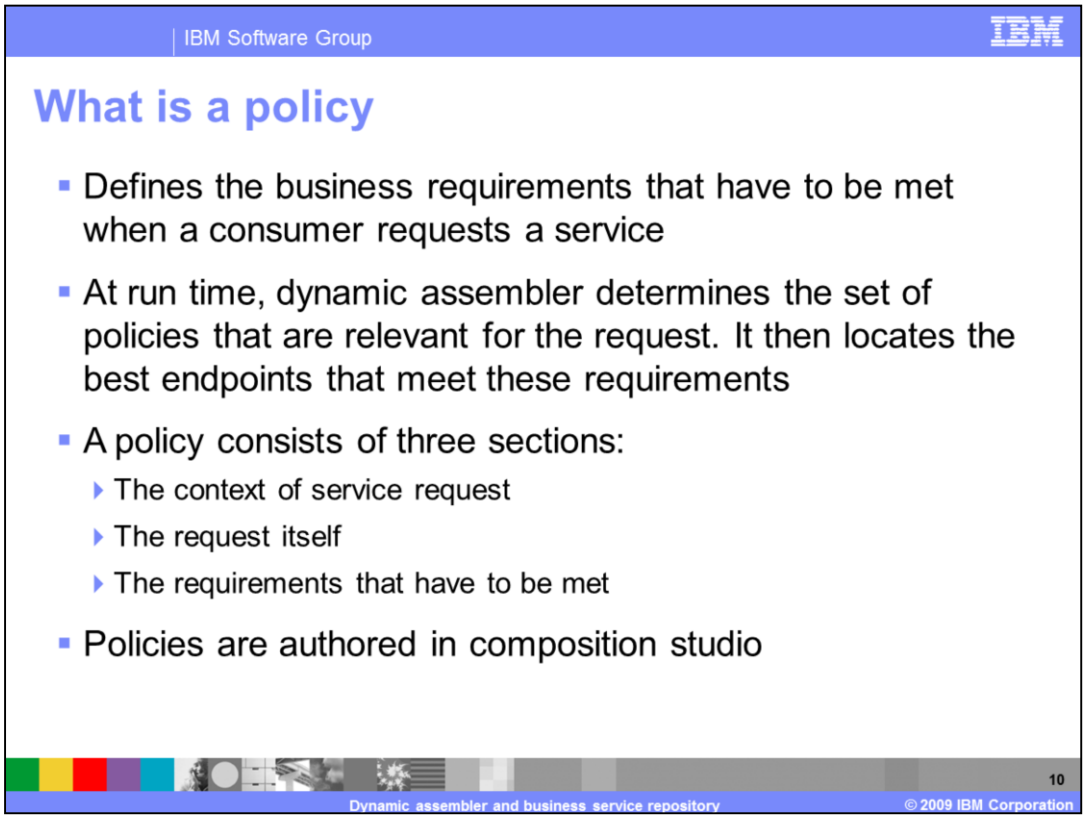



Next you look into content. By definition it is information housed in the payload from the requester. This includes both the header and body of the request message. The dynamic assembler uses this information in the selection of endpoints and execution of business policies. WebSphere Business Services Fabric provides the ability to the add content into the requesters message that is then for routing the message appropriately by the dynamic assembler. This is called content- based routing. The addition or extension to content can be done through the fabric modeling tool or through business space. Examples of content are claim value, line of business and account limit.

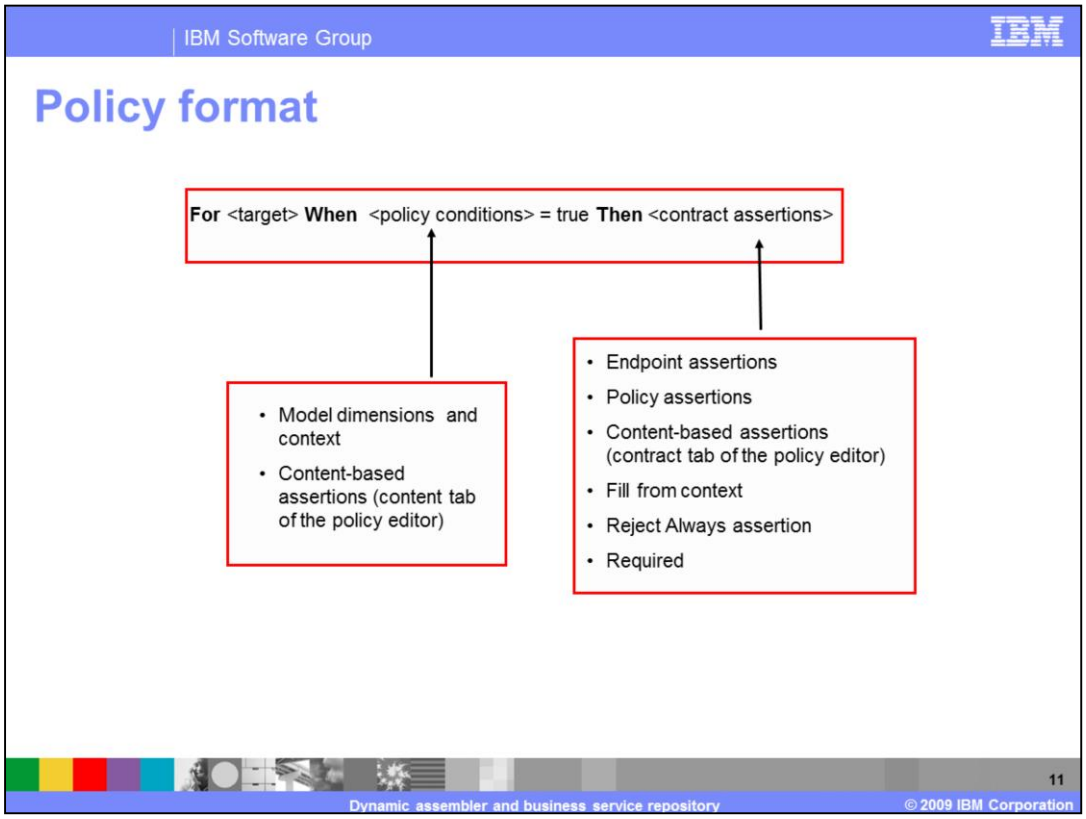



You have now covered context and content. Next you look into contract. This will give the three C's or the basics of creating business policies. Contract is defined as a set of requirements or assertions that have to be met by service provider at runtime based on the context and content. It describes the capabilities, restrictions and preferences for a composite business application or business service. The contract is defined at runtime and is a combination of the metadata and policies that are relevant for the particular circumstances. If a service provider or endpoint satisfies this set of requirements or contract it is eligible to process the incoming request. The contract can be defined explicitly or implicitly through endpoint assertions. Examples of contract is hours of operation.

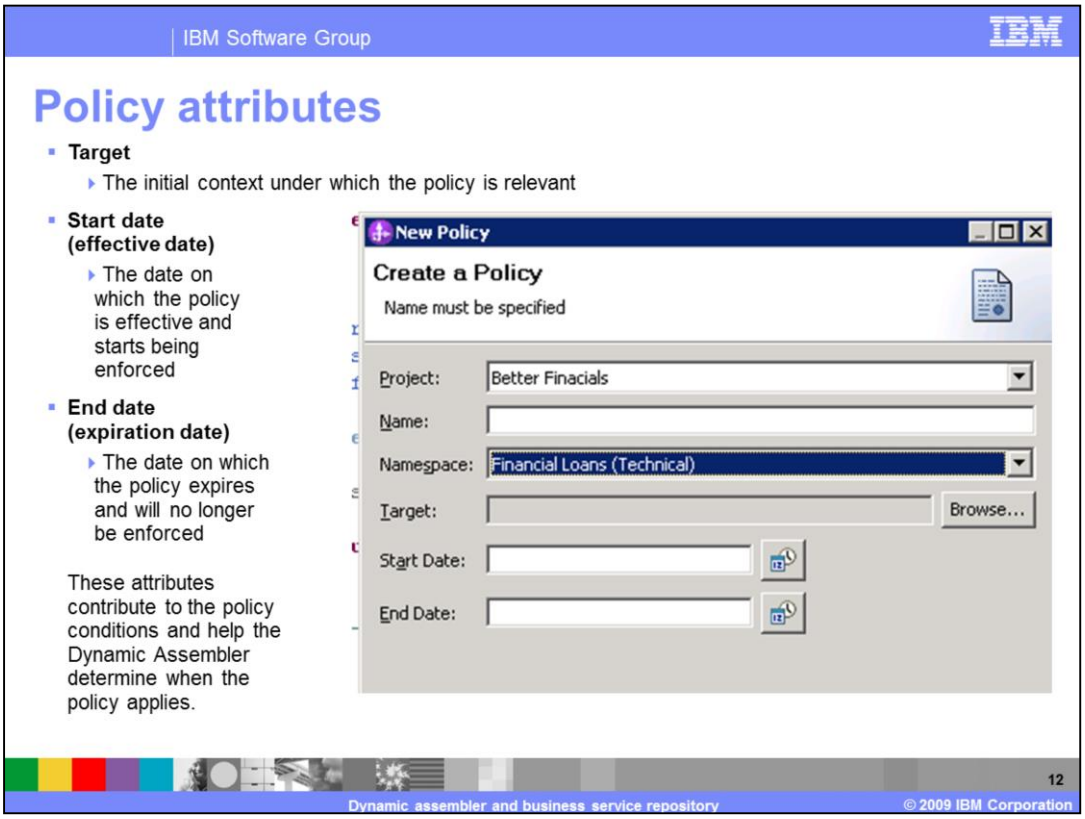
Next you shall look into the basics of a business policy and how content, context and contract help define business policies.



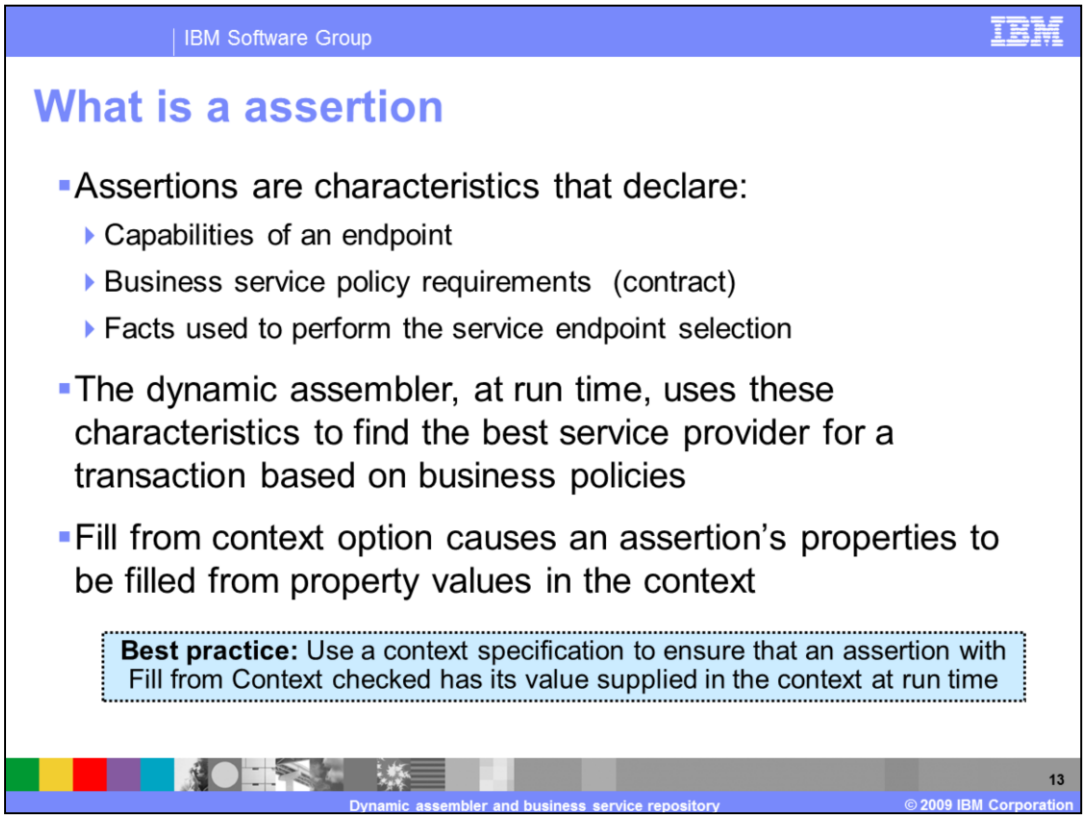

A policy in the fabric represents a key piece of domain knowledge that must be represented well, that is, it is easily understood while also having the ability to be applied directly to a given situation that has to be met when a consumer requests a service. A policy is a set of assertions that represent requirements, constraints or capabilities for a business service. For example, "a business service must cost less than five dollars per transaction," is the example of a policy. At run time, dynamic assembler determines the set of policies that are relevant for the request, and then locates the best endpoints that meet these requirements. A policy consists of three sections; the context of service request, the request payload and the requirements that have to be met. Further, each policy is targeted to a model dimension much as application suites, applications, business services, interfaces and other types of resources. These policies are authored in composition studio.



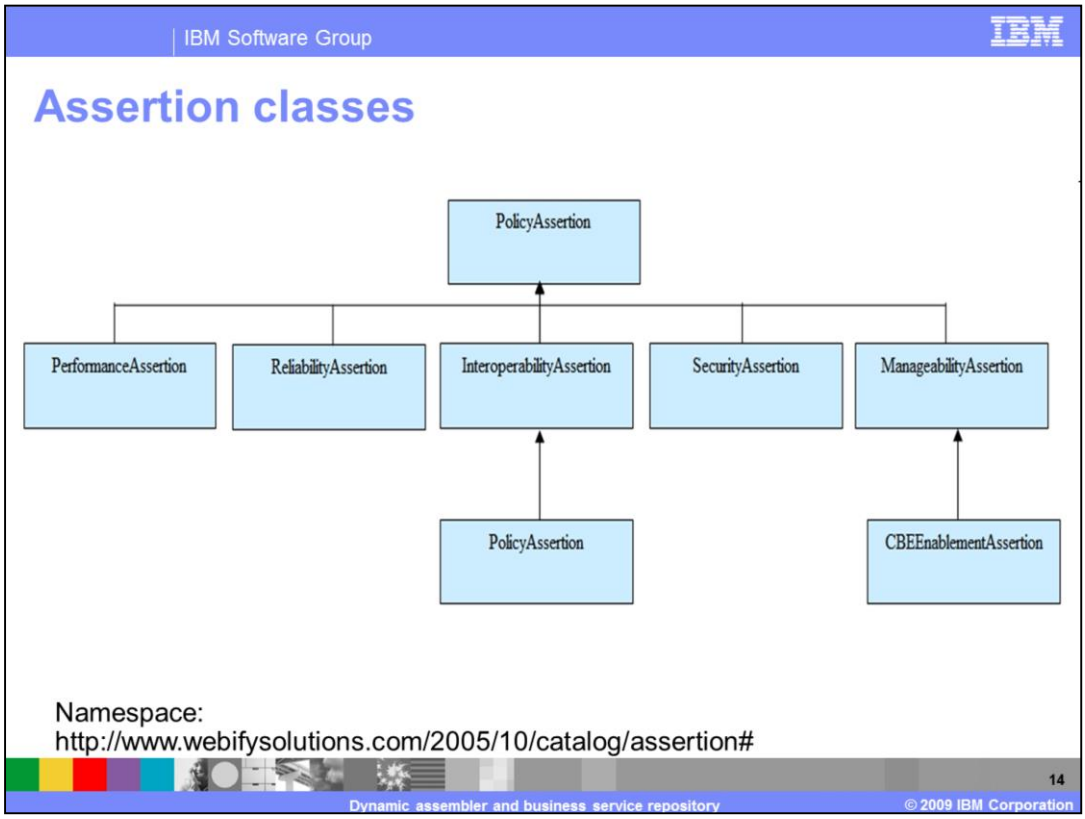

"For" and "When" and "then" are often used as programming commands. So "A policy can be thought of as a "For, when, and then" condition. That is, for a particular target, 'when' the policy conditions are satisfied, 'then' establish this contract." Now the target for the policy can be any model dimension.



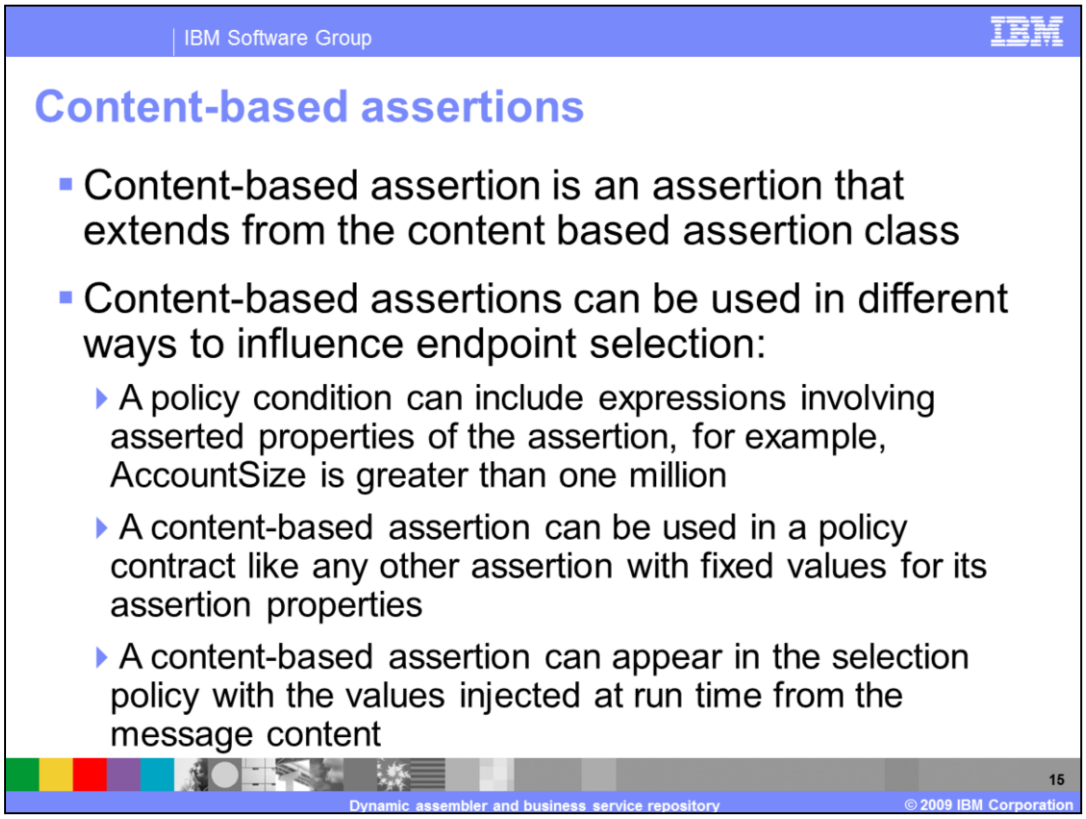

Composition studio provides a simple wizard approach to creating policies. The screen capture provided is the create policy screen. Here you define the project, name, namespace and target this policy is associated with. Optionally effective dates can be established for the policy. The dynamic assembler takes all this information into account when determining which policies applies to a given business request.



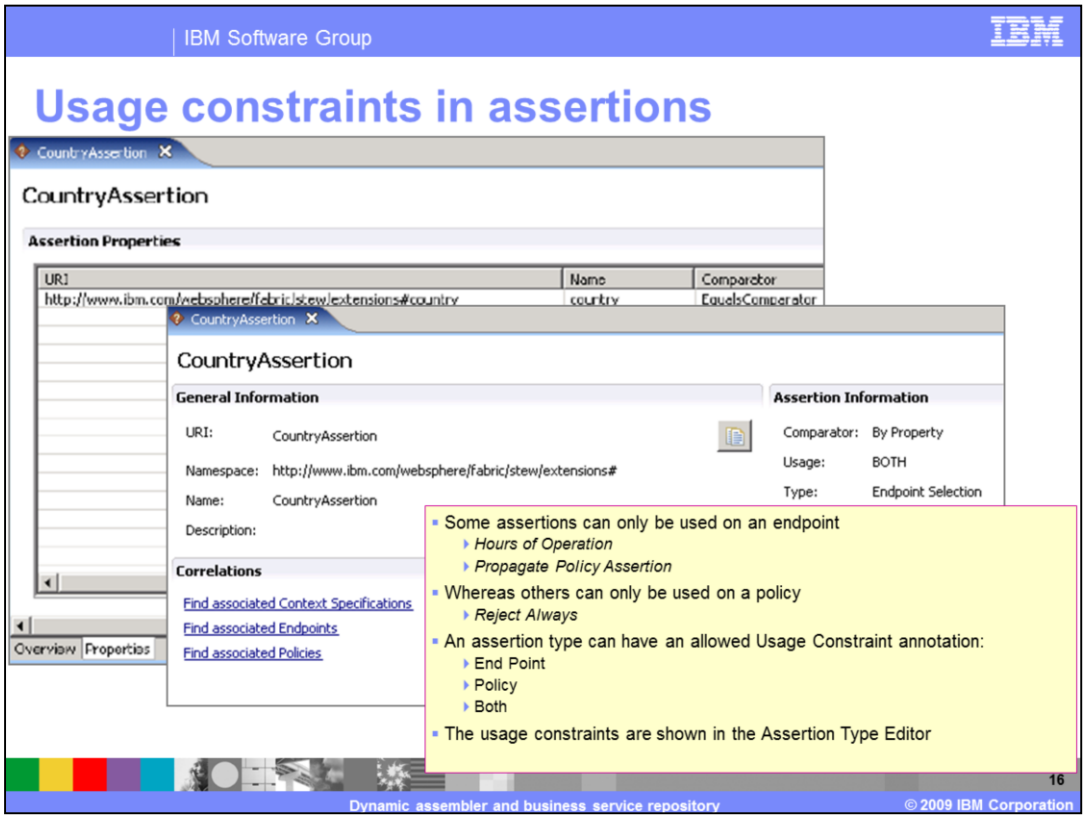

You have now seen what a policy is and the role assertions play in creating policies. Next you shall take a deeper look into assertions. Assertions describe the capabilities of an endpoint or a contract of a policy. At run time, the Business Services Dynamic Assembler uses these characteristics to find the best suited endpoint or service realization for a consumer based on their business requirements or policies. Assertions can be attached to endpoints to specify a particular characteristic of an endpoint, for example, maximum transaction time available. Assertions are also used in policy contract. During run time, if a policy is applicable then the contract is applied. When the policy target, context, and content conditions are satisfied, the contract is enforced. Assertions on a policy can either be used for endpoint selection during run time or for other non-endpoint selection purposes, like transformation assertion or data format assertion. WebSphere Business Services Fabric provides further flexibility in using assertions. The fill from context option enables an assertion's properties to be filled from property values in the context. The assertion acts like a template on a policy and has its values supplied dynamically from the context at run time. It is a recommended best practice to use a context specification to ensure that an assertion with fill from context option checked and has its value supplied in the context at run time.




You can define these assertions or capabilities along five dimensions: performance, reliability, interoperability, security, and manageability. The most common type of assertion that can be extended is the Content Based Assertion. It allows assertions based on service request message content to be defined naturally in terms of the content. Assertions in the core Business Service Model belong to the namespace http://www.webifysolutions.com/2005/10/catalog/assertion#.

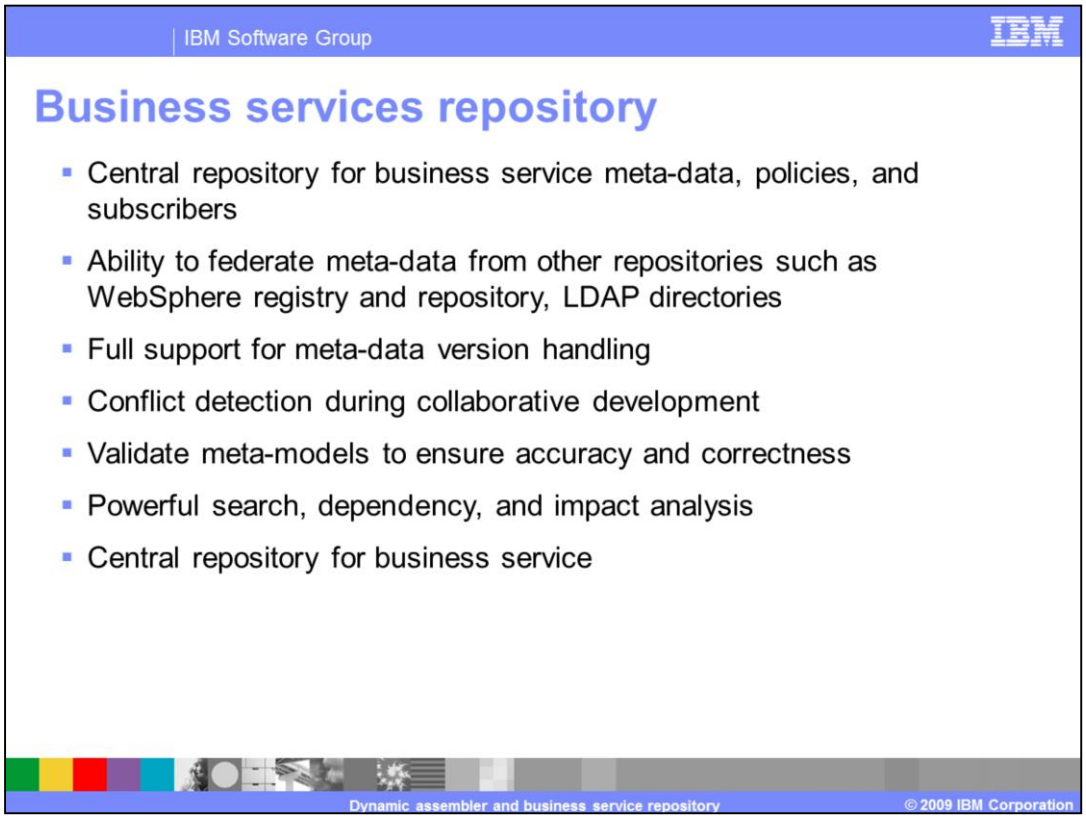



Content based assertions as you saw is the most commonly used type of assertion. It provides the ability to use information from the incoming message to make selection decision in the dynamic assembler. It extends the content based assertion class. These extensions are done by extending the fabric business model using the fabric modeling tool. From version 6.2 business users can use the business space widgets to accomplish the same task. Content based assertions can be used in different ways to influence endpoint selection such as: First, a policy condition can include expressions involving asserted properties of the assertion, for example, AccountSize is greater than one million. Second, a content-based assertion can be used in a policy contract like any other assertion with fixed values for its assertion properties. And lastly a content-based assertion can appear in the selection policy with the values injected at run time from the message content.




Certain assertions can only be used for endpoint selection or in policy but not both. Endpoint assertions are hours of operation and propagate policy assertion. Policy assertions is a reject always assertion. The exact usage for each assertions can be found in the composition studio description of the assertion in the assertion explorer. Usage can be endpoint, policy or both.


This presentation begins with providing insight into the business services repository.




The WebSphere Business Services Fabric Business Services Repository is a standards-based, enterprise SOA metadata repository for business service descriptions, subscribers and policies. It enables the rich description, discovery and federation of data across universal description, discovery and integration registries, repositories and Lightweight Directory Access Protocol systems. This module provides version control, ability to segment business services meta-data by namespace, automated rule based validation on client and server, ability to do fine-grained roll-backs, conflict detection during collaborative development, powerful search, dependency, and impact analysis.

# WebSphere Business Services Fabric and WebSphere Service Registry and Repository

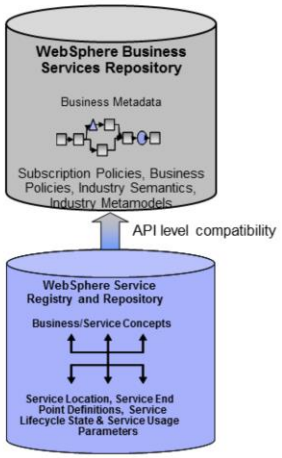
Comprehensive registry/repository for both business and technical metadata to support dynamic service selection

**WebSphere Business Service Repository**

- Includes business-related policies, semantics, metadata, and subscriptions

**WebSphere Service Registry and Repository**

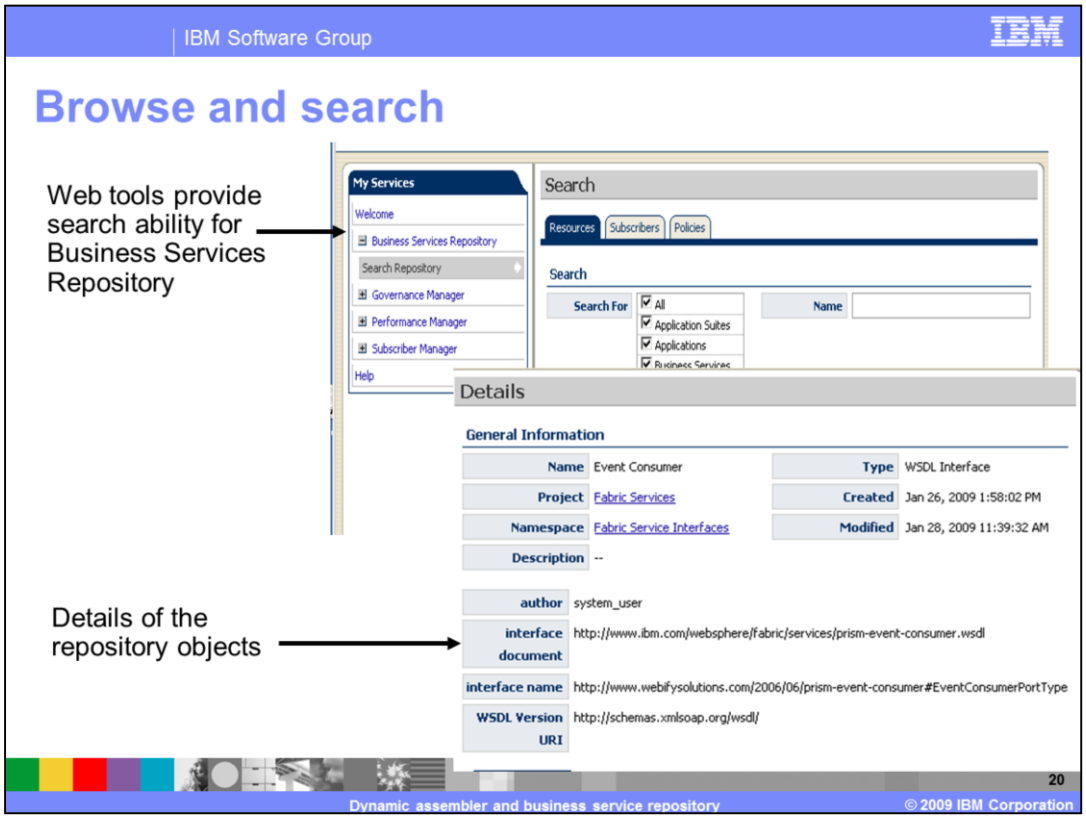
- Includes technical metadata associated with infrastructure implementation: WSDL, XSD, WS-Policy and so forth



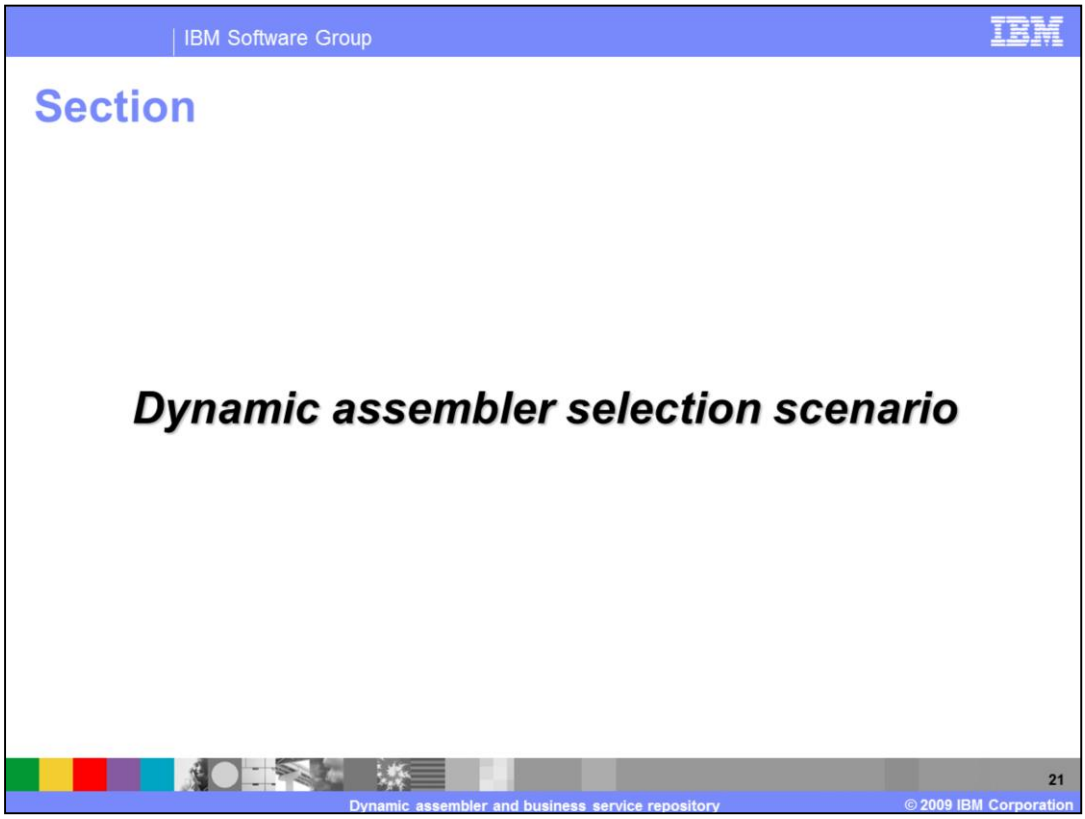




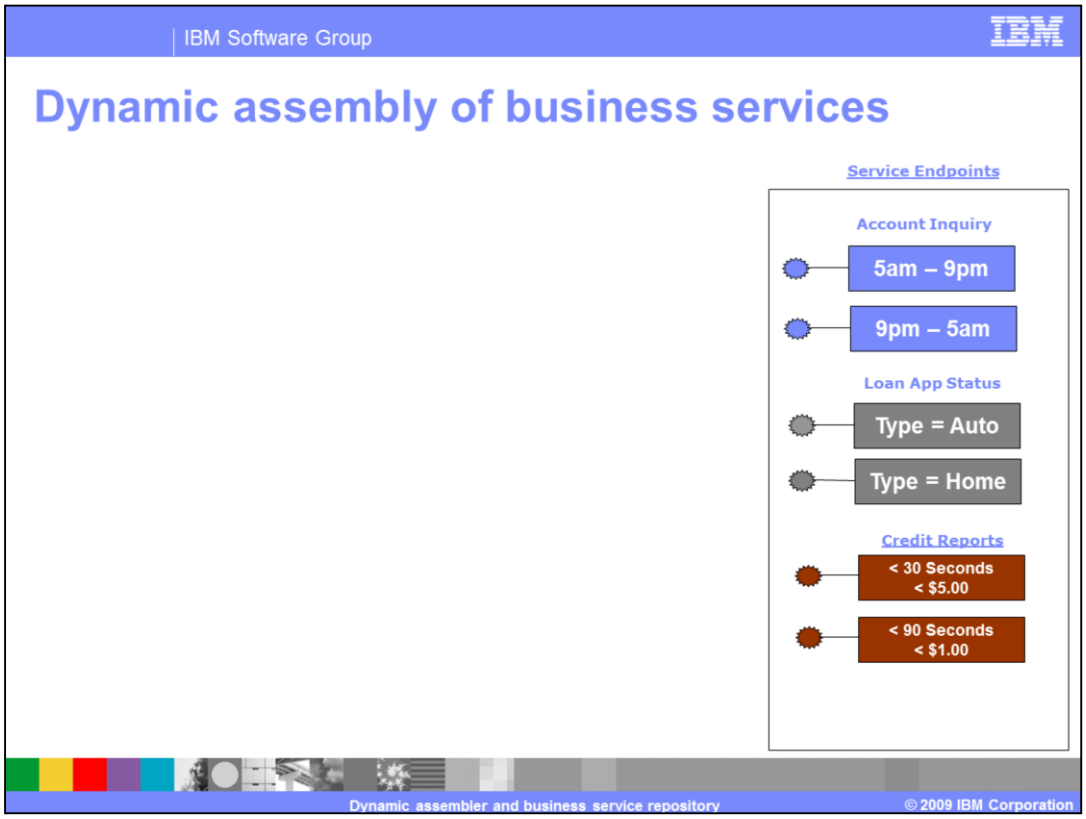
The Business Service Repository should not be confused or substituted for the WebSphere Service Registry and Repository. The Business Service Repository is local to the WebSphere Business Services Fabric and stores business related policies, semantics, metadata and subscriptions. It is not exposed publically. The WebSphere Service Registry and Repository alternatively is a central repository that is publically accessible and stores technical metadata associated with infrastructure implementation like WSDL, XSD and WS-Policy. As stated earlier the data from WebSphere Service Registry and Repository can be federated into the Business Service Repository and used while creating business policies in composition studio.




The Web tools in Web Sphere Business Services Fabric provides users the ability to search Business Services Repository and view the details of resources, subscribers and policies through powerful filtering options. After the search results are retrieved, you can view the details of any given policy, subscriber, or resource by clicking the name link in the results table. The details page helps provide you with detailed information on the repository entities that is the resources, subscribers and policies. The details page also contains the list of the objects linked to the chosen resource, subscriber or policy. From the details page, you can click links to view the author's information, or information on other projects that are associated with the selected resource, subscriber or policy.




The next set of slides will put together all the concepts that have been covered so far and apply them into a real business scenario.



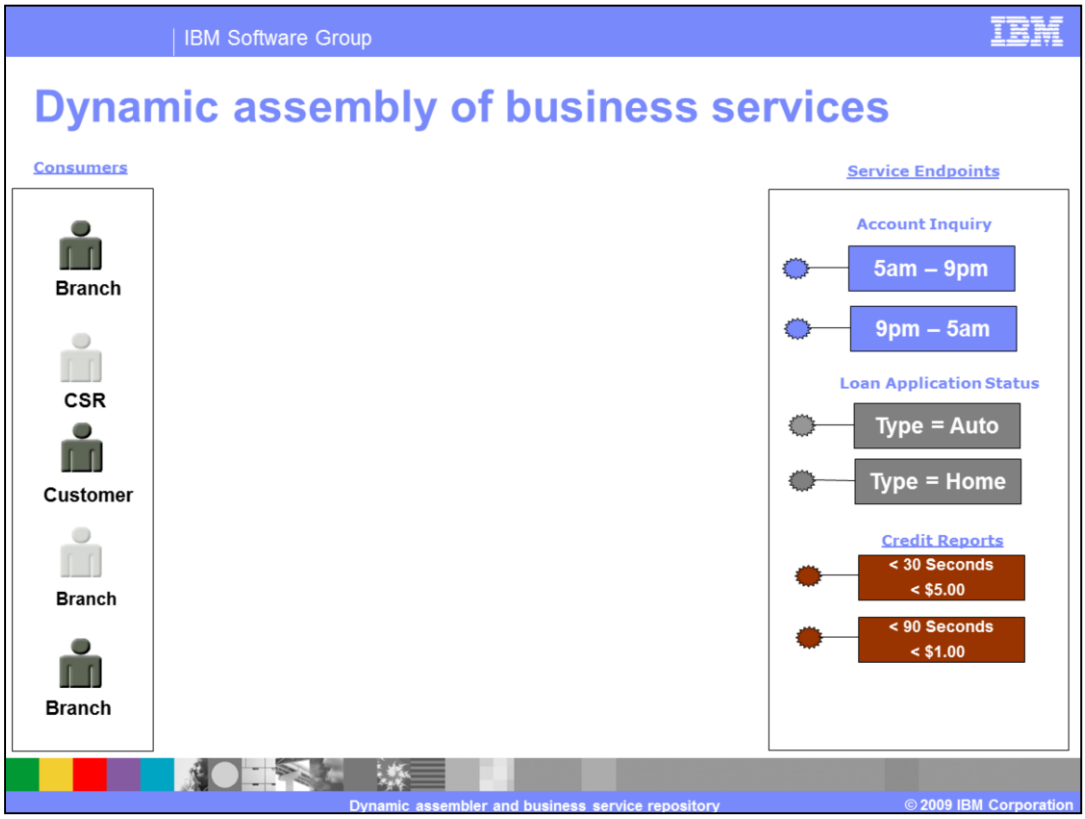

The business scenario being discussed deals with a loan application solution in the banking industry. Let's examine the landscape for assembling this SOA solution. To start off with, there needs to be some services for your discussion. Let's say there are a couple of account inquiry service endpoints, a couple of endpoints that deliver loan application status, and a couple of endpoints that provide credit report information.

One of the things with enterprise software is that services almost always come with some constraints around their use. Let's talk about some example constraints. Let's say the account inquiry service runs on a system that needs to go down a portion of the day for maintenance, and yet the business requires 24 hours a day seven days a week operations. How is that dealt with? The account inquiry service is mirrored, and by doing so – there is one service available from 9am – 5pm and another available from 5pm – 9am. That's one example of a constraint – Hours of Operation.

Another example of a constraint might be the product type that you're dealing with. You will almost certainly deliver your loan application status differently depending on whether you are dealing with you're auto or home loan product. So that is another example – Product Line or Product Type.

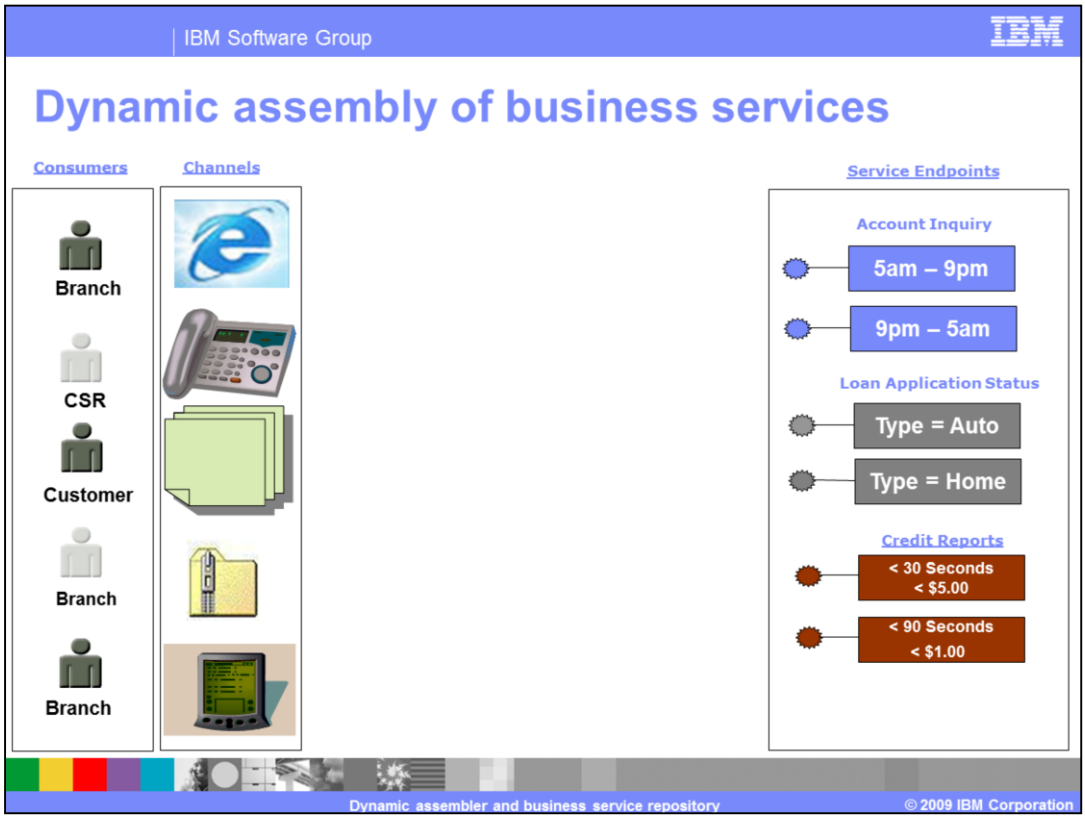
Finally, you can have some constraints around business level Performance. Let's say you have one credit report service that responds in less than 30 seconds and another that responds in less than 90 seconds. Maybe you add cost in there as well. Therefore the service responding in less than 30 seconds costs you five dollars a transaction, whereas the other costs just one dollar.

All of these are examples of constraints. For the discussion here, there is only one constraint type per service shown, but in practice you can have multiple constraints per service.



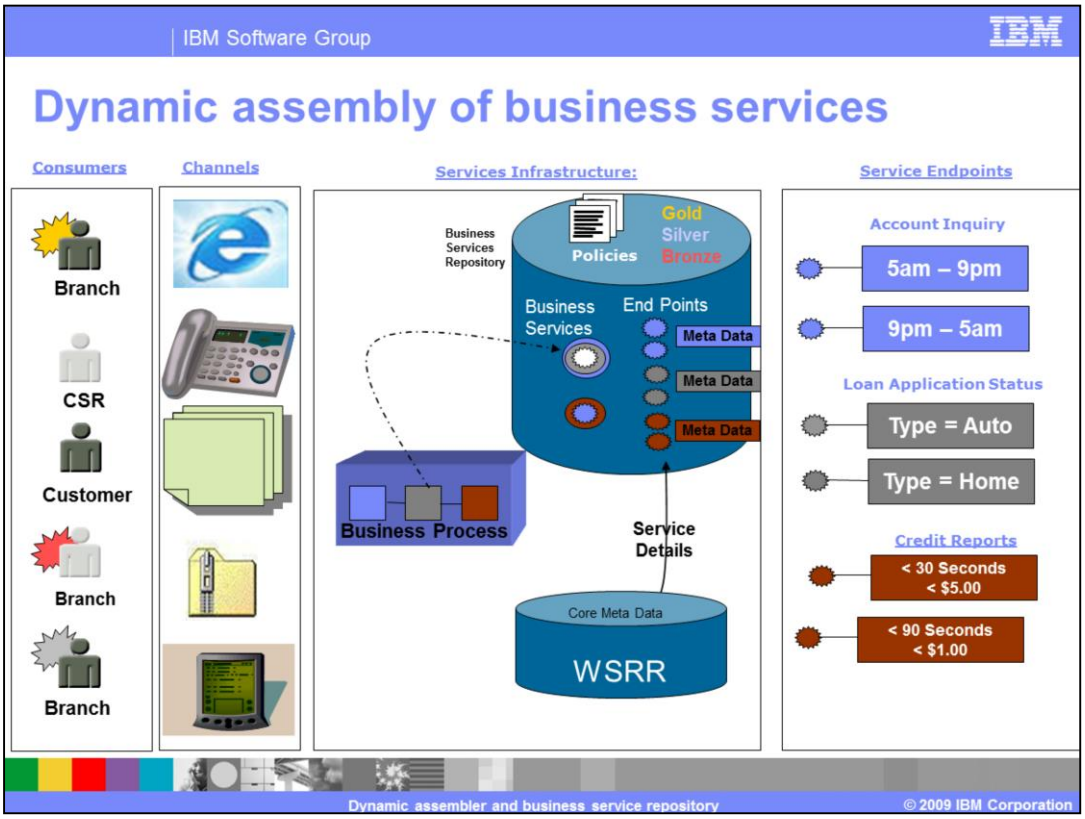

Next let's look at the potential consumers of this SOA solution. Again, this is a banking example, so let's look at some roles that make sense for a banking solution. Let's say there is a Customer role, a Branch role, and a corporate Customer Service Representative role.

The roles can also be duplicated. Let's say that there are branch users in different geographies; that should be treated differently, in that case you need a variety of different branch roles to account for that variability.




This lays out the solution consumers. The next thing that needs to be considered is the channels that the solution is delivered through. A simple application might only need to be delivered through for example a portal channel. Other examples of channels might be an Integrated Voice Response channel, a paper or fax or electronic document channel. Maybe there is a business partner gateway setup for interactions with the partner network that might have a proprietary data format channel. Last, there can be a channel for wireless interactions from a smart phone.

So, now the landscape has been laid out and there is quite a bit of variability to deal with. How does one implement a SOA solution so that it can respond to and operate against all of this variability? And what if things change over time, then how does the solution deal with that?

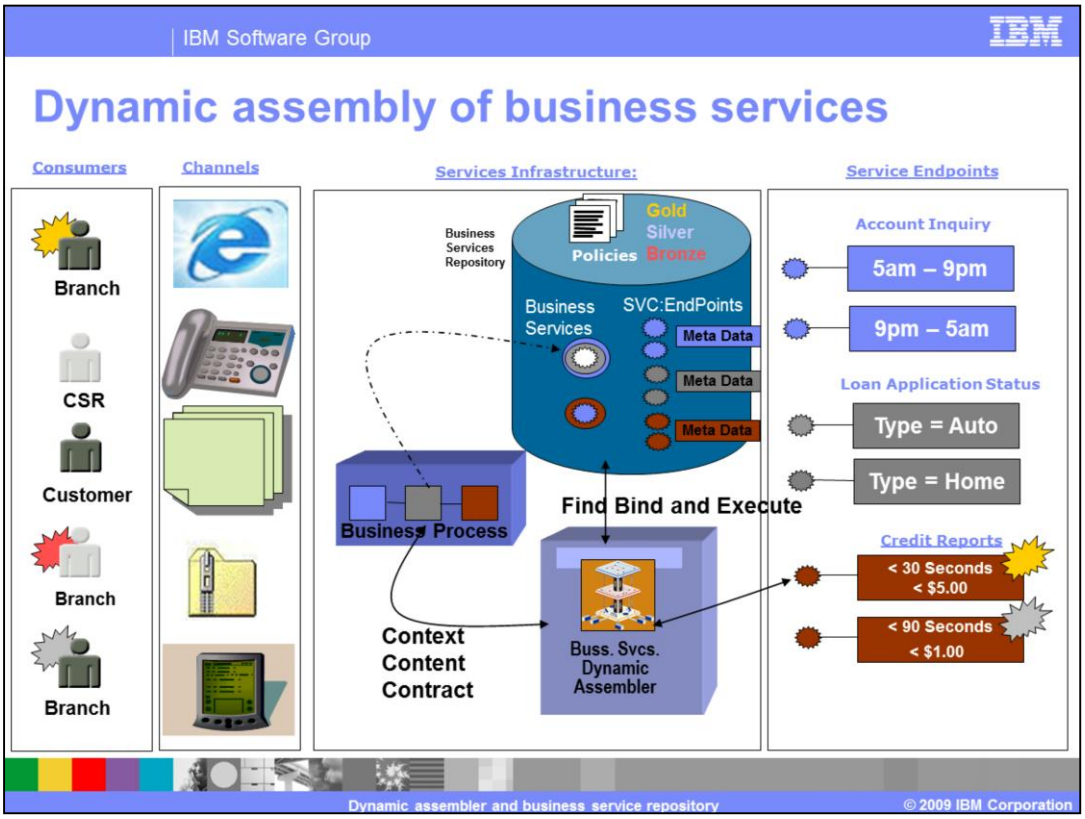



That is where you can use dynamic assembly. With the business services approach, you'll be collecting information about all of this variability and modeling it in your Business Services Repository. You can then use this information to understand how the business services are being used and which operational capabilities should be used based on different invocation contexts.

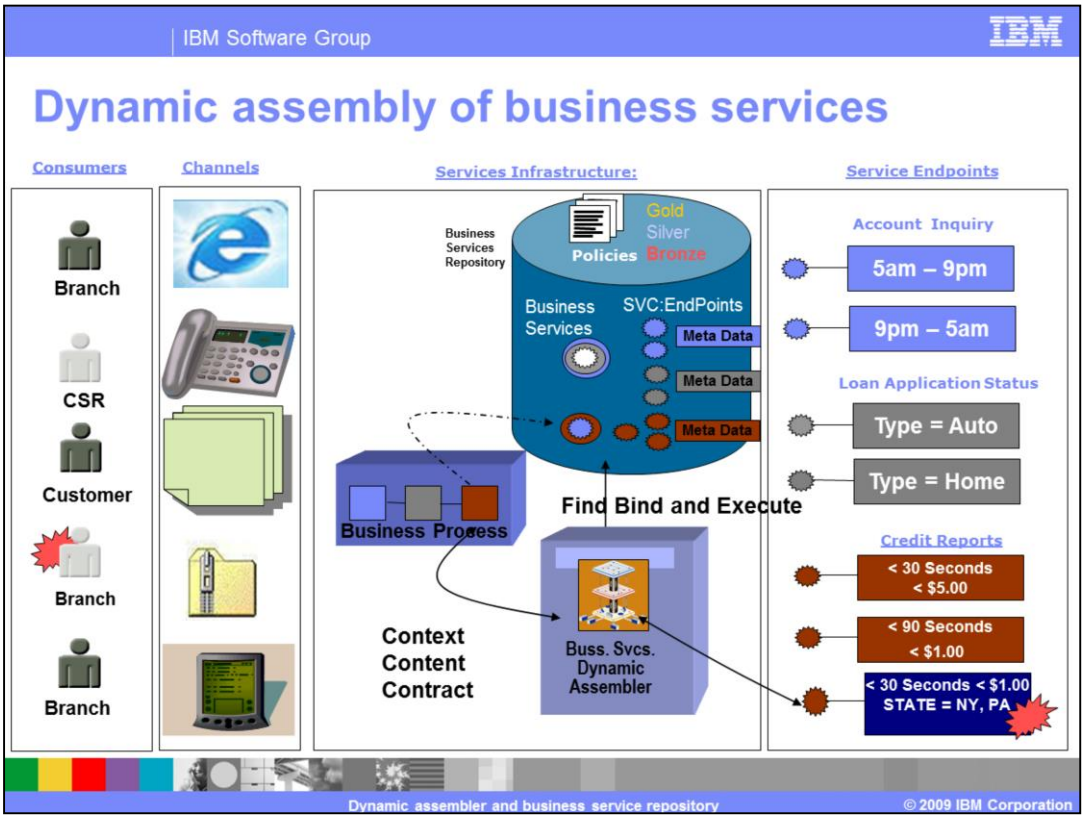
Back to the scenario, the first thing you'll want to do is gain an appraisal of all of the technical services that you're going to use in your solution. You do this by federating technical service details from WebSphere Service Registry and Repository into the Business Services Repository. Next, you want to add metadata around these technical service capabilities to describe their business level usage. Let's start by adding the constraints that were talked about earlier. Therefore you can add hours of operation for each of the account inquiry services, the line of business for each of the loan application status services and the response time for each of the credit report services. You can also add some constraints around the different channels and roles that have access to these services. Next you want to associate each fine grained end point with a higher level coarse grain business service without going in to the metadata level details. For example you have just one loan application status business service with out going into the line of business details. Another example is just one account inquiry business service without differentiating between two different hours of operations for each of the endpoints.

In addition to associating the endpoints to the business service you need to establish business policies also. For example, if you need to establish an elite access system for the branches such that certain branches can have a gold status, silver status or bronze status. Now for gold level branch that serves the high profile client you can want to provide a faster response time at a higher cost while providing a slower less expensive response time to the silver and bronze level branches. These business policies are created in the business service repository.

If you had to create a business process for the loan application status you now have the ability to create simpler business process by leveraging the metadata and business policies. The business decision points have now been abstracted and so the process will not need separate paths for example the type of loan application or the response time for the credit report service. Instead you can create a simple step business process that leverages the higher level business services with dynamic assembler binding the service endpoint based on the business policies at runtime. This is not a complete solution but it is sufficient for the discussion. Finally, you're ready to deploy and use this business process. The solution is deployed on WebSphere Process Server that contains the dynamic assembler.




Next you will observe this simpler business process at runtime and get an understanding of how the business services with the underlying dynamic assembler help adapt each business request to the right endpoint. Let's say the gold branch invokes the business process for a loan application through the portal. The first step of the request is handled by account inquiry business service. The context of this business service that is the service identifier, gold access level and the portal channel are sent to the dynamic assembler. The dynamic assembler then scans the business service repository for business policies associated with this context. It will discover the hours of operation business policy and apply that. The next step is to scan the business service repository for possible endpoints. The dynamic assembler will find two endpoints, however because of the hours of operation business policy will select only one depending on the time of the day. Hence the actual endpoint invocation is abstracted from the business process which makes the process more flexible and loosely coupled. Similarly in the next step the dynamic assembler will set the line of business metadata to auto or home from the incoming content or request payload and then invoke the appropriate endpoint.




In the end, let's take a look at how powerful this can be as things change over time. Let's take the example of a new service endpoint being added for the credit report service. This endpoint performs just the same as the existing endpoint that costs five dollars a transaction, but comes at a cost of just one dollar a transaction. Unfortunately, this service endpoint is only available for transactions that originate in Pennsylvania and New York.

With the business services fabric, you can add this new endpoint to the business services repository and describe its constraints. Once published, the endpoint is considered in invocations that are processed by the dynamic assembler. Notice you did not have to add new workflow in the business process, change code or even redeploy the business process in order to add this new endpoint and establish it's constraints.

At runtime when a branch from new york will invoke the credit report business service the dynamic assembler will route the message to this new endpoint based on the fact that it met the state constraint. You should now have a high level understanding of dynamic assembler.

# Trademarks, copyrights, and disclaimers

IBM, the IBM logo, ibm.com, and the following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

CICS WebSphere

If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of other IBM trademarks is available on the Web at "Copyright and trademark information" at http://www.ibm.com/legal/copytrade.shtml

Java, and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements or changes in the products or programs described herein at any time without notice. Any statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business. Any reference to an IBM Program Product in this document is not intended to state or imply that only that program product may be used. Any functionally equivalent program, that does not infringe IBM's intellectual property rights, may be used instead.

THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (for example, IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products.

IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.